Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide

Version 6.0
Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide

Version 6.0
Before using this document, be sure to read the information in "Notices" on page 1161.
Contents

Figures ............................................ ix

Tables ................................................ xi

About this document ............................. xvii
Revision history ..................................... xvii
Edition April 2018, CCA Support Program . xvii
Edition March 2017, CCA Support Program . xvii
Edition July 2016, CCA Support Program . xix
Releases 4.4 and 5.2 ................................. xix
Who should use this document .................. xx
Distribution-specific information ............... xx
Terminology ......................................... xxi
Hardware requirements ............................ xxi
How to use this document ......................... xxiii
Where to find more information .................. xxv
Cryptography publications ....................... xxvi
Do you have problems, comments, or suggestions? xxviii

Part 1. IBM CCA programming ..................... 1

Chapter 1. Introduction to programming for the IBM Common Cryptographic Architecture ................. 3

Available Common Cryptographic Architecture (CCA) verbs ........................................ 3
Common Cryptographic Architecture functional overview ............................................. 4
How application programs obtain service ................................................................. 8
Overscanned processing and load balancing .............................................................. 9
Domain selection capabilities ................................................................. 10
Multi-coprocessor selection capabilities .......................... 11
CPACF support ....................................... 14
Environment variables that affect CPACF usage .......................... 14
CSU_HCPUACLR .................................... 15
CSU_HCPUAPRT .................................... 15
Access control points that affect CPACF protected key operations .......................... 16
CPACF operation (protected key) .......................... 16
Using keys with CPACF, protected key .......................... 18
Using keys with CPACF, clear key or no key .......................... 18
CCA library CPACF preparation at startup .......................... 18
Interaction between the default card and use of protected key CPACF .................... 18
Using the AUTOSELECT option and the use of protected key CPACF .................... 19
Security API programming fundamentals .......................... 19
Verbs, variables, and parameters .......................... 19
Commonly encountered parameters .......................... 22
Parameters common to all verbs .......................... 22

Chapter 2. Running CCA verbs ..................... 21

The rule_array and other keyword parameters .......................... 22
Key tokens, key labels, and key identifiers .......................... 23
How to compile and link CCA application programs ............................................. 24
Using the Master Key Process (CSNBMPK) verb ............................................. 25
Building C applications using the CCA libraries ............................................. 25
Using the CCA JNI .................................. 25
Calling the CCA JNI using the Java package infrastructure .................................. 26
Entry points and data types used in the JNI ............................................. 26
JNI sample modules and sample code ............................................. 26
Preparing your Java environment ............................................. 27
Building Java applications using the CCA JNI ............................................. 28
Building the Java byte code ............................................. 28
Running the Java byte code ............................................. 28

Chapter 3. Using AES, DES, and HMAC cryptography and verbs .......................... 33

Functions of the AES, DES, and HMAC cryptographic keys ........................................ 33
Key separation ........................................ 33
Master key variant for fixed-length tokens ............................................. 34
Transport key variant for fixed-length tokens ............................................. 34
Key forms ............................................ 34
Key token ............................................. 35
Key wrapping ........................................ 37
AES key wrapping ..................................... 37
DES key wrapping ..................................... 37
Variable length token (AESKW method) ............................................. 40
Control vector ....................................... 40
Types of keys ........................................ 41
Verbs for managing AES and DES key storage files ............................................. 51
Verbs for managing the PKA key storage file and PKA keys in the cryptographic engine ............................................. 51
EC Diffie-Hellman key agreement models ............................................. 52
Improved remote key distribution ............................................. 53
Remote key loading ..................................... 53
Verbs supporting Secure Sockets Layer (SSL) ............................................. 55
Managing data integrity and message authentication ............................................. 55
Processing message authentication code ............................................. 55
Hashing functions ..................................... 56
Processing personal identification numbers ............................................. 56
Secure messaging ..................................... 57
Trusted Key Entry support ............................................. 57
Typical sequences of CCA verbs ............................................. 57
Using the CCA node and master key management verbs ............................................. 58
Summary of the CCA nodes and resource control verbs ............................................. 59
Summary of the AES, DES, and HMAC verbs ............................................. 61
PKA Key Record List (CSNDKRL) .
PKA Key Record Read (CSNDKRR) .
PKA Key Record Write (CSNDKRW)
Retained Key Delete (CSNDRKD) .
Retained Key List (CSNDRKL) . .

.
.
.
.
.

.
.
.
.
.

.
.
.
.
.

.
.
.
.
.

.
.
.
.
.

478
480
482
484
487

Chapter 12. Financial services . . . . 491
How personal identification numbers (PINs) are
used . . . . . . . . . . . . . . .
How Visa card verification values are used . .
Translating data and PINs in networks . . . .
Working with Europay-Mastercard-Visa Smart
cards . . . . . . . . . . . . . . .
PIN verbs . . . . . . . . . . . . .
ANSI X9.8 PIN restrictions . . . . . . . .
The PIN profile. . . . . . . . . . . .
Visa Format Preserving Encryption . . . . .
Authentication Parameter Generate (CSNBAPG)
Clear PIN Encrypt (CSNBCPE) . . . . . .
Clear PIN Generate (CSNBPGN) . . . . . .
Clear PIN Generate Alternate (CSNBCPA) . . .
CVV Generate (CSNBCSG) . . . . . . . .
CVV Key Combine (CSNBCKC) . . . . . .
CVV Verify (CSNBCSV) . . . . . . . . .
Encrypted PIN Generate (CSNBEPG) . . . .
Encrypted PIN Translate (CSNBPTR) . . . .
Encrypted PIN Translate Enhanced (CSNBPTRE)
Encrypted PIN Verify (CSNBPVR) . . . . .
FPE Decipher (CSNBFPED). . . . . . . .
FPE Encipher (CSNBFPEE) . . . . . . . .
FPE Translate (CSNBFPET) . . . . . . . .
PIN Change/Unblock (CSNBPCU) . . . . .
Recover PIN from Offset (CSNBPFO) . . . .
Secure Messaging for Keys (CSNBSKY). . . .
Secure Messaging for PINs (CSNBSPN) . . .
Transaction Validation (CSNBTRV) . . . . .

|
|

. 491
. 492
. 492
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.
.
.
.
.
.
.
.
.

Chapter 14. Using digital signatures
Digital Signature Generate (CSNDDSG)
Digital Signature Verify (CSNDDSV) .

.
.

.
.

.
.
.
.
.
.
.
.
.
.
.
.

Key Export to TR31 (CSNBT31X) . . .
TR31 Key Import (CSNBT31I) . . . .
TR31 Key Token Parse (CSNBT31P) . .
TR31 Optional Data Build (CSNBT31O)
TR31 Optional Data Read (CSNBT31R) .

492
493
495
497
502
508
511
515
518
523
526
531
535
539
545
555
560
568
575
583
591
595
599
603

.
.

. 675
. 680

Chapter 15. Managing PKA
cryptographic keys . . . . . . . . . 689
PKA Key Generate (CSNDPKG) .
PKA Key Import (CSNDPKI) . .

.
.

.
.

.
.

.
.

.
.

. 689
. 696

.
.
.
.
.
.
.
.

.
.
.
.
.
.
.
.

699
710
713
720
722
734
738
744

.
.
.
.
.

.
.
.
.
.

.
.
.
.
.

.
.
.
.
.

755
781
804
808
811

Chapter 17. Utility verbs . . . . . . . 817
Code Conversion (CSNBXEA) .

.

.

.

.

.

.

. 817

Part 3. Reference information . . . 823
Chapter 18. Return codes and reason
codes. . . . . . . . . . . . . . . 825
Return codes .
Reason codes .
Reason codes
Reason codes
Reason codes
Reason codes
Reason codes

.
.
that
that
that
that
that

. . . . . . . . . .
. . . . . . . . . .
accompany return code 0 .
accompany return code 4 .
accompany return code 8 .
accompany return code 12
accompany return code 16

.
.
.
.
.

825
825
826
827
827
844
845

Chapter 19. Key token formats . . . . 847

609
610
610
617
623
630
637
649
653
659
662
668

675

.
.
.
.
.
.
.
.

Chapter 16. TR-31 symmetric key
management verbs . . . . . . . . . 755

Chapter 13. Financial services for DK
PIN methods. . . . . . . . . . . . 609
Weak PIN table. . . . . . . . . . .
DK PIN methods . . . . . . . . . .
DK Deterministic PIN Generate (CSNBDDPG)
DK Migrate PIN (CSNBDMP) . . . . . .
DK PAN Modify in Transaction (CSNBDPMT)
DK PAN Translate (CSNBDPT) . . . . .
DK PIN Change (CSNBDPC) . . . . . .
DK PIN Verify (CSNBDPV). . . . . . .
DK PRW Card Number Update (CSNBDPNU)
DK PRW CMAC Generate (CSNBDPCG) . .
DK Random PIN Generate (CSNBDRPG) . .
DK Regenerate PRW (CSNBDRP). . . . .

PKA Key Token Build (CSNDPKB) . . .
PKA Key Token Change (CSNDKTC) . .
PKA Key Translate (CSNDPKT) . . . .
PKA Public Key Extract (CSNDPKX) . .
Remote Key Export (CSNDRKX) . . . .
Trusted Block Create (CSNDTBC) . . .
Public Infrastructure Certificate (CSNDPIC)
Public Infrastructure Manage (CSNDPIM) .

|
|

AES internal fixed-length key token . . . . .
Token validation value . . . . . . . .
DES internal key token . . . . . . . . .
DES external key token . . . . . . . . .
External RKX DES key tokens . . . . . . .
DES null key token . . . . . . . . . .
RSA public key token . . . . . . . . .
RSA private key token . . . . . . . . .
RSA private external key token . . . . .
RSA private internal key token . . . . .
RSA private key token, 1024-bit
Modulus-Exponent . . . . . . . . .
RSA private key token, 1024-bit
Modulus-Exponent format with OPK section .
RSA private key token, 4096-bit
Modulus-Exponent . . . . . . . . .
RSA private key, 4096-bit Modulus-Exponent
format with AES encrypted OPK section . .
RSA private key, 2048-bit Chinese Remainder
Theorem . . . . . . . . . . . . .
RSA private key, 4096-bit Chinese Remainder
Theorem with OPK . . . . . . . . .
RSA private key, 4096-bit Chinese Remainder
Theorem format with AES encrypted OPK
section . . . . . . . . . . . . .

.
.
.
.
.
.
.
.
.
.

847
849
849
850
851
852
853
853
854
855

. 857
. 859
. 861
. 863
. 865
. 867

. 869

Contents

v


Chapter 21. Control vectors and changing control vectors with the Control Vector Translate verb. 989
Control vector table. 989
Control-vector-base bit maps. 991
Key Form Bits, fff. 995
Specifying a control-vector-base value. 995
Changing control vectors with the Control Vector Translate verb. 1000
Providing the control information for testing the control vectors. 1000
Mask array preparation. 1000
Selecting the key-half processing mode. 1002
When the target key-token CV is null. 1004
Control vector translate example. 1004

Chapter 22. PIN formats and algorithms. 1005
PIN notation. 1005
PIN block formats. 1005
PIN extraction rules. 1008
IBM PIN algorithms. 1010
VISA PIN algorithms. 1017

Chapter 23. Cryptographic algorithms and processes. 1021
Cryptographic key-verification techniques. 1021
Master-key verification algorithms. 1021
SHA-1 based master-key verification method. 1021
z/OS-based master-key verification method. 1022
SHA-256 based master-key verification method. 1022
Asymmetric master key MDC-based verification method. 1022
Key-token verification patterns. 1022
CCA DES-key verification algorithm. 1023
Encrypt zeros AES-key verification algorithm. 1023
Encrypt zeros DES-key verification algorithm. 1024
Modification Detection Code calculation. 1024
Ciphering methods. 1026
General data-encryption processes. 1027
Single-DES and Triple-DES encryption algorithms for general data. 1027
ANSI X3.106 Cipher Block Chaining (CBC) method. 1027
ANSI X9.23 cipher block chaining. 1029
Triple-DES ciphering algorithms. 1030
MAC calculation methods. 1033
ANSI X9.9 MAC. 1033
ANSI X9.19 Optional Procedure 1 MAC. 1034
EMV MAC. 1034
ISO 16609 TDES MAC. 1034
Keyed-hash MAC (HMAC). 1035
RSA key-pair generation. 1035
Multiple decipherment and encipherment. 1036
Multiple encipherment of single-length keys. 1037
Multiple decipherment of single-length keys. 1037
Multiple encipherment of double-length keys. 1038
Multiple decipherment of double-length keys. 1039

Chapter 20. Key forms and types used in the Key Generate verb. 985
Generating an operational key. 985
Generating an importable key. 985
Generating an exportable key. 985
Examples of single-length keys in one form only. 985
Examples of OPIM single-length, double-length, and triple-length keys in two forms. 986
Examples of OPEX single-length, double-length, and triple-length keys in two forms. 986
Examples of IMEX single-length and double-length keys in two forms. 987
Examples of EXEX single-length and double-length keys in two forms. 987

RSA private key token, 1024-bit
Modulus-Exponent internal format for cryptographic coprocessor feature. 872
RSA variable Modulus-Exponent token. 873
ECC key token. 874
PKA key tokens. 880
PKA null key token. 880
PKA key token sections. 881
Integrity of PKA private key sections containing an encrypted RSA key. 884
Number representation in PKA key tokens. 884
PKA public-key certificate section. 885
HMAC key token. 888
Variable-length symmetric key tokens. 889
General format of a variable-length symmetric key token. 889
AES CIPHER variable-length symmetric key token. 904
AES MAC variable-length symmetric key token. 911
HMAC MAC variable-length symmetric key token. 920
AES EXPORTER and IMPORTER variable-length symmetric key token. 927
AES PINPROT, PINCALC, and PINPRW variable-length symmetric key token. 938
AES DESUSECV variable-length symmetric key token. 947
AES DKYGENKY variable-length symmetric key token. 951
AES SECMMSG variable-length symmetric key token. 960
TR-31 optional block data. 967
Trusted blocks. 968
Trusted block organization. 969
Trusted block integrity. 970
Number representation in trusted blocks. 971
Trusted block sections. 971
Trusted block section X’1’. 972
Trusted block section X’12’. 972
Trusted block section X’12’ subsections. 974
Trusted block section X’13’. 979
Trusted block section X’14’. 980
Trusted block section X’14’ subsections. 980
Trusted block section X’15’. 982
Figures

1. CCA security API, access layer, and cryptographic engine ............... 5
2. CPACF ........................................... 17
3. Control Vector Generate and Key Token Build CV keyword combinations for fixed-length DES key tokens .............. 48
4. PKA key management ........................................... 73
5. Key Token Build2 keyword combinations for AES CIPHER keys .............. 270
6. Key Token Build2 keyword combinations for AES DKYGENKY keys .......... 273
7. Key Token Build2 keyword combinations for AES EXPORTER keys .......... 277
8. Key Token Build2 keyword combinations for AES IMPORTER keys .......... 281
9. Key Token Build2 keyword combinations for AES MAC keys .......... 285
10. Key Token Build2 keyword combinations for HMAC MAC keys .......... 288
11. Key Token Build2 keyword combinations for AES PINCALC keys .......... 291
12. Key Token Build2 keyword combinations for AES PINPROT keys .......... 293
13. Key Token Build2 keyword combinations for AES PINPRW keys .......... 296
14. Key Token Build2 keyword combinations for AES SECMG keys .......... 298
15. Control vector base bit map (common bits and key-encrypting keys) .......... 992
16. Control vector base bit map (data operation keys) .......... 993
17. Control vector base bit map (PIN processing keys and cryptographic variable-encrypting keys) .......... 994
18. Control vector base bit map (key generating keys) .......... 995
19. Control Vector Translate verb mask_array processing .......... 1002
20. Control Vector Translate verb ........................................... 1003
21. ISO-3 PIN-block format ........................................... 1006
22. 3624 PIN generation algorithm ........................................... 1101
23. GBP PIN generation algorithm ........................................... 1012
24. PIN-Offset generation algorithm ........................................... 1013
25. PIN verification algorithm ........................................... 1015
26. GBP PIN verification algorithm ........................................... 1017
27. PVV generation algorithm ........................................... 1018
28. Triple-DES data encryption and decryption ........................................... 1027
29. Enciphering using the ANSI X3.106 CBC method ........................................... 1028
30. Deciphering using the CBC method ........................................... 1029
31. Enciphering using the ANSI X9.23 method ........................................... 1030
32. Deciphering using the ANSI X9.23 method ........................................... 1030
33. Triple-DES CBC encryption process ........................................... 1031
34. Triple-DES CBC decryption process ........................................... 1032
35. EDE algorithm ........................................... 1032
36. DED process ........................................... 1033
37. MAC calculation method ........................................... 1034
38. Multiple decipherment of single-length keys ........................................... 1037
39. Multiple decipherment of single-length keys ........................................... 1038
40. Multiple decipherment of double-length keys ........................................... 1039
41. Multiple decipherment of double-length keys ........................................... 1040
42. Multiple decipherment of triple-length keys ........................................... 1041
43. Multiple decipherment of triple-length keys ........................................... 1042
44. Access-control-point list example ........................................... 1078
45. Role data structure example ........................................... 1079
46. Syntax, sample routine in C ........................................... 1093
47. Syntax, sample routine in Java ........................................... 1098
48. ACP setting for a role ........................................... 1131
49. Querying an adapter's compliance mode (part 1) ........................................... 1133
50. Querying an adapter's compliance mode (part 2) ........................................... 1134
Tables

- New verbs for CCA Releases 6.0 and 5.3  
- Updated verbs for CCA Releases 6.0 and 5.3  
- New verbs for CCA Releases 4.4 and 5.2  
- Updated verbs for CCA Releases 4.4 and 5.2  
- Verbs that ignore AUTOSELECT  
- Key types  
- Key subtypes specified by the rule_array  
- DES control vector key-subtype and key-usage keywords  
- Access control points used by ATM remote key loading  
- Summary of CCA nodes and resource control verbs  
- Summary of CCA AES, DES, and HMAC verbs  
- Summary of PKA verbs  
- TR-31 symmetric key management verbs  
- Keywords for Access Control Maintenance control information  
- Meaning of the name parameter  
- Meaning of the output_data parameter  
- Keywords for Access Control Tracking control information  
- Role tracking data header format  
- GETDATA output_data format  
- GETSTATE output_data format  
- Keywords for Cryptographic Facility Query control information  
- Cryptographic Facility Query information returned in the rule_array  
- Output data format for the GETCOMPD keyword  
- Output data format for the SIZEWPIN keyword  
- Output data format for the STATDECT keyword  
- Output data format for STATICS operational key parts  
- Output data format for STATICSB operational key parts  
- Output data format for STATICSE operational key parts  
- Output data format for STATICSX operational key parts  
- Output data format for STATKPR operational key parts  
- Output data format for STATOAHL keyword  
- Output data format for STATVKPR operational key parts  
- Output data format for STATWPIN keyword  
- CSUACFQ weak PIN entry structure (type X'30')  
- Keywords for Cryptographic Resource Allocate control information  
- Keywords for Cryptographic Resource Deallocate control information  
- Keywords for Key Storage Initialization control information  
- Keywords for Log Query control information  
- Meaning of log_number_or_level for the requested service (keyword)  
- Meaning of log_data_length for the requested service (keyword)  
- Required commands for the Log Query verb  
- Keywords for Master Key Process control information  
- Keywords for Random Number Tests control information  
- Keywords for Multiple Clear Key Import control information  
- Keywords for Control Vector Generate control information  
- Keywords for Control Vector Translate control information  
- Keywords for Diversified Key Generate control information  
- Keyword for Diversified Key Generate2 control information  
- Generating and generated key tokens  
- CSNDEDH skeleton key-tokens  
- CSNDEDH concatenation string format for DERIV01  
- DERIV01 supplied public information  
- CSNDEDH concatenation string format for DERIV02  
- Keywords for EC Diffie-Hellman control information  
- Valid key bit lengths and minimum curve size  
- Keywords for the Key Generate verb key_form parameter  
- Key length values for the Key Generate verb  
- Key Generate - key lengths for each key type  
- Keywords for Key Generate, valid key types and key forms for a single key  
- Keywords for Key Generate, valid key types and key forms for a key pair  
- Keywords for Key Generate2 control information  
- Keywords and associated algorithms for key_type_1/2 parameter  
- ACPs supporting DK keys for the Key Generate2 verb  
- Key Generate2 key_type and key_form keywords for one AES or HMAC key  
- Key Generate2 key_type and key_form keywords for a pair of AES or HMAC keys  
- AES KEK strength required for generating an HMAC key under an AES KEK  

© Copyright IBM Corp. 2007, 2018
67. CSNBKGN2 access control requirements for
   DK enabled keys .......................... 232
68. Keywords for Key Part Import control
   information .................................. 239
69. Keywords for Key Part Import2 control
   information ................................. 242
70. Key Test parameter changes ...................... 245
71. Key Test GENERATE outputs and VERIFY
   inputs ...................................... 246
72. Keywords for Key Test control information ... 247
73. Keywords for Key Test2 control information .. 250
74. Required commands for CSNBKTYT2 .......... 253
75. Keywords for Key Test Extended control
   information ................................. 256
76. Keywords for Key Token Build control
   information .................................. 260
77. Keywords for Key Token Build2 ............... 266
78. Related key usage fields when Key Token
   Build2 builds a DKYGENKY key-token .... 269
79. Key Token Build2 rule array keywords for
   AES CIPHER keys ........................... 270
80. Key Token Build2 rule array keywords for
   AES DKYGENKY keys ....................... 274
81. Key Token Build2 rule array keywords for
   AES EXPTOR keys ........................... 278
82. Key Token Build2 rule array keywords for
   AES IMPORTE keys ........................... 282
83. Key Token Build2 rule array keywords for
   AES MAC keys .................................. 285
84. Key Token Build2 rule array keywords for
   HMAC MAC keys .............................. 288
85. Key Token Build2 rule array keywords for
   AES PINCALC keys ........................... 291
86. Key Token Build2 rule array keywords for
   AES PINPROT keys ........................... 294
87. Key Token Build2 PINPRW related key words .. 296
88. Key Token Build2 SECMMSG related key words ... 299
89. Keywords for Key Token Change control
   information .................................. 302
90. Keywords for Key Token Change2 control
   information .................................. 305
91. Keywords for Key Token Parse control
   information .................................. 309
92. Keywords for Key Token Parse2 .................. 314
93. Keywords for Key Translate2 control
   information .................................. 324
94. Keywords for PKA Decrypt control
   information .................................. 328
95. Keywords for PKA Encrypt control
   information .................................. 331
96. Keywords for Restrict Key Attribute control
   information .................................. 338
97. Keywords for Random Number Generate
   form parameter ............................... 341
98. Keywords for Random Number Generate Long
   control information ........................... 343
99. Keywords for Symmetric Key Export control
   information .................................. 345
100. AES EXPORTER strength required for
    exporting an HMAC key under an AES
    EXPORTER ................................... 349
101. Minimum RSA modulus strength required to
    contain a PKOAEP2 block when exporting an
    AES key .................................... 349
102. Minimum RSA modulus length to adequately
    protect an AES key .......................... 349
103. Keywords for Symmetric Key Export with
    Data control information ..................... 350
104. Required commands for the Symmetric Key
    Export with Data verb ....................... 352
105. Keywords for Symmetric Key Generate
    control information .......................... 354
106. Keywords for Symmetric Key Import control
    information ................................... 359
107. Symmetric Key Import2 key-wrapping
    method of target key when system default is
    ECB (Legacy) ................................. 363
108. Symmetric Key Import2 key-wrapping
    method of target key when system default is
    CBC (Enhanced) ............................... 364
109. Keywords for Symmetric Key Import2 control
    information ................................... 366
110. PKCS#1 OAEP encoded message layout
    (PKOAEP2) .................................. 368
111. Keywords for Unique Key Derive control
    information ................................... 371
112. Valid Control Vectors for Derived Keys ....... 375
113. Required commands for the Symmetric Key
    Export with Data verb ....................... 376
114. Derivation variants .......................... 376
115. Keywords for Decipher control information .... 383
116. Keywords for Encipher control information .... 388
117. Keywords for Symmetric Algorithm Decipher
    control information .......................... 392
118. Keywords for Symmetric Algorithm Encipher
    control information .......................... 400
119. Keywords for Cipher Text Translate2 control
    information ................................... 406
120. Restrictions for cipher_text_in_length and
    cipher_text_out_length ...................... 411
121. Cipher Text Translate2 key usage ................ 414
122. Keywords for HMAC Generate control
    information ................................... 420
123. Hash methods and HMAC key sizes ............ 421
124. Keywords for HMAC Verify control
    information ................................... 423
125. Keywords for MAC Generate control
    information ................................... 428
126. Keywords for MAC Generate2 control
    information ................................... 431
127. CSNBMGN2 access control points ............... 432
128. Keywords for MAC Verify control
    information ................................... 435
129. Keywords for MAC Verify2 control
    information ................................... 438
130. Required commands for CSNBMV2R2 ............ 440
131. Keywords for MDC Generate control
    information ................................... 443
132. Keywords for One-Way Hash control
    information ................................... 445
133. Valid symbols for the name token ............... 451
134. Key labels that are not valid .................... 451
About this document

The presented information describes how to use a variety of services for cryptography and data-security provided in the Common Cryptographic Architecture (CCA).

CCA functions described in this edition apply to the CCA host library (RPM) version 6.0, shortly referred to as CCA (Release) 6.0. The new host library 6.0 is tested on CCA firmware versions 4.4.x, 5.3.x, and 6.0.x. Note that there is no CCA firmware or host library version 4.5.

The CCA functions perform cryptographic operations using the following adapters in CCA coprocessor mode:
- IBM® Crypto Express6 (CEX6C), Feature Code 0893
- IBM Crypto Express5 (CEX5C), Feature Code 0890
- IBM Crypto Express4 (CEX4C), Feature Code 0865
- IBM Crypto Express3 (CEX3), Feature Code 0864

See “Concurrent installations” on page 1117 for details.

This publication is for planning and programming purposes only.

The CCA host software provides an application programming interface through which applications request secure, high-speed cryptographic services from the hardware cryptographic features.

This publication continues to document CCA Releases 5.0 up to 5.2. Where CCA Releases 6.0 or 5.3 have been changed or enhanced from the previously documented CCA Release 5.2, these changes are indicated with revision marks.

Applications linked with prior CCA host software will continue to function if the CCA host software is upgraded in place. However, IBM always recommends full testing of upgrades before implementing production roll-out.

For the supported environments and product ordering information, see:

http://www.ibm.com/security/cryptocards/

Revision history

Track the changes of this document for each CCA Support Program release.

Edition April 2018, CCA Support Program Releases 6.0 and 5.3

This edition describes the IBM CCA Basic Services API for Release 6.0 including those for Release 5.3.

For Release 6.0, changes to the CCA API include the following new verbs:

<table>
<thead>
<tr>
<th>Verb</th>
<th>Service name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNDPIC</td>
<td>“Public Infrastructure Certificate (CSNDPIC)” on page 738</td>
<td>Managing PKA cryptographic keys</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 2007, 2018
Table 1. New verbs for CCA Releases 6.0 and 5.3 (continued)

<table>
<thead>
<tr>
<th>Verb</th>
<th>Service name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNDPIM</td>
<td>“Public Infrastructure Manage (CSNDPIM)” on page 744</td>
<td>Managing PKA cryptographic keys</td>
</tr>
</tbody>
</table>

The following verbs provide new keywords:

Table 2. Updated verbs for CCA Releases 6.0 and 5.3

<table>
<thead>
<tr>
<th>Verb</th>
<th>Service name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSUACFQ</td>
<td>“Cryptographic Facility Query (CSUACFQ)” on page 100</td>
<td>Using the CCA nodes and resource control verbs</td>
</tr>
<tr>
<td>CSNBKYT2</td>
<td>“Key Test2 (CSNBKYT2)” on page 249</td>
<td>Managing AES and DES cryptographic keys</td>
</tr>
<tr>
<td>CSNBKT2</td>
<td>“Key Token Build (CSNBKT2)” on page 259</td>
<td>Managing AES and DES cryptographic keys</td>
</tr>
<tr>
<td>CSNBKT2</td>
<td>“Key Token Build2 (CSNBKT2)” on page 264</td>
<td>Managing AES and DES cryptographic keys</td>
</tr>
<tr>
<td>CSNBKTR2</td>
<td>“Key Translate2 (CSNBKTR2)” on page 323</td>
<td>Managing AES and DES cryptographic keys</td>
</tr>
<tr>
<td>CSNDDSG</td>
<td>“Digital Signature Generate (CSNDDSG)” on page 675</td>
<td>Using Digital Signatures</td>
</tr>
<tr>
<td>CSNDDSV</td>
<td>“Digital Signature Verify (CSNDDSV)” on page 680</td>
<td>Using Digital Signatures</td>
</tr>
<tr>
<td>CSNDPKB</td>
<td>“PKA Key Token Build (CSNDPKB)” on page 699</td>
<td>Managing PKA cryptographic keys</td>
</tr>
<tr>
<td>CSNDPKT</td>
<td>“PKA Key Translate (CSNDPKT)” on page 713</td>
<td>Managing PKA cryptographic key</td>
</tr>
</tbody>
</table>

In addition, the following changes apply to CCA Release 6.0:

- When configured as a CCA coprocessor, a CEX6C adapter is capable of running in PCI-HSM 2016 compliance mode. Where applicable, CCA has updated its services to produce or use keys that adhere to the PCI-HSM 2016 standards. Currently, only internal, fixed-length TDES key tokens can be compliant-tagged. You can migrate existing CCA applications to make use of the PCI-HSM 2016 compliant services.
- CCA 6.0 now supports X.509 certificates with CSNDDSV based on a Public Key Infrastructure (PKI) hosted in the adapter. The PKI is managed from a Trusted Key Entry workstation version 9.0 or later.
- The new CSNDPIC verb can create a PKCS #10 Certificate Signing Request based on your input private key, suitable to have a Certificate Authority to create a signed certificate of the public key associated with the private key in the token.
- CCA now supports export of AES CIPHER keys to CPACF. This is the first strongly typed key that can be exported as a CPACF protected key. Export is controlled by a new management bit in the AES CIPHER variable key token structure. Therefore to exploit this feature, new AES CIPHER key tokens need to be created which have the management bit enabled. Use the CSNBKT2 verb to create the skeleton key tokens with this bit enabled. Use the CSNBKGN2 verb to populate a key to the skeleton.
- The panel.exe utility now presents a more usable command line interface, which, among others, supports services for the PCI-HSM 2016 compliance mode, for certificates, master keys, and roles.
• Domain selection capabilities are added in the CCA host library. In the application environment, a user may select which domain is to be used by that application. The domain used must be one of the domains configured available for the LPAR.

• The default wrapping method for DES key tokens was ORIGINAL for CEX*C firmware before CCA 6.0. Now with CEX6C, the default wrapping method is ENHANCED (see also “DES key wrapping” on page 37).

The following change applies to CCA Release 5.3:

• For the CSNDDSG and CSNDDSV verbs, a new digital-signature hash method PKCS-PSS is available. This method supports the PKCS #1 v2.2 standard for the RSA Probabilistic Signature Scheme (RSA-PSS). RSA-PSS is based on the RSA crypto system and provides increased security assurance.

The CSNDDSV verb can now verify hashes using a variety of SHA-<xxx> methods. Also, CSNDDSV supports new certificate validation methods.

Edition March 2017, CCA Support Program Releases 4.4 and 5.2

This publication is the second edition for CCA Support Program Releases 5.2 and 4.4. It now includes installation instructions for the Ubuntu DEB package for IBM CCA.

Edition July 2016, CCA Support Program Releases 4.4 and 5.2

This edition describes the IBM CCA Basic Services API for Release 5.2, including those for Release 4.4.

For Release 5.2, changes to the CCA API include the following new verbs:

<table>
<thead>
<tr>
<th>Verb entry-point name</th>
<th>Service name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSUAACM</td>
<td>“Access Control Maintenance (CSUAACM)” on page 89</td>
<td>Using the CCA nodes and resource control verbs</td>
</tr>
<tr>
<td>CSUAACT</td>
<td>“Access Control Tracking (CSUAACT)” on page 93</td>
<td>Using the CCA nodes and resource control verbs</td>
</tr>
</tbody>
</table>

The following verbs provide new keywords:

<table>
<thead>
<tr>
<th>Entry-point</th>
<th>Service name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNBKGN2</td>
<td>“Key Generate2 (CSNBKGN2)” on page 221</td>
<td>Managing AES and DES cryptographic keys</td>
</tr>
<tr>
<td>CSNBKYT2</td>
<td>“Key Test2 (CSNBKYT2)” on page 249</td>
<td>Managing AES and DES cryptographic keys</td>
</tr>
<tr>
<td>CSNDEDH</td>
<td>“EC Diffie-Hellman (CSNDEDH)” on page 195</td>
<td>Managing AES and DES cryptographic keys</td>
</tr>
<tr>
<td>CSNBKTB2</td>
<td>“Key Token Build2 (CSNBKTB2)” on page 264</td>
<td>Managing AES and DES cryptographic keys</td>
</tr>
<tr>
<td>CSNBKTP2</td>
<td>“Key Token Parse2 (CSNBKTP2)” on page 311</td>
<td>Managing AES and DES cryptographic keys</td>
</tr>
<tr>
<td>CSNBSAD</td>
<td>“Symmetric Algorithm Decipher (CSNBSAD)” on page 390</td>
<td>Protecting data</td>
</tr>
<tr>
<td>CSNBSAE</td>
<td>“Symmetric Algorithm Encipher (CSNBSAE)” on page 397</td>
<td>Protecting data</td>
</tr>
</tbody>
</table>
Table 4. Updated verbs for CCA Releases 4.4 and 5.2 (continued)

<table>
<thead>
<tr>
<th>Entry-point</th>
<th>Service name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNDPKB</td>
<td>“PKA Key Token Build (CSNDPKB)” on page 699</td>
<td>Managing PKA cryptographic keys</td>
</tr>
<tr>
<td>CSNDPKG</td>
<td>“PKA Key Generate (CSNDPKG)” on page 689</td>
<td>Managing PKA cryptographic keys</td>
</tr>
</tbody>
</table>

CSNKBKT: The keywords CV-KEK and NOCV-KEK introduced with CCA 5.0 produced unexpected results during further testing. These issues are fixed in CCA 5.2.

In addition, the following change applies to CCA Release 5.2

- For the JNI version of the verbs, starting with this release, a new data type \texttt{hikmNativeLong} is replacing the old type \texttt{hikmNativeInteger}. Both types inherit from an abstract class \texttt{hikmNativeNumber}. Thus, type \texttt{hikmNativeInteger} is still supported, so you can run existing applications with this deprecated data type. However, start using type \texttt{hikmNativeLong} for new applications instead, because \texttt{hikmNativeInteger} may be removed in the future.

Applications linked with prior CCA host software will continue to function if the CCA host software is upgraded in place. However, IBM always recommends full testing of upgrades before implementing production roll-out.

A note on the document structure: There is a new topic called “RSA private key token” on page 853 that describes in its subtopics the RSA private key tokens for both the external and internal format, combined into one table for each token type.

Who should use this document

This document is intended for application programmers who are responsible for writing application programs that use the security application programming interface (API) to access cryptographic functions.

Distribution-specific information

Common Cryptographic Architecture for Linux on IBM Z® provides support for the CEX6S feature since Release 6.0. In order to use the full set of CCA, starting with Release 6.0, a Linux distribution with support for the CEX6S feature is required.

For the most current distribution-specific information about CCA Release 6.0 and prior releases, always refer to the Release Notes on the software-package selection page. The contained information may be more up-to-date than in this publication.

CCA 6.0 is supported on the following distributions:

- Red Hat Enterprise Linux 6.9 (with update)
- Red Hat Enterprise Linux 7.4
- Red Hat Enterprise Linux 8
- SUSE Linux Enterprise Server 11 SP4 (with update)
- SUSE Linux Enterprise Server 12 SP3
- SUSE Linux Enterprise Server 15
- Ubuntu 16.04 (with update)

To exploit the full functionality of CCA 6.0 introduced with CEX6S, you need an IBM z14™ with a CEX6S installed.
Note: In order to fully exploit the CCA 6.0 functionality, CEX6S toleration by the Linux on Z operating system is sufficient. All distributions listed above provide sufficient CEX6S support on the required hardware.

For restrictions and recommendations about the usage of CCA prior to release 6.0 with Linux on Z distributions, refer to:

IBM 4767 CCA software releases

Note: Only 64-bit versions of this software are provided. 31-bit support is not provided.

Terminology

The following terms are used in this document for CCA releases and features.

CEX3C
An IBM 4765 Crypto Express3 feature (CEX3), configured in CCA coprocessor mode. This publication does not cover other modes of the CEX3.

CEX4C
An IBM 4765 Crypto Express4 feature (CEX4S), configured in CCA coprocessor mode. This publication does not cover other modes of the CEX4.

CEX5C
An IBM 4767 Crypto Express5 feature (CEX5S), configured in CCA coprocessor mode. This publication does not cover other modes of the CEX5.

CEX6C
An IBM 4768 Crypto Express6 feature (CEX6S), configured in CCA coprocessor mode. This publication does not cover other modes of the CEX6.

CEX*C
Either the CEX3C, CEX4C, CEX5C, CEX6C, or (if plural) any combination of these.

Hardware requirements

In order to make use of the verbs provided in the Common Cryptographic Architecture (CCA) API for Linux on Z, your hardware must meet certain minimum requirements.

This is the minimum supported hardware configuration:

- zEC12
- One CEXC4 feature, with one CEXC4 adapter mapped to the z/VM® image or LPAR that uses it. The CEXC4 must have CCA 4.2.0z or greater firmware loaded, configured in co-processor mode.
- If you plan to use a Trusted Key Entry (TKE) workstation, you must have a TKE V9.0 to see CEX6C adapters and to exploit PCI-HSM 2016 compliance mode.
  - You need a TKE 6.0 workstation or higher to support CEX3C adapters.
  - You need a TKE 7.2 workstation or higher in order to see supported CEX4C adapters.
  - A TKE V8.0 workstation is required to manage CEX5C adapters.
For more detailed information about required TKE versions for accessing the various CEX*C features, see “CEX6C information” on page 1117.

This is the maximum supported hardware configuration:

- IBM z14
  - 16 CEX6S adapters total in the machine

This hardware configuration is also supported:

- IBM z13®
  - 16 CEX5S adapters total in the machine
  - all 16 adapters may also be mapped to an LPAR, up to 85 total LPARs
  - there is one adapter per feature code.
- IBM zEnterprise® EC12
  - two CEX4C adapter features, with up to 4 CEX4C adapters mapped to a single z/VM image or LPAR.

See “Concurrent installations” on page 1117 for details about a mixed environment of CEX6S and previous CEX*C features. Note that CEX4Cs are only supported by TKE 7.2 or later if the Linux driver reports them as CEX4s. If you are running in toleration mode, and the Linux driver reports them as CEX3Cs, then TKE 6.0 can manage them as CEX3Cs.

To determine whether a Crypto Express adapter available to Linux on Z is a CEX3C, a CEX4C, a CEX5C, or a CEX6C, see “Listing CCA coprocessors” on page 13.

Apply the following APAR fixes on the appropriate z/VM systems to achieve the applicable cryptographic coprocessor support:

VM64656
Introduces Crypto Express3 support.

VM64727
Fixes problem with shared coprocessors.

VM64793
Introduces protected key CPACF support.

On z/VM 6.2, apply the following APAR fixes:

VM65007
Introduces Crypto Express4 support.

VM65577
Introduces Crypto Express5 support.

On z/VM 6.3, apply the following APAR fixes:

VM65577
Introduces Crypto Express5 support.

VM65942
Introduces Crypto Express6 support.

On z/VM 6.4, apply the following APAR fix:

VM65942
Introduces Crypto Express6 support.
How to use this document

Read an overview of the information in each section of this document.

For encryption, CCA supports Advanced Encryption Standard (AES), Data Encryption Standard (DES), public key cryptography (PKA or RSA), and Elliptic Curve Cryptography (ECC). These are very different cryptographic systems. Additionally, CCA provides APIs for generating and verifying Message Authentication Codes (MACs), Hashed Message Authentication Codes (HMACs), hashes, and PINS, as well as other cryptographic functions.

Part 1, “IBM CCA programming,” on page 1 includes the following chapters:

• Chapter 1, “Introduction to programming for the IBM Common Cryptographic Architecture,” on page 3 describes the programming considerations for using the CCA verbs. It also explains the syntax and parameter definitions used in verbs. Information about concurrency is also provided.
• Chapter 2, “Running CCA verbs in PCI-HSM 2016 compliance mode,” on page 31 describes the support for compliance-tagged tokens introduced with CCA 6.0. It gives links to further related information in this document, and discusses the new warning mode to aid installations in detecting migration concerns.
• Chapter 3, “Using AES, DES, and HMAC cryptography and verbs,” on page 33 gives an overview of AES, DES, and HMAC cryptography, and provides general guidance information on how these verbs use different key types and key forms.
• Chapter 4, “Introducing PKA cryptography and using PKA verbs,” on page 71 introduces Public Key Algorithm (PKA) support and describes programming considerations for using the CCA PKA and ECC verbs, such as the PKA key token structure and key management.
• Chapter 5, “TR-31 symmetric key management,” on page 79 introduces TR-31 support and how CCA uses an IBM-defined optional block in a TR-31 key block.
• Chapter 6, “Understanding and managing master keys,” on page 83 provides information about master key management.

Part 2, “CCA verbs,” on page 87 includes the following topics:

• Chapter 7, “Using the CCA nodes and resource control verbs,” on page 89 describes using the CCA resource control verbs.
• Chapter 8, “Managing AES, DES, and HMAC cryptographic keys,” on page 165 describes the verbs for generating and maintaining DES and AES cryptographic keys, the Random Number Generate verb (which generates 8-byte random numbers), the Random Number Generate Long verb (which generates up to 8192 bytes of random content), and the Secure Sockets Layer (SSL) security protocol. This chapter also describes utilities to build DES and AES tokens, generate and translate control vectors, and describes the PKA verbs that support DES and AES key distribution.
• Chapter 9, “Protecting data,” on page 379 describes the verbs for enciphering and deciphering data.
• Chapter 10, “Verifying data integrity and authenticating messages,” on page 417 describes the verbs for generating and verifying Message Authentication Codes (MACs), generating and verifying Hashed Message Authentication Codes (HMACs), generating Modification Detection Codes (MDCs), and generating hashes (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, MD5, RIPEMD-160).
• Chapter 11, “Key storage mechanisms,” on page 449 describes the use of key storage, key tokens, and associated verbs.
Chapter 12, “Financial services,” on page 491 describes the verbs for use in support of finance-industry applications. This includes several categories.

- Verbs for generating, verifying, and translating personal identification numbers (PINS).
- Verbs that generate and verify VISA card verification values and American Express card security codes.
- Verbs to support smart card applications using the EMV (Europay MasterCard Visa) standards.

Chapter 13, “Financial services for DK PIN methods,” on page 609 describes the verbs for PIN methods and requirements for financial services specified by the German Banking Industry Committee, Deutsche Kreditwirtschaft (DK).

Chapter 14, “Using digital signatures,” on page 675 describes the verbs that support using digital signatures to authenticate messages.

Chapter 15, “Managing PKA cryptographic keys,” on page 689 describes the verbs that generate and manage PKA keys.

Chapter 16, “TR-31 symmetric key management verbs,” on page 755 describes the verbs used to manage TR-31 key blocks and TR-31 functions.

Chapter 17, “Utility verbs,” on page 817 describes the utility verb CSNBXEA which is provided for code conversion.

Part 3, “Reference information,” on page 823 includes the following information:

- Chapter 18, “Return codes and reason codes,” on page 825 explains the return and reason codes returned by the verbs.
- Chapter 19, “Key token formats,” on page 847 describes the formats for AES, DES internal, external, and null key tokens, for PKA public, private external, and private internal key tokens containing Rivest-Shamir-Adleman (RSA) information, PKA null key tokens, ECC key tokens, HMAC key tokens, Transaction Validation Values (TVVs), and trusted blocks.
- Chapter 20, “Key forms and types used in the Key Generate verb,” on page 985 describes the key forms and types used by the Key Generate verb.
- Chapter 21, “Control vectors and changing control vectors with the Control Vector Translate verb,” on page 989 contains a table of the default control vector values that are associated with each key type and describes the control information for testing control vectors, mask array preparation, selecting the key-half processing mode, and an example of using the Control Vector Translate verb.
- Chapter 22, “PIN formats and algorithms,” on page 1005 describes the PIN notation, formats, extraction rules, and algorithms.
- Chapter 23, “Cryptographic algorithms and processes,” on page 1021 describes various ciphering and key verification algorithms, as well as the formatting of hashes and keys.
- Chapter 24, “Access control points and verbs,” on page 1047 lists the access control points and their corresponding verbs.
- Chapter 25, “Access control data structures,” on page 1073 documents the data structures that are used in the access control system.
- Chapter 26, “Using verbs and applications in PCI-HSM 2016 compliance mode,” on page 1081 informs about how to manage keys, verbs and applications that should apply the PCI-HSM 2016 compliance standard.
- Chapter 27, “Sample verb call routines,” on page 1093 contains sample verb call routines, both in C and Java, that illustrates the practical application of CCA verb calls.
• Chapter 28, “Initial system set-up tips,” on page 1103 includes tips to help you set up your system for the first time.

• Chapter 29, “CCA installation instructions,” on page 1107 includes RPM installation, configuration, and uninstallation instructions.

• Chapter 30, “Coexistence of CEX6C and previous CEX*C features,” on page 1117 includes information about using various versions of CEX*C features in the same system, and other restrictions.

• Chapter 31, “Utilities,” on page 1119 describes the ivp.e and panel.exe utilities.

• Chapter 32, “Security API command and sub-command codes,” on page 1135 contains an alphabetical list of security API command and sub-command codes returned by the output rule-array for option STATDIAG of the Cryptographic Facility Query verb.

• Chapter 33, “openCryptoki support,” on page 1139 provides information about openCryptoki which is an open source implementation of the Cryptoki API as defined by the industry-wide PKCS #11 Cryptographic Token Interface Standard.

• Chapter 34, “List of abbreviations,” on page 1151 contains definitions of abbreviations used.

Where to find more information

You can find other CCA product publications that might be helpful when developing applications and systems that exploit the IBM Crypto Express features.

While there is substantial commonality in the API supported by the CCA products, and while this document seeks to guide you to the subset supported by Linux on Z, other individual product publications might provide further insight into potential issues of compatibility.

IBM 4768 Cryptographic Coprocessors - CEX6S

All of the IBM 4768-related publications can be obtained from the product web site:

IBM PCIeCC3 library

There are the following categories:

HSM CEX6S general documentation
The listed manuals apply to the IBM CEX6S Cryptographic Coprocessor.

PCieCC3 Enterprise PKCS #11 (EP11)
This manual describes the capabilities of the cryptographic application programming interface (API) provided with the Enterprise PKCS#11 (EP11) Library for Linux on Z.

Related products
This category includes information about
• the IBM CPACF Enablement crypto feature
• the IBM Cryptographic Coprocessor Facility (CCF)

IBM 4767 Cryptographic Coprocessors - CEX5S

All of the IBM 4767-related publications can be obtained from the product web site:

HSM 4767 - IBM PCIeCC2 library

There are the following categories:
HSM 4767 general documentation

These manuals apply to the IBM 4767 PCIe Cryptographic Coprocessor. Among these is the IBM 4767 PCIe Cryptographic Coprocessor Installation Manual which describes how to install CCA.

PCleCC2 CCA Support Program

The CCA Basic Services and Secure Key Solution manuals describe the capabilities of the security application programming interface (API) provided with the CCA Support Program.

IBM 4767 custom programming

This category includes programming information for creating applications that perform within the IBM 4767 cryptographic coprocessor.

IBM 4767 optional smart cards and readers for Linux users

The Smart Card User Guide describes the IBM Smart Card Utility Program (SCUP) and SCUP-enabled Cryptographic Node Management (CNM) utility.

IBM 4767 CCA utilities

The CCA Utilities User Guide describes CCA backup/restore, CCA initialization, and CCA HSM.

Other documents referenced in this book are:

- CCA Basic Services Reference and Guide for the IBM 4767 and IBM 4765 PCIe Cryptographic Coprocessors
- z/OS Cryptographic Services ICSF Trusted Key Entry Workstation User’s Guide
- Cryptographic Services Integrated Cryptographic Service Facility Application Programmer’s Guide
- Cryptographic Services Integrated Cryptographic Service Facility System Programmer’s Guide
- For Linux on Z:
  - Device Drivers, Features, and Commands, SC33-8411
- See the following web site for information about your distribution of Linux:
  - Distribution hints

Cryptography publications

The publications listed in this topic describe cryptographic standards, research, and practices relevant to the coprocessor.

- American National Standards Institute (ANSI). ANSI is the official U.S. representative to the International Organization for Standardization (ISO) and, via the U.S. National Committee, the International Electrotechnical Commission (IEC). ANSI is also a member of the International Accreditation Forum (IAF).


• ECC Brainpool Standard Curves and Curve Generation, v.1.0, October 19, 2005

• Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation (RFC 5639), Manfred Lochter and Johannes Merkle, IETF Trust, March 2010.
  Available at [http://www.rfc-editor.org/rfc/rfc5639.txt]

• Federal Information Processing Standards (FIPS), issued by the U.S. National Institute of Standards and Technology (NIST, see [http://www.nist.gov/itl/]). The listed FIPS publications are available from this web site: FIPS PUBLICATIONS.

• International Organization for Standardization (ISO). ISO is the world’s largest developer and publisher of International Standards. ISO is a network of the national standards institutes of many countries, one member per country, with a Central Secretariat in Geneva, Switzerland, that coordinates the system.
    – ISO 9564-4:2016 Financial services -- Personal Identification Number (PIN) management and security -- Part 4: Requirements for PIN handling in eCommerce for Payment Transactions
    – ISO 20038:2017 Banking and related financial services -- Key wrap using AES

• International Organization for Standardization/International Electrotechnical Commission (ISO/IEC)

• National Institute of Standards and Technology (NIST) Special Publications (SP), U.S. Dept. of Commerce


---

**Do you have problems, comments, or suggestions?**

Your suggestions and ideas can contribute to the quality and the usability of this document.

If you have problems using this document, or if you have suggestions for improving it, complete and mail the Reader's Comment Form found at the back of the document.
Part 1. IBM CCA programming

Read the information in this part of the document which introduces programming for the IBM CCA, AES, DES, HMAC, and PKA cryptography.

The topics in this part explain how to use CCA nodes and AES, DES, HMAC, and PKA verbs.

• Chapter 1, “Introduction to programming for the IBM Common Cryptographic Architecture,” on page 3 describes the programming considerations for using the CCA verbs. It also explains the syntax and parameter definitions used in the verbs. Information about concurrency is also provided.

• Chapter 2, “Running CCA verbs in PCI-HSM 2016 compliance mode,” on page 31 describes the support for compliance-tagged tokens introduced with CCA 6.0. It gives links to further related information in this document, and discusses the new warning mode to aid installations in detecting migration concerns.

• Chapter 3, “Using AES, DES, and HMAC cryptography and verbs,” on page 33 gives an overview of AES, DES, and ECC cryptography and provides general guidance information on how these verbs use different key types and key forms.

• Chapter 4, “Introducing PKA cryptography and using PKA verbs,” on page 71 introduces Public Key Algorithm (PKA) and Elliptic Curve Cryptography (ECC) support, and describes programming considerations for using the CCA PKA verbs, such as the PKA key token structure and key management.


• Chapter 6, “Understanding and managing master keys,” on page 83 provides information about master key management.
Chapter 1. Introduction to programming for the IBM Common Cryptographic Architecture

By using the IBM CCA application programming interface (API), you can obtain cryptographic and other services from the CEX*C feature and CCA.

The following subtopics are discussed:

- “Available Common Cryptographic Architecture (CCA) verbs”
- “Common Cryptographic Architecture functional overview” on page 4
- “CPACF support” on page 14
- “Security API programming fundamentals” on page 19
- “How to compile and link CCA application programs” on page 24

Available Common Cryptographic Architecture (CCA) verbs

CCA products provide a variety of cryptographic processes and data-security techniques.

Your application program can call verbs (sometimes called services) to perform the following functions:

**Data confidentiality**
Encrypt and decrypt information, typically using the AES or DES algorithms in Cipher Block Chaining (CBC) mode to enable data confidentiality.

**Data integrity**
Hash data to obtain a digest, or process the data to obtain a Message Authentication Code (MAC) or keyed hash MAC (HMAC), that is useful in demonstrating data integrity.

**Nonrepudiation**
Generate and verify digital signatures using either the RSA algorithm or the ECDSA algorithm, to demonstrate data integrity and form the basis for nonrepudiation.

**Authentication**
Generate, encrypt, translate, and verify finance industry personal identification numbers (PINs) and American Express, MasterCard, and Visa card security codes with a comprehensive set of finance-industry-specific services.

**Key management**
Manage the various AES, DES, ECC, and RSA keys necessary to perform the mentioned cryptographic operations.

**Java interaction**
Interact with the Java Native Interface (JNI). The CCA verbs have a specific version that can be used for JNI work.

**Note:** In this JNI, a new data type `hikmNativeLong` is replacing the old type `hikmNativeInteger` since CCA version 5.2. Both types inherit from an abstract class `hikmNativeNumber`. Thus, type `hikmNativeInteger` is still supported, so you can run existing applications with this deprecated data.
type. However, start using type \texttt{hikmNativeLong} for new applications instead, because \texttt{hikmNativeInteger} may be removed in the future.

**CCA management**

Control the initialization and operation of CCA.

This publication groups the many available verbs by topics. Each topic lists the verbs in alphabetical order by verb pseudonym.

---

**Common Cryptographic Architecture functional overview**

You use the CCA security API to access a common cryptographic architecture.

[Figure 1 on page 5](#) provides a conceptual framework for positioning the CCA security API, which you use to access a common cryptographic architecture. Application programs make procedure calls to the CCA security API to obtain cryptographic and related I/O services. You can issue a call to the CCA security API from essentially any high-level programming language. The call, or request, is forwarded to the cryptographic services access layer and receives a synchronous response. Your application program loses control until the access layer returns a response after processing your request.
The products that implement the CCA support consist of both hardware and software components.

**CCA software support:** The software consists of application development and runtime software components.

- The application development software primarily consists of language bindings that can be included in new applications to assist in accessing services available at the API. Language bindings are provided for the C and Java programming languages.
- The runtime software can be divided into the following categories:
  - Service-requesting programs, including application and utility programs.
  - The security API, an agent function that is logically part of the calling application program or utility.
  - The cryptographic services access layer: an environment-dependent request routing function, key-storage support services, and device driver to access one or more hardware cryptographic engines.
- The cryptographic engine software that gives access to the cryptographic engine hardware.

The cryptographic engine is implemented in the hardware of the CEX*C coprocessor. Security-sensitive portions of CCA are implemented in the cryptographic engine software running in the protected coprocessor environment.

- Utility programs and tools provide support for administering CCA secret keys, interacting with CCA managed symmetric and public key cryptography key storage, and configuring the software support.

You can create application programs that employ the CCA security API or you can purchase applications from IBM or other sources that use the products. This document is the primary source of information for designing systems and application programs that use the CCA security API with the cryptographic coprocessors.

**Cryptographic engine:** The CCA architecture defines a cryptographic subsystem that contains a cryptographic engine operating within a protected boundary. The coprocessor's tamper-resistant, tamper-responding environment provides physical security for this boundary and the CCA architecture provides the logical security needed for the full protection of critical information.

**CEX3C Coprocessors:** The coprocessor provides a secure programming and hardware environment wherein AES, DES, RSA, Elliptic Curve, and HMAC processes are performed. Each cryptographic coprocessor includes a general-purpose processor, non-volatile storage, and specialized cryptographic electronics. These components are encapsulated in a protective environment to enhance security. The IBM CCA Support Program enables applications to employ a set of AES, DES, RSA, Elliptic Curve, and HMAC-based cryptographic services utilizing the coprocessor hardware. Services include:

- DES, AES, RSA, Elliptic Curve, and HMAC key-pair generation
- DES, AES, RSA, Elliptic Curve, and HMAC host-based key record management
- Digital signature generation and verification
- Cryptographic key wrapping and unwrapping
- Data encryption, decryption and MAC generation/verification
- PIN processing for the financial services industry
- Other services, including DES key-management based on control-vector-enforced key separation.

**CEX4C Coprocessor:** The coprocessor provides the same cryptographic functions as the CEX3C coprocessor.

**CEX5C Coprocessor:** The coprocessor provides the same functions as the CEX4C coprocessor with more algorithms moved from hardware-enhanced to fully hardware-accelerated. CMAC, VFPE, AESKW, and other algorithms are added.

**CEX6C Coprocessor:** The coprocessor provides domain support for up to 85 logical partitions. When configured as a CCA coprocessor, it includes secure key functions with emphasis on the specialized functions required for banking and payment card systems. It is optionally programmable to add custom functions and algorithms by using User Defined Extensions (UDX). A new mode, called Payment Card Industry (PCI) PIN Transaction Security (PTS) Hardware Security Module (HSM), shortened to PCI-HSM 2016 compliance mode, is available exclusively for Crypto Express6S.
in CCA mode. Exploitation of PCI-HSM 2016 mode with compliance-tagged secure key tokens simplifies compliance with PCI requirements for hardware security modules.

**CCA**: Common Cryptographic Architecture (CCA) is the basis for a consistent cryptographic product family. Applications employ the CCA security API to obtain services from, and to manage the operation of, a cryptographic system that meets CCA specifications.

**CCA access control**: Each CCA node has an access-control system enforced by the hardware and protected software. The robust UNIX style access controls integrated into the Linux operating system are used to protect the integrity of the underlying CCA hardware environment. The specialized processing environment provided by the cryptographic engine can be kept secure because selected services are provided only when certain requirements are met or a Trusted Key-Entry console is used to enable access. The access-control decisions are performed within the secured environment of the cryptographic engine and cannot be subverted by rogue code that might run on the main computing platform.

**Coprocessor certification**: After quality checking a newly manufactured coprocessor, IBM loads and certifies the embedded software. Following the loading of basic, authenticated software, the coprocessor generates an RSA key-pair and retains the private key within the cryptographic engine. The associated public key is signed by a certification key securely held at the manufacturing facility and then the certified device key is stored within the coprocessor. The manufacturing facility key has itself been certified by a securely held key unique to the CEX*C product line.

The private key within the coprocessor, known as the device private key, is retained in the coprocessor. From this time on, if tampering is detected or if the coprocessor batteries are removed or lose power in the absence of bus power, the coprocessor sets all security-relevant keys and data items to zero. This process is irreversible and results in the permanent loss of the factory-certified device key, the device private key, and all other data stored in battery-protected memory. Security-sensitive data stored in the coprocessor flash memory is encrypted. The key used to encrypt such data is itself retained in the battery-protected memory.

**CCA master key**: When using the CCA architecture, working keys, including session keys and the RSA and ECC private keys used at a node to form digital signatures or to unwrap other keys, are generally stored outside the cryptographic-engine protected environment. These working keys are wrapped (DES triple-encrypted or AES encrypted) by the CCA master key. The master key is held in the clear (not enciphered) within the cryptographic engine.

The number of keys usable with a CCA subsystem is thus restricted only by the host server storage, not by the finite amount of storage within the coprocessor secure module. In addition, the working keys can be used by additional CCA cryptographic engines which have the same master key. This CCA characteristic is useful in high-availability and high-throughput environments where multiple cryptographic processors must function in parallel.

**Establishing a CCA master key**: To protect working keys, the master key must be generated and initialized in a secure manner. One method uses the internal random-number generator for the source of the master key. In this case, the master key is never external to the node as an entity and no other node has the same master key unless master-key cloning is authorized and in use (unless, out of all
the possible values, another node randomly generates the same master-key data). If an uncloned coprocessor loses its master key - for example, the coprocessor detects tampering and destroys the master key - there is no way to recover the working keys that it wrapped. The number of possible values is:

- For DES and RSA master keys, \(2^{168}\)
- For AES and APKA master keys, \(2^{256}\)

Another master-key-establishment method enables authorized users to enter multiple, separate key parts into the cryptographic engine. As each part is entered, that part is XORed with the contents of the new master-key register. When all parts have been accumulated, a separate command is issued to promote the contents of the current master-key register to the old master-key register and to promote the contents of the new master-key register to the current master-key register. The length of the key parts is:

- For DES and RSA master keys, 168 bits
- For AES and APKA master keys, 256 bits

CCA verbs: An application or utility program obtains service from the CCA Support Program by issuing service requests (verb calls or procedure calls) to the runtime subsystem (see [Chapter 27, “Sample verb call routines,” on page 1093](#)). To fulfill these requests, the Support Program in turn obtains service from the coprocessor software and hardware.

The available services are collectively described as the CCA security API. All the software and hardware accessed through the CCA security API should be considered an integrated subsystem. A command processor performs the verb request within the cryptographic engine.

Commands and access control, roles, profiles: In order to ensure that only designated individuals (or programs) can run commands such as master-key loading, each command processor that performs sensitive processing interrogates one or more control-point values within the cryptographic engine access-control system for permission to perform the request.

The access-control system includes one or more roles. Each role defines the permissible control points for users of that role. In the IBM Z environment, all application programs run using the permissions defined in the default role for their domain. The default role can only be modified using the TKE workstation. For a description of the functions that are permitted by the default version of the default role, see [z/OS Cryptographic Services ICSF Trusted Key Entry Workstation User’s Guide](#).

### How application programs obtain service

Application programs and utility programs obtain services from the security product by issuing service requests to the runtime subsystem of software and hardware. Use a procedure call according to the rules of your application language.

When the cryptographic services access layer receives requests concurrently from multiple application programs, it serializes the requests and returns a response for each request. There are other multiprocessing implications arising from the existence of a common master-key and a common key-storage facility. These implications are covered by separate topics of this publication.
The way application programs and utilities are linked to the API services depends on the computing environment. In the Linux environment, the operating system dynamically links application security API requests to the subsystem shared object library code. Compile application programs that use CCA and link the compiled programs to the CCA library. The library and its default distribution location is /usr/lib64/libcsulcca.so.

Together, the security API shared library and the environment-dependent request routing mechanism act as an agent on behalf of the application and present a request to the server. Requests can be issued by one or more programs. Each request is processed by the server as a self-contained unit of work. The programming interface can be called concurrently by applications running as different processes. The security API can be used by multiple threads in a process and is thread safe.

In each server environment, a device driver provided by IBM supplies low-level control of the hardware and passes the request to the hardware device. Requests can require one or more I/O commands from the security server to the device driver and hardware.

The security server and a directory server manage key storage. Applications can store locally used cryptographic keys in a key-storage facility. This is especially useful for long-life keys. Keys stored in key storage are referenced using a key label. Before deciding whether to use the key-storage facility or to let the application retain the keys, consider system design trade-off factors, such as key backup, the impact of master-key changing, the lifetime of a key, and so forth.

Overlapped processing and load balancing

You can maximize throughput by organizing your application or applications to make multiple, overlapping calls to the CCA API.

Calls to the CCA security API are synchronous, that is, your program loses control until the verb completes. Multiple processing-threads can make concurrent calls to the API.

You can maximize throughput by organizing your application or applications to make multiple, overlapping calls to the CCA API. You can also increase throughput by employing multiple coprocessors, each with CCA. You can maximize throughput by organizing your application (or applications) to make multiple, overlapping calls to the CCA API. You can also increase throughput by employing multiple coprocessors, each with CCA. Another way to maximize throughput is to make use of the AUTOSELECT option for automatic load-balancing. See “Multi-coprocessor selection capabilities” on page 11.

Within the coprocessor, the CCA software is organized into multiple threads of processing. This multiprocessing design is intended to enable concurrent use of the coprocessor’s main engine, PCIe communications, DES and Secure Hash Algorithm-1 (SHA-1) engine, and modular-exponentiation engine.

Host-side key caching

Calls to the CCA security API are synchronous, that is, your program loses control until the verb completes. Multiple processing-threads can make concurrent calls to the API.
CCA provides caching of key records obtained from key storage within the CCA host code. However, the host cache is unique for each host process. If different host processes access the same key record, an update to a key record caused in one process does not affect the contents of the key cache held for other processes. Caching of key records within the key-storage system can be suppressed so all processes access the most current key-records. To suppress caching of key records, use the SET command to set the environment variable CSUCACHE to NO. If this environment variable is not set, or is set to anything other than NO, caching of key records will not be suppressed. The CSUCACHE environment variable does not impact CPACF translated key caching.

**Domain selection capabilities**

Beginning with CCA 6.0 and Linux kernel version 4.10, the user can exploit multiple domains. To find out the available domains on your partition use the `1szcrypt` command:

```
# 1szcrypt

<table>
<thead>
<tr>
<th>CARD.DOMAIN</th>
<th>TYPE</th>
<th>MODE</th>
<th>STATUS</th>
<th>REQUEST_CNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>CE5C</td>
<td>CCA-Coproc</td>
<td>online</td>
<td>36</td>
</tr>
<tr>
<td>00.0005</td>
<td>CE5C</td>
<td>CCA-Coproc</td>
<td>online</td>
<td>10</td>
</tr>
<tr>
<td>00.001F</td>
<td>CE5C</td>
<td>CCA-Coproc</td>
<td>online</td>
<td>7</td>
</tr>
</tbody>
</table>
```

In the example, there is a single card, designated ID 0. For card 0 there are two domains available, X’05’ and X’1f’ (31 decimal).

There is a default domain for the system, which can be retrieved using the following command:

```
# cat /sys/bus/ap/ap_domain
```

CCA uses this default domain if no other choice is made using the CSU_DEFAULT_DOMAIN environment variable.

You can configure a CCA application to use the multi-domain capabilities with the environment variable CSU_DEFAULT_DOMAIN. The CCA library scans this variable one time when the application starts, but not for every request. CCA does not try to fix bad values for the variable. An error is returned for each request sent to an unconfigured domain, and information is placed into the system log.

**Using CSU_DEFAULT_DOMAIN**

- **Selecting a single domain**
  
  To set a single domain for use by the current application, whether that is a configuration utility (such as panel.exe) or a production business application, set the CSU_DEFAULT_DOMAIN variable to the decimal number for that domain. Continuing with the previous example, if domain 31 is desired, use this command:

  ```
  # export CSU_DEFAULT_DOMAIN=31
  ```

  Use cases for a single-domain configuration in a multi-domain system image
When setting master keys for multiple domains on the same card using the `panel.exe` utility, set the default domain to the desired value in between configuration actions to target the correct domain.

If there are two applications that run on the same system but require unique master keys and unique key storage locations:
- Configure domain X on all cards with the master key for `application_1`, and configure domain Y on all cards with the master key for `application_2`.
- Create unique key storage for each application.
- Set the CSU_DEFAULT_DOMAIN environment variable for each application.

### Selecting any domain

Similar to the multi-card load-balancing approach, there is a way to request that the Linux device driver load-balances your requests across all domains. It is best used in concert with the multi-card load-balancing approach since there is no extra performance capacity available by using a second domain from the same card. It is very important that all used domains used are configured with the same master key. Also, further required configuration (such as decimalization tables) must be the same for all domains, if they are made available to a system that exploits multi-domain load-balancing.

To configure multi-domain load-balancing, use this command:

```
# export CSU_DEFAULT_DOMAIN=DOM-ANY
```

### Multi-coprocessor selection capabilities

Multi-coprocessor selection capabilities allow you to employ more than one CCA coprocessor.

When more than one CCA coprocessor is installed, an application program can control which CCA coprocessor to use. It can explicitly select a CCA coprocessor, it can switch on the AUTOSELECT option, or it can optionally employ the default CCA coprocessor.

#### AUTOSELECT option

If switched on, the AUTOSELECT option overrides an explicit CCA coprocessor selection and default CCA coprocessor selection for all verbs (except those listed in “Verbs that ignore AUTOSELECT” on page 12).

When the AUTOSELECT option is switched on, the CCA coprocessor to be used by a verb will be selected by the operating system (the Linux device driver) from the set of available CCA coprocessors, including any coprocessors loaded with CCA user defined function (UDX) code. The Linux device driver chooses a CCA coprocessor based on a policy for load balancing.

To switch on the AUTOSELECT option, use the Cryptographic Resource Allocate verb (CSUACRA). Alternatively, the AUTOSELECT option can be switched on at program start by setting the environment variable CSU_DEFAULT_ADAPTER to the value DEV-ANY. For example:

```
export CSU_DEFAULT_ADAPTER=DEV-ANY
```

To switch off the AUTOSELECT option, use the Cryptographic Resource Deallocate verb (CSUACRD).
Master key coherence for AUTOSELECT

When using the AUTOSELECT option, all CCA coprocessors accessible by the operating system must have the same state. In particular, they must be configured with the same master key as appropriate for the services in use. For example, if your application uses only DES functions and you enable AUTOSELECT, then the SYM-MK should be the same across all accessible CCA coprocessors. If you use any RSA functions from PKA verbs, then the ASYM-MKs must be the same. For AES usage the AES-MKs must match, and for ECC the APKA-MK must match.

Verbs that ignore AUTOSELECT

The following verbs ignore the AUTOSELECT option and use the explicitly selected or default CCA coprocessor instead. These verbs act as if the AUTOSELECT option does not exist, acting exactly as they did in prior releases in which AUTOSELECT was not present.

Table 5. Verbs that ignore AUTOSELECT

<table>
<thead>
<tr>
<th>Verb</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSUACFQ</td>
<td>When querying CCA coprocessor state it is important to retrieve data from the explicitly queried CCA coprocessor.</td>
</tr>
<tr>
<td>CSUARNT</td>
<td>When testing adapter health it is also important to receive explicit results.</td>
</tr>
<tr>
<td>CSNDRKL, CSNDRKD</td>
<td>Listing and managing the retained keys for a CCA coprocessor requires dealing with explicitly selected CCA coprocessors. Note that it is specifically not recommended to use any retained key functions (such as choosing to use retained keys with PKA verbs) when AUTOSELECT is active. By their very nature retained keys are tied to the specific CCA coprocessor where they were created</td>
</tr>
<tr>
<td>CSNBMKP</td>
<td>Managing the Master Keys for a CCA coprocessor also requires explicit allocation and de-allocation to achieve expected results.</td>
</tr>
<tr>
<td>HOST-only verbs</td>
<td>Some verbs are not impacted by AUTOSELECT simply because they never use the CCA coprocessor. These verbs take actions that do not involve secrets guarded by the card Master Keys. These include:</td>
</tr>
<tr>
<td></td>
<td>CSUACFV, CSNBSKSI, CSNBCVG, CSNBKTB, CSNBKTB2, CSNBKTP, CSNBKTP2, CSNBT31O, CSNBT31R, CSNBT31P, CSNBAKRC, CSNBAKRD, CSNBAKRL, CSNBAKRR, CSNBAKRW, CSNBKRC, CSNBKRC, CSNBKRD, CSNBKRR, CSNBKRW, CSNDKRC, CSNDKRD, CSNDKRL, CSNDKRR, CSNDKRW, CSNDPKB, CSNDPKX</td>
</tr>
<tr>
<td>CPACF using verbs</td>
<td>Some verbs use the CPACF according to configuration of appropriate environment variables. These verbs will ignore AUTOSELECT if told to use CPACF. These include:</td>
</tr>
<tr>
<td></td>
<td>CSNBOWH, CSNBDEC, CSNBENC, CSNBSAE, CSNBSAD, CSNBMGN, CSNBMVR</td>
</tr>
</tbody>
</table>

Explicit CCA coprocessor selection

To explicitly select a CCA coprocessor, use the Cryptographic Resource Allocate verb (CSUACRA). This verb allocates a CCA coprocessor loaded with the CCA software. When a CCA coprocessor is allocated and the AUTOSELECT option is not on, CCA requests are routed to it until it is de-allocated. Similarly, when a CCA coprocessor is allocated and one of the verbs that ignore the AUTOSELECT option is used, then CCA requests are routed to it until it is de-allocated.
To de-allocate an allocated coprocessor, use the Cryptographic Resource Deallocate verb (CSUACRD). When a coprocessor is not allocated (either before an allocation occurs or after the cryptographic resource is de-allocated), requests are routed to the default coprocessor unless the AUTOSELECT option is on.

Note: The scope of the Cryptographic Resource Allocate and the Cryptographic Resource Deallocate verbs is to a thread. A multi-threaded application program can use all of the installed CCA coprocessors simultaneously. A program thread can use only one of the installed coprocessors at any given time, but it can switch to a different installed coprocessor as needed. To perform the switch, a program thread must deallocate an allocated cryptographic resource, if any, and then it must allocate the desired cryptographic resource. The Cryptographic Resource Allocate verb fails if a cryptographic resource is already allocated.

Listing CCA coprocessors

With the first call to CCA from a process, CCA associates coprocessor designators CRP01, CRP02, and so on with specific coprocessors. The host determines the total number of coprocessors installed through a call to the coprocessor device driver. Adding, removing, or relocating coprocessors can alter the number associated with a specific coprocessor. The host then polls each coprocessor in turn to determine which ones contain the CCA firmware. As each coprocessor is evaluated, the CCA host associates the identifiers CRP01, CRP02, and so forth to the CCA coprocessors. CCA coprocessors loaded with a UDX extension are also assigned a CRPnn identifier.

For a specific device driver, names such as these are used: CRPnn, cardnn, APnn, and so forth, where the nn values normally do not match (for example, some start with 0, others start with 1).

To determine the coprocessor type of a card (one out of CEX3C through CEX6C), use one of these methods:

- Invoke the Cryptographic Facility Query verb (see “Determining if a card is a CEX3C, CEX4C, CEX5C, or a CEX6C” on page 101).
- Use the lszcrypt command (see “Confirming your cryptographic devices” on page 1105).
- Use the hwtype attribute of your cryptographic devices in sysfs (see the version of Device Drivers, Features, and Commands, that applies to your distribution).
- Run panel.exe -x using the panel.exe utility installed with the RPM, to get a quick summary of cards available and their status. See “The panel.exe utility” on page 1121.
- Run 1vp.e, another utility installed with the RPM, which gives more detailed information about each card available. See Chapter 31, “Utilities,” on page 1119.

Note that the mapping of logical card identifiers such as CRP01 and CRP02 to physical cards in your machine is not defined. This is because the mapping can change depending on the machine and its configuration. If your application needs to identify specific coprocessor cards, you can do one of the following:

- Use the Cryptographic Facility Query verb (see “Cryptographic Facility Query (CSUACFQ)” on page 100) with the STATCARD or STATCRD2 rule_array keyword
- Use the panel.exe utility program with option -x, in order to read a card’s serial number (see “The panel.exe utility” on page 1121).
Default CCA coprocessor

The selection of a default device occurs with the first CCA call to a coprocessor. When the default device is selected, it remains constant throughout the life of the thread. Changing the value of the environment variable after a thread uses a coprocessor does not affect the assignment of the default coprocessor. If a thread with an allocated coprocessor ends without first de-allocating the coprocessor, excess memory consumption results. It is not necessary to deallocate a cryptographic resource if the process itself ends. It is suggested only if individual threads end while the process continues to run.

You can alter the default designation by explicitly setting the CSU_DEFAULT_ADAPTER environment variable. This is accomplished by issuing the following command:

```
export CSU_DEFAULT_ADAPTER=CRPxx
```

Replace CRPxx with the identifier for the resource you wish to use, such as CRP02. The rpm limits the maximum value of xx such that CRP01 through CRP08 are the valid identifiers for the default coprocessor (limited by the number of coprocessors actually configured).

When cards of multiple types (CEX*C) are active in the same system, note the following:

- The CCA library detects CEX*C adapters and intermingle them in the CRPxx adapter instance list. This is a list of all available adapters, in the order that they were discovered by the device driver.
- The default adapter will be the lowest numbered CEX*C instance, that is, the newest adapter level, found by the device driver. For example, if the ordered discovery list is: [CEX3C, CEX4C, CEX3C, CEX4C], the default adapter will be the first CEX4C.
- A user can specify the proper CRPxx number to allocate and work with any card (CEX*C).
- For a specific device driver, names such as these are used: CRPnn, cardxx, APxx, and so forth, where the xx values normally do not match (for example, some start with 0, others start with 1).

CPACF support

Central Processor Assist for Cryptographic Functions (CPACF) must be configured and enabled on the system before you can use it.

CPACF support has these features:

- "Environment variables that affect CPACF usage"
- "Access control points that affect CPACF protected key operations" on page 16
- "CPACF operation (protected key)" on page 16
- "CCA library CPACF preparation at startup" on page 18
- "Interaction between the default card and use of protected key CPACF" on page 19

Environment variables that affect CPACF usage

The CSU_HCPUACLR and CSU_HCPUAPRT environment variables control whether the CPACF is used for certain CCA functions.
These variables are overridden by the explicit use of the Cryptographic Resource Allocate (CSUACRA) and Cryptographic Resource Deallocate (CSUACRD) verbs to enable or disable these access patterns. To avoid confusion, the environment variables are given similar names to the keywords used by Cryptographic Resource Allocate (CSUACRA) and Cryptographic Resource Deallocate (CSUACRD).

**Note:** The default values listed here are valid even if these environment variables are not defined. Their settings represent default policy decisions made in the library code.

**CSU_HCPUACLR**  
Use the CSU_HCPUACLR variable to allow CPACF for clear key operations and hashing algorithms.

Set CSU_HCPUACLR to 1 in a profile setup file or with this command:
```
export CSU_HCPUACLR=1
```

Setting this variable to any other value (except for the case where the variable has not been set, as noted above) results in disabling the use of the CPACF for clear key operations and hashing algorithms. The default is 1, meaning that the function is enabled.

**Affected verbs**
- MDC Generate (CSNBMDG)
- One-Way Hash (CSNBOWH)
- Symmetric Algorithm Decipher (CSNBSAD) (clear key AES)
- Symmetric Algorithm Encipher (CSNBSAE) (clear key AES)

**CSU_HCPUAPRT**  
Use the CSU_HCPUAPRT variable to use CPACF for protected key (translated secure key) operations.

Set CSU_HCPUAPRT to 1 in a profile setup file or with this command:
```
export CSU_HCPUAPRT=1
```

Setting this variable to any other value (except for the case where the variable has not been set, as noted above) results in disabling the use of the CPACF for protected key (translated secure key) operations. The default is 0, meaning that the function is disabled.

**Affected verbs**
- Decipher (CSNBDEC)
- Encipher (CSNBENC)
- MAC Generate (CSNBMGN)
- MAC Verify (CSNBMVR)
- Symmetric Algorithm Decipher (CSNBSAD) (clear key AES, protected key AES)
- Symmetric Algorithm Encipher (CSNBSAE) (clear key AES, protected key AES)
Access control points that affect CPACF protected key operations

There are two access points that enable the protected key feature.

These two access points are:

High-performance secure DES keys

This is bit X'0295', and is set ON by default.

This ACP enables translating DES keys for use with the CPACF. Without this bit set ON, the call to the CEX*C to rewrap the key under the CPACF wrapping key will fail with a return code 8 and reason code 90, which will in turn imply disabling the use of this function by the host user. This error will not be returned to the user, instead the operation will be sent to the CEX*C. Because the default value of the bit is ON, it is assumed that the user will know that it is set OFF on purpose. A return code 8 and reason code 90 will cause no further requests to go to the CEX*C verb that translates keys, in an effort to preserve normal path performance.

High-performance secure AES keys

This is bit X'0296', and is set ON by default.

This ACP enables translating AES keys for use with the CPACF. Without this bit set ON, the call to the CEX*C to rewrap the key under the CPACF wrapping key will fail with a return code 8 and reason code 90, which will in turn imply disabling the use of this function by the host user. This error will not be returned to the user, instead the operation will be sent to the CEX*C. Because the default value of the bit is ON, it is assumed that the user knows that it is set OFF on purpose. A return code 8 and reason code 90 do not cause further requests to go to the CEX*C verb that translates keys, in an effort to preserve normal path performance.

CPACF operation (protected key)

These are details for Central Processor Assist for Cryptographic Functions (CPACF) usage by the host library.

Note that at system power-on, the CPACF generates a new Key Encryption Key (KEK, kek-t) for wrapping translated keys.

Figure 2 on page 17 illustrates the CPACF layer as it relates to the security access API and cryptographic engine. The CPACF exploitation layer examines commands received by the security server to see if they can be redirected to the CPACF. If so, this layer makes preparations (including translating secure keys to protected keys), and then call the CPACF directly. If all preparations and the CPACF operations are successful, the results are returned as a normal return through the security server. For any errors, the command is redirected back through the security server to the normal path, using the allocated CEX*C for the thread making the call.
Clear key or No key: For operations that do not use keys (such as hash algorithms) or operations that use keys that are not encrypted under the card master key, (called clear keys), no translation is necessary and the CPACF is used immediately.

Protected key: The device driver and the other layers are used for protected key support, for translating keys. This relationship is similar to the 'directory server' relationship: a translation layer invisible to the customer. After translation the 'translated-key' is stored in an invisible runtime cache so that the next use of the key can avoid the translation step. For protected key usage, a CEX*C feature must be available and allocated for use by the thread.

CPACF service actions and running applications: The CPACF is an independent hardware unit, like the CEX*C itself, and can be independently configured available or unavailable while a Linux instance is running by service technicians performing service actions. If the CPACF is cycled it will generate a new wrapping key for translated keys, invalidating all of the keys in the CCA library key translation cache. Therefore, it is never advisable to attempt such a service action while there are system instances with applications running that use the CPACF.

If such an action is undertaken, applications should be stopped and restarted so that the libcsulcca.so is unloaded from memory and reloaded. This will cause the key cache to be cycled. A more complete measure would be to reboot system images. If these precautions are disregarded and a CPACF service action is undertaken as described, application crashes may ensue with a SIGSEGV error. This could occur due to translated keys wrapped under outdated CPACF wrapping keys being used.
A normal system-wide power cycle will cause the CPACF to generate a new wrapping key by design, however, this action also of course cycles all of the hosted system LPARs and VM system images so there is no problem; translated keys are not cached in permanent storage.

**Using keys with CPACF, protected key**
Follow the steps in this procedure to use keys with CPACF, protected key.

**Procedure**

1. An eligible CCA verb call (see lists in “Access control points that affect CPACF protected key operations” on page 16) specifying a key token or key identifier for a key token that is a normal internal CCA key token, called key-e here, comes into the CCA library.
2. The CCA library verifies that a CEXC is available for key translation. If not, then the standard no-available-device error is returned.
3. The CCA library tries to find an already translated version (key-t) that matches the key-e passed into the CCA library.
   - The user application (CCA library in this case) must cache translated key-t objects in RAM, using the key-e tokens as references.
4. If a key-t is not found for the key-e used:
   - The CCA library translates the key-e to a key-t for use with the CPACF using CCA secure services, then caches the key pair.
5. At this point, either a fresh key-t has been obtained, or a key-t was found in RAM cache for the operation.
6. The CCA library directs the operation to the CPACF using the key-t.

**Results**

The `panel.exe --list-cpacf` command displays all the supported CPACF functions. This is especially useful on a z/VM system, to make sure that the protected key functions are available. For details, see “The panel.exe utility” on page 1121.

**Using keys with CPACF, clear key or no key**
Follow the steps in this procedure to use keys with CPACF, clear key or no key.

**Procedure**

1. An eligible CCA verb call (see lists in “Access control points that affect CPACF protected key operations” on page 16) comes into the CCA library.
2. No CEXC is necessary, so no check for availability or Cryptographic Resource Allocate (CSUACRA) call will be implied.
3. The CCA library prepares an appropriate CPACF clear key (key-c) structure using the clear key passed to the CCA verb (key-v).
4. The CCA library directs the operation to the CPACF using the key-c.

**CCA library CPACF preparation at startup**

When the CCA library first starts up, it must take some initialization steps to prepare for using the Central Processor Assist for Cryptographic Functions (CPACF).
Procedure

1. Check configuration options to see if either is set to ON, allowing some use of the CPACF. If neither is on, skip the rest of initialization.

2. Check for existence and configuration of the CPACF.

Interaction between the default card and use of protected key CPACF

While the CPACF can be used to encrypt and decrypt data in the absence of a CEX*C, for protected key operations, a CEX*C is still necessary and it must be the allocated or default adapter for the thread doing the processing.

Using the AUTOSELECT option and the use of protected key CPACF

While the CPACF can be used to encrypt and decrypt data in the absence of a CEX*C, for protected key operations a CEX*C is still necessary, because the users' key tokens are translated with a service only available on the CEX*C for use with the CPACF.

Therefore, when enabling the AUTOSELECT option, all CCA coprocessors available to the operating system must be CEX3C or newer. See “Multi-coprocessor selection capabilities” on page 11 for more information.

Security API programming fundamentals

You obtain CCA cryptographic services from the coprocessor through procedure calls to the CCA security application programming interface (API).

Most of the services provided are considered an implementation of the IBM Common Cryptographic Architecture (CCA). Most of the extensions that differ from other IBM CCA implementations are in the area of the access-control services. If your application program is used with other CCA products, compare the product literature for differences.

Your application program requests a service through the security API by using a procedure call for a verb. The term verb implies an action that an application program can initiate. Other publications might use the term callable service instead. The procedure call for a verb uses the standard syntax of a programming language, including the entry-point name of the verb and the parameters of the verb. Each verb has an entry-point name and a fixed-length parameter list.

The security API is designed for use with high-level languages, such as C, COBOL, or RPG and for low-level languages, such as assembler language. It is also designed to enable you to use the same verb entry-point names and variables in the various supported environments. Therefore, application code you write for use in one environment generally can be ported to additional environments with minimal change.

Verbs, variables, and parameters

Certain information is included for each verb, including the entry-point name and the parameter list.
Each verb has an entry-point name and a fixed-length parameter list. Part 2, “CCA verbs,” on page 87 describes each verb, and includes the following information for each verb:

- Pseudonym
- Entry-point name
- Description
- Format
- Parameters
- Restrictions
- Required commands
- Usage notes
- Related information
- JNI version

**Pseudonym**

Also known as a general-language name or verb name. This name describes the function that the verb performs, such as Key Generate.

**Entry-point name**

Also known as a computer-language name. This name is used in your program to call the verb. Each verb’s 7 or 8 character entry-point name begins with one of the following prefixes:

- **CSNB** Generally, the AES and DES verbs
- **CSND** Public key cryptography verbs, including RSA and Elliptic Curve
- **CSUA** Cryptographic-node and hardware-control verbs

The last three or four letters in the entry-point name after the prefix identify the specific verb in a group and are often the first letters of the principal words in the verb pseudonym.

When verbs are described throughout this publication, they are sometimes referred to by the pseudonym, and at other times by the pseudonym followed by the entry-point name in parenthesis. An example of this is: Key Generate (CSNBKGN).

The verb prefixes used here are different from those used by the Integrated Cryptographic Service Facility (ICSF).

**Description**

The verb is described in general terms. Be sure to read the parameter descriptions because these add additional detail.

**Format**

The format section for each verb lists the entry-point name followed by the list of parameters for the verb. You must code all the parameters, and they must be in the order listed.

```c
entry-point name(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    parameter_5,
    parameter_6,
    ...
    parameter_n)
```
Parameters

All information exchanged between your application program and a verb is through the variables identified by the parameters in the procedure call. These parameters are pointers to the variables contained in application program storage that contain information to be exchanged with the verb. Each verb has a fixed-length parameter list and though all parameters are not always used by the verb, they must be included in the call.

The first four parameters are the same for all of the verbs. For a description of these parameters, see “Parameters common to all verbs” on page 22. For the description of the remaining parameters, see the definitions with the individual verbs.

In the description for each parameter, data flow direction and data type are indicated, as follows.

Direction:   direction
Type:       data type

direction The parameter descriptions use the following terms to identify the flow of information:

Input  The application program sends the variable to the verb (to the called routine).

Output  The verb returns the variable to the application program.

Input/Output  The application program sends the variable to the verb or the verb returns the variable to the application program, or both.

data type Data identified by a verb parameter can be a single value or a one-dimensional array. If a parameter identifies an array, each data element of the array is of the same data type. If the number of elements in the array is variable, a preceding parameter identifies a variable that contains the actual number of elements in the associated array. Unless otherwise stated, a variable is a single value, not an array.

For each verb, the parameter descriptions use the following terms to describe the type of variable:

Integer  A CCA integer (CCAINT). On Linux on Z, this is defined as the system type long. On other platforms this has been defined as a 4-byte (32-bit), signed, two's-complement binary number. (CCA for Linux on Z has always defined the CCA integer as long).

String  A series of bytes where the sequence of the bytes must be maintained. Each byte can take on any bit configuration. The string consists only of the data bytes. No string terminators, field-length values, or typecasting parameters are included. Individual verbs can restrict the byte values within the string to characters or numerics.

Character data must be encoded in the native character set of the computer where the data is used. Exceptions to this rule are noted where necessary.

Array  An array of values, which can be integers or strings. Only
one-dimensional arrays are permitted. For information about the parameters that use arrays, see “The rule_array and other keyword parameters.”

Restrictions
Any restrictions are noted.

Required commands
Any access control points required to use the verb are described here.

Usage notes
Usage notes about this verb are listed.

Related information
Any related information is noted.

JNI version
If the verb has a Java Native Interface version, it is described.

Commonly encountered parameters
Some parameters are common to all verbs and other parameters are used with many of the verbs.

- “Parameters common to all verbs”
- “The rule_array and other keyword parameters”
- “Key tokens, key labels, and key identifiers” on page 23

Parameters common to all verbs
A parameter is an address pointer to the associated variable in application program storage.

The first four parameters (return_code, reason_code, exit_data_length, and exit_data) are the same for all verbs:

return_code
The return code specifies the general result of the verb. Chapter 18, “Return codes and reason codes,” on page 825 lists the return codes.

reason_code
The reason code specifies the result of the verb that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Chapter 18, “Return codes and reason codes,” on page 825 lists the reason codes.

exit_data_length
A pointer to an integer value containing the length of the string (in bytes) that is returned by the exit_data value. This parameter should point to a value of zero, to ensure compatibility with any future extension or other operating environment.

exit_data
The data that is passed to an installation exit. Exits are not supported and no exit data is allowed in this parameter.

Restriction: The exit_data_length and exit_data variables must be declared in the parameter list. The exit_data_length parameter should be set to B'0'.

The rule_array and other keyword parameters
The rule_array parameter and some other parameters use keywords to transfer information.
Generally, a rule_array consists of a variable number of data elements that contain keywords that control specific details of the verb process. Almost all keywords, in a rule_array or otherwise, are eight bytes in length, and should be uppercase, left-aligned, and padded on the right with space characters. Because not all implementations fold lowercase characters to uppercase, you should always code the keywords in uppercase.

The number of keywords in a rule_array is specified by a rule_array_count variable, an integer that defines the number of 8-byte elements in the array.

In some cases, a rule_array is used to convey information other than keywords between your application and the server. This is, however, an exception. For a list of key types that are passed in the rule_array keyword, see Table 7 on page 46.

### Key tokens, key labels, and key identifiers

Essentially all cryptographic operations employ one or more keys. In CCA, keys are retained within a structure called a key token.

A verb parameter can point to a variable that contains a key token. Generally you do not need to be concerned with the details of a key token. You can deal with it as an entity.

Key tokens are described as either internal, operational, or external, as follows:

#### Internal

A key token that contains an encrypted key for local use. The cryptographic engine decrypts an internal key to use the key in a local operation. When a key is entered into the system, it is always encrypted if it appears outside the protected environment of the cryptographic engine. The engine has a special key-encrypting key, called a master key. This key is held within the engine to wrap and unwrap locally used keys.

#### Operational

An internal key token that is complete and ready for use and contains a key that is encrypted under a master key. During entry of a key, the internal key-token can have a flag set indicating the key information is incomplete.

#### External

A key token that contains a key that is either in the clear or is encrypted by some key-encrypting key other than the master key. Generally, when a key is to be transported from place to place or is to be held for a significant period of time, the key must be encrypted with a transport key. A key wrapped by a (transport) key-encrypting key is designated as being external.

RSA and ECC public-keys are not encrypted values and, when not accompanied by private-key information, are retained in an external key-token.

Internal key tokens can be stored in a file maintained by the directory server. These key tokens are referenced by use of a key label. A key label is an alphanumeric string you place in a variable and reference with a verb parameter.

Parameter descriptions specify how you can provide a key using these terms:

#### Key token

The parameter must contain a proper key-token structure.
Key label
The parameter must contain a key-label string used to locate a key record in key storage.

Key identifier
The parameter must contain either a key token or a key label. The first byte in the parameter indicates whether it contains a key token or a key label.

- X'00' indicates a DES null key-token.
- range X'01' - X'1F' indicates that the variable is processed as a key token.
- range X'20' - X'FE' indicates that the variable is processed as a key label. There are additional restrictions on the value of a key label.
- X'FF' raises an error condition when passed to the API.

How to compile and link CCA application programs

The CCA RPM includes C libraries that you can use to build CCA C applications.

One of these libraries also supports a Java Native Interface (JNI) that you can use to build CCA Java applications. The CCA RPM also includes Java libraries with front end classes for the JNI.

The C libraries and their default distribution locations are:

/usr/lib64/libcsulcca.so.*
The libcsulcca.so library is the main interface library to most CCA interface calls. It contains all CCA verbs apart from the Master Key Process (CSNBMKP) verb. The names of the verbs and their parameters are listed in the C header file csulinc1.h.

This library also contains the C support for the JNI.

/usr/lib64/libcsulccamk.so.*
This library contains the Master Key Process (CSNBMKP) verb and is required for applications that use the master key process (see "Using the Master Key Process (CSNBMKP) verb" on page 25).

CCA 5.0 includes two versions of the Java libraries:

New version introduced with CCA 4.2
As of CCA 4.2, new Java libraries that use the Java package infrastructure are included. These Java libraries and their default distribution locations are:

/opt/IBM/CCA/cnm/CNM.jar
This library contains the data classes used by the JNI (see “Entry points and data types used in the JNI” on page 26) and most JNI verb front end classes.

/opt/IBM/CCA/cnm/CNMMK.jar
This library contains the JNI front end class for the Master Key Process (CSNBMKP) verb.

Deprecated version - removed starting with CCA 5.0
Note that the CCA 5.0 rpm no longer contains the files mentioned in this section. However, CCA 4.2 still includes the Java libraries of earlier CCA versions to support your existing applications. These libraries contain all verbs available with CCA 4.2 but future additions will not be available in these libraries. The default distribution locations of these deprecated libraries are:

```
/opt/IBM/CCA/cnm/HIKM.zip
This library contains the data classes used by the JNI (see “Entry points and data types used in the JNI” on page 26) and most JNI verb front end classes.
```

```
/opt/IBM/CCA/cnm/HIKMMK.zip
This library contains the JNI front end class for the Master Key Process (CSNBMKP) verb.
```

See “Calling the CCA JNI using the Java package infrastructure” on page 26 for more information about the two versions of the Java libraries.

The CCA RPM also includes C and Java sample programs that help you develop your application (see “Building C applications using the CCA libraries” and “JNI sample modules and sample code” on page 26).

### Using the Master Key Process (CSNBMKP) verb

```
/usr/lib64/libcsulccamk.so contains the Master Key Process (CSNBMKP) verb.
```

Any use of the `libcsulccamk.so` library is restricted because the library is installed so that only the root user (user ID of 0) and members of the group `cca_admin` have read access. The `cca_admin` group is added by the CCA RPM installation procedure. This is done to limit the ability of an untrusted user to copy the library with the purpose of reverse-engineering the master-key access methods inside it.

Furthermore, use of some specific access methods through the Master Key Process (CSNBMKP) verb are restricted to a corresponding Linux group membership of the user trying to make that access. Table 315 on page 1115 contains a list of the groups and their functions.

Users without the required group membership are denied use. For more information, see Master key load (Step 7 on page 1113).

### Building C applications using the CCA libraries

Perform these steps to build a C program from a make file.

**Procedure**

1. Change to the directory that contains the make file.
2. Issue this command to compile the program:
   ```
   make -f <makefile>
   ```

   For example, to use the make file of “Sample program in C” on page 1093, issue:
   ```
   make -f makefile.lnx
   ```

### Using the CCA JNI

You can use the Java native interface (JNI) to work with the CCA.
Calling the CCA JNI using the Java package infrastructure

Starting with CCA 5.0, there is only one method for calling the CCA JNI.

Deprecated methods are no longer available, and therefore documented only in prior editions of this publication.

For more information, see topic “JNI sample modules and sample code” about available sample programs.

When calling the CCA JNI, you need import statements in the Java source files. The JNI JAR files used by this method are CNM.jar and, for the master key process, CNMMK.jar.

See “Building the Java byte code” on page 28 for details about compiling and running Java applications with this method.

Entry points and data types used in the JNI

The Java entry points of CCA verbs are similar to the C entry points, except that a letter J is appended to the entry point name.

For example, CSNBKGN is the C entry point for the Key Generate verb, and CSNBKGNJ is the Java entry point for this verb. The detailed verb descriptions in Part 2, “CCA verbs,” on page 87 include a section for the JNI interface.

The following data types are defined and used in the JNI:

- **hikmNativeNumber**
  - Abstract data type. Parent for hikmNativeLong and hikmNativeInteger.

- **hikmNativeLong**
  - 64-bit native signed integer (type long), matching the C interface

- **Byte** *
  - general pointer type to unsigned byte

- **hikmNativeInteger**
  - This data type is deprecated.

**Note:** The data type **hikmNativeLong** is replacing the old type **hikmNativeInteger** since CCA version 5.2. Both types inherit from abstract class **hikmNativeNumber**. Thus, type **hikmNativeInteger** is still supported, so you can run existing applications with this deprecated data type. However, start using type **hikmNativeLong** for new applications instead, because **hikmNativeInteger** may be removed in the future.

The following documentation files (per default located in /opt/IBM/CCA/doc) provide information about these classes:

- hikmNativeNumber.html
- hikmNativeLong.html
- hikmNativeInteger.html

A file named hikmNativeNumber.html provides information about this class. The default location of this file is /opt/IBM/CCA/doc directory.

**JNI sample modules and sample code**

To illustrate the two JNI access methods and also how to use the CCA JNI to call CCA verbs, sample modules are provided.
mac.java
illustrates JNI calls versus C calls to CCA.

This sample program calls the same CCA verbs as the sample C language program named mac.c.

This sample program uses the package infrastructure of the new JNI access method. For more information about this sample program, see "Sample program in Java" on page 1097.

RNGpk.java
illustrates the method available for calling the JNI.

- Java class implementations (*.java files) that call CCA JNI functions need an import line, such as this:
  ```java
  import com.ibm.crypto.cca.jni.*;
  ```
- The Java classpath must point to CNM.jar.

The default location of the sample code is /opt/IBM/CCA/samples for both, SUSE and Red Hat distributions.

Preparing your Java environment
Before you can compile and run Java applications that use the CCA Java Native Interface (JNI), you must install a Java version that is supported by your distribution.

Install Java from the distribution installation media or from other authorized sources for that distribution.

The CCA JNI has been tested with IBM Java version 1.8.0 for the following distributions:

- Red Hat Enterprise Linux 7.4
- Red Hat Enterprise Linux 6.9
- SUSE Linux Enterprise Server 11 SP4
- SUSE Linux Enterprise Server 12 SP3
- Ubuntu 16.04.4

Later versions of Java, as provided with a distribution, might work, although they have not been tested.

For compiling and running applications that use the CCA JNI, you need access to the java and javac executables. Use one of the following methods to ensure that you can call the command without preconditions:

- Add the path to the java and javac executable to your PATH environment variable.
- Create soft links from the java and javac executables from wherever they are located to a directory that is in your PATH environment variable by default, such as /usr/bin.

Note: IBM Java 1.7.0 interaction

Some users reported an issue with IBM Java, support item IV52646. The following exception occurs:
Exception in thread "main" java.lang.UnsatisfiedLinkError: nio
(/usr/lib64/jvm/java-1.7.0-ibm-1.7.0/jre/lib/s390x/libnio.so:
symbol NET_Bind, version SUNWprivate_1.1 not defined in file libnet.so
with link time reference)

This occurs because the s390x version of the libnio.so library is being loaded
when a Java program is run. The correct version is:
/usr/lib64/jvm/java-1.7.0-ibm-1.7.0/jre/lib/s390x/libnio.so

To ensure that the application uses the correct version, do the following in the
application environment as a test, and then add an appropriate adjustment to the
application environment file (such as .bashrc or .bash_profile):
export LD_LIBRARY_PATH=/usr/lib64/jvm/java-1.7.0-ibm-1.7.0/jre/lib/s390x:$LD_LIBRARY_PATH

**Building Java applications using the CCA JNI**

Call the CCA JNI using the Java package infrastructure.

**Building the Java byte code:**

Perform these steps to compile a Java source file.

**Procedure**

1. Ensure that your LD_LIBRARY_PATH variable points to the CCA libraries. This
element points to the default location:
   export LD_LIBRARY_PATH=/usr/lib64

2. Change to the directory that contains the source code file of the program you
   want to compile.

3. Issue this command to compile a Java source code file `<program>.java`:
   javac -classpath `<fullpath>/CNM.jar` `<program>.java`
   where `<fullpath>` is the location of CNM.jar.
   The following example uses the default path and the sample program of
   "Sample program in Java" on page 1097
   `javac -classpath /opt/IBM/CCA/cnm/CNM.jar RNpk.java`

   If the program uses the Master Key Process (CSNBMKP) verb, you must also
   include CNMJKM.jar in the class path (see "Using the Master Key Process
   (CSNBMKP) verb" on page 25). Issue this command to compile such programs:
   `javac -classpath `<fullpath>/CNM.jar:<fullpath>/CNMJKM.jar` `<program>.java`

   **Tip:** Instead of using the -classpath option, you can set the CLASSPATH
   variable to point to CNM.jar and, if required, CNMJKM.jar.

**Running the Java byte code:**

Perform these steps to run a compiled Java program.

**Procedure**

1. Change to the directory that contains the compiled Java program you want to
   run.

2. Issue this command to run a program `<program>.class`:
   `java -classpath `<fullpath>/CNM.jar:. `<program>.class`
where `<fullpath>` is the location of CNM.jar. The period (.) at the end of the class path ensures that Java can find `<program>.class` in the current directory.

The following example uses the default path and the compiled sample program of “Sample program in Java” on page 1097:

```
javac -classpath /opt/IBM/CCA/cnm/CNM.jar:. RNGpk.class
```

If the program uses the Master Key Process (CSNBMKP) verb in addition to other verbs, you must also include CNMMK.jar in the class path (see “Using the Master Key Process (CSNBMKP) verb” on page 25). Issue this command to run such programs:

```
java -classpath <fullpath>/CNM.jar:<fullpath>/CNMMK.jar:. <program>.class
```

**Tip:** Instead of using the `-classpath` option, you can set the CLASSPATH variable to point to CNM.jar and, if required, CNMMK.jar.
Chapter 2. Running CCA verbs in PCI-HSM 2016 compliance mode

Starting with the Crypto Express6 feature (CEX6S), the IBM Z cryptographic hardware offers the possibility to run CCA applications using keys that conform to the PIN Transaction Security (PTS) standards of the Payment Card Industry (PCI). This PCI PTS standard implemented on a hardware security module as of June 2016 is referred to in this documentation as PCI-HSM 2016 compliance mode.

When configured as a CCA coprocessor, a CEX6C adapter is capable of running in PCI-HSM 2016 compliance mode. In order for the requirements of PCI-HSM 2016 to apply to a workload, the workload must be using compliant-tagged key tokens. Such a workload can be processed in the domain of a cryptographic coprocessor which is also switched into PCI-HSM 2016 compliance mode.

You can use a TKE V9.0 or higher to set a domain on the adapter to this PCI-HSM 2016 compliance mode. Then you can use the TKE to manually enter or create operational secure keys that have a compliance-tag, or you can create compliance-tagged keys directly from an application using verbs such as CSNBKTB or CSNBKGN. Thus, the CCA applications that invoke services that use PCI-HSM 2016 compliant keys, also apply to the PTS standards of the Payment Card Industry (PCI).

Such a compliant-tagged key token is a key token that adheres to the requirements of the PCI-HSM 2016 compliance mode. The compliant tag is indicated by bit 58 in the control vector of the key token. For more information about the compliant tag, see Chapter 21, “Control vectors and changing control vectors with the Control Vector Translate verb,” on page 989.

For verbs that support PCI-HSM 2016 compliant-tagged key tokens, their description in Part 2, “CCA verbs,” on page 87 contains a note that indicates this support. Also, Table 312 on page 1083 provides a compact list of these verbs.

Restriction: Only internal, fixed-length TDES key tokens can be tagged to be compliant to PCI-HSM 2016.

Watch the TKE Demonstration Video to learn how to use a TKE V9.0 to set a domain on a crypto adapter to PCI-HSM 2016 compliance mode.

The PCI-HSM 2016 compliance mode places certain restrictions on the use of compliant-tagged key tokens. For more detail, see “Impact of the PCI-HSM 2016 compliance mode on the callable verbs” on page 1082.

Existing applications prior to CCA release 6.0 that exploit verbs that are now PCI-HSM 2016 compliant, can be migrated to apply to PCI-HSM 2016 compliance mode. For more information, see “Migrating applications to PCI-HSM 2016 compliance mode” on page 1088.
Compliance warning events

CCA can generate warning events for operations that might need modifications to meet requirements for the request to be compliant. In the system log, the compliance warning indicates whether the request was compliant or not, while the request processes normally with no change in behavior noticeable to the application.

You set the environment variable CSU_CMP_WARN_MODE=SYSLOG to indicate that the generation of warning events is desired.

- Warning events are generated for requests that can be successfully made compliant. When the request is potentially compliant, you can convert the used key tokens to compliant-tagged key tokens and the operation is successful.
- When the request is non-compliant, the key tokens, the verb, or the verb and rule combination must be updated to be compliant before the key tokens used can be converted to be compliant-tagged.

Warning events are generated for potentially successful requests where at least one of the key tokens that are used is an internal, version 00 or version 01 DES key token and none of the key tokens is already compliant-tagged. This includes external key tokens and output key tokens. For services that do not accept compliant-tagged key tokens, only internal key tokens that are used as input to the service are included in the event. For services that accept compliant-tagged key tokens, all key tokens that are used are included in the event.

Warning events are generated in the form of SYSLOG entries. See "Generating warning events" on page 1085 for more information and examples.

For detailed information about the format of generated warning events and the required steps for making applications compliant, read "Migrating applications to PCI-HSM 2016 compliance mode" on page 1088.

Restriction: Cryptographic coprocessors earlier than CEX6C do not respond to a specification of desired warning events. In addition, Linux sometimes buffers log entries before posting them, thus causing a short delay.
Chapter 3. Using AES, DES, and HMAC cryptography and verbs

You can use AES, DES, and HMAC cryptographic functions that CCA provides. CCA also provides cryptographic key functions, and you can use CCA to build key tokens.

The CEX*C protects data from unauthorized disclosure or modification. This coprocessor protects data stored within a system, stored in a file off a system on magnetic tape, and sent between systems. The coprocessor also authenticates the identity of customers in the financial industry and authenticates messages from originator to receiver. The coprocessor uses cryptography to perform these functions.

The CCA API for the coprocessor provides access to cryptographic functions through verbs. A verb is a routine that receives control using a function call from an application program. Each verb performs one or more cryptographic functions, including:

- Generating and managing cryptographic keys
- Enciphering and deciphering data with encrypted keys using either the U.S. National Institute of Standards and Technology (NIST) Data Encryption Standard (DES) or Advanced Encryption Standard (AES)
- Re-enciphering text from encryption under one key to encryption under another key
- Encoding and decoding data with clear keys
- Generating random numbers
- Ensuring data integrity and verifying message authentication
- Generating, verifying, and translating personal identification numbers (PINs) that identify a customer on a financial system

Functions of the AES, DES, and HMAC cryptographic keys

The CCA API provides functions to create, import, and export AES, DES, and HMAC keys.

Key separation

The cryptographic coprocessor controls the use of keys by separating them into unique types, allowing you to use a specific type of key only for its intended purpose.

For example, a key used to protect data cannot be used to protect a key.

A CCA system has only one DES or AES master key. However, to provide for key separation, the cryptographic coprocessor automatically encrypts each type of key in a fixed-length token under a unique variation of the master key. Each variation of the master key encrypts a different type of key. Although you enter only one master key, you have a unique master key to encrypt all other keys of a certain type.
Master key variant for fixed-length tokens

Whenever the master key is used to encipher a key, the cryptographic coprocessor produces a variation of the master key according to the type of key that the master key will encipher.

These variations are called master key variants. The cryptographic coprocessor creates a master key variant by XORing a fixed pattern, called a control vector, onto the master key. A unique control vector is associated with each type of key. For example, all the different types of data-encrypting, PIN, MAC, and transport keys each use a unique control vector which is XORed with the master key in order to produce the variant. The different key types are described in “Types of keys” on page 41.

Each master key variant protects a different type of key. It is similar to having a unique master key protect all the keys of a certain type.

The master key, in the form of master key variants, protects keys operating on the system. A key can be used in a cryptographic function only when it is enciphered under a master key. When systems want to share keys, transport keys are used to protect keys sent outside of systems. When a key is enciphered under a transport key, the key cannot be used in a cryptographic function. It must first be brought on to a system and enciphered under the system’s master key, or exported to another system where it will then be enciphered under that system’s master key.

Transport key variant for fixed-length tokens

Like the master key, the coprocessor creates variations of a transport key to encrypt a key according to its type.

This allows for key separation when a key is transported off the system. A transport key variant, also called key-encrypting key variant, is created the same way a master key variant is created. The transport key’s clear value is XORed with a control vector associated with the key type of the key it protects.

Note: To exchange keys with systems that do not recognize transport key variants, the coprocessor allows you to encrypt selected keys under a transport key itself, not under the transport key variant. For more information, see NOCV Importers and Exporters on page “NOCV importers and exporters” on page 43.

Key forms

A key that is protected under the master key is in operational form, which means the coprocessor can use it in cryptographic functions on the system.

When you store a key with a file or send it to another system, the key is enciphered under a transport key rather than the master key. The transport key is a key shared by your system and another system for the purpose of securely exchanging other keys. When CCA enciphers a key under a transport key, the key is not in operational form and cannot be used to perform cryptographic functions.

When a key is enciphered under a transport key, the sending system considers the key in exportable form. The receiving system considers the key in importable form. When a key is re-enciphered from under a transport key to under a system’s master key, it is in operational form again.
Enciphered keys appear in three forms. The form you need depends on how and when you use a key.

- **Operational** key form is used at the local system. Many verbs can use an operational key form. The Key Generate, Key Import, Data Key Import, Clear Key Import, and Multiple Clear Key Import verbs can create an operational key form.

- **Exportable** key form is transported to another cryptographic system. It can be passed only to another system. The CCA verbs cannot use it for cryptographic functions. The Key Generate, Data Key Export, and Key Export verbs produce the exportable key form.

- **Importable** key form can be transformed into operational form on the local system. The Key Import verb (CSNBKIM) and the Data Key Import verb (CSNBKDM) can use an importable key form. Only the Key Generate verb (CSNBKGN) can create an importable key form.

For more information about the key types, see "Functions of the AES, DES, and HMAC cryptographic keys" on page 33. See Chapter 20, "Key forms and types used in the Key Generate verb," on page 985 for more information about key form.

### Symmetric key (DES, AES) flow

The conversion from one key to another key is considered to be a one-way flow. An operational key form cannot be turned back into an importable key form. An exportable key form cannot be turned back into an operational or importable key form. The flow of CCA key forms can be in only one direction:

IMPORTABLE → OPERATIONAL → EXPORTABLE

### Key token

CCA supports two types of symmetric key tokens, fixed-length and variable-length.

An AES or DES fixed-length token is a 64-byte field composed of a key value and control information in the control vector. An HMAC key token is a variable-length token composed of a key value and control information. The control information is assigned to the key when the coprocessor creates the key. The key token can be either an internal key token, an external key token, or a null key token. Through the use of key tokens, CCA can do the following:

- Support continuous operation across a master key change
- Control use of keys in cryptographic services

If the first byte of the key identifier is X’01’, the key identifier is interpreted as an **internal key token**. An internal key token is a token that can be used only on the CCA system that created it or another CCA system with the same host master key. It contains a key that is encrypted under the master key.

An application obtains an internal key token by using one of the verbs such as those listed below. The verbs are described in detail in Chapter 8, “Managing AES, DES, and HMAC cryptographic keys,” on page 165.

- AES Key Record Read
- Clear Key Import
- Data Key Import
- DES Key Record Read
- Key Generate
The master key could be dynamically changed between the time that you invoke a verb, such as the Key Import verb, to obtain a key token, and the time that you pass the key token to the Encipher verb. When a change to the master key occurs, the coprocessor will still successfully use the key, because it stores a copy of the old master key as well as the new one.

**Attention:** If an internal key token held in user storage is not used while the master key is changed twice, the internal key token is no longer usable. A return code of 0 with a reason code of 10001 notifies you that the master key used to decrypt the key used in your operation was an old master key, as a reminder that you should use one of the Key Token Change verbs to re-encipher your key under the current or new master key (as desired, see verbs for description).

If the first byte of the key identifier is X'02', the key identifier is interpreted as an **external key token**. By using the external key token, you can exchange keys between systems. It contains a key that is encrypted under a key-encrypting key.

An external key token contains an encrypted key and control information to allow compatible cryptographic systems to:

- Have a standard method of exchanging keys
- Control the use of keys through the control vector
- Merge the key with other information needed to use the key

An application obtains the external key token by using one of the verbs such as those listed below. They are described in detail in Chapter 8, “Managing AES, DES, and HMAC cryptographic keys,” on page 165.

- Key Generate
- Key Export
- Data Key Export
- Symmetric Key Export

If the first byte of the key identifier is X'00', the key identifier is interpreted as a **null key token**. Use the null key token to import a key from a system that cannot produce external key tokens. That is, if you have an 8 or 16-byte key that has been encrypted under an import key, but is not imbedded within a token, place the encrypted key in a null key token and then invoke the Key Import verb to get the key in operational form.

For debugging information, see Chapter 19, “Key token formats,” on page 847 for the format of internal, external, or null key tokens.
Key wrapping

CCA supports two methods of wrapping the key value in a fixed-length symmetric key token for DES and AES keys: the original ECB wrapping and an enhanced CBC wrapping method, which is ANSI X9.24 compliant.

These methods use the 64-byte token. HMAC keys use a variable length token with associated data and the payload wrapping method. Variable-length tokens are wrapped by using the AESKW wrapping method that is defined in ANSI X9.102.

AES key wrapping
The key value in AES tokens are wrapped using the AES algorithm and cipher block chaining (CBC) mode of encryption.

The key value is left-aligned in a 32-byte block, padded on the right with zero, and encrypted.

The enhanced wrapping of an AES key (*K) using an AES *MK is defined as:

\[ e_{\text{c} \text{b} \text{c} \text{K}}(*K) = e_{\text{c} \text{b} \text{c} \text{K}}(*K) \]

where:

- \( e_k(m) \) or \( e_{\text{ek}}(*k) \): message \( m \) is encrypted (e) with key \(*k\) or key \(*k\) is encrypted with key encrypting key \(*\text{kek}\)

- \( \text{ecbc}_{\text{h}}(m) \): message \( m \) is encrypted (e) with key \(*k\) using the cipher block chaining (cbc) mode of operation

DES key wrapping
The key value in a DES key token are wrapped using one of several possible methods.

The methods are:

**Original method**
The key value in DES tokens are encrypted using triple-DES encryption, and key parts are encrypted separately. The original method is the default on CEX*C before CEX6C.

**Enhanced method**
The key value for keys is bundled with other token data and encrypted using triple-DES encryption and cipher block chaining mode. The enhanced method applies only to DES key tokens. The enhanced method of symmetric key wrapping is designed to be ANSI X9.24 and PCI-HSM 2016 compliant. The enhanced method is the default as of CEX6C.

**Compliant-tagged method**
This method is the same as the enhanced method except for a few important differences:
- The master-key-to-wrapping-key derivation process uses a different input string, giving a different actual wrapping key.
- The process for binding the key halves together before encryption uses SHA-256 instead of SHA-1. This method was introduced with CCA 6.0 and is exclusive to compliant-tagged DES keys.
When the Allow weak wrapping of compliance-tagged keys by DES MK (X'02EB') access control point is enabled, any service which attempts to wrap a compliant-tagged key token with a weaker DES master key in terms of effective key strength (1 or more 56 bit key parts are repeated) will succeed.

**ECB wrapping of DES keys (original method)**

In ECB wrapping, a double length key (*K) is wrapped using a double-length key-encrypting key (*KEK).

The definition is as follows:

\[
ed_{KEK}(KL) \| ed_{KEK}(KR) = ed_{KEK}(d_{KEK}(ed_{KEK}(KL))) \| ed_{KEK}(d_{KEK}(ed_{KEK}(KR)))
\]

where:

- **KL** is the left 64 bits of *K
- **KR** is the right 64 bits of *K
- **KEKL** is the left 64 bits of *KEK
- **KEKR** is the right 64 bits of *KEK
- **||** means concatenation
- \(d_k(m)\) or \(e_k(m)\) means that message \(m\) is decrypted (d) or encrypted (e) with key \(k\).

**Enhanced CBC wrapping of DES keys**

The enhanced CBC wrapping method uses triple-DES encryption, an internal chaining of the key value, and CBC mode.

The enhanced wrapping of a double-length key (*K) using a double-length key-encrypting key (*KEK) is defined as:

\[
ed_{KEK}(KL) = ecbc_{KEKL}(dcbc_{KEKR}(ecbc_{KEKL}(KLPRIME || KR)))
\]

Where:

- **KL** is the left 64 bits of *K
- **KR** is the right 64 bits of *K
- **KLPRIME** is the 64-bit modified value of KL
- **KEKL** is the left 64 bits of *KEK
- **KEKR** is the right 64 bits of *KEK
- **SHA1(X)** is the 160-bit SHA-1 hash of X
- **||** means concatenation
- **XOR** means bitwise exclusive OR
- **ecbc** means encryption using cipher block chaining mode
- **dcbc** means decryption using cipher block chaining mode.
Wrapping key derivation for enhanced wrapping of DES keys

The wrapping key is exactly the same key that is used by the legacy wrapping method (the only method used by CCA 4.0.0), with one exception.

Instead of using the base key itself (master key or key-encrypting key), a key that is derived from that base key is used. The derived key will have the control vector applied to it in the standard CCA manner, and then use the resulting key to wrap the new-format target key token.

The reason for using a derived key is to ensure that no attacks against this wrapping scheme are possible using the existing CCA functions. For example, it was observed that an attack was possible by copying the wrapped key into an ECB CCA key token, if the wrapping key was used instead of a derivative of that key.

The key will be derived using a method defined in the U.S. National Institute of Standards and Technology (NIST) standard SP 800-108, Recommendation for Key Derivation Using Pseudorandom Functions (October, 2009). Derivation will use the method KDF in counter mode using pseudo-random function (PRF) HMAC-SHA256. This method provides sufficient strength for deriving keys for any algorithm used.

The HMAC algorithm is defined as:

\[ \text{HMAC}(K, \text{text}) = H((K0 \text{ XOR } \text{opad}) \ || \ H((K0 \text{ XOR } \text{ipad}) \ || \ \text{text})) \]

where:

- \( H \) is an approved hash function.
- \( K \) is a secret key shared between the originator and the intended receivers.
- \( K0 \) is the key \( K \) after any necessary preprocessing to form a key of the proper length.
- \( \text{ipad} \) is the constant X'36' repeated to form a string the same length as \( K0 \)
- \( \text{opad} \) is the constant X'5C' repeated to form a string the same length as \( K0 \)
- \( \text{text} \) is the text to be hashed.
- \( || \) means concatenation
- \( \text{XOR} \) means bitwise exclusive OR

If the key \( K \) is equal in length to the input block size of the hash function (512 bits for SHA-256), \( K0 \) is set to the value of \( K \). Otherwise, \( K0 \) is formed from \( K \) by hashing or padding.

The Key Derivation Function (KDF) specification calls for inputs optionally including two byte strings, Label and Context. The Context will not be used. The Label will contain information on the usage of this key, to distinguish it from other derivations that CCA may use in the future for different purposes. Because the security of the derivation process is rooted in the security of the derivation key and in the HMAC and Key Derivation Functions (KDF) themselves, it is not necessary for this label string to be of any particular minimum size. The separation indicator byte of X'00' specified in the NIST document will follow the label.

The label value will be defined so that it is unique to derivation for this key wrapping process. This means that any future designs that use the same KDF must use a different value for the label. The label will be the 16 byte value consisting of the following ASCII characters:
ENHANCEDWRAP2010 (X'454E4841 4E434544 57524150 32303130 ')

The parameters for the counter mode KDF defined in NIST standard SP 800-108 are:

**Fixed values:**
- \( h \) Length of output of PRF, 256 bits
- \( r \) Length of the counter, in bits, 32. The counter is an unsigned 4-byte value.

**Inputs:**
- \( KI \) (input key) - The key we are deriving from.
- \( Label \) - The value shown above (ASCII ENHANCEDWRAP2010).
- Separator byte - X'00' following the label value.
- Context - A null string. No context is used.
- \( L \) - The length of the derived key to be produced, rounded up to the next multiple of 256.
- \( PRF \) - HMAC-SHA256.

**Variable length token (AESKW method)**
The wrapping method for the variable-length key tokens with AESKW is defined in standard ANSI X9.102.

The wrapping of the payload of a variable length key (*K) using an AES *MK is defined as:

\[
e_{MK}(*K) = e_{AESKW+MK}(P)
\]

\( P = ICV || Pad\ length || Hash\ length || Hash\ options || Data\ hash || *K || Padding\)

**Where:**
- **ICV** Is the 6 byte constant X'A6A6A6A6A6A6'.
- **Pad length** Is the length of the padding in bits.
- **Hash length** Is the length of the Data Hash in bytes.
- **Hash options** Is a 4-byte field.
- **Data hash** Is the hash of the associated data block.
- **Padding** Is the number of bytes of X'00' used to make the overall length of \( P \) a multiple of 16.

**eAESKW**
Means encryption using the AESKW method.

**Control vector**
A unique control vector exists for each type of CCA key.

For an internal key token, the coprocessor XORs the master key with the control vector associated with the type of key the master key will encipher. The control vector ensures that an operational key is used only in cryptographic functions for
which it is intended. For example, the control vector for an input PIN-encrypting key ensures that such a key can be used only in the Encrypted PIN Translate and Encrypted PIN Verify functions.

Types of keys
The cryptographic keys are grouped into the following categories based on the functions that they perform.

Symmetric keys master key (SYM-MK)
The SYM-MK master key is a triple-length (192-bit) key that is used only to encrypt other DES keys on the coprocessor. The administrator installs and changes the SYM-MK master key using the `panel.exe` utility, the clear key entry panels, the z/OS® clear key entry panels, or the optional Trusted Key Entry (TKE) workstation. The master key always remains within the secure boundary of the coprocessor. It is used only to encipher and decipher keys that are in operational form.

Note: If the coprocessor is shared with z/OS, the SYM-MK key must be a double-length (128-bit) key. This means that the first 64 bits and the last 64 bits of the key must be identical. If the master key is loaded by z/OS CCA or from a TKE workstation, it will automatically be a double-length key.

AES keys master key (AES-MK)
The AES-MK master key is a 256-bit key that is used to encrypt other AES keys and HMAC keys on the coprocessor. The administrator installs and changes the AES-MK master key using the `panel.exe` utility, the clear key entry panels, the z/OS clear key entry panels, or the optional Trusted Key Entry (TKE) workstation. The master key always remains within the secure boundary of the coprocessor. It is used only to encipher and decipher keys that are in operational form.

Asymmetric keys master key (ASYM-MK)
The ASYM-MK is a triple-length (192-bit) key that is used to protect RSA private keys on the coprocessor. The administrator installs and changes the ASYM-MK master key using the `panel.exe` utility, the clear key entry panels, the z/OS clear key entry panels, or the optional Trusted Key Entry (TKE) workstation. The master key always remains within the secure boundary of the coprocessor. It is used only to encipher and decipher keys that are in operational form.

AES CIPHER keys
The AES cipher keys are 128-bit, 192-bit, and 256-bit keys that protect data privacy. If you intend to use a cipher key for an extended period, you can store it in key storage so that it will be re-enciphered if the master key is changed.

AES PKA master key (APKA-MK)
The APKA-MK key, introduced to CCA beginning with Release 4.1.0, is used to encrypt and decrypt the Object Protection Key (OPK) that is itself used to wrap the key material of an Elliptic Curve Cryptography (ECC) key. ECC keys are asymmetric. The APKA-MK is a 256-bit (32-byte) value. The administrator installs and changes the APKA-MK master key using the `panel.exe` utility, the clear key entry panels, the z/OS clear key entry panels, or the optional Trusted Key Entry (TKE) workstation.

Data-encrypting keys
The data-encrypting keys are single-length DES (64-bit), double-length DES (128-bit), or triple-length DES (192-bit) keys, or 128-bit, 192-bit or 256-bit
AES keys that protect data privacy. Single-length DES data-encrypting keys can also be used to encode and decode data and authenticate data sent in messages. If you intend to use a data-encrypting key for an extended period of time, you can store it in the CCA key storage file so that it will be re-enciphered if the master key is changed.

You can use single-length DES data-encrypting keys in the Encipher and Decipher verbs to manage data, and also in the MAC Generate and MAC Verify verbs. Double-length DES and triple-length DES data-encrypting keys can be used in the Encipher and Decipher verbs for more secure data privacy. DATAC is also a double-length DES data encrypting key.

AES data-encrypting keys can be used in services similar to DES data-encrypting key services.

**DES CIPHER keys**
These consist of CIPHER, ENCIPHER, and DECIPHER keys. They are single and double length DES keys for enciphering and deciphering data.

**Ciphertext translation keys**
These ciphertext translation keys consist of CIPHERXI, CIPHERXL, and CIPHERXO keys. They protect data that is transmitted through intermediate systems when the originator and receiver do not share a common key. Data that is enciphered under one ciphertext translation key is re-enciphered under another ciphertext translation key on the intermediate node. During this process, the data never appears in the clear. These keys are double-length.

**HMAC keys**
HMAC keys are variable-length symmetric keys. The length is in the range of 80 - 2024. HMAC keys are used to generate and verify HMACs using the FIPS-198 algorithm, with the HMAC Generate and HMAC Verify verbs.

- Operational keys will be encrypted under the AES master key
- HMAC keys can be imported and exported under an RSA key.
- HMAC keys will be stored in the AES key storage file. The AES master key must be active.

For more information about HMAC keys and verb processing, see **Chapter 10, “Verifying data integrity and authenticating messages,” on page 417.**

**MAC keys**
The MAC keys are single-length DES (64-bits - DATAM, DATAMV, MAC, and MACVER, ) and double-length DES (128-bits - DATAM, DATAMV, MAC, and MACVER) keys used for the verbs that generate and verify MACs.

**PIN keys**
The personal identification number (PIN) is a basis for verifying the identity of a customer across financial industry networks. PIN keys are used in cryptographic functions to generate, translate, and verify PINs, and protect PIN blocks. They are all double-length DES (128 bits) keys. PIN keys are used in the Clear PIN Generate, Encrypted PIN Verify, and Encrypted PIN Translate verbs.

For installations that do not support double-length DES 128-bit keys, effective single-length DES keys are provided. For a single-length DES key, the left key half of the key equals the right key half.
“Processing personal identification numbers” on page 56 gives an overview of the PIN algorithms you need to know to write your own application programs.

AES transport keys (or key-encrypting keys)
Transport keys are also known as key-encrypting keys. They are used to protect AES and HMAC keys when you distribute them from one system to another.

There are two types of AES transport keys:

**Exporter key-encrypting key**
This type of key protects keys of any type that are sent from your system to another system. The exporter key at the originator is the same key as the importer key of the receiver.

**Importer key-encrypting key**
This type of key protects keys of any type that are sent from another system to your system. It also protects keys that you store externally in a file that you can import to your system at another time. The importer key at the receiver is the same key as the exporter key at the originator.

DES transport keys (or key-encrypting keys)
Transport keys are also known as key-encrypting keys. They are double-length (128 bits) DES keys used to protect keys when you distribute them from one system to another.

There are several types of DES transport keys:

**Exporter or OKEYXLAT key-encrypting key**
This type of key protects keys of any type that are sent from your system to another system. The exporter key at the originator is the same key as the importer key of the receiver.

**Importer or IKEYXLAT key-encrypting key**
This type of key protects keys of any type that are sent from another system to your system. It also protects keys that you store externally in a file that you can import to your system later. The importer key at the receiver is the same key as the exporter key at the originator.

NOCV importers and exporters
These keys are key-encrypting keys used to exchange keys with systems that do not recognize key-encrypting key variants. There are some requirements and restrictions for the use of NOCV key-encrypting keys:

- The use of NOCV IMPORTERs and EXPORTERs is controlled by access control points in the coprocessor’s role-based access control system.
- Only programs in system or supervisor state can use the NOCV key-encrypting key in the form of tokens in verbs. Any program can use NOCV key-encrypting keys with label names from the key storage.
- Access to NOCV key-encrypting keys should be carefully controlled, because use of these keys can reduce security in your key management process.
- NOCV key-encrypting key can be used to encrypt single or double length DES keys with standard CVs for key types DATA,
DATAC, DATAM, DATAMV, DATAXLAT, EXPORTER, IKEYXLAT, IMPORTER, IPINENC, single-length MAC, single-length MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER.

- NOCV key-encrypting keys can be used with triple length DATA keys. Because DATA keys have 0 CVs, processing will be the same as if the key-encrypting keys are standard key-encrypting keys (not the NOCV key-encrypting key).

**Note:** A key-encrypting key should be as strong or stronger than the key that it is wrapping.

You use key-encrypting keys to protect keys that are transported using any of the following verbs: Data Key Export, Key Export, Key Import, Clear Key Import, Multiple Clear Key Import, Key Generate, Key Generate2, Key Translate and Key Translate2.

For installations that do not support double-length key-encrypting keys, effective single-length keys are provided. For an effective single-length key, the clear key value of the left key half equals the clear key value of the right key half.

**Key-generating keys**

Key-generating keys are double-length keys used to derive other keys. This is often used in smart card applications.

**Clear keys**

A clear key is the base value of a key, and is not encrypted under another key. Encrypted keys are keys whose base value has been encrypted under another key.

To convert a clear key to an encrypted data key in operational form, use either the Clear Key Import verb or the Multiple Clear Key Import verb.

Table 6 describes the key types.

<table>
<thead>
<tr>
<th>Key type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AESDATA</td>
<td>Data encrypting key. Use the AES 128-bit, 192-bit, or 256-bit key to encipher and decipher data.</td>
</tr>
<tr>
<td>AESTOKEN</td>
<td>Can contain an AES key.</td>
</tr>
<tr>
<td>CIPHER</td>
<td>AES 128-bit, 192-bit, or 256-bit key is used to encrypt or decrypt data. It can be used in the Symmetric Algorithm Decipher and Symmetric Algorithm Encipher verbs.</td>
</tr>
<tr>
<td></td>
<td>DES 128-bit, 192-bit, or 256-bit key is used to encrypt or decrypt data. It can be used in the Encipher and Decipher verbs.</td>
</tr>
<tr>
<td></td>
<td>Used only to encrypt or decrypt data. This is a single or double length key and can be used in the Encipher or Decipher verbs.</td>
</tr>
<tr>
<td>CIPHERXI</td>
<td>Usable with the Cipher Text Translate2 verb (translate inbound key only)</td>
</tr>
<tr>
<td>CIPHERXL</td>
<td>Usable with the Cipher Text Translate2 verb (translate inbound or outbound key)</td>
</tr>
<tr>
<td>CIPHERXO</td>
<td>Usable with the Cipher Text Translate2 verb (translate outbound key only)</td>
</tr>
<tr>
<td>CLRAES</td>
<td>Data encrypting key. The key value is not encrypted. Use this AES 128-bit, 192-bit, or 256-bit key to encipher and decipher data.</td>
</tr>
<tr>
<td>CVARDEC</td>
<td>The cryptographic variable decipher service, which is available in some CCA implementations, uses a CVARDEC key to decrypt plaintext by using the Cipher Block Chaining (CBC) method. This is a single-length key.</td>
</tr>
<tr>
<td>Key type</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CVARENC</td>
<td>The cryptographic variable encipher service, which is available in some CCA implementations, uses a CVARENC key to encrypt plaintext by using the Cipher Block Chaining (CBC) method. This is a single-length key.</td>
</tr>
<tr>
<td>CVARPINE</td>
<td>Used to encrypt a PIN value for decryption in a PIN-printing application. This is a single-length key.</td>
</tr>
<tr>
<td>CVARXCVL</td>
<td>Used to encrypt special control values in DES key management. This is a single-length key.</td>
</tr>
<tr>
<td>CVARXCVR</td>
<td>Used to encrypt special control values in DES key management. This is a single-length key.</td>
</tr>
<tr>
<td>DATA</td>
<td>Data encrypting key. Use this DES single-length, double-length, or triple-length key to encipher and decipher data. Use the AES 128-bit, 192-bit, or 256-bit key to encipher and decipher data.</td>
</tr>
<tr>
<td>DATAC</td>
<td>Used to specify a DATA-class key that will perform in the Encipher and Decipher verbs, but not in the MAC Generate or MAC Verify verbs. This is a double-length key. Only available with a CEX'C.</td>
</tr>
<tr>
<td>DATAM</td>
<td>Key-encrypting keys that have a control vector with this attribute formerly could only be used to transport keys with a key type of DATA, CIPHER, ENCRYPTER, DECIPHER, MAC, and MACVER. The meaning of this keyword has been discontinued and its usage is allowed for backward compatibility reasons only.</td>
</tr>
<tr>
<td>DATAMV</td>
<td>Used to specify a DATA-class key that performs in the MAC Verify verb, but not in the MAC Generate, Encipher, or Decipher verbs.</td>
</tr>
<tr>
<td>DATAXLAT</td>
<td>Data translation key. Use this single-length key to reencipher text from one DATA key to another.</td>
</tr>
<tr>
<td>DECIPHER</td>
<td>Used only to decrypt data. DECIPHER keys cannot be used in the Encipher (CSNBENC) verb. This is a single-length key.</td>
</tr>
<tr>
<td></td>
<td>This is a single or double length key and can be used in the Decipher verb.</td>
</tr>
<tr>
<td>DKEYGENKY</td>
<td>Used to generate a diversified key based on the key-generating key. This is a double-length key.</td>
</tr>
<tr>
<td>ENCRYPTER</td>
<td>Used only to encrypt data. ENCRYPTER keys cannot be used in the Decipher (CSNBDEC) verb. This is a single-length key.</td>
</tr>
<tr>
<td></td>
<td>This is a single or double length key and can be used in the Encipher verb.</td>
</tr>
<tr>
<td>EXPORTER</td>
<td>Exporter key-encrypting key. Use this double-length DES key or 128-bit, 192-bit or 256-bit AES key to convert a key from operational form into exportable form.</td>
</tr>
<tr>
<td>HMAC</td>
<td>Variable-length HMAC generation key. Use this key to generate or verify a Message Authentication Code using the keyed-hash MAC algorithm.</td>
</tr>
<tr>
<td>HMACVER</td>
<td>Variable-length HMAC verification key. Use this key to verify a Message Authentication Code using the keyed-hash MAC algorithm.</td>
</tr>
<tr>
<td>IKEYXLAT</td>
<td>Used to decrypt an input key in the Key Translate and Key Translate2 verbs. This is a double-length key.</td>
</tr>
<tr>
<td>IMPORTER</td>
<td>Importer key-encrypting key. Exporter key-encrypting key. Use this double-length DES key or 128-bit, 192-bit or 256-bit AES key to convert a key from importable form into operational form.</td>
</tr>
<tr>
<td>IMP-PKA</td>
<td>Double-length limited-authority importer key used to encrypt PKA private key values in PKA external tokens.</td>
</tr>
<tr>
<td>IPINENC</td>
<td>Double-length input PIN-encrypting key. PIN blocks received from other nodes or automatic teller machine (ATM) terminals are encrypted under this type of key. These encrypted PIN blocks are the input to the Encrypted PIN Translate, Encrypted PIN Verify, and Clear PIN Generate Alternate verbs.</td>
</tr>
<tr>
<td>KEYGENKY</td>
<td>Used to generate a key based on the key-generating key. This is a double-length key.</td>
</tr>
<tr>
<td>MAC</td>
<td>Single, double-length, or variable-length MAC generation key. Use this key to generate a message authentication code.</td>
</tr>
</tbody>
</table>
### Table 6. Key types (continued)

<table>
<thead>
<tr>
<th>Key type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MACVER</td>
<td>Single, double-length, or variable-length MAC verification key. Use this key to verify a message authentication code.</td>
</tr>
<tr>
<td>OKEYXLAT</td>
<td>Used to encrypt an output key in the Key Translate and Key Translate2 verbs. This is a double-length key.</td>
</tr>
<tr>
<td>OPINENC</td>
<td>Output PIN-encrypting key. Use this double-length output key to translate PINs. The output PIN blocks from the Encrypted PIN Translate, Encrypted PIN Generate, and Clear PIN Generate Alternate verbs are encrypted under this type of key.</td>
</tr>
<tr>
<td>PINGEN</td>
<td>PIN generation key. Use this double-length key to generate PINs.</td>
</tr>
<tr>
<td>PINVER</td>
<td>PIN verification key. Use this double-length key to verify PINs.</td>
</tr>
<tr>
<td>SECMSG</td>
<td>Used to encrypt PINs or keys in a secure message. This is a double-length key.</td>
</tr>
<tr>
<td>TOKEN</td>
<td>A key token that might contain a key.</td>
</tr>
</tbody>
</table>

### Table 7. Key subtypes specified by the rule_array keyword

<table>
<thead>
<tr>
<th>rule_array keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMEX-CSC</td>
<td>A MAC key that can be used for the AMEX CSC transaction validation process MAC calculation method, used with the Transaction Validation (CSNBTRV) verb.</td>
</tr>
<tr>
<td>ANSI X9.9</td>
<td>A MAC key that can be used for the ANSI X9.9 MAC calculation method, either for MAC Generate (CSNBMGN), MAC Verify (CSNBMVR), or Transaction Validation (CSNBTRV). Other Control Vector bits could limit these usages.</td>
</tr>
<tr>
<td>ANY</td>
<td>Key-encrypting keys that have a control vector with this attribute can be used to transport any type of key. The meaning of this keyword has been discontinued, and its usage is allowed for backward compatibility reasons only.</td>
</tr>
<tr>
<td>ANY-MAC</td>
<td>Can be used with any function or MAC calculation method that uses a MAC key, such as MAC Generate (CSNBMGN), MAC Verify (CSNBMVR), or Transaction Validation (CSNBTRV). This is the default configuration for a MAC key control vector.</td>
</tr>
<tr>
<td>CVVKEY-A</td>
<td>Can be used as ‘Key A’ in either the CVV Generate (CSNBCSG) or CVV Verify (CSNBCSV) verbs, as controlled by the CVV generation and verification Control Vector bits (bits 20 and 21 respectively).</td>
</tr>
<tr>
<td>CVVKEY-B</td>
<td>Can be used as ‘Key B’ in either the CVV Generate (CSNBCSG) or CVV Verify (CSNBCSV) verbs, as controlled by the CVV generation and verification Control Vector bits (bits 20 and 21 respectively).</td>
</tr>
<tr>
<td>DATA</td>
<td>Data encrypting key. Use this 8-byte, 16-byte or 24-byte DES key or 16-byte, 24-byte or 32-byte AES key to encipher and decipher data.</td>
</tr>
<tr>
<td>EPINGENA</td>
<td>Legacy key subtype, used to turn on bit 19 of a PIN Generating Key Control Vector. The default PIN Generating Key type will have this bit on. No PIN generating or processing behavior is currently influenced by this key subtype parameter. EPINGENA is no longer supported, although the bit retains this definition for compatibility. There is no Encrypted Pin Generate Alternate verb</td>
</tr>
<tr>
<td>LMTD-KEK</td>
<td>Key-encrypting keys that have a control vector with this attribute formerly could only be used to exchange keys with key-encrypting keys that carry NOT-KEK, PIN, or DATA key-type ciphering restrictions. The usage of this keyword has been discontinued and its usage is allowed for backward compatibility reasons only.</td>
</tr>
<tr>
<td>NOT-KEK</td>
<td>Key-encrypting keys that have a control vector with this attribute formerly could not be used to transport key-encrypting keys. The meaning of this keyword has been discontinued and its usage is allowed for backward compatibility reasons only.</td>
</tr>
</tbody>
</table>
Table 7. Key subtypes specified by the rule_array keyword (continued)

<table>
<thead>
<tr>
<th>rule_array keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIN</td>
<td>Key-encrypting keys that have a control vector with this attribute formerly could only be used to transport keys with a key type of PINVER, IPINENC, and OPINENC. The usage of this keyword has been discontinued and its usage is allowed for backward compatibility reasons only.</td>
</tr>
</tbody>
</table>
Figure 3. Control Vector Generate and Key Token Build CV keyword combinations for fixed-length DES key tokens
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key-encrypting keys</strong></td>
<td></td>
</tr>
<tr>
<td>OPIM</td>
<td>IMPORTER keys that have a control vector with this attribute can be used in the Key Generate verb when the key form is OPIM.</td>
</tr>
<tr>
<td>IMEX</td>
<td>IMPORTER and EXPORTER keys that have a control vector with this attribute can be used in the Key Generate verb when the key form is IMEX.</td>
</tr>
<tr>
<td>IMIM</td>
<td>IMPORTER keys that have a control vector with this attribute can be used in the Key Generate verb when the key form is IMIM.</td>
</tr>
<tr>
<td>IMPORT</td>
<td>IMPORTER keys that have a control vector with this attribute can be used to import a key in the Key Import verb.</td>
</tr>
<tr>
<td>OPEX</td>
<td>EXPORTER keys that have a control vector with this attribute can be used in the Key Generate verb when the key form is OPEX.</td>
</tr>
<tr>
<td>EXEX</td>
<td>EXPORTER keys that have a control vector with this attribute can be used in the Key Generate verb when the key form is EXEX.</td>
</tr>
<tr>
<td>EXPORT</td>
<td>EXPORTER keys that have a control vector with this attribute can be used to export a key in the Key Export verb.</td>
</tr>
<tr>
<td>Xlate</td>
<td>IMPORTER and EXPORTER keys that have a control vector with this attribute can be used in the Key Translate or Key Translate2 verbs.</td>
</tr>
<tr>
<td>ANY</td>
<td>Key-encrypting keys that have a control vector with this attribute can be used to transport any type of key. The meaning of this keyword has been discontinued, and its usage is allowed for backward compatibility reasons only.</td>
</tr>
<tr>
<td>NOT-KEK</td>
<td>Key-encrypting keys that have a control vector with this attribute formerly could not be used to transport key-encrypting keys. The meaning of this keyword has been discontinued and its usage is allowed for backward compatibility reasons only.</td>
</tr>
<tr>
<td>DATA</td>
<td>Key-encrypting keys that have a control vector with this attribute formerly could only be used to transport keys with a key type of DATA, CIPHER, ENCIPHER, DECIPHER, MAC, and MACVER. The meaning of this keyword has been discontinued and its usage is allowed for backward compatibility reasons only.</td>
</tr>
<tr>
<td>PIN</td>
<td>Key-encrypting keys that have a control vector with this attribute formerly could only be used to transport keys with a key type of PINVER, IPINENC, and OPINENC. The meaning of this keyword has been discontinued and its usage is allowed for backward compatibility reasons only.</td>
</tr>
<tr>
<td>LMTD-KEK</td>
<td>Key-encrypting keys that have a control vector with this attribute formerly could only be used to exchange keys with key-encrypting keys that carry NOT-KEK, PIN, or DATA key-type ciphering restrictions. The meaning of this keyword has been discontinued and its usage is allowed for backward compatibility reasons only.</td>
</tr>
<tr>
<td><strong>MAC keys</strong></td>
<td></td>
</tr>
<tr>
<td>ANY-MAC</td>
<td>Any MAC verb can use this key.</td>
</tr>
<tr>
<td>ANSIx9.9</td>
<td>The meaning of this keyword has been discontinued and its usage is allowed for backward compatibility reasons only.</td>
</tr>
<tr>
<td>CVVKEY-A</td>
<td>Restricts the usage to single-length key-A key or double-length key-A and key-B keys for the CVV Generate and CVV Verify verbs.</td>
</tr>
<tr>
<td>CVVKEY-B</td>
<td>Restricts the usage to single-length key-B key for the CVV Generate and CVV Verify verbs.</td>
</tr>
<tr>
<td><strong>Data operation keys</strong></td>
<td></td>
</tr>
<tr>
<td>SMKEY</td>
<td>Enable the encryption of keys in an EMV secure message.</td>
</tr>
<tr>
<td>SMPIN</td>
<td>Enable the encryption of PINs in an EMV secure message.</td>
</tr>
<tr>
<td><strong>PIN keys</strong></td>
<td></td>
</tr>
<tr>
<td>NO-SPEC</td>
<td>The control vector does not require a specific PIN-calculation method.</td>
</tr>
</tbody>
</table>
## Table 8. DES control vector key-subtype and key-usage keywords (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM-PIN</td>
<td>Select the IBM 3624 PIN-calculation method.</td>
</tr>
<tr>
<td>IBM-PINO</td>
<td>Select the IBM 3624 PIN-calculation method with offset processing.</td>
</tr>
<tr>
<td>GBP-PIN</td>
<td>Select the IBM German Bank Pool PIN-calculation method.</td>
</tr>
<tr>
<td>GBP-PINO</td>
<td>Select the IBM German Bank Pool PIN-calculation method with institution-PIN input or output.</td>
</tr>
<tr>
<td>VISA-PVV</td>
<td>Select the Visa PVV PIN-calculation method.</td>
</tr>
<tr>
<td>INBK-PIN</td>
<td>Select the InterBank PIN-calculation method.</td>
</tr>
<tr>
<td>NOOFFSET</td>
<td>Indicates that a PINGEN or PINVER key cannot participate in the generation or verification of a PIN when an offset process is requested.</td>
</tr>
<tr>
<td>CPINGEN</td>
<td>The key can participate in the Clear PIN Generate verb.</td>
</tr>
<tr>
<td>CPINGENA</td>
<td>The key can participate in the Clear PIN Generate Alternate verb.</td>
</tr>
<tr>
<td>EPINGEN</td>
<td>The key can participate in the Encrypted PIN Generate verb.</td>
</tr>
<tr>
<td>EPINVER</td>
<td>The key can participate in the Encrypted PIN Verify verb.</td>
</tr>
<tr>
<td>CPINENC</td>
<td>The key can participate in the Clear PIN Encrypt verb.</td>
</tr>
<tr>
<td>REFORMAT</td>
<td>The key can participate in the Encrypted PIN Translate verb in the Reformat mode.</td>
</tr>
<tr>
<td>TRANSLAT</td>
<td>The key can participate in the Encrypted PIN Translate verb in the <em>translate</em> mode.</td>
</tr>
</tbody>
</table>

*Key-generating keys*

<table>
<thead>
<tr>
<th>CLR8-ENC</th>
<th>The key can be used to multiply encrypt eight bytes of clear data with a generating key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DALL</td>
<td>The key can be used to generate keys with the following key types: DATA, DATAC, DATAM, DATAMV, DMKEY, DMPIN, EXPORTER, IKEYXLAT, IMPORTER, MAC, MACVER, OKEYXLAT, and PINVER.</td>
</tr>
<tr>
<td>DDATA</td>
<td>The key can be used to generate a single-length or double-length DATA or DATAC key.</td>
</tr>
<tr>
<td>DEXP</td>
<td>The key can be used to generate an EXPORTER or an OKEYXLAT key.</td>
</tr>
<tr>
<td>DIMP</td>
<td>The key can be used to generate an IMPORTER or an IKEYXLAT key.</td>
</tr>
<tr>
<td>DMAC</td>
<td>The key can be used to generate a MAC or DATAM key.</td>
</tr>
<tr>
<td>DMKEY</td>
<td>The key can be used to generate a SECMSG with SMKEY secure messaging key for encrypting keys.</td>
</tr>
<tr>
<td>DMPIN</td>
<td>The key can be used to generate a SECMSG with SMPIN secure messaging key for encrypting PINs.</td>
</tr>
<tr>
<td>DMV</td>
<td>The key can be used to generate a MACVER or DATAMV key.</td>
</tr>
<tr>
<td>DPVR</td>
<td>The key can be used to generate a PINVER key.</td>
</tr>
<tr>
<td>DKYL0</td>
<td>A DKYGENKY key with this subtype can be used to generate a key based on the key-usage bits.</td>
</tr>
<tr>
<td>DKYL1</td>
<td>A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL0.</td>
</tr>
<tr>
<td>DKYL2</td>
<td>A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL1.</td>
</tr>
<tr>
<td>DKYL3</td>
<td>A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL2.</td>
</tr>
<tr>
<td>DKYL4</td>
<td>A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL3.</td>
</tr>
<tr>
<td>DKYL5</td>
<td>A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL4.</td>
</tr>
</tbody>
</table>
Table 8. DES control vector key-subtype and key-usage keywords (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DKYL6</td>
<td>A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL5.</td>
</tr>
<tr>
<td>DKYL7</td>
<td>A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL6.</td>
</tr>
</tbody>
</table>

**Key lengths**

<table>
<thead>
<tr>
<th>MIXED</th>
<th>Indicates that the key can be either a replicated single-length key or a double-length key with two different, random 8-byte values.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SINGLE, KEYLN8</td>
<td>Specifies the key as a single-length key.</td>
</tr>
<tr>
<td>DOUBLE, KEYLN16</td>
<td>Specifies the key as a double-length key.</td>
</tr>
</tbody>
</table>

**Miscellaneous attributes**

<table>
<thead>
<tr>
<th>XPORT-OK</th>
<th>Permits the key to be exported by Key Export or Data Key Export. Also permits the key to be exported by the Key Export to TR31 verb, unless NOT31XPT is set (CV bit 27 = B'1').</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO-XPORT</td>
<td>Prohibits the key from being exported by Key Export or Data Key Export.</td>
</tr>
<tr>
<td>KEY-PART</td>
<td>Specifies the control vector is for a key part.</td>
</tr>
<tr>
<td>ENH-ONLY</td>
<td>Prohibits the key from being wrapped with the legacy method once it has been wrapped with the enhanced method.</td>
</tr>
<tr>
<td>T31XPTOK</td>
<td>Permits the key to be exported by the Key Export to TR31 verb.</td>
</tr>
<tr>
<td>NOT31XPT</td>
<td>Prohibits the key from being exported by the Key Export to TR31 verb.</td>
</tr>
</tbody>
</table>

**Verbs for managing AES and DES key storage files**

CCA provides API functions to allow application programs to manage the AES and DES key storage file, where key tokens are stored when the program references them by key label name.

The following verbs are used to manage the AES and DES key storage files:

- AES Key Record Create (CSNBAKRC)
- AES Key Record Delete (CSNBAKRD)
- AES Key Record List (CSNBAKRL)
- AES Key Record Read (CSNBAKRR)
- AES Key Record Write (CSNBAKRW)
- DES Key Record Create (CSNBKRC)
- DES Key Record Delete (CSNBKRD)
- DES Key Record List (CSNBKRL)
- DES Key Record Read (CSNBKRR)
- DES Key Record Write (CSNBKRW)

**Verbs for managing the PKA key storage file and PKA keys in the cryptographic engine**

The PKA key storage file is a repository for RSA keys, similar to the AES and DES key storage files.
An application can store keys in the key storage file and refer to them by label when using any of the verbs which accept RSA key tokens as input. The following verbs are used to manage the PKA key storage file, or PKA keys stored in the cryptographic engine:

- PKA Key Record Create (CSNDKRC)
- PKA Key Record Delete (CSNDKRD)
- PKA Key Record List (CSNDKRL)
- PKA Key Record Read (CSNDKRR)
- PKA Key Record Write (CSNDKRW)
- Retained Key Delete (CSNDRKD)
- Retained Key List (CSNDRKL)

**EC Diffie-Hellman key agreement models**

You can specify an EC Diffie-Hellman agreement model, or obtain the shared secret value without deriving a key.

**Token agreement scheme**

The caller must have both the required key tokens and both party's identifiers, including a randomly generated nonce.

Combine the exchanged nonce and Party Info into the party identifier. (Both parties must combine this information in the same format.) Then call the EC Diffie-Hellman verb, where “EC” means Elliptic Curve. Specify a skeleton token or the label of a skeleton token as the output key identifier to be used as a container for the computed symmetric key material. Note, both parties must specify the same key type in their skeleton key tokens.

- Specify rule-array keyword **DERIV01** to denote the Static Unified Model key agreement scheme.
- Specify an ECC token as the private key identifier containing this party’s ECC public-private key pair.
- Optionally specify a private KEK key identifier, if the key pair is in an external key token.
- Specify an ECC token as the public key identifier containing the other party's ECC public key part.
- Specify a skeleton token as the output key identifier to be used as a container for the computed symmetric key material.
- Optionally specify an output KEK key identifier, if the output key is to be in an external key token.
- Specify the combined party info (including nonce) as the party identifier.
- Specify the desired size of the key to be derived (in bits) as the key bit length.

**Obtaining the raw "Z" value**

The caller must then derive the final key material using a method of their choice. Do not specify any party info.

- Specify rule array keyword **PASSTHRU** to denote no key agreement scheme.
- Specify an ECC token as the private key identifier containing this party's ECC public-private key pair.
• Optionally specify a private KEK key identifier, if the key pair is in an external key token.
• Specify an ECC token as the public key identifier containing the other party’s ECC public key part.
• The output key identifier will be populated with the resulting shared secret material.

Improved remote key distribution

New methods have been added for securely transferring symmetric encryption keys to remote devices, such as Automated Teller Machines (ATMs), PIN-entry devices, and point of sale terminals.

These methods can also be used to transfer symmetric keys to another cryptographic system of any type, such as a different kind of Hardware Security Module (HSM) in an IBM or non IBM computer server.

This change replaces expensive human operations with network transactions that can be processed quickly and inexpensively. This method makes significant interoperability improvements to related cryptographic key-management functions.

In "Remote key loading," an ATM scenario illustrates the operation of the new methods. Other uses of this method are also possible.

Remote key loading

Remote key loading is the process of installing symmetric encryption keys into a remotely located device from a central administrative site.

This encompasses two phases of key distributions:
• Distribution of initial key encrypting keys (KEKs) to a newly installed device. A KEK is a type of symmetric encryption key that is used to encrypt other keys so that they can be securely transmitted over unprotected paths.
• Distribution of operational keys or replacement KEKs, enciphered under a KEK currently installed in the device.

Access control points are assigned to roles to control keyword usage in the services provided for ATM remote key loading. Table 9 lists the access control points used by the ATM remote key loading function.

<table>
<thead>
<tr>
<th>Verb name</th>
<th>Entry point</th>
<th>Offset</th>
<th>Access Control Point name and comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trusted Block Create</td>
<td>CSNDTBC</td>
<td>X'030F'</td>
<td>Trusted Block Create - Create Block in inactive form</td>
</tr>
<tr>
<td>Trusted Block Create</td>
<td>CSNDTBC</td>
<td>X'0310'</td>
<td>Trusted Block Create - Activate an inactive block</td>
</tr>
<tr>
<td>PKA Key Import</td>
<td>CSNDPKI</td>
<td>X'0311'</td>
<td>PKA Key Import - Import an external trusted block</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Convert a trusted block from external to internal format</td>
</tr>
<tr>
<td>PKA Key Import</td>
<td>CSNDPKI</td>
<td>X'0104'</td>
<td>PKA Key Import</td>
</tr>
<tr>
<td>Remote Key Export</td>
<td>CSNDRKX</td>
<td>X'0312'</td>
<td>Remote Key Export - Gen or export a non-CCA node key</td>
</tr>
</tbody>
</table>
Remote key loading methods

New remote key loading methods have been developed to overcome some of the shortcomings of the old manual key loading methods.

These new methods define acceptable techniques using public key cryptography to load keys remotely. Using these new methods, initial KEKs can be loaded without sending people to the remote device. This will reduce labor costs, be more reliable, and be much less expensive to install and change keys.

The new cryptographic features provide new methods for the creation and use of the special key forms needed for remote key distribution of this type. In addition, the new cryptographic features provide ways to solve long-standing barriers to secure key exchange with non-IBM cryptographic systems.

After an ATM is in operation, new keys can be installed as needed, by sending them enciphered under a KEK installed previously. This is straightforward in concept, but the cryptographic architecture in ATMs is often different from that of the host system that is sending the keys, and it is difficult to export the keys in a form understood by the ATM. For example, cryptographic architectures often enforce key-usage restrictions in which a key is bound to data describing limitations on how it can be used (for encrypting data, for encrypting keys, for operating on Message Authentication Codes (MACs), and so forth). The encoding of these restrictions and the method used to bind them to the key itself differs among cryptographic architectures, and it is often necessary to translate the format to that understood by the target device prior to a key being transmitted. It is difficult to do this without reducing security in the system; typically it is done by making it possible to arbitrarily change key-usage restrictions.

The methods described here provide a mechanism through which the system owner can securely control these translations, preventing the majority of attacks that could be mounted by modifying usage restrictions.

A data structure called a trusted block is defined to facilitate the remote key loading methods. The trusted block is the primary vehicle supporting these new methods. See "Trusted blocks" on page 968.
Verbs supporting Secure Sockets Layer (SSL)

The Secure Sockets Layer (SSL) protocol, developed by Netscape Development Corporation, provides communications privacy over the Internet. Client/server applications can use the SSL protocol to provide secure communications and prevent eavesdropping, tampering, or message forgery.

CCA provides verbs that support the RSA encryption and decryption of PKCS 1.2-formatted symmetric key data to produce symmetric session keys. These session keys can then be used to establish an SSL session between the sender and receiver. The verbs provide SSL support:

- PKA Decrypt (CSNDPKD)
- PKA Encrypt (CSNDPKE)

Managing data integrity and message authentication

To ensure the integrity of transmitted messages and stored data, CCA provides DES-based Message Authentication Code (MAC) functions and several hashing functions, including Modification Detection Code (MDC), SHA-1, RIPEMD-160 and MD5.


The choice of verb depends on the security requirements of the environment in which you are operating. If you need to ensure the authenticity of the sender and also the integrity of the data, consider Message Authentication Code processing. If you need to ensure the integrity of transmitted data in an environment where it is not possible for the sender and the receiver to share a secret cryptographic key, consider hashing functions.

Processing message authentication code

The process of verifying the integrity and authenticity of transmitted messages is called message authentication.

Message authentication code (MAC) processing allows you to verify that a message was not altered or a message was not fraudulently introduced onto the system. You can check that a message you have received is the same one sent by the message originator. The message itself can be in clear or encrypted form. The comparison is performed within the cryptographic coprocessor. Because both the sender and receiver share a secret cryptographic key used in the MAC calculation, the MAC comparison also ensures the authenticity of the message.

In a similar manner, MACs can be used to ensure the integrity of data stored on the system or on removable media, such as tape.

CCA key typing makes it possible to give one party a key that can only be used to generate a MAC, and to give another party a corresponding key that can only be used to verify the MAC. This ensures that the second party cannot impersonate the first by generating MACs with their version of the key.

The coprocessor provides support for both single-length and double-length MAC generation and MAC verification keys. With the ANSI X9.9-1 single key algorithm, use the single-length MAC and MACVER keys.
CCA provides support for the use of data-encrypting keys in the MAC Generate and MAC Verify verbs, and also the use of a MAC generation key in the MAC Verify verb. This support permits CCA MAC verbs to interface more smoothly with non-CCA key distribution system.


These verbs are used to process MACs:
- MAC Generate (CSNBMGN)
- MAC Verify (CSNBMVR)

**Hashing functions**
Hashing functions are provided by two verbs.

These verbs are:
- MDC Generate (CSNBMDG)
- One-Way Hash (CSNBOWH)

**Processing personal identification numbers**

The process of validating personal identities in a financial transaction system is called personal authentication.

The personal identification number (PIN) is the basis for verifying the identity of a customer across the financial industry networks. The financial industry needs functions to generate, translate, and verify PINs. These functions prevent unauthorized disclosures when organizations handle personal identification numbers.

The coprocessor supports the following algorithms for generating and verifying personal identification numbers:
- IBM 3624
- IBM 3624 PIN offset
- IBM German Bank Pool
- IBM German Bank Pool PIN Offset (GBP-PINO)
- VISA PIN validation value
- Interbank

You can translate PIN blocks from one format to another without the PIN being exposed in cleartext form. The coprocessor supports the following formats:
- ANSI X9.8
- ISO formats 0, 1, 2, 3
- VISA formats 1, 2, 3, 4
- IBM 4704 Encrypting PINPAD format
- IBM 3624 formats
- IBM 3621 formats
- ECI formats 1, 2, 3
With the capability to translate personal identification numbers into different PIN block formats, you can use personal identification numbers on different systems.

Secure messaging

The following verbs assist applications in encrypting secret information such as clear keys and PIN blocks in a secure message.

These verbs execute within the secure boundary of the cryptographic coprocessor:
- Secure Messaging for Keys (CSNBSKY)
- Secure Messaging for PINs (CSNBSNP)

Trusted Key Entry support

The Trusted Key Entry workstation (TKE) provides a secure method of initializing and administering cryptographic coprocessors.

It is an optional IBM Z feature, but it is mandatory if z/OS and CCA are not available on your system. Initialization of the coprocessor can be done through CCA for both the z/OS and Linux environments, either with or without TKE.

TKE Version 6.0 or higher is required in order to administer the CEX*C coprocessor features earlier than CEX5C. You can use the TKE workstation to load DES master keys, PKA master keys, and operational keys in a secure way. TKE Version 6.0 and 7.0 can also set AES master keys on a CEX*C coprocessor earlier than CEX5C. For CEX5C, TKE 8.0 is required. For CEX6C, TKE 9.0 is required, especially for exploiting PCI-HSM 2016 compliance mode.

You can load keys remotely and for multiple coprocessors, which can be in a single machine or in multiple machines. The TKE workstation eases the administration for using one coprocessor as a production machine and as a test machine at the same time, while maintaining security and reliability.

The TKE workstation can be used for enabling and disabling access control points for verbs executed on the cryptographic coprocessor. See Chapter 24, “Access control points and verbs,” on page 1047 for additional information.

For complete details about the TKE workstation, see z/OS Cryptographic Services ICSF Trusted Key Entry Workstation User’s Guide.

Typical sequences of CCA verbs

View some sample sequences in which CCA verbs might be called.

Combination A (DATA keys only)
1. Random number generate
2. Clear key import or multiple clear key import
3. Encipher/decipher
4. Data key export or key export (optional step)

Combination B
1. Random number generate
2. Secure key import or multiple secure key import
3. Any service
4. Data key export for DATA keys, or key export in the general case (optional step)

**Combination C**
1. Key generate (OP form only)
2. Any service
3. Key export (optional)

**Combination D**
1. Key generate (OPEX form)
2. Any service

**Combination E**
1. Key Generate (IM form only)
2. Key Import
3. Any service
4. Key Export (optional)

**Combination F**
1. Key Generate (IMEX form)
2. Key Import
3. Any service

**Combination G**
1. Key Generate
2. AES or DES Key Record Create
3. AES or DES Key Record Write
4. Any service (passing label of the key just generated)

**Combination H**
1. Key Import
2. AES or DES Key Record Create
3. AES or DES Key Record Write
4. Any service (passing label of the key just generated)

**Notes**
1. An example of “any service” is CSNBENC.
2. These combinations exclude verbs that can be used on their own; for example, Key Export or encode, or using the Key Generate verb to generate an exportable key.
3. These combinations do not show key communication, or the transmission of any output from an CCA verb.

The key forms are described in Chapter 20, “Key forms and types used in the Key Generate verb,” on page 985 and “Key Generate (CSNBKGN)” on page 211.

---

**Using the CCA node and master key management verbs**

Use these verbs for the CCA node and master key management functions.
- Cryptographic Facility Query (CSUACFQ)
- Cryptographic Facility Version (CSUACFV)
- Cryptographic Resource Allocate (CSUACRA)
Summary of the CCA nodes and resource control verbs

A table of the CCA nodes and resource control verbs, including references to the verb descriptions.

Table 10. Summary of CCA nodes and resource control verbs

<table>
<thead>
<tr>
<th>Entry point</th>
<th>Verb name</th>
<th>Description</th>
<th>Topic/Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSUAACM</td>
<td>Access Control Maintenance</td>
<td>Queries or controls installed roles.</td>
<td>“Access Control Maintenance (CSUAACM)” on page 89</td>
</tr>
<tr>
<td>CSUAACT</td>
<td>Access Control Tracking</td>
<td>Tracks the usage of ACPs for a defined span of time, and also queries that tracking information.</td>
<td>“Access Control Tracking (CSUAACT)” on page 93</td>
</tr>
<tr>
<td>CSUACFQ</td>
<td>Cryptographic Facility Query</td>
<td>Retrieves information about the coprocessor and the CCA application program in that coprocessor.</td>
<td>“Cryptographic Facility Query (CSUACFQ)” on page 100</td>
</tr>
</tbody>
</table>

Chapter 7, “Using the CCA nodes and resource control verbs,” on page 89

Chapter 3. AES, DES, and HMAC cryptography and verbs 59
<table>
<thead>
<tr>
<th>Entry point</th>
<th>Verb name</th>
<th>Description</th>
<th>Topic/Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSUACFV</td>
<td>Cryptographic Facility Version</td>
<td>Retrieve the Security Application Program Interface (SAPI) version and build date.</td>
<td>“Cryptographic Facility Version (CSUACFV)” on page 143</td>
</tr>
<tr>
<td>CSUACRA</td>
<td>Cryptographic Resource Allocate</td>
<td>Allocates specific CCA coprocessor for use by the thread or process, depending on the scope of the verb.</td>
<td>“Cryptographic Resource Allocate (CSUACRA)” on page 144</td>
</tr>
<tr>
<td>CSUACRD</td>
<td>Cryptographic Resource Deallocate</td>
<td>De-allocates a specific CCA coprocessor that is allocated by the thread or process, depending on the scope of the verb.</td>
<td>“Cryptographic Resource Deallocate (CSUACRD)” on page 147</td>
</tr>
</tbody>
</table>
| CSNBKSI     | Key Storage Initialization | This verb initializes a key-storage file using the current symmetric or asymmetric master-key. The initialized key storage does not contain any preexisting key records.  
The name and path of the key storage data and index file are established differently in each operating environment. Note that HMAC keys are not supported for key storage. | “Key Storage Initialization (CSNBKSI)” on page 149 |
| CSUALGQ     | Log Query           | This verb retrieves system log (SYSLOG) message data and CCA log message data from the coprocessor. SYSLOG data is available for one of the five latest boot cycles (current boot cycle and up to four previous boot cycles). CCA log message data is optionally available during the current boot cycle. The verb supports service personnel and developers in testing and debugging issues. | “Log Query (CSUALGQ)” on page 152 |
| CSNBMPK     | Master Key Process  | Operates on the three master-key registers: new, current, and old.                                                                                                                                    | “Master Key Process (CSNBMPK)” on page 157          |
| CSUARNT     | Random Number Tests | Invokes the USA NIST FIPS PUB 140-1 specified cryptographic operational tests. These tests, selected by a rule_array keyword, consist of known-answer tests of DES, RSA, and SHA-1 processes and, for random numbers, monobit test, poker test, runs test, and log-run test. | “Random Number Tests (CSUARNT)” on page 161          |
Summary of the AES, DES, and HMAC verbs

Hash Message Authentication Code (HMAC) support was added in CCA Release 4.1.0. All of the HMAC verbs and features listed in this summary require CCA 4.1.0 or CCA 4.2.0 in order to run.

Table 11 lists the AES, DES, and HMAC verbs described in this document. The table also references the chapter that describes the verb.

<table>
<thead>
<tr>
<th>Entry point</th>
<th>Verb name</th>
<th>Description</th>
<th>Topic/Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 8, “Managing AES, DES, and HMAC cryptographic keys,” on page 165</td>
<td><strong>Clear Key Import</strong> (CSNBCKI)</td>
<td>Imports an 8-byte clear DATA key, enciphers it under the master key, and places the result into an internal key token. This verb converts the clear key into operational form as a DATA key.</td>
<td>“Clear Key Import (CSNBCKI)” on page 166</td>
</tr>
<tr>
<td>CSNBCKM</td>
<td><strong>Multiple Clear Key Import</strong></td>
<td>Imports a single-length, double-length, or triple-length clear DATA key that is used to encipher or decipher data. It accepts a clear key and enciphers the key under the host master key, returning an encrypted DATA key in operational form in an internal key token.</td>
<td>“Multiple Clear Key Import (CSNBCKM)” on page 167</td>
</tr>
<tr>
<td>CSNBCVG</td>
<td><strong>Control Vector Generate</strong></td>
<td>Builds a control vector from keywords specified by the key_type and rule_array parameters.</td>
<td>“Control Vector Generate (CSNBCVG)” on page 170</td>
</tr>
<tr>
<td>CSNBDKM</td>
<td><strong>Data Key Import</strong></td>
<td>Imports an encrypted source DES single-length or double-length DATA key and creates or updates a target internal key token with the master key enciphered source key.</td>
<td>“Data Key Import (CSNBDKM)” on page 181</td>
</tr>
<tr>
<td>CSNBDKX</td>
<td><strong>Data Key Export</strong></td>
<td>Re-enciphers a DATA key from encryption under the master key to encryption under an exporter key-encrypting key, making it suitable for export to another system.</td>
<td>“Data Key Export (CSNBDKX)” on page 180</td>
</tr>
<tr>
<td>CSNBDKG</td>
<td><strong>Diversified Key Generate</strong></td>
<td>Generates a key based upon the key-generating key, the processing method, and the parameter data that is supplied. The control vector of the key-generating key also determines the type of target key that can be generated.</td>
<td>“Diversified Key Generate (CSNBDKG)” on page 183</td>
</tr>
<tr>
<td>CSNBDKG2</td>
<td><strong>Diversified Key Generate2</strong></td>
<td>Generates an AES key based on a function of a key-generating key, the process rule, and data that you supply.</td>
<td>“Diversified Key Generate2 (CSNBDKG2)” on page 189</td>
</tr>
<tr>
<td>Entry point</td>
<td>Verb name</td>
<td>Description</td>
<td>Topic/Page</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>CSNDEDH</td>
<td>EC Diffie-Hellman</td>
<td>Creates symmetric key material from a pair of Elliptic Curve Cryptography (ECC) keys using the Elliptic Curve Diffie-Hellman (ECDH) protocol.</td>
<td>“EC Diffie-Hellman (CSNDEDH)” on page 195</td>
</tr>
<tr>
<td>CSNBKEX</td>
<td>Key Export</td>
<td>Re-enciphers a key from encryption under a master key variant to encryption under the same variant of an exporter key-encrypting key, making it suitable for export to another system.</td>
<td>“Key Export (CSNBKEX)” on page 209</td>
</tr>
<tr>
<td>CSNBKGN</td>
<td>Key Generate</td>
<td>Generates a 64-bit, 128-bit, 192-bit, or 256-bit odd parity key, or a pair of keys; and returns them in encrypted forms (operational, exportable, or importable). Key Generate does not produce keys in plaintext.</td>
<td>“Key Generate (CSNBKGN)” on page 211</td>
</tr>
<tr>
<td>CSNBKGN2</td>
<td>Key Generate2</td>
<td>Generates either one or two HMAC keys. This verb does not produce keys in clear form and all keys are returned in encrypted form. When two keys are generated, each key has the same clear value, although this clear value is not exposed outside the secure cryptographic feature. This verb returns variable-length CCA key tokens and uses the AESKW wrapping method. Operational keys will be encrypted under the AES master key.</td>
<td>“Key Generate2 (CSNBKGN2)” on page 221</td>
</tr>
<tr>
<td>CSNBKIM</td>
<td>Key Import</td>
<td>Re-enciphers a key from encryption under an importer key-encrypting key to encryption under the master key. The re-enciphered key is in the operational form.</td>
<td>“Key Import (CSNBKIM)” on page 234</td>
</tr>
<tr>
<td>CSNBKPI</td>
<td>Key Part Import</td>
<td>Combines the clear key parts of any key type and returns the combined key value in an internal key token or an update to the CCA key storage file.</td>
<td>“Key Part Import (CSNBKPI)” on page 237</td>
</tr>
<tr>
<td>CSNBKPI2</td>
<td>Key Part Import2</td>
<td>Combines the clear key parts of any HMAC key type from an internal variable-length symmetric key-token, and returns the combined key value in an internal variable-length symmetric key-token or an update to the CCA key storage file.</td>
<td>“Key Part Import2 (CSNBKPI2)” on page 241</td>
</tr>
<tr>
<td>CSNBKYT</td>
<td>Key Test</td>
<td>Generates or verifies (depending on keywords in the rule_array) a secure verification pattern for keys. This verb requires the tested key to be in the clear or encrypted under the master key.</td>
<td>“Key Test (CSNBKYT)” on page 245</td>
</tr>
<tr>
<td>Entry point</td>
<td>Verb name</td>
<td>Description</td>
<td>Topic/Page</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>CSNBKYT2</td>
<td>Key Test2</td>
<td>Generates or verifies (depending on keywords in the <code>rule_array</code>) a secure cryptographic verification pattern for keys contained in a variable-length symmetric key-token. The key to test can be in the clear or encrypted under a master key. Requires the tested key to be in the clear or encrypted under the master key.</td>
<td>&quot;Key Test2 (CSNBKYT2)&quot; on page 249</td>
</tr>
</tbody>
</table>
| CSNBKYTX     | Key Test Extended | This verb is essentially the same as Key Test, except for the following:  
  • In addition to operating on internal keys and key parts, this verb also operates on external keys and key parts.  
  • This verb does not operate on clear keys, and does not accept `rule_array` keywords CLR-A128, CLR-A192, CLR-A256, KEY-CLR, and KEY-CLRD. | "Key Test Extended (CSNBKYTX)" on page 254 |
| CSNBKTB      | Key Token Build | Builds an internal or external token from the supplied parameters. You can use this verb to build CCA key tokens for all key types that CCA supports. The resulting token can be used as input to the Key Generate, and Key Part Import verbs. | "Key Token Build (CSNBKTB)" on page 259  |
| CSNBKTB2     | Key Token Build2 | Builds variable-length internal or external key tokens for all key types that the coprocessor supports. The key token is built based on parameters that you supply. The resulting token can be used as input to the Key Generate2, and Key Part Import2 verbs. A clear key token built by this verb can be used as input to the Key Test2 verb.  
  This verb supports internal HMAC tokens, both as clear key tokens and as skeleton tokens containing no key. | "Key Token Build2 (CSNBKTB2)" on page 264 |
| CSNBKTC      | Key Token Change | Re-enciphers a DES key from encryption under the old master key to encryption under the current master key, and to update the keys in internal DES key-tokens. | "Key Token Change (CSNBKTC)" on page 301 |
| CSNBKTC2     | Key Token Change2 | Re-enciphers a variable-length HMAC key from encryption under the old master key to encryption under the current master key. This verb also updates the keys in internal HMAC key-tokens. | "Key Token Change2 (CSNBKTC2)" on page 304 |
Table 11. Summary of CCA AES, DES, and HMAC verbs (continued)

<table>
<thead>
<tr>
<th>Entry point</th>
<th>Verb name</th>
<th>Description</th>
<th>Topic/Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNBKTP</td>
<td>Key Token Parse</td>
<td>Disassembles a key token into separate pieces of information. This verb can disassemble an external key-token or an internal key-token in application storage.</td>
<td>“Key Token Parse (CSNBKTP)” on page 307</td>
</tr>
<tr>
<td>CSNBKTP2</td>
<td>Key Token Parse2</td>
<td>Disassembles a variable-length symmetric key-token into separate pieces of information. The verb can disassemble an external or internal variable-length symmetric key-token in application storage. The verb returns some of the key-token information in a set of variables identified by individual parameters, and returns the remaining information as keywords in the rule array.</td>
<td>“Key Token Parse2 (CSNBKTP2)” on page 311</td>
</tr>
<tr>
<td>CSNBKTR</td>
<td>Key Translate</td>
<td>Uses one key-encrypting key to decipher an input key and then enciphers this key using another key-encrypting key within the secure environment.</td>
<td>“Key Translate (CSNBKTR)” on page 321</td>
</tr>
<tr>
<td>CSNBKTR2</td>
<td>Key Translate2</td>
<td>Uses one key-encrypting key to decipher an input key and then enciphers this key using another key-encrypting key within the secure environment. This verb differs from the Key Translate verb in that Key Translate2 can process both fixed-length and variable-length symmetric key tokens.</td>
<td>“Key Translate2 (CSNBKTR2)” on page 323</td>
</tr>
<tr>
<td>CSNDPKD</td>
<td>PKA Decrypt</td>
<td>Uses an RSA private key to decrypt the RSA-encrypted key value and return the clear key value to the application.</td>
<td>“PKA Decrypt (CSNDPKD)” on page 327</td>
</tr>
<tr>
<td>CSNDPKE</td>
<td>PKA Encrypt</td>
<td>Encrypts a supplied clear key value under an RSA public key. The supplied key can be formatted using the PKCS 1.2 or ZERO-PAD methods prior to encryption.</td>
<td>“PKA Encrypt (CSNDPKE)” on page 330</td>
</tr>
<tr>
<td>CSNBPEX</td>
<td>Prohibit Export</td>
<td>Modifies the control vector of a CCA key token so that the key cannot be exported. This verb operates only on internal key tokens.</td>
<td>“Prohibit Export (CSNBPEX)” on page 334</td>
</tr>
<tr>
<td>CSNBPEXX</td>
<td>Prohibit Export Extended</td>
<td>Modifies an external DES key-token so that the key can no longer be exported after it has been imported. This verb operates only on internal key tokens.</td>
<td>“Prohibit Export Extended (CSNBPEXX)” on page 335</td>
</tr>
<tr>
<td>CSNBRKA</td>
<td>Restrict Key Attribute</td>
<td>Modifies an operational variable-length key so that it cannot be exported.</td>
<td>“Restrict Key Attribute (CSNBRKA)” on page 337</td>
</tr>
<tr>
<td>Entry point</td>
<td>Verb name</td>
<td>Description</td>
<td>Topic/Page</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>CSNBRNG</td>
<td>Random Number Generate</td>
<td>Generates an 8-byte cryptographic-quality random number suitable for use as an encryption key or for other purposes. The output can be specified in three forms of parity: RANDOM, ODD, and EVEN.</td>
<td>“Random Number Generate (CSNBRNG)” on page 341</td>
</tr>
<tr>
<td>CSNBRNGL</td>
<td>Random Number Generate Long</td>
<td>Generates a cryptographic-quality random number suitable for use as an encryption key or for other purposes, ranging from 1 - 8192 bytes in length. The output can be specified in three forms of parity: RANDOM, ODD, and EVEN.</td>
<td>“Random Number Generate Long (CSNBRNGL)” on page 342</td>
</tr>
<tr>
<td>CSNDSYX</td>
<td>Symmetric Key Export</td>
<td>Transfer an application-supplied symmetric key (a DATA key) from encryption under the AES, DES or HMAC master key to encryption under an application-supplied RSA public key. The application-supplied DATA key must be an AES, DES or HMAC internal key token, or the label of an AES or DES key token in the CCA key storage file. The Symmetric Key Import and Symmetric Key Import2 verb can import the PKA-encrypted key form at the receiving node. Support for HMAC key was added beginning with CCA 4.1.0.</td>
<td>“Symmetric Key Export (CSNDSYX)” on page 344</td>
</tr>
<tr>
<td>CSNDSXD</td>
<td>Symmetric Key Export with Data</td>
<td>Export a symmetric key, along with some application supplied data, encrypted using an RSA key.</td>
<td>“Symmetric Key Export with Data (CSNDSXD)” on page 350</td>
</tr>
<tr>
<td>CSNDSYG</td>
<td>Symmetric Key Generate</td>
<td>Generate a symmetric key (a DATA key) and return the key in two forms: DES-encrypted and encrypted under an RSA public key. The DES-encrypted key can be an internal token encrypted under a host DES master key, or an external form encrypted under a KEK. (You can use the Symmetric Key Import verb to import the PKA-encrypted form.)</td>
<td>“Symmetric Key Generate (CSNDSYG)” on page 353</td>
</tr>
<tr>
<td>CSNDSYI</td>
<td>Symmetric Key Import</td>
<td>Import a symmetric AES or DES DATA key enciphered under an RSA public key into operational form enciphered under a DES master key.</td>
<td>“Symmetric Key Import (CSNDSYI)” on page 358</td>
</tr>
<tr>
<td>Entry point</td>
<td>Verb name</td>
<td>Description</td>
<td>Topic/Page</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CSNDSYI2</td>
<td>Symmetric Key Import2</td>
<td>Use this verb to import an HMAC key that has been previously formatted and enciphered under an RSA public key by the Symmetric Key Export verb. The formatted and RSA-enciphered key is contained in an external variable-length symmetric key-token. The key is deciphered using the associated RSA private-key. The recovered HMAC key is re-enciphered under the AES master-key. The re-enciphered key is then returned in an internal variable-length symmetric key-token. The key algorithm for this verb is HMAC.</td>
<td>“Symmetric Key Import2 (CSNDSYI2)” on page 362</td>
</tr>
<tr>
<td>CSNBUKD</td>
<td>Unique Key Derive</td>
<td>Performs the key derivation process as defined in ANSI X9.24 Part 1. The process derives keys from two values: the base derivation key and the derivation data. Rule array keywords determine the types and number of keys derived on a particular call.</td>
<td>Chapter 9, “Protecting data,” on page 379</td>
</tr>
<tr>
<td>CSNBDEC</td>
<td>Decipher</td>
<td>Deciphers data using cipher block chaining mode of DES. The result is called plaintext.</td>
<td>“Decipher (CSNBDEC)” on page 381</td>
</tr>
<tr>
<td>CSNBENC</td>
<td>Encipher</td>
<td>Enciphers data using the cipher block chaining mode of DES. The result is called ciphertext.</td>
<td>“Encipher (CSNBENC)” on page 385</td>
</tr>
<tr>
<td>CSNBSAD</td>
<td>Symmetric Algorithm</td>
<td>Deciphers data using the AES cipher block chaining mode.</td>
<td>“Symmetric Algorithm Decipher (CSNBSAD)” on page 390</td>
</tr>
<tr>
<td></td>
<td>Decipher</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNBSAE</td>
<td>Symmetric Algorithm</td>
<td>Enciphers data using the AES cipher block chaining mode</td>
<td>“Symmetric Algorithm Encipher (CSNBSAE)” on page 397</td>
</tr>
<tr>
<td></td>
<td>Encipher</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNBCTT2</td>
<td>Cipher Text Translate2</td>
<td>Deciphers encrypted data (ciphertext) under one ciphertext translation key and re-enciphers it under another ciphertext translation key without having the data appear in the clear outside the cryptographic coprocessor.</td>
<td>“Cipher Text Translate2 (CSNBCTT2)” on page 404</td>
</tr>
<tr>
<td>CSNBHMG</td>
<td>HMAC Generate</td>
<td>Generates a keyed hash message authentication code (HMAC) for the text string provided as input. See Chapter 10, “Verifying data integrity and authenticating messages,” on page 417.</td>
<td>“HMAC Generate (CSNBHMG)” on page 419</td>
</tr>
</tbody>
</table>

Chapter 9, “Protecting data,” on page 379

Chapter 10, “Verifying data integrity and authenticating messages,” on page 417
<table>
<thead>
<tr>
<th>Entry point</th>
<th>Verb name</th>
<th>Description</th>
<th>Topic/Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNBHMOV</td>
<td>HMAC Verify</td>
<td>Verifies a keyed hash message authentication code (HMAC) for the text string provided as input. See Chapter 10, “Verifying data integrity and authenticating messages,” on page 417.</td>
<td>“HMAC Verify (CSNBHMOV)” on page 423</td>
</tr>
<tr>
<td>CSNBMGN</td>
<td>MAC Generate</td>
<td>Generates a 4, 6, or 8-byte Message Authentication Code (MAC) for a text string that the application program supplies. The MAC is computed using either the ANSI X9.9-1 algorithm or the ANSI X9.19 optional double key algorithm and padding could be applied according to the EMV specification.</td>
<td>“MAC Generate (CSNBMGN)” on page 426</td>
</tr>
<tr>
<td>CSNBMGN2</td>
<td>MAC Generate2</td>
<td>Generates a keyed hash message authentication code (HMAC) or a ciphered message authentication code (CMAC) for the message string provided as input. A MAC key with key usage that can be used for generate is required to calculate the MAC.</td>
<td>“MAC Generate2 (CSNBMGN2)” on page 430</td>
</tr>
<tr>
<td>CSNBVRR</td>
<td>MAC Verify</td>
<td>Verifies a 4, 6, or 8-byte Message Authentication Code (MAC) for a text string that the application program supplies. The MAC is computed using either the ANSI X9.9-1 algorithm or the ANSI X9.19 optional double key algorithm and padding could be applied according to the EMV specification. The computed MAC is compared with a user-supplied MAC.</td>
<td>“MAC Verify (CSNBVRR)” on page 433</td>
</tr>
<tr>
<td>CSNBVRR2</td>
<td>MAC Verify2</td>
<td>Verifies a keyed hash message authentication code (HMAC) or a ciphered message authentication code (CMAC) for the message text provided as input. A MAC key with key usage that can be used for verify is required to verify the MAC.</td>
<td>“MAC Verify2 (CSNBVRR2)” on page 438</td>
</tr>
<tr>
<td>CSNBMDG</td>
<td>MDC Generate</td>
<td>Creates a 128-bit hash value (Modification Detection Code) on a data string whose integrity you intend to confirm.</td>
<td>“MDC Generate (CSNBMDG)” on page 441</td>
</tr>
<tr>
<td>CSNBOWH</td>
<td>One-Way Hash</td>
<td>Generates a one-way hash on specified text.</td>
<td>“One-Way Hash (CSNBOWH)” on page 445</td>
</tr>
<tr>
<td><strong>Chapter 12, “Financial services,” on page 491</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNBAPG</td>
<td>Authentication Parameter Generate</td>
<td>Generates an authentication parameter (AP) and returns it encrypted using the key supplied in an input parameter.</td>
<td>“Authentication Parameter Generate (CSNBAPG)” on page 508</td>
</tr>
<tr>
<td>Entry point</td>
<td>Verb name</td>
<td>Description</td>
<td>Topic/Page</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>CSNBCPE</td>
<td>Clear PIN Encrypt</td>
<td>Formats a PIN into a PIN block format and encrypts the results. You can also use this verb to create an encrypted PIN block for transmission. With the RANDOM keyword, you can have the verb generate random PIN numbers.</td>
<td>“Clear PIN Encrypt (CSNBCPE)” on page 511</td>
</tr>
</tbody>
</table>
| CSNBPGN     | Clear PIN Generate               | Generates a clear personal identification number (PIN), a PIN verification value (PVV), or an offset using one of the following algorithms:  
- IBM 3624 (IBM-PIN or IBM-PINO)  
- IBM German Bank Pool (GBP-PIN or GBP-PINO)  
- VISA PIN validation value (VISA-PVV)  
- Interbank PIN (INBK-PIN)                                                                                           | “Clear PIN Generate (CSNBPGN)” on page 515 |
| CSNBCPA     | Clear PIN Generate Alternate     | Generates a clear VISA PIN validation value (PVV) from an input encrypted PIN block. The PIN block might have been encrypted under either an input or output PIN encrypting key. The IBM-PINO algorithm is supported to produce a 3624 offset from a customer selected encrypted PIN. The PIN block must be encrypted under either an input PIN-encrypting key (IPINENC) or output PIN-encrypting key (OPINENC). | “Clear PIN Generate Alternate (CSNBCPA)” on page 518 |
| CSNBCSG     | CVV Generate                     | Generates a VISA Card Verification Value (CVV) or a MasterCard Card Verification Code (CVC) as defined for track 2.                                                                                                                                                                                                                           | “CVV Generate (CSNBCSG)” on page 523 |
| CSNBCKC     | CVV Key Combine                  | Combine two single-length operational DES keys that are suitable for use with the CVV (card-verification value) algorithm into one operational TDES key.                                                                                                                                                                                        | “CVV Key Combine (CSNBCKC)” on page 526 |
| CSNBCSV     | CVV Verify                       | Verifies a VISA Card Verification Value (CVV) or a MasterCard Card Verification Code (CVC) as defined for track 2.                                                                                                                                                                                                                           | “CVV Verify (CSNBCSV)” on page 531 |
| CSNBEPG     | Encrypted PIN Generate           | Generates and formats a PIN and encrypts the PIN block.                                                                                                                                                                                                                                                                                     | “Encrypted PIN Generate (CSNBEPG)” on page 535 |
Table 11. Summary of CCA AES, DES, and HMAC verbs (continued)

<table>
<thead>
<tr>
<th>Entry point</th>
<th>Verb name</th>
<th>Description</th>
<th>Topic/Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNBPTR</td>
<td>Encrypted PIN Translate</td>
<td>Re-enciphers a PIN block from one PIN-encrypting key to another and, optionally, changes the PIN block format. UKPT keywords are supported. You must identify the input PIN-encrypting key that originally enciphers the PIN. You also need to specify the output PIN-encrypting key that you want the verb to use to encipher the PIN. If you want to change the PIN block format, specify a different output PIN block format from the input PIN block format.</td>
<td>“Encrypted PIN Translate (CSNBPTR)” on page 539</td>
</tr>
<tr>
<td>CSNBPTRE</td>
<td>Encrypted PIN Translate Enhanced</td>
<td>Reformats a PIN into a different PIN-block format using an enciphered PAN field. You can use this verb in an interchange-network application, or to change the PIN block to conform to the format and encryption key used in a PIN-verification database.</td>
<td>“Encrypted PIN Translate Enhanced (CSNBPTRE)” on page 545</td>
</tr>
<tr>
<td>CSNBPVR</td>
<td>Encrypted PIN Verify</td>
<td>Verifies a supplied PIN using one of the following algorithms: • IBM 3624 (IBM-PIN or IBM-PINO) • IBM German Bank Pool (GBP-PIN or GBP-PINO) • VISA PIN validation value (VISA-PVV) • Interbank PIN (INBK-PIN) UKPT keywords are supported.</td>
<td>“Encrypted PIN Verify (CSNBPVR)” on page 555</td>
</tr>
<tr>
<td>CSNBFPED</td>
<td>FPE Decipher</td>
<td>Decrypts payment card data for the Visa Data Secure Platform (VDSP) processing.</td>
<td>“FPE Decipher (CSNBFPED)” on page 560</td>
</tr>
<tr>
<td>CSNBFPEE</td>
<td>FPE Encipher</td>
<td>Encrypts payment card data for the Visa Data Secure Platform (VDSP) processing.</td>
<td>“FPE Encipher (CSNBFPEE)” on page 568</td>
</tr>
<tr>
<td>CSNBFPET</td>
<td>FPE Translate</td>
<td>Translates payment data from encryption under one key to encryption under another key with a possibly different format.</td>
<td>“FPE Translate (CSNBFPET)” on page 575</td>
</tr>
<tr>
<td>CSNBPCU</td>
<td>PIN Change/Unblock</td>
<td>Supports the PIN change algorithms specified in the VISA Integrated Circuit Card Specification; available only on an IBM z890 or IBM z990 with May 2004 or later version of Licensed Internal Code (LIC).</td>
<td>“PIN Change/Unblock (CSNBPCU)” on page 583</td>
</tr>
<tr>
<td>CSNBPFO</td>
<td>Recover PIN from Offset</td>
<td>Calculates the encrypted customer-entered PIN from a PIN generating key, account information, and an IBM-PINO Offset.</td>
<td>“Recover PIN from Offset (CSNBPFO)” on page 591</td>
</tr>
<tr>
<td>CSNSKY</td>
<td>Secure Messaging for Keys</td>
<td>Encrypts a text block, including a clear key value decrypted from an internal or external DES token.</td>
<td>“Secure Messaging for Keys (CSNSKY)” on page 595</td>
</tr>
</tbody>
</table>
Table 11. Summary of CCA AES, DES, and HMAC verbs (continued)

<table>
<thead>
<tr>
<th>Entry point</th>
<th>Verb name</th>
<th>Description</th>
<th>Topic/Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNBSPN</td>
<td>Secure Messaging for PINs</td>
<td>Encrypts a text block, including a clear PIN block recovered from an encrypted PIN block.</td>
<td>“Secure Messaging for PINs (CSNBSPN)” on page 599</td>
</tr>
<tr>
<td>CSNBTRV</td>
<td>Transaction Validation</td>
<td>Supports the generation and validation of American Express card security codes; available only on an IBM z890 or IBM z990 with May 2004 or later version of Licensed Internal Code (LIC).</td>
<td>“Transaction Validation (CSNBTRV)” on page 603</td>
</tr>
</tbody>
</table>
Chapter 4. Introducing PKA cryptography and using PKA verbs

Read the provided introduction to Public Key Algorithms (PKA) and Elliptic Curve Cryptography (ECC). When you use the CCA PKA verbs, take note of these programming considerations, such as the PKA key token structure and key management.

You can use PKA support to exchange symmetric (secret) keys securely, and to compute digital signatures for authenticating messages to users.

The preceding chapters focused on AES or DES cryptography or secret-key cryptography. This cryptography is symmetric (senders and receivers use the same key, which must be exchanged securely in advance, to encipher and decipher data).

Public key cryptography does not require exchanging a secret key. It is asymmetric (the sender and receiver each have a pair of keys, a public key and a different but corresponding private key).

PKA key algorithms

Public key cryptography uses a key pair consisting of a public key and a private key.

The PKA public key uses one of the following algorithms:

Rivest-Shamir-Adleman (RSA)

The RSA algorithm is the most widely used and accepted of the public key algorithms. It uses three quantities to encrypt and decrypt text: a public exponent (PU), a private exponent (PR), and a modulus (M). Given these three and some cleartext data, the algorithm generates ciphertext as follows:

\[ \text{ciphertext} = \text{cleartext}^{PU} \pmod{M} \]

Similarly, the following operation recovers cleartext from ciphertext:

\[ \text{cleartext} = \text{ciphertext}^{PR} \pmod{M} \]

Elliptic Curve Digital Signature Algorithm (ECDSA)

The ECDSA algorithm uses elliptic curve cryptography (an encryption system based on the properties of elliptic curves) to provide a variant of the Digital Signature Algorithm.

PKA master keys

On the cryptographic coprocessor, PKA keys are protected by the Asymmetric-Keys Master Key (ASYM-MK).

The ASYM-MK is a triple-length DES key used to protect PKA private keys. On the cryptographic coprocessor, the ASYM-MK protects RSA private keys.

There are two PKA master keys: the ASYM-MK mentioned above, and the 256-bit AES PKA Master Key (APKA-MK), used to protect ECC private keys stored in ECC key tokens.
In order for PKA verbs to function on the processor, the hash pattern of the ASYM-MK must match the hash pattern of the SYM-MK on the cryptographic coprocessor. The administrator installs the PKA master keys and the ASYM-MK on the cryptographic coprocessor by using either the pass phrase initialization routine, the Clear Master Key Entry panels, or the optional Trusted Key Entry workstation (TKE).

**Operational private keys**

Operational private keys are protected under two layers of DES encryption.

They are encrypted under an Object Protection Key (OPK) that in turn is encrypted under the ASYM-MK. You dynamically generate the OPK for each private key at import time or when the private key is generated on a CEX*C. CCA provides a public key storage file for the storage of application PKA keys. Although you cannot change PKA master keys dynamically, the PKA Key Token Change verb can be run to change a private PKA token (RSA or ECC) from encryption under the old ASYM-MK (or APKA-MK) to encryption under the current ASYM-MK (or APKA-MK).

**PKA verbs**

The CEX*C features provide application programming interfaces to PKA functions.

These PKA functions are:
- RSA digital signature functions
- Key management and key generation functions
- DES key distribution functions
- Data encryption functions
- ECC digital signature functions
- ECC key management and key generation functions
- ECC-based and RSA-based services for:
  - DES and AES key derivation
  - Diffie-Hellman key agreement for DES and AES
  - Key distribution functions for DES and AES keys

Starting with the CEX3C feature, the CEX*C adapters add application programming interfaces to the following PKA functions:
- ECC digital signature functions
- ECC key management and key generation functions
- ECC-based and RSA-based services for:
  - DES and AES key derivation
  - Diffie-Hellman key agreement for DES and AES
  - Key distribution functions for DES and AES keys

**Verbs supporting digital signatures**

CCA provides verbs that support digital signatures.

These verbs are:
- Digital Signature Generate (CSNDDSG)
- Digital Signature Verify (CSNDDSV)
PKA key management

You can generate RSA and ECC keys using the PKA Key Generate verb (CSNDPKG), or a comparable product from another vendor.

You can use the PKA Key Generate verb to generate internal and external PKA tokens. You can also generate RSA keys on another system and then import them to the cryptographic coprocessor. To input a clear RSA key, create the token with the PKA Key Token Build verb and import it using the PKA Key Import verb. To input an encrypted RSA key, use the PKA Key Import verb.

In either case, use the PKA Key Token Build verb to create a skeleton key token as input (see “PKA Key Token Build (CSNDPKB)” on page 699).

The PKA Key Import verb uses the clear token from the PKA Key Token Build verb or a clear or encrypted token from the CCA system to securely import the key token into operational form for the coprocessor to use. CCA does not permit the export of the imported PKA key.

The PKA Public Key Extract verb builds a public key token from a private key token.

Application RSA public and private keys can be stored in the PKA key storage file.
Verbs for PKA key management

CCA provides the following verbs for PKA key management:

- PKA Key Generate (CSNDPKG)
- PKA Key Import (CSNPKII)
- PKA Key Token Build (CSNDPKB)
- PKA Key Token Change (CSNDKTC)
- PKA Key Translate (CSNDPKT)
- PKA Public Key Extract (CSNDPKX)
- Remote Key Export (CSNDRKX)
- Trusted Block Create (CSNDTBC)

Key identifier for PKA key token

A key identifier for a PKA key token is a variable length (maximum allowed size is 2500 bytes) area that contains either a key label or a key token.

- A key label identifies keys that are in the PKA key storage file.
- A key token can be either an internal key token, an external key token, or a null key token. Key tokens are generated by an application (for example, using the PKA Key Generate verb), or received from another system that can produce external key tokens.

An internal key token can be used only on the local system, because the PKA master key encrypts the key value. Internal key tokens contain keys in operational form only.

An external key token can be exchanged with other systems because a transport key that is shared with the other system encrypts the key value. External key tokens contain keys in either exportable or importable form.

A null key token consists of eight bytes of binary zeros. The PKA Key Record Create verb can be used to write a null token to the key storage file. This record can subsequently be identified as the target token for the PKA Key Import or PKA Key Generate verb.

The term key identifier is used when a parameter could be one of the above items, and indicates that different inputs are possible. For example, you might want to specify a specific parameter as either an internal key token or a key label. The key label is, in effect, an indirect reference to a stored internal key token.

Key label

If the first byte of the key identifier is greater than X'20' but less than X'FF', the field is considered to be holding a key label.

The contents of a key label are interpreted as the identifier of a key entry in the PKA storage file. The key label is an indirect reference to an internal key token.

If the first byte of the key identifier is X'FF', the identifier is not valid. If the first byte is less than X'20', the identifier is treated as a key token as described below.

A key label is specified on verbs with the key_identifier parameter as a 64-byte character string, left-aligned, and padded on the right with blanks. In most cases, the verb does not check the syntax of the key label other than the first byte.

A key label has the following form:
Key token

A key token is a variable length (maximum allowed size is 3500 bytes) field composed of key value and control information.

PKA keys can be either public or private RSA, or ECC keys. Each key token can be either an internal key token (the first byte of the key identifier is X'1F'), an external key token (the first byte of the key identifier is X'1E'), or a null PKA private key token (the first byte of the key identifier is X'00').

For descriptions of the PKA key tokens and for debugging information, see Chapter 19, "Key token formats," on page 847.

Internal key token

An internal key token is a token that can be used only on the system that created it or another system with the same PKA master key.

It contains a key that is encrypted under the PKA master key.

An application obtains an internal key token by using one of the verbs such as those listed below. The verbs are described in detail in Chapter 15, "Managing PKA cryptographic keys," on page 689.

- PKA Key Generate
- PKA Key Import

The PKA Key Token Change verb can re-encipher private internal tokens from encryption under the old ASYM-MK to encryption under the current ASYM-MK. PKA key storage Re-encipher/Activate options are available to re-encipher RSA and ECC internal tokens in the PKA key storage when the SYM-MK/ASYM-MK (or APKA-MK) keys are changed.

PKA master keys cannot be changed dynamically.

External key token

If the first byte of the key identifier is X'1E', the key identifier is interpreted as an external key token.

An external PKA key token contains key (possibly encrypted) and control information. By using the external key token, you can exchange keys between systems.

An application obtains the external key token by using one of the verbs such as those listed below. They are described in detail in Chapter 15, "Managing PKA cryptographic keys," on page 689.

- PKA Public Key Extract
- PKA Key Token Build
- PKA Key Generate
Null key token

If the first byte of the key identifier is X’00’, the key identifier is interpreted as a null key token.

Summary of the PKA verbs

Use this table of the PKA verbs to map them to their corresponding verb names and descriptions.

The PKA verb names start with CSND. This table also references the topic that describes the verb.

Table 12. Summary of PKA verbs

<table>
<thead>
<tr>
<th>Entry point</th>
<th>Verb name</th>
<th>Description</th>
<th>Topic/Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 14, “Using digital signatures,” on page 675</td>
<td>CSNDDSG</td>
<td>Digital Signature Generate</td>
<td>Generates a digital signature using an RSA or ECC private key.</td>
</tr>
<tr>
<td></td>
<td>CSNDDSV</td>
<td>Digital Signature Verify</td>
<td>Verifies a digital signature using an RSA or ECC public key.</td>
</tr>
<tr>
<td>Chapter 15, “Managing PKA cryptographic keys,” on page 689</td>
<td>CSNDPKG</td>
<td>PKA Key Generate</td>
<td>Generates an RSA or ECC public-private key-pair</td>
</tr>
<tr>
<td></td>
<td>CSNDPKI</td>
<td>PKA Key Import</td>
<td>Imports a key token containing either a clear key or an RSA or ECC key enciphered under a transport key.</td>
</tr>
<tr>
<td></td>
<td>CSNDPKB</td>
<td>PKA Key Token Build</td>
<td>Creates an external PKA key token containing a clear private RSA key. Using this token as input to the PKA Key Import verb returns an operational internal token containing an enciphered private key. Using PKA Key Token Build on a clear public RSA key, returns the public key in a token format that other PKA verbs can directly use. PKA Key Token Build can also be used to create a skeleton token for input to the PKA Key Generate verb for the generation of an internal RSA key token.</td>
</tr>
<tr>
<td></td>
<td>CSNDKTC</td>
<td>PKA Key Token Change</td>
<td>Changes PKA key tokens from encipherment with the old asymmetric-keys master key to encipherment with the current asymmetric-keys master key. This verb changes only private internal tokens.</td>
</tr>
<tr>
<td></td>
<td>CSNDPKT</td>
<td>PKA Key Translate</td>
<td>Translates PKA key tokens from encipherment under the old Asymmetric-Keys Master Key to encipherment under the current Asymmetric-Keys Master Key. This verb changes only Private Internal PKA Key Tokens.</td>
</tr>
<tr>
<td>Entry point</td>
<td>Verb name</td>
<td>Description</td>
<td>Topic/Page</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CSNDPKX</td>
<td>PKA Public Key Extract</td>
<td>Extracts a PKA public key token from a supplied PKA internal or external private key token. Performs no cryptographic verification of the PKA private token.</td>
<td>“PKA Public Key Extract (CSNDPKX)” on page 720</td>
</tr>
<tr>
<td>CSNDRKX</td>
<td>Remote Key Export</td>
<td>Secure transport of DES keys using asymmetric techniques from a security module (for example, the CEX*C) to a remote device such as an Automated Teller Machine (ATM).</td>
<td>“Remote Key Export (CSNDRKX)” on page 722</td>
</tr>
<tr>
<td>CSNDTBC</td>
<td>Trusted Block Create</td>
<td>Creates an external trusted block under dual control. A trusted block is an extension of CCA PKA key tokens using new section identifiers.</td>
<td>“Trusted Block Create (CSNDTBC)” on page 734</td>
</tr>
</tbody>
</table>
Chapter 5. TR-31 symmetric key management


The provided information is an extension of Chapter 8, “Managing AES, DES, and HMAC cryptographic keys,” on page 165. For additional information on symmetric keys, including DES control vectors, see Chapter 8, “Managing AES, DES, and HMAC cryptographic keys,” on page 165.

The TR-31 key block is a format defined by the ANSI Standards Committee to support interchange of symmetric keys in a secure manner and with key attributes included in the exchanged data. Currently, this format supports only DES keys. AES keys are not supported.

TR-31 is a Technical Report. This is different from a standard, which is a mandatory set of rules that must be followed. A Technical Report is not mandatory, but provides guidance to those who are using the standards. In this case, TR-31 is a companion to the standard X9.24-1, which defines requirements for key management performed using symmetric key techniques. TR-31 shows a method that complies with the various requirements that are defined in X9.24-1, and because no other specific method has been defined by the standards committee, the TR-31 method is becoming the apparent standard through which financial organizations will exchange keys.

Prior to TR-31, there were problems with the interchange of symmetric keys. In the banking environment, it is very important that each symmetric key have a specific set of attributes attached to it, specifying such things as the cryptographic operations for which that key can be used. CCA implements these attributes in the form of the control vector (CV), but other vendors implement attributes in their own proprietary ways. Thus, if you are exchanging keys between CCA systems, you can securely pass the attributes using CCA functions and data structures. If, however, that same key were sent to a non-CCA system, there would be no secure way to do that. This is because the two cryptographic architectures have no common key format that could be used to pass both the key and its attributes. As a result, the normal approach has been to strip the attributes and send just the encrypted key, then attach attributes again at the receiving end.

The above scenario has major security problems because it allows an insider to obtain the key without its designated attributes. The insider can then attach other attributes to it, thereby compromising the security of the system. For example, assume the exchanged key is a key-encrypting key (KEK). The attributes of a KEK should restrict its use to key management functions that are designed to prevent exposure of the keys that the KEK is used to encrypt. If that KEK is transmitted without any attributes, an attacker on the inside can turn the key into a type used for data decryption. Such a key can then be used to decipher all of the keys that were previously protected using the KEK. It is clearly very desirable to have a way of exchanging keys that prevents this modification of the attributes. TR-31 provides such a method.

The TR-31 key block has a set of defined key attributes. These attributes are securely bound to the key so that they can be transported together between any two systems that both understand the TR-31 format. This is much of the reason for
its gain in popularity. There are two supported cryptographic methods for protecting the key block. The original version of TR-31 defined a method that encrypted the key field in CBC mode and computed a TDES MAC over the header and key field. The encryption and MAC operations used different keys, created by applying predefined variants to the input key block protection key. This method is identified by a Key Block Version ID value of A (X’41’). An update to TR-31 adds a more modern method, identified by a Key Block Version ID value of B (X’42’) or C (X’43’). The B method uses an authenticated encryption scheme and uses cryptographic key derivation methods to produce the encryption and MAC keys. The C method is exactly the same as the A method in terms of wrapping keys. However, the field values are expected to conform to the updated standard.

Not surprisingly, TR-31 uses some key attributes that are different from those in the CCA control vector. In some cases, there is a one-to-one correspondence between CCA and TR-31 attributes. For these cases, conversion is simple and straightforward. In other cases, the correspondence is one-to-many or many-to-one and the application program must provide information to help the CCA verbs decide how to perform the translation between CCA and TR-31 attributes. There are also CCA attributes that simply cannot be represented using TR-31. CCA keys with those attributes are not eligible for conversion to TR-31 format.

The TR-31 key block has these two important features:

1. The key is protected in such a way that it meets the key bundling requirements of various standards. These standards state that the individual 8-byte blocks of a double-length or triple-length TDES key must be bound in such a way that they cannot be individually manipulated. TR-31 accomplishes this mainly by computation of a MAC across the entire structure, excluding the MAC value itself.

2. Key usage attributes, defined to control how the key can be used, are securely bound to the key itself. This makes it possible for a key and its attributes to be securely transferred from one party to another while assuring that the attributes of the key cannot be modified to suit the needs of an attacker.


**Note:** CCA 6.0 introduces compliance-tagged key support for domains that enter compliance mode. Compliance-tagged DES keys may be converted to or from TR-31 key blocks. The wrapping key encrypting key (KEK) must be compliance-tagged as well as the key to be exported or imported. A compliance-tagged KEK cannot be used to import or export a key that is not compliance-tagged.

<table>
<thead>
<tr>
<th>Verb</th>
<th>Page</th>
<th>Service</th>
<th>Entry point</th>
<th>Service location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Export to TR31</td>
<td>“Key Export to TR31 (CSNBT31X)” on page 755</td>
<td>Exports a CCA external or internal fixed-length symmetric key-token, converting it into an external X9 TR-31 key block format.</td>
<td>CSNBT31X</td>
<td>cryptographic engine</td>
</tr>
<tr>
<td>TR31 Key Import</td>
<td>“TR31 Key Import (CSNBT31I)” on page 781</td>
<td>Imports an external X9 TR-31 key block, converting it into a CCA external or internal fixed-length symmetric key-token.</td>
<td>CSNBT31I</td>
<td>cryptographic engine</td>
</tr>
<tr>
<td>Verb</td>
<td>Page</td>
<td>Service</td>
<td>Entry point</td>
<td>Service location</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td>-------------</td>
<td>------------------</td>
</tr>
<tr>
<td>TR31 Key Token Parse</td>
<td>&quot;TR31 Key Token Parse (CSNBT31P)&quot; on page 804</td>
<td>Parses the information from the standard predefined fields of the TR-31 key block header without importing the key.</td>
<td>CSNBT31P</td>
<td>security API host software</td>
</tr>
<tr>
<td>TR31 Optional Data Build</td>
<td>&quot;TR31 Optional Data Build (CSNBT31O)&quot; on page 808</td>
<td>Constructs the optional blocks of a TR-31 key block, one block at a time.</td>
<td>CSNBT31O</td>
<td>security API host software</td>
</tr>
<tr>
<td>TR31 Optional Data Read</td>
<td>&quot;TR31 Optional Data Read (CSNBT31R)&quot; on page 811</td>
<td>Obtains the contents of any optional fields of a TR-31 key block header.</td>
<td>CSNBT31R</td>
<td>security API host software</td>
</tr>
</tbody>
</table>
Chapter 6. Understanding and managing master keys

In a CCA node, AES, DES, APKA, and PKA master keys are used to wrap or unwrap working keys. These master keys are also used to wrap or unwrap the object protection keys (OPKs) of the working keys that have an OPK defined. These keys are used by the node and can appear outside of the cryptographic engine, and therefore need wrapping.

The ECC keys are wrapped by using 32-byte AES keys. The DES and RSA working keys are wrapped using Triple-DES encryption. DES working keys can be wrapped by a more secure method of Triple-DES by using CBC mode. This method is called the enhanced key-wrapping method. These methods of securing keys enable a node to operate on an unlimited number of working keys, without concern for storage space within the confines of the secured cryptographic engine.

The CCA design supports a set of three master-key registers for each master key: new, current, and old. While a master key is being assembled, it is accumulated in the new-master-key register. Then the Master Key Process verb is used to transfer the contents of the new-master-key register to the current-master-key register.

Working keys are normally encrypted by their associated current master key. To facilitate continuous operations, CCA also has an old-master-key register. When a new master key is transferred to the current-master-key register, the preexisting contents, if any, of the current-master-key register are transferred to the old-master-key register. Whenever a working key must be decrypted by the master key in CCA, the master key verification pattern information that is included in the key token is used to determine whether the current or the old master key must be used to recover the working key. Special status (return code 0, reason code 10001) is returned if the old master key is used. Thus, application programs can arrange to update the working key by encryption with the current master key (with the help of the Key Token Change and PKA Key Token Change verbs). Whenever a working key is encrypted for local use, the key or its OPK is wrapped with the current master key.

Symmetric and asymmetric master keys

Read the contained information on how CCA handles symmetric and asymmetric master keys.

CCA incorporates the following sets of master-key registers:

• The DES master-key register set is used to wrap and unwrap DES (symmetric) working keys.
• The PKA master-key register set is used to wrap and unwrap RSA (asymmetric) private working keys, or the object protection keys (OPKs) of the RSA working keys that have an OPK defined, excluding private key sections X’30’ and X’31’, which have their OPKs wrapped and unwrapped by the APKA master key.
• The AES master-key register set is used to wrap and unwrap AES (symmetric) fixed-length, and AES and HMAC variable-length, symmetric working keys.
• The APKA master-key register set is used to wrap and unwrap the Object Protection Key (OPK) that is itself used to wrap the key material of an Elliptic Curve Cryptography (ECC) key or the OPK of RSA private key sections X’30’ and X’31’. ECC keys are asymmetric.
The verbs that operate on the master keys permit you to specify a register set (with keywords AES-MK, APKA-MK, SYM-MK and ASYM-MK). For DES and PKA master keys, if applications that modify these master-key registers never explicitly select a register set, the master keys in the two register sets are modified in the same way and contain the same keys. However, if at any time you modify only one of the DES or PKA register sets, applications thereafter need to manage the two register sets independently.

**Establishing master keys**

The preferred and most secure method of establishing master keys in the coprocessor is to use a Trusted Key Entry (TKE) workstation. The TKE leverages user smart cards to establish a secure connection all the way to the firmware in the coprocessor, with a unique session key. Your key parts are secured from the smart card all the way to the target coprocessor with this solution.

An AES master key is established from clear key parts (components). An APKA master key is also established from clear key parts.

DES and PKA master keys, on the other hand, are established in one of these ways:
- from clear key parts
- through random generation internal to the coprocessor

**Establishing a master key from clear information**

Individual key parts are supplied as clear information, and the parts are exclusive-ORed within the cryptographic engine. Knowledge of a single part gives no information about the final key when multiple, random-valued parts are exclusive-ORed.

A common technique is to record the values of the parts (typically on paper or diskette) and independently store these values in locked safes. When installing the master key, individuals trusted to not share the key-part information, retrieve the parts and enter the information into the cryptographic engine. Use the Master Key Process verb for this operation.

Entering the first and subsequent parts is authorized by two different control points so that a cryptographic engine, the coprocessor, can enforce that two different roles, and thus profiles, are activated to install the master-key parts. This requires that roles exist that enforce this separation of responsibility.

Setting the master key uses a unique command with its own control point. You can set up the access-control system to require the participation of at least three individuals or three groups of individuals.

You can check the contents of any of the master-key registers, and the key parts as they are entered into the new-master-key register, using the Key Test verb. The verb performs a one-way function on the key-of-interest, the result of which is either returned or compared to a known correct result.
Establishing a DES or PKA master key from an internally generated random value

The Master Key Process verb can be used to randomly generate a new DES or PKA master key within the cryptographic engine. The value of the new master-key is not available outside of the cryptographic engine. The verb does not support random generation of an AES master key.

This random method, which is a separately authorized command invoked through use of the Master Key Process verb, ensures that no one has access to the value of the master key. Randomly generating a master key is useful when keys shared with other nodes are distributed using public key techniques, or when DES transport keys are established between nodes. In these cases, there is no need to reestablish a master key with the same value. In general, IBM does not recommend to use a random master key, because workload sharing and backup solutions become very difficult.
Part 2. CCA verbs

CCA verbs implement a variety of cryptographic processes and data-security techniques. Each available verb is documented in detail in a separate topic.

For each documented verb, a Java Native Interface (JNI) is defined that you can use for JNI work.

Note: In this JNI, a new data type hikmNativeLong is replacing the old type hikmNativeInteger since CCA version 5.2. Both types inherit from an abstract class hikmNativeNumber. Thus, type hikmNativeInteger is still supported, so you can run existing applications with this deprecated data type. However, start using type hikmNativeLong for new applications instead, because hikmNativeInteger may be removed in the future.

The following topics are contained:

- Chapter 7, “Using the CCA nodes and resource control verbs,” on page 89 describes how to use the CCA resource control verbs.
- Chapter 8, “Managing AES, DES, and HMAC cryptographic keys,” on page 165 describes the verbs for generating and maintaining AES, DES, and HMAC cryptographic keys, the Random Number Generate verb (which generates 8-byte random numbers), the Random Number Generate Long verb (which generates up to 8192 bytes of random content), and the Secure Sockets Layer (SSL) security protocol. This chapter also describes utilities to build DES and AES tokens, generate and translate control vectors, and describes the PKA verbs that support DES and AES key distribution.
- Chapter 9, “Protecting data,” on page 379 describes the verbs for enciphering and deciphering data.
- Chapter 10, “Verifying data integrity and authenticating messages,” on page 417 describes the verbs for generating and verifying Message Authentication Codes (MACs), generating Modification Detection Codes (MDCs) and generating hashes (SHA-1, MD5, RIPEMD-160).
- Chapter 11, “Key storage mechanisms,” on page 449 describes the use of key storage, key tokens, and associated verbs.
- Chapter 12, “Financial services,” on page 491 describes the verbs for use in support of finance-industry applications. This includes several categories.
  - Verbs for generating, verifying, and translating personal identification numbers (PINS).
  - Verbs that generate and verify VISA card verification values and American Express card security codes.
  - Verbs to support smart card applications using the EMV (Europay MasterCard Visa) standards.
- Chapter 13, “Financial services for DK PIN methods,” on page 609 contains information on financial services that are based on the PIN methods and requirements specified by the German Banking Industry Committee (Deutsche Kreditwirtschaft (DK)).
- Chapter 14, “Using digital signatures,” on page 675 describes the verbs that support using digital signatures to authenticate messages.
- Chapter 15, “Managing PKA cryptographic keys,” on page 689 describes the verbs that generate and manage PKA keys.
• Chapter 16, “TR-31 symmetric key management verbs,” on page 755 describes the verbs that manage TR-31 functions.

• Chapter 17, “Utility verbs,” on page 817 describes the Code Conversion (CSNBXEA) utility. Use this verb to convert text strings from ASCII to EBCDIC or from EBCDIC to ASCII.
Chapter 7. Using the CCA nodes and resource control verbs

Use these verbs to control CCA nodes and their resources.

The following verbs are described:

- “Access Control Maintenance (CSUAACM)”
- “Access Control Tracking (CSUAACT)” on page 93
- “Cryptographic Facility Query (CSUACFQ)” on page 100
- “Cryptographic Facility Version (CSUACFV)” on page 143
- “Cryptographic Resource Allocate (CSUACRA)” on page 144
- “Cryptographic Resource Deallocate (CSUACRD)” on page 147
- “Key Storage Initialization (CSNBKSI)” on page 149
- “Log Query (CSUALGQ)” on page 152
- “Master Key Process (CSNBMKP)” on page 157
- “Random Number Tests (CSUARNT)” on page 161

Access Control Maintenance (CSUAACM)

Use the Access Control Maintenance verb to query or control installed roles.

You can use this verb to perform the following services:

- Retrieve a list of the installed roles.
- Retrieve the non-secret data for a selected role.

You select which service to perform by specifying the corresponding keyword in the input rule-array. You can only perform one of these services per verb call.

Chapter 25, “Access control data structures,” on page 1073 describes the roles and role structures and provides examples of access control data structures for the roles.

Note: You can also perform the services provided by the CSUAACM verb with the help of the panel.exe utility. Refer to “panel.exe default syntax as of CCA 6.0” on page 1121 and to “Using panel.exe to show the active role and ACPs” on page 1129.

Format

The format of CSUAACM.

```
CSUAACM(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    name,
    output_data_length,
    output_data)
```
Parameters

The parameter definitions for CSUAACM.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

`rule_array_count`

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1.

`rule_array`

- **Direction:** Input
- **Type:** String array

The `rule_array` parameter is a pointer to a string variable containing an array of keywords. The keywords are eight bytes in length and must be left-aligned and padded on the right with space characters. The `rule_array` keywords are described in Table 14.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTROLES</td>
<td>Retrieves a list of the roles installed in the coprocessor.</td>
</tr>
<tr>
<td>GET-ROLE</td>
<td>Retrieves the non-secret part of a role definition from the coprocessor.</td>
</tr>
</tbody>
</table>

Table 14. Keywords for Access Control Maintenance control information

`name`

- **Direction:** Input
- **Type:** String

The `name` parameter is a pointer to a string variable containing the name of a role or user profile which is the target of the request.

The manner in which this variable is used depends on the service being performed.

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Contents of name parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTROLES</td>
<td>The <code>name</code> parameter is unused.</td>
</tr>
<tr>
<td>GET-ROLE</td>
<td>The <code>name</code> parameter contains the 8-character role ID for the role definition that is to be retrieved or deleted. A role ID cannot start with a space character.</td>
</tr>
</tbody>
</table>

`output_data_length`

- **Direction:** Input/Output
- **Type:** Integer

The `output_data_length` parameter is a pointer to an integer variable containing the number of bytes of data in the `output_data` variable. The value must be a multiple of four bytes.
On input, the `output_data_length` parameter must be set to the total size of the variable pointed to by the `output_data` parameter. On output, this variable contains the number of bytes of data returned by the verb in the `output_data` variable.

`output_data`

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>String</td>
</tr>
</tbody>
</table>

The `output_data` parameter is a pointer to a string variable containing data returned by the verb. Any integer value returned in the `output_data` variable is in big-endian format; the high-order byte of the value is in the lowest-numbered address in storage. Authentication data structures are described in Chapter 25, “Access control data structures,” on page 1073.

The manner in which this variable is used depends on the function being performed.

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Contents of output_data parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTROLES</td>
<td>Contains a list of the role IDs for all the roles stored in the coprocessor.</td>
</tr>
</tbody>
</table>

Access Control Maintenance (CSUAACM)
Table 16. Meaning of the output_data parameter (continued)

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Contents of output_data parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET-ROLE</td>
<td>The variable contains the non-secret portion of the selected role. This includes the following data, in the order listed.</td>
</tr>
<tr>
<td></td>
<td><strong>Role version</strong></td>
</tr>
<tr>
<td></td>
<td>Two bytes containing 2 one-byte integer values, where the first byte contains the major version number and the second byte contains the minor version number.</td>
</tr>
<tr>
<td></td>
<td><strong>Comment</strong></td>
</tr>
<tr>
<td></td>
<td>A 20-character variable padded on the right with spaces, containing a comment which describes the role. This variable is not X'00' terminated.</td>
</tr>
<tr>
<td></td>
<td><strong>Required authentication-strength level</strong></td>
</tr>
<tr>
<td></td>
<td>A 2-byte integer defining how secure the user authentication must be in order to authorize this role.</td>
</tr>
<tr>
<td></td>
<td><strong>Lower time-limit</strong></td>
</tr>
<tr>
<td></td>
<td>The earliest time of day that this role can be used. The time limit consists of two 1-byte integer values, a 1-byte hour, followed by a 1-byte minute. The hour can range from 0 - 23, and the minute can range from 0 - 59.</td>
</tr>
<tr>
<td></td>
<td><strong>Upper time-limit</strong></td>
</tr>
<tr>
<td></td>
<td>The latest time of day that this role can be used. The format is the same as the Lower time-limit.</td>
</tr>
<tr>
<td></td>
<td><strong>Valid days of the week</strong></td>
</tr>
<tr>
<td></td>
<td>A 1-byte variable defining which days of the week this role can be used. Seven bits of the byte are used to represent Sunday through Saturday, where a 1 bit means that the day is allowed, while a 0 bit means it is not.</td>
</tr>
<tr>
<td></td>
<td>The first bit (most significant bit, MSB) is for Sunday, and the last significant bit (LSB) is unused and is set to B'0'.</td>
</tr>
<tr>
<td></td>
<td><strong>Access-control-point list</strong></td>
</tr>
<tr>
<td></td>
<td>The access-control-point bit map defines which functions a user with this role is permitted to run.</td>
</tr>
</tbody>
</table>

**Restrictions**

The restrictions for CSUAACM.

None.

**Required commands**

The CSUAACM required commands.

The Access Control Maintenance verb requires the shown command to be enabled in the active role, based on the following rule_array keywords:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTROLES, GET-ROLE</td>
<td>X'0116'</td>
<td>Access Control Manager - Read role</td>
</tr>
</tbody>
</table>
Usage notes

Usage notes for CSUAACM.

Only roles relevant to the domain which originates the command are listed or returned.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSUAACMJ.

See "Building Java applications using the CCA JNI" on page 28.

Format

```java
public native void CSUAACMJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] name,
    hikmNativeNumber output_data_length,
    byte[] output_data);
```

Access Control Tracking (CSUAACCT)

The Access Control Tracking verb allows you to track the usage of ACPs for a defined span of time, and also allows you to query that tracking information.

Use the Access Control Tracking verb to start or stop tracking of access control points (ACPs) that are queried by applications. You can also use this verb to maintain ACP tracking information. Some key points to consider are:

- Tracking is started, stopped, and maintained on a per role basis, for one or more role IDs at a time.
- Tracking of ACPs occurs between START and STOP requests.
- Whenever an application, that is running under a role ID that is being tracked, causes an ACP to be queried, this ACP is tracked.
- ACP information is returned on the basis of the selected role ID.
  - Data size: CSUAACCT uses the same access-control-point list data structure and transfer semantics as the Access Control Maintenance (CSUAACM) verb. Both verbs also share the same transfer size limitations, if any. It is deemed unlikely that role tracking data for one role passes the maximum data transfer size for a platform since typically data for a single role is returned.
  - If a SIZEDATA request for a given list of roles results in an error that indicates the total size is too large to transfer, try reducing the size of the list of roles by dividing it into multiple queries.
- Tracking configuration and data is retained inside the cryptographic coprocessor until cleared by a specific request or re-initialization.

To use this verb, do the following:

1. Specify the desired function to perform in the rule array. Choose whether to start or stop access control tracking, whether to retrieve collected tracking data or the size of that data. You can also choose to query if tracking is enabled, or clear tracking information.
Access Control Tracking (CSUAACT)

2. Use the role_ID parameter to optionally identify one or more 8-character role IDs to be tracked. If no role IDs are specified (that is, the role_ID_length variable is 0), the verb operates on the default role, which all users may use.

3. Specify a buffer large enough to receive any output data.

Note:
1. Tracking of role IDs has minor performance implications. From the initialization of a role ID tracking structure after a START until it is cleared, either by the CLRDATA option of this verb or by a reinitialization of the CCA application in the coprocessor, two identical copies of the tracking structure are maintained. One is an in-memory copy, and the other is an in-storage copy. The in-memory copy serves to provide the least impact on performance, while the in-storage copy serves to survive a power outage. During the period that a role ID is being tracked, any attempt by that role to access an ACP causes the role ID access control point list of the in-memory copy to be checked. When an ACP (offset) in the list is checked and found to be set to B'1' (at least one access attempt for this ACP, whether successful or not, has been recorded), no further processing is performed. Otherwise, the offset in both the in-memory and in-storage lists is changed to B'1' from B'0'. This update to both lists only occurs once per ACP accessed.

2. Chapter 25, “Access control data structures,” on page 1073 describes the roles and role structures and provides examples of access control data structures for the roles.

3. You can also perform ACP tracking with the help of the panel.exe utility. Refer to “panel.exe default syntax as of CCA 6.0” on page 1121 and to “Using panel.exe to control ACP tracking” on page 1131.

Format
The format of CSUAACT.

```
CSUAACT(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    role_ID_length,
    role_ID,
    output_data_length,
    output_data,
    reserved1_length,
    reserved1,
    reserved2_length,
    reserved2)
```

Parameters
The parameter definitions for CSUAACT.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

Direction: Input
Type: Integer
A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1.

**rule_array**

**Direction:** Input  
**Type:** String array

The `rule_array` parameter is a pointer to a string variable containing an array of keywords. The keywords are eight bytes in length and must be left-aligned and padded on the right with space characters. The `rule_array` keywords are described in [Table 17](#).

### Table 17. Keywords for Access Control Tracking control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function to perform (required)</strong>. Setting the <code>role_ID_length</code> variable to 0 causes the function to operate on the default role ID.</td>
<td></td>
</tr>
</tbody>
</table>
| CLRDATA  | Specifies to clear ACP tracking information for the role IDs identified by the `role_ID` parameter.  
**Note:**  
1. No data is returned in the buffer identified by the `output_data` parameter.  
2. This function does not start or stop tracking information. Use the START or STOP keywords for this purpose.  
3. It is possible for some data to be gathered after CLRDATA, but before a subsequent STOP. To avoid this, use STOP before CLRDATA.  
4. This function generates an ACP query which gets logged before any tracking information is cleared. |
| GETDATA  | Note: Specifies to return the tracking information for the role IDs identified by the `role_ID` parameter. This tracking information is returned into the buffer identified by the `output_data` parameter, provided that the buffer is large enough to receive all of the data.  
1. This function does not clear tracking information. Use CLRDATA for this purpose.  
2. This function does not stop ACP tracking. Use STOP instead.  
3. Use SIZEDATA to determine the minimum required size for the buffer identified by the `output_data` parameter. |
| GETSTATE | Specifies to return the state (either enabled or not enabled) of ACP tracking for the role IDs identified by the `role_ID` parameter. The state is returned into the buffer identified by the `output_data` parameter, provided that the buffer is large enough to receive all of the data.  
**Note:** This function does not start or stop ACP tracking. Use START or STOP instead. |
| SIZEDATA | Specifies to return the size of the data in bytes that GETDATA would return as a single integer value into the `output_data_length` variable.  
**Note:**  
1. No data is returned in the buffer identified by the `output_data` parameter.  
2. The `output_data_length` variable can be greater on output than on input.  
3. Provide the same list of role IDs to be used for GETDATA. |
Table 17. Keywords for Access Control Tracking control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
</table>
| START   | Specifies to start ACP tracking for the role IDs identified by the role_ID parameter.  
Note:  
1. No data is returned in the buffer identified by the output_data parameter.  
2. Use of this function for a role ID that is being tracked has no effect and does not cause an error or warning. |
| STOP    | Specifies to stop ACP tracking for the role IDs identified by the role_ID parameter.  
Note:  
1. No data is returned in the buffer identified by the output_data parameter.  
2. Use of this keyword on a role ID that is not being tracked has no effect and does not cause an error or warning.  
3. ACP tracking stops after the ACP check for the required commands of this verb are recorded for the appropriate role ID. |

**role_ID_length**
- Direction: Input
- Type: Integer

The role_ID_length parameter is a pointer to an integer variable containing the number of bytes of data in the role_ID variable. This value must be greater than or equal to 0 and a multiple of 8.

**Note:** Setting the role_ID_length variable to 0 causes the function to operate on the default role ID.

**role_ID**
- Direction: Input
- Type: String

This parameter is a pointer to a string variable containing an optional array of role IDs of the role definitions to process. The role IDs are 8 bytes in length and must be left-aligned and padded on the right with space characters.

**output_data_length**
- Direction: Input/Output
- Type: Integer

This parameter is a pointer to an integer variable containing the number of bytes of data in the output_data variable. For keyword SIZEDATA, this value is allowed to be less on input than on output since no data is returned in the buffer identified by the output_data parameter. Otherwise, set this value to at least the number of data bytes to be returned in that buffer.

**output_data**
- Direction: Output
- Type: String

The output_data parameter is a pointer to a string variable containing data returned by the verb. The size and content of the output data depends on the function to perform based on the rule array. Only rule array keywords
GETDATA and GETSTATE return data in this variable (buffer). Keyword SIZEDATA does not return any data in this buffer, even if the returned output_data_length variable is greater than 0.

The GETSTATE option returns a concatenation of role tracking data headers, one for each role ID identified by the role_ID parameter. Refer to Table 18. No additional data follows the header for GETSTATE.

**Table 18. Role tracking data header format**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>02</td>
<td>Role tracking data (RTD) header version (X'0100').</td>
</tr>
<tr>
<td>02</td>
<td>02</td>
<td>RTD length in bytes (big endian): rtdln. Length includes this header (16 bytes) + length of data that follows the header, if any.</td>
</tr>
<tr>
<td>04</td>
<td>08</td>
<td>Role ID of role tracking data</td>
</tr>
<tr>
<td>12</td>
<td>01</td>
<td>Role tracking flag byte for the role ID (value at offset 4)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>B'xxxx xxx0'</td>
<td>Tracking is not enabled.</td>
</tr>
<tr>
<td>B'0000 0001'</td>
<td>Tracking is enabled.</td>
</tr>
</tbody>
</table>

All unused bits are reserved and must be zero.

| 13 | 03 | Reserved, binary zero. |

**Note:**
1. For GETDATA, the returned output_data is a concatenation of a role tracking data header and an ACP list pair, one for each role ID identified by the role_ID parameter.
2. For GETSTATE, the returned output_data is a concatenation of role tracking data headers, one for each role ID identified by the role_ID parameter.

The GETDATA option returns a role tracking data header for each role ID, with an additional ACP list concatenated to each header. Refer to Table 308 on page 1075. Table 19 provides the format of the output returned by the GETDATA option.

**Table 19. GETDATA output_data format**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role_id</td>
<td>Length (bytes)</td>
<td>Description</td>
</tr>
<tr>
<td>role_id 1 of n, where n = 1 for role_ID_length = 0. Otherwise n = role_ID_length / 8.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>02</td>
<td>Role tracking data (RTD) header version (X'0100').</td>
</tr>
<tr>
<td>02</td>
<td>02</td>
<td>RTD length in bytes (big endian): rtdln#1. Length includes this header (16 bytes) + length of ACP list structure.</td>
</tr>
<tr>
<td>04</td>
<td>08</td>
<td>Role ID of role tracking data</td>
</tr>
</tbody>
</table>
### Table 19. GETDATA output_data format (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>01</td>
<td>Role tracking flag byte for the role ID (value at offset 4)</td>
</tr>
</tbody>
</table>

**Value Meaning**

- \( B''xxxx xxx0' \)
  - Tracking is not enabled.
- \( B''0000 0001' \)
  - Tracking is enabled.

All unused bits are reserved and must be zero.

<table>
<thead>
<tr>
<th>13</th>
<th>03</th>
<th>Reserved, binary zero.</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>rtdln#1 - 16</td>
<td>Access-control-point list structure for role ID 1. Refer to Table 308 on page 1075</td>
</tr>
</tbody>
</table>

The meaning of each bit valued in the bit-map data segments of the ACP list structure has the following meaning for role tracking data:

**Value Meaning**

- \( B''0' \)
  - No recorded access attempt has been made.
- \( B''1' \)
  - This ACP has had at least one access attempt while being tracked. The value does not indicate the success of any access attempt.

**Note:**

1. Use the Access Control Maintenance (CSUAACM) verb with the GET-ROLE rule-array keyword to determine which offsets this role ID has enabled. Compare the CSUAACM list of enabled ACPs to the tracking list of the role ID to determine which ACP access attempts were successful.

2. The gathering of data for a given role ID can be impacted by multiple STARTs and STOPs. If CLRDATA is not performed between a STOP and a START.

For role ID 2 of n, for role ID length \( \geq 2^8 \) where \( n = \text{role ID length} / 8 \)

<table>
<thead>
<tr>
<th>rtdln#1</th>
<th>02</th>
<th>RTD header version.</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtdln#1 + 02</td>
<td>02</td>
<td>RTD overall length in bytes (big endian): rtdln#2</td>
</tr>
<tr>
<td>rtdln#1 + 04</td>
<td>08</td>
<td>Role ID of role tracking data.</td>
</tr>
<tr>
<td>rtdln#1 + 12</td>
<td>01</td>
<td>Role tracking flag byte for role ID (value at offset rtdln#n-1 + 4).</td>
</tr>
<tr>
<td>rtdln#1 + 13</td>
<td>03</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>rtdln#1 + 16</td>
<td>rtdln#n - 16</td>
<td>Access-control-point list structure for role ID n</td>
</tr>
</tbody>
</table>

**Note:**

1. For GETDATA, the returned output_data is a concatenation of a role tracking data header and ACP list pair, one for each role ID identified by the role_ID parameter.

2. All integer values returned in the output_data variable are in big-endian format.
Access Control Tracking (CSUAACT)

The GETSTATE option returns a concatenation of role tracking data headers, one for each role ID identified by the **role ID** parameter. Refer to Table 20. No additional data follows the header for GETSTATE.

**Table 20. GETSTATE output_data format**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>02</td>
<td>Role tracking data (RTD) header version (X'0100').</td>
</tr>
<tr>
<td>02</td>
<td>02</td>
<td>RTD structure length in bytes (big endian). Length includes this header (16 bytes) + length of data that follows the header (zero for GETSTATE).</td>
</tr>
<tr>
<td>04</td>
<td>08</td>
<td>Role ID of role tracking data</td>
</tr>
<tr>
<td>12</td>
<td>01</td>
<td>Role tracking flag byte for the role ID (value at offset 4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx xxx0'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0000 0001'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
<tr>
<td>13</td>
<td>03</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

**Note:** For GETSTATE, the returned output_data is a concatenation of role tracking data headers, one for each role ID identified by the role_ID parameter.

**reserved1_length**

**Direction:** Input/Output  
**Type:** Integer

The **reserved1_length** parameter is a pointer to an integer variable containing the number of bytes of data in the **reserved1** variable. This value must be zero.

**reserved1**

**Direction:** Output  
**Type:** String

The **reserved1** parameter is a pointer to a string variable. This parameter is reserved for future use.

**reserved2_length**

**Direction:** Input/Output  
**Type:** Integer

The **reserved2_length** parameter is a pointer to an integer variable containing the number of bytes of data in the **reserved2** variable. This value must be zero.

**reserved2**

**Direction:** Output  
**Type:** String

The **reserved2** parameter is a pointer to a string variable. This parameter is reserved for future use.
Access Control Tracking (CSUAACT)

Restrictions
The restrictions for CSUAACT.
None.

Required commands
The CSUAACT required commands.
The Access Control Tracking verb requires the following command to be enabled in the active role, based on the following rule_array keywords:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLRDATA, GETDATA, SIZEDATA, START, or STOP</td>
<td>X'01CC'</td>
<td>Access Control Tracking - Enable</td>
</tr>
</tbody>
</table>

Usage notes
Usage notes for CSUAACT.
None.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSUAACTJ.
See “Building Java applications using the CCA JNI” on page 28.

Format
public native void CSUAACTJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber role_ID_length
    byte[] role_ID,
    hikmNativeNumber output_data_length,
    byte[] output_data,
    hikmNativeNumber reserved1_length,
    byte[] reserved1,
    hikmNativeNumber reserved2_length,
    byte[] reserved2);

Cryptographic Facility Query (CSUACFQ)
The Cryptographic Facility Query verb is used to retrieve information about the coprocessor and the CCA application program in that coprocessor.
This information includes the following:
• General information about the coprocessor, its operating system, and CCA application
• The Environment Identifier (EID)
• Diagnostic information from the coprocessor
• Export-control information from the coprocessor
• Time and date information from the coprocessor
The contents and size of the authorized PIN decimalization tables loaded onto the coprocessor

On input, you specify:
- A `rule_array_count` of 1 or 2
- Optionally, a `rule_array` keyword of ADAPTER1 (for backward compatibility)
- The class of information queried with a `rule_array` keyword

This verb returns information elements in the `rule_array` and sets the `rule_array_count` variable to the number of returned elements.

**Determining if a card is a CEX3C, CEX4C, CEX5C, or a CEX6C**

Using the Cryptographic Facility Query, the output `rule_array` for option STATCCA is the most accurate way to determine the cryptographic coprocessor type.

An updated device driver might not be available yet for all distributions where the current RPM is usable. The CCA host library uses this mechanism to determine card version, and we recommend that the application developer also use this method. Where this Cryptographic Facility Query output from the `rule_array` for option STATCCA and the device driver disagree about the version of a particular card, it is the device driver that will be out of date because the Cryptographic Facility Query data is not interpreted in any way. It comes directly from the adapter.

- If first character of the CCA application version field is a number 4 or greater, then this card is a CEX3C or higher. For example, a 4 in the first character indicates a CEX3C or CEX4C.
- If the first character of the CCA application version field is the number 5, then this card is a CEX5C.
- If the first character of the CCA application version field is the number 6, then this card is a CEX6C.

The commands `ivp.e` and `panel.exe -x` also tell you the cryptographic coprocessor type, by calling the Cryptographic Facility Query verb for all available adapters.

For details about `panel.exe`, see “The panel.exe utility” on page 1121.

**Format**

The format of CSUACFQ.

```python
CSUACFQ(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    verb_data_length,
    verb_data)
```

**Parameters**

The parameter definitions for CSUACFQ.
Cryptographic Facility Query (CSUACFQ)

For the definitions of the \textit{return\_code}, \textit{reason\_code}, \textit{exit\_data\_length}, and \textit{exit\_data} parameters, see “Parameters common to all verbs” on page 22.

\textbf{rule\_array\_count}

\begin{itemize}
  \item \textbf{Direction:} Input/Output
  \item \textbf{Type:} Integer
\end{itemize}

A pointer to an integer variable containing the number of elements in the \textit{rule\_array} variable. On input, this value must be 1 or 2.

On output, the verb sets the variable to the number of \textit{rule\_array} elements it returns to the application program.

\textbf{Tip:} With this verb, the number of returned \textit{rule\_array} elements can exceed the \textit{rule\_array\_count} you specified on input. Be sure you allocate adequate memory to receive all the information elements according to the information class you select on input with the information-to-return keyword in the \textit{rule\_array}.

\textbf{rule\_array}

\begin{itemize}
  \item \textbf{Direction:} Input/Output
  \item \textbf{Type:} String array
\end{itemize}

The \textit{rule\_array} parameter is a pointer to a string variable containing an array of keywords. The keywords are eight bytes in length and must be left-aligned and padded on the right with space characters.

On input, set the \textit{rule\_array} to specify the type of information to retrieve. There are two input \textit{rule\_array} elements, as described in Table 21. This table also indicates to which parameter, \textit{rule\_array} or \textit{verb\_data}, output data is returned for each keyword.

\textit{Table 21. Keywords for Cryptographic Facility Query control information}

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Output in \textit{rule_array}</th>
<th>Output in \textit{verb_data}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adapter to use (Optional)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADAPTER1</td>
<td>This keyword is ignored. It is accepted for backward compatibility.</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Information to return (One required)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETCOMPD</td>
<td>Get compliance data. Returns official card and domain compliance information. See the SIGNSTAT keyword later in this table for a method to have the entire bundle hashed and signed by adapter firmware outbound authentication firmware keys.</td>
<td>None.</td>
<td>See “GETCOMPD” on page 117.</td>
</tr>
<tr>
<td>GET-UDX</td>
<td>Obtains UDX identifiers. This keyword applies only when using Linux on Z.</td>
<td>None.</td>
<td>See “GET-UDX” on page 117.</td>
</tr>
</tbody>
</table>
Table 21. Keywords for Cryptographic Facility Query control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Output in rule_array</th>
<th>Output in verb_data</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOREMKST</td>
<td>This keyword returns additional master key information. It is in a second rule array class and used to signify that additional data is to be returned with one or more existing keywords. This keyword is only allowed when one of STATICSFB, STATCCA, or STATCCAE is also present. Any other use results in an error return code of (8 / 0x22 [34]). When present, the reply rule array elements for the SYM and ASYM master keys have additional bytes populated. The master key status is still present in the same format in offset 0 of the element. When STATICSFB is present, an ASCII '1' is in offset 7 of the reply rule array element if ACP '0x0330' was set when the master key parts were loaded. This forces the user to enter 24-bytes of key material. This is only present for the SYM MK. Example output for STATICSFB MOREMKST (truncated to show only the SYM reply): Rule array kw 0: Card Serial Number Rule array kw 1: &quot;3 1&quot; SYM MK new mk reg is full and 24-bytes entered. Rule array kw 2: &quot;2 &quot; SYM MK valid (contains a key) current mk. Rule array kw 3: &quot;2 &quot; SYM MK valid old mk.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUM-DECT</td>
<td>Returns the number of bytes of data required for the verb_data variable when the STATDECT rule-array keyword is specified. <strong>Note:</strong> A TKE is used to securely load PIN decimalization tables.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QPENDING</td>
<td>TKE uses this rule_array keyword to request information about pending changes previously submitted by this TKE or another TKE to this adapter. Only TKE can submit changes to be stored in the Pending Change Buffer queried with this command. The keyword is available for normal users of Cryptographic Facility Query, for informational or debugging reasons (no secrets are exposed). This keyword applies only when using Linux on Z.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| SIGNSTAT | This keyword causes the returned data to have a signature post-pended to the reply data which covers the reply data. It works in combination with the GETCOMPD and STATOAHL keywords. It always returns data prefixed with a signed_data_t structure that defines where the data is (following the signed_data_t) and where an optional signature is, (after the data BLOB).  
• If SIGNSTAT is not present, the signed_data_t beginning structure is still returned, however the signature fields show offset, length, and sig-type of 0x00.  
• If SIGNSTAT is present, adapter firmware signs the data, puts a signature after the data, and fixes the signature fields to allow for the signature fields and payload. |
| SIZEWPIN | Get the number of bytes of storage required for the output of a STATWPIN request. |

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Output in rule_array</th>
<th>Output in verb_data</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM-DECT</td>
<td>Returns the number of bytes of data required for the verb_data variable when the STATDECT rule-array keyword is specified. <strong>Note:</strong> A TKE is used to securely load PIN decimalization tables.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QPENDING</td>
<td>TKE uses this rule_array keyword to request information about pending changes previously submitted by this TKE or another TKE to this adapter. Only TKE can submit changes to be stored in the Pending Change Buffer queried with this command. The keyword is available for normal users of Cryptographic Facility Query, for informational or debugging reasons (no secrets are exposed). This keyword applies only when using Linux on Z.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| SIGNSTAT | This keyword causes the returned data to have a signature post-pended to the reply data which covers the reply data. It works in combination with the GETCOMPD and STATOAHL keywords. It always returns data prefixed with a signed_data_t structure that defines where the data is (following the signed_data_t) and where an optional signature is, (after the data BLOB).  
• If SIGNSTAT is not present, the signed_data_t beginning structure is still returned, however the signature fields show offset, length, and sig-type of 0x00.  
• If SIGNSTAT is present, adapter firmware signs the data, puts a signature after the data, and fixes the signature fields to allow for the signature fields and payload. |
| SIZEWPIN | Get the number of bytes of storage required for the output of a STATWPIN request. |

**Note:** A TKE is used to securely load PIN decimalization tables. See the extension of the GETCOMPD and STATOAHL output in "GETCOMPD" on page 117 and "STATOAHL" on page 133.
### Table 21. Keywords for Cryptographic Facility Query control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Output in rule_array</th>
<th>Output in verb_data</th>
</tr>
</thead>
<tbody>
<tr>
<td>STAES</td>
<td>Obtains status information on AES master-key registers and AES key-length enablement.</td>
<td>See Table 22 on page 106</td>
<td>None.</td>
</tr>
<tr>
<td>STAPKA</td>
<td>Obtains status information on APKA master-key registers and APKA key-length enablement.</td>
<td>See Table 22 on page 106</td>
<td>None.</td>
</tr>
<tr>
<td>STATCARD</td>
<td>Obtains coprocessor-related basic status information. This keyword is provided for backwards compatibility. The STATCRD2 should be used instead of STATCARD.</td>
<td>See Table 22 on page 106</td>
<td>None.</td>
</tr>
<tr>
<td>STATCCA</td>
<td>Obtains CCA-related status information.</td>
<td>See Table 22 on page 106</td>
<td>None.</td>
</tr>
<tr>
<td>STATCCAE</td>
<td>Obtains CCA-related extended status information.</td>
<td>See Table 22 on page 106</td>
<td>None.</td>
</tr>
<tr>
<td>STATCRD2</td>
<td>Obtains extended basic status information about the coprocessor.</td>
<td>See Table 22 on page 106</td>
<td>None.</td>
</tr>
<tr>
<td>STATDECT</td>
<td>Obtains the information on all of the authorized PIN decimalization tables that are currently stored on the coprocessor. Output is returned in the verb_data variable. Note: A TKE is used to securely load PIN decimalization tables.</td>
<td>None.</td>
<td>See “STATDECT” on page 123.</td>
</tr>
<tr>
<td>STATDIAG</td>
<td>Obtains diagnostic information.</td>
<td>See Table 22 on page 106</td>
<td>None.</td>
</tr>
<tr>
<td>STATEID</td>
<td>Obtains the Environment Identifier (EID).</td>
<td>See Table 22 on page 106</td>
<td>None.</td>
</tr>
<tr>
<td>STATEXPT</td>
<td>Obtains function control vector-related status information.</td>
<td>See Table 22 on page 106</td>
<td>None.</td>
</tr>
<tr>
<td>STATICSA</td>
<td>Obtains the indicated master key hash and verification patterns to be returned for the master keys loaded in the current domain. This keyword applies only when using Linux on Z.</td>
<td>See Table 22 on page 106</td>
<td>See “STATICSA” on page 123.</td>
</tr>
<tr>
<td>STATICSB</td>
<td>Obtains the indicated master key hash and verification patterns to be returned for the master keys loaded in the current domain. See “STATICSB” on page 126.</td>
<td>See Table 22 on page 106</td>
<td>See “STATICSB” on page 126.</td>
</tr>
<tr>
<td>STATICSE</td>
<td>Obtains the indicated master key hash and verification patterns to be returned for the master keys loaded in the current domain. This keyword applies only when using Linux on Z.</td>
<td>See Table 22 on page 106</td>
<td>See “STATICSE” on page 128.</td>
</tr>
<tr>
<td>STATICSF</td>
<td>This keyword returns the adapter serial number and status information about the SYM (DES) and ASYM (RSA) master-key registers, including whether a valid key is present in each of the old, current, and new registers. This keyword applies only when using Linux on Z.</td>
<td>See Table 22 on page 106</td>
<td>None.</td>
</tr>
<tr>
<td>STATICSX</td>
<td>Obtains the indicated master key hash and verification patterns to be returned for the master keys loaded in the current domain. This keyword applies only when using Linux on Z.</td>
<td>See Table 22 on page 106</td>
<td>See “STATICSX” on page 130.</td>
</tr>
<tr>
<td>STATKPR</td>
<td>Obtains non-secret information about an operational key part. This keyword applies only when using Linux on Z.</td>
<td>None.</td>
<td>See “STATKPR” on page 132.</td>
</tr>
<tr>
<td>STATKPR</td>
<td>Obtains the names of the operational key parts. This keyword applies only when using Linux on Z.</td>
<td>None.</td>
<td>“STATKPR” on page 133.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Description</td>
<td>Output in rule_array</td>
<td>Output in verb_data</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>STATMOFN</td>
<td>Obtains master-key shares distribution information.</td>
<td>See Table 22 on page 106</td>
<td>None.</td>
</tr>
<tr>
<td>STATOAHL</td>
<td>Returns the adapter health for firmware, with full bootloader fields. This data structure matches the signed health_t structure, which is returned by the secure bootloader of the HSM when a QueryHealth command is issued. (Such a command is issued by low level IBM Z firmware which then caches the information for the management console). Thus, this STATOAHL keyword returns the same health_t structure through CCA, signed by an OA key.</td>
<td>None. See</td>
<td>STATOAHL on page 133</td>
</tr>
<tr>
<td>STATVKPL</td>
<td>Obtains the names of all the operational key parts for variable length key token preparation. This keyword applies only when using Linux on Z.</td>
<td>None. See</td>
<td>STATVKPL on page 140</td>
</tr>
<tr>
<td>STATVKPR</td>
<td>Obtains non-secret information about an operational key part. This is different from STATKPR in that a register for creating a key in a variable length key token is described. This keyword applies only when using Linux on Z.</td>
<td>None. See</td>
<td>STATVKPR on page 140</td>
</tr>
<tr>
<td>STATWPIN</td>
<td>Returns the state information on all of the weak PIN entries that are currently stored on the coprocessor.</td>
<td>None. See</td>
<td>Table 33 on page 141</td>
</tr>
<tr>
<td>TIMEDATE</td>
<td>Reads the current date, time, and day of the week from the secure clock within the coprocessor.</td>
<td>See Table 22 on page 106</td>
<td>None.</td>
</tr>
<tr>
<td>TKESTATE</td>
<td>Indicates whether TKE access is enabled or not. This keyword applies only when using Linux on Z.</td>
<td>See Table 22 on page 106</td>
<td>None.</td>
</tr>
<tr>
<td>WRAPMTHD</td>
<td>Obtains the default key wrapping method.</td>
<td>See Table 22 on page 106</td>
<td>None.</td>
</tr>
</tbody>
</table>

Different sets of rule_array elements are returned, depending on the input keyword. Table 22 on page 106 describes these rule_array elements for keywords that result in output data in the rule_array parameter.

For rule_array elements that contain numbers, those numbers are represented by numeric characters which are left-aligned and padded on the right with space characters. For example, a rule_array element that contains the number 2 contains the character string “2” (the number 2 followed by seven space characters).

For some keywords, there is output data in the verb_data variable. This output data is described in “Verb data returned for CSUACFQ keywords” on page 117.
<table>
<thead>
<tr>
<th>Element number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Output rule_array for option QPENDING</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Change type (ASCII number)</td>
<td>An ASCII number that indicates the type of pending change stored in the adapter (if there is one)</td>
</tr>
<tr>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>No pending change</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Role load</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Profile load</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Role delete</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Profile delete</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Domain zeroize</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Enable</td>
</tr>
<tr>
<td>2</td>
<td>user ID (string)</td>
<td>A string of eight ASCII characters for the user ID of the user who initiated the pending change.</td>
</tr>
<tr>
<td></td>
<td><strong>Output rule_array for option STATAES</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>AES NMK status</td>
<td>State of the AES new master key register:</td>
</tr>
<tr>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Register is clear</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Register contains a partially complete key</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Register contains a complete key</td>
</tr>
<tr>
<td>2</td>
<td>AES CMK status</td>
<td>State of the AES current master key register:</td>
</tr>
<tr>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Register is clear</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Register contains a key</td>
</tr>
<tr>
<td>3</td>
<td>AES OMK status</td>
<td>State of the AES old master key register:</td>
</tr>
<tr>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Register is clear</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Register contains a key</td>
</tr>
<tr>
<td>4</td>
<td>AES key length enablement</td>
<td>The maximum AES key length that is enabled by the function control vector. The value is 0 (if no AES key length is enabled in the function control vector (FCV)), 128, 192, or 256.</td>
</tr>
<tr>
<td></td>
<td><strong>Output rule_array for option STATAPKA</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>ECC NMK status</td>
<td>The state of the ECC new master key register:</td>
</tr>
<tr>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Register is clear</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Register contains a partially complete key</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Register contains a complete key</td>
</tr>
<tr>
<td>2</td>
<td>ECC CMK status</td>
<td>The state of the ECC current master key register:</td>
</tr>
<tr>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Register is clear</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Register contains a key</td>
</tr>
<tr>
<td>3</td>
<td>ECC OMK status</td>
<td>The state of the ECC old master key register:</td>
</tr>
<tr>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Register is clear</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Register contains a key</td>
</tr>
<tr>
<td>4</td>
<td>ECC key length enablement</td>
<td>The maximum ECC curve size that is enabled by the function control vector. The value is 0 (if no ECC keys are enabled in the function control vector (FCV)) and 521 for the maximum size.</td>
</tr>
</tbody>
</table>
### Table 22. Cryptographic Facility Query information returned in the rule_array (continued)

<table>
<thead>
<tr>
<th>Element number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Number of installed adapters</td>
<td>A numeric character string containing the number of active coprocessors installed in the machine. This includes only coprocessors that have CCA software loaded (including those with CCA UDX software). Non-CCA coprocessors are not included in this number.</td>
</tr>
<tr>
<td>2</td>
<td>DES hardware level</td>
<td>A numeric character string containing an integer value identifying the version of DES hardware on the coprocessor.</td>
</tr>
<tr>
<td>3</td>
<td>RSA hardware level</td>
<td>A numeric character string containing an integer value identifying the version of RSA hardware on the coprocessor.</td>
</tr>
<tr>
<td>4</td>
<td>POST version</td>
<td>A character string identifying the version of the coprocessor’s Power-On Self Test (POST) firmware. The first four characters define the POST0 version and the last four characters define the POST1 version.</td>
</tr>
<tr>
<td>5</td>
<td>Coprocessor operating system name</td>
<td>A character string identifying the operating system firmware on the coprocessor.</td>
</tr>
<tr>
<td>6</td>
<td>Coprocessor operating system version</td>
<td>A character string identifying the version of the coprocessor’s operating system firmware.</td>
</tr>
<tr>
<td>7</td>
<td>Coprocessor part number</td>
<td>A character string containing the 8 character part number identifying the version of the coprocessor.</td>
</tr>
<tr>
<td>8</td>
<td>Coprocessor EC level</td>
<td>A character string containing the 8 character engineering change (EC) level for this version of the coprocessor.</td>
</tr>
<tr>
<td>9</td>
<td>Miniboot version</td>
<td>A character string identifying the version of the coprocessor’s miniboot firmware. This firmware controls the loading of programs into the coprocessor. The first four characters define the MiniBoot0 version and the last four characters define the MiniBoot1 version.</td>
</tr>
<tr>
<td>10</td>
<td>CPU speed</td>
<td>A numeric character string containing the operating speed of the microprocessor chip, in megahertz.</td>
</tr>
<tr>
<td>11</td>
<td>Adapter ID (see also element number 15)</td>
<td>A unique identifier manufactured into the coprocessor. The coprocessor adapter ID is an 8-byte binary value.</td>
</tr>
<tr>
<td>12</td>
<td>Flash memory size</td>
<td>A numeric character string containing the size of the flash EPROM memory on the coprocessor, in 64 KB increments.</td>
</tr>
<tr>
<td>13</td>
<td>DRAM memory size</td>
<td>A numeric character string containing the size of the dynamic RAM (DRAM) memory on the coprocessor, in kilobytes.</td>
</tr>
<tr>
<td>14</td>
<td>Battery-backed memory size</td>
<td>A numeric character string containing the size of the battery-backed RAM on the coprocessor, in kilobytes.</td>
</tr>
<tr>
<td>15</td>
<td>Serial number</td>
<td>A character string containing the unique serial number of the coprocessor. The serial number is factory installed.</td>
</tr>
</tbody>
</table>

#### Output rule_array for option STATCCA

<table>
<thead>
<tr>
<th>Element number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NMK status</td>
<td>The state of the new master-key register:</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>The register is clear.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>The register contains a partially complete key.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>The register contains a key.</td>
<td></td>
</tr>
</tbody>
</table>
Table 22. Cryptographic Facility Query information returned in the rule_array (continued)

<table>
<thead>
<tr>
<th>Element number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>CMK status</td>
<td>The state of the current master-key register:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>OMK status</td>
<td>The state of the old master-key register:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>CCA application version</td>
<td>A character string that identifies the version of the CCA application program running in the coprocessor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the first character of the CCA application version field is a number, such as 4 or greater, then this card is a CEX3C or higher. For example, a 4 in the first character indicates a CEX3C or CEX4C. A 5 in the first character indicates a CEX5C. A 6 in the first character indicates a CEX6C.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The results of this query come directly from the card itself. If the host device driver is not up to date, it could incorrectly identify a wrong CEX*C. Therefore, looking at this field resolves all questions.</td>
</tr>
<tr>
<td>5</td>
<td>CCA application build date</td>
<td>A character string containing the build date for the CCA application program running in the coprocessor.</td>
</tr>
<tr>
<td>6</td>
<td>User role</td>
<td>A character string containing the role identifier which defines the host application user's current authority.</td>
</tr>
</tbody>
</table>

Output rule_array for option STATCCAE

<table>
<thead>
<tr>
<th>Element number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Symmetric NMK status</td>
<td>The state of the symmetric new master-key register:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Symmetric CMK status</td>
<td>The state of the symmetric current master-key register:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Symmetric OMK status</td>
<td>The state of the symmetric old master-key register:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>CCA application version</td>
<td>A character string that identifies the version of the CCA application program that is running in the coprocessor.</td>
</tr>
<tr>
<td>5</td>
<td>CCA application build date</td>
<td>A character string containing the build date for the CCA application program that is running in the coprocessor.</td>
</tr>
<tr>
<td>6</td>
<td>User role</td>
<td>A character string containing the role identifier which defines the host application user's current authority.</td>
</tr>
<tr>
<td>7</td>
<td>Asymmetric NMK status</td>
<td>The state of the asymmetric new master-key register:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
Table 22. Cryptographic Facility Query information returned in the rule_array (continued)

<table>
<thead>
<tr>
<th>Element number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Asymmetric CMK status</td>
<td>The state of the asymmetric current master-key register:</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>The register is clear.</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>The register contains a key.</td>
</tr>
<tr>
<td>9</td>
<td>Asymmetric OMK status</td>
<td>The state of the asymmetric old master-key register:</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>The register is clear.</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>The register contains a key.</td>
</tr>
</tbody>
</table>

Output rule_array for option STATCRD2

<table>
<thead>
<tr>
<th>Number of installed adapters</th>
<th>A numeric character string containing the number of active coprocessors installed in the machine. This includes only coprocessors that have CCA software loaded (including those with CCA UDX software). Non-CCA coprocessors are not included in this number.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES hardware level</td>
<td>A numeric character string containing an integer value identifying the version of DES hardware on the coprocessor.</td>
</tr>
<tr>
<td>RSA hardware level</td>
<td>A numeric character string containing an integer value identifying the version of RSA hardware on the coprocessor.</td>
</tr>
<tr>
<td>POST version</td>
<td>A character string identifying the version of the coprocessor's Power-On Self Test (POST) firmware. The first four characters define the POST0 version and the last four characters define the POST1 version.</td>
</tr>
<tr>
<td>Coprocessor operating system name</td>
<td>A character string identifying the operating system firmware on the coprocessor.</td>
</tr>
<tr>
<td>Coprocessor operating system version</td>
<td>A character string identifying the version of the coprocessor's operating system firmware.</td>
</tr>
<tr>
<td>Coprocessor part number</td>
<td>A character string containing the 8 character part number identifying the version of the coprocessor.</td>
</tr>
<tr>
<td>Coprocessor EC level</td>
<td>A character string containing the 8 character engineering change (EC) level for this version of the coprocessor.</td>
</tr>
<tr>
<td>Miniboot version</td>
<td>A character string identifying the version of the coprocessor's miniboot firmware. This firmware controls the loading of programs into the coprocessor. The first four characters define the MiniBoot0 version and the last four characters define the MiniBoot1 version.</td>
</tr>
<tr>
<td>CPU speed</td>
<td>A numeric character string containing the operating speed of the microprocessor chip, in megahertz.</td>
</tr>
<tr>
<td>Adapter ID (see also element number 15)</td>
<td>A unique identifier manufactured into the coprocessor. The coprocessor adapter ID is an 8-byte binary value.</td>
</tr>
<tr>
<td>Flash memory size</td>
<td>A numeric character string containing the size of the flash EPROM memory on the coprocessor, in 64 KB increments.</td>
</tr>
<tr>
<td>DRAM memory size</td>
<td>A numeric character string containing the size of the dynamic RAM (DRAM) memory on the coprocessor, in kilobytes.</td>
</tr>
<tr>
<td>Battery-backed memory size</td>
<td>A numeric character string containing the size of the battery-backed RAM on the coprocessor, in kilobytes.</td>
</tr>
</tbody>
</table>
### Table 22. Cryptographic Facility Query information returned in the rule_array (continued)

<table>
<thead>
<tr>
<th>Element number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Serial number</td>
<td>A character string containing the unique serial number of the coprocessor. The serial number is factory installed.</td>
</tr>
<tr>
<td>16</td>
<td>POST2 version</td>
<td>A character string identifying the version of the coprocessor's POST2 firmware. The first four characters define the POST2 version, and the last four characters are reserved and valued to space characters.</td>
</tr>
</tbody>
</table>

**Output rule_array for option STATDIAG**

<table>
<thead>
<tr>
<th>Element number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Battery state</td>
<td>A numeric character string containing a value which indicates whether the battery on the coprocessor needs to be replaced:</td>
</tr>
<tr>
<td></td>
<td>Value Description</td>
<td>Description</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1 The battery is good.</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2 The battery should be replaced.</td>
</tr>
<tr>
<td>2</td>
<td>Intrusion latch state</td>
<td>A numeric character string containing a value which indicates whether the intrusion latch on the coprocessor is set or cleared:</td>
</tr>
<tr>
<td></td>
<td>Value Description</td>
<td>Description</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1 The latch is cleared.</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2 The latch is set.</td>
</tr>
<tr>
<td>3</td>
<td>Error log status</td>
<td>A numeric character string containing a value which indicates whether there is data in the coprocessor CCA error log:</td>
</tr>
<tr>
<td></td>
<td>Value Description</td>
<td>Description</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1 The error log is empty.</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2 The error log contains abnormal termination data, but is not yet full.</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>3 The error log is full and cannot hold any more data.</td>
</tr>
<tr>
<td>4</td>
<td>Mesh intrusion</td>
<td>A numeric character string containing a value to indicate whether the coprocessor has detected tampering with the protective mesh that surrounds the secure module. This indicates a probable attempt to physically penetrate the module:</td>
</tr>
<tr>
<td></td>
<td>Value Description</td>
<td>Description</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1 No intrusion has been detected.</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2 An intrusion attempt has been detected.</td>
</tr>
<tr>
<td>5</td>
<td>Low voltage detected</td>
<td>A numeric character string containing a value to indicate whether a power-supply voltage was below the minimum acceptable level. This might indicate an attempt to attack the security module:</td>
</tr>
<tr>
<td></td>
<td>Value Description</td>
<td>Description</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1 Only acceptable voltages have been detected.</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2 A voltage has been detected below the low-voltage tamper threshold.</td>
</tr>
<tr>
<td>6</td>
<td>High voltage detected</td>
<td>A numeric character string containing a value indicates whether a power-supply voltage was greater than the maximum acceptable level. This might indicate an attempt to attack the security module:</td>
</tr>
<tr>
<td></td>
<td>Value Description</td>
<td>Description</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1 Only acceptable voltages have been detected.</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2 A voltage has been detected greater than the high-voltage tamper threshold.</td>
</tr>
<tr>
<td>7</td>
<td>Temperature range exceeded</td>
<td>A numeric character string containing a value to indicate whether the temperature in the secure module was outside of the acceptable limits. This might indicate an attempt to attack the security module:</td>
</tr>
<tr>
<td></td>
<td>Value Description</td>
<td>Description</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1 The temperature is acceptable.</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2 The temperature has been detected outside of an acceptable limit.</td>
</tr>
</tbody>
</table>
Table 22. Cryptographic Facility Query information returned in the rule_array (continued)

<table>
<thead>
<tr>
<th>Element number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Radiation detected</td>
<td>A numeric character string containing a value to indicate whether radiation was detected inside the secure module. This might indicate an attempt to attack the security module:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>9, 11, 13, 15, 17</td>
<td>Last 5 commands run</td>
<td>These five rule_array elements contain the last five commands that were run by the coprocessor CCA application. They are in chronological order, with the most recent command in element 9. Each element contains the security API command code in the first four characters and the subcommand code in the last four characters. See Table 317 on page 1135</td>
</tr>
<tr>
<td>10, 12, 14, 16, 18</td>
<td>Last 5 return codes</td>
<td>These five rule_array elements contain the security API return codes and reason codes corresponding to the five commands in rule_array elements 9, 11, 13, 15, and 17. Each element contains the return code in the first four characters and the reason code in the last four characters.</td>
</tr>
</tbody>
</table>

**Output rule_array for option STATEID**

| 1, 2 | EID | The two elements, when concatenated, provide the 16-byte Environment Identifier (EID) value. |

**Output rule_array for option STATEXPT**, when a function control vector (FCV) is loaded. Returns space characters in each of the six rule array elements if an FCV is not loaded.

| 1 | Base CCA services availability | A numeric character string containing a value to indicate whether base CCA services are available: |
|   |                               | **Value**  | **Description** |
|   |                               | 0          | Base CCA services are not available. |
|   |                               | 1          | Base CCA services are available. |

| 3 | 56-bit DES availability | A numeric character string containing a value to indicate whether 56-bit DES encryption is available: |
|   |                           | **Value**  | **Description** |
|   |                           | 0          | 56-bit DES encryption is not available. |
|   |                           | 1          | 56-bit DES encryption is available. |

| 4 | Triple-DES availability | A numeric character string containing a value to indicate whether Triple-DES encryption is available: |
|   |                           | **Value**  | **Description** |
|   |                           | 0          | Triple-DES encryption is not available. |
|   |                           | 1          | Triple-DES encryption is available. |

| 5 | SET services availability | A numeric character string containing a value to indicate whether SET (secure electronic transaction) services are available: |
|   |                           | **Value**  | **Description** |
|   |                           | 0          | SET services are not available. |
|   |                           | 1          | SET services are available. |
|   | Note: The SET services are not supported in the Linux on Z environment. |

| 6 | Maximum modulus for symmetric key encryption | A numeric character string containing the maximum modulus size enabled for the encryption of symmetric keys. This defines the longest public-key modulus that can be used for key management of symmetric-algorithm keys. |

**Output rule_array for option STATICCSA**

This keyword also has verb data returned in the verb_data field. See “Verb data returned for CSUACFQ keywords” on page 117.
Table 22. Cryptographic Facility Query information returned in the rule_array (continued)

<table>
<thead>
<tr>
<th>Element number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Card serial number</td>
<td>Eight ASCII characters for the adapter serial number</td>
</tr>
<tr>
<td>2</td>
<td>DES new master-key register state</td>
<td>An ASCII number showing the state of the DES new master-key register:</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Empty</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Partially full</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Full</td>
</tr>
<tr>
<td>3</td>
<td>DES current master-key register state</td>
<td>An ASCII number showing the state of the DES current master-key register:</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Valid</td>
</tr>
<tr>
<td>4</td>
<td>DES old master-key register state</td>
<td>An ASCII number showing the state of the DES old master-key register:</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Valid</td>
</tr>
<tr>
<td>5</td>
<td>PKA new master-key register state</td>
<td>An ASCII number showing the state of the PKA new master-key register:</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Empty</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Partially full</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Full</td>
</tr>
<tr>
<td>6</td>
<td>PKA current master-key register state</td>
<td>An ASCII number showing the state of the PKA current master-key register:</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Valid</td>
</tr>
<tr>
<td>7</td>
<td>PKA old master-key register state</td>
<td>An ASCII number showing the state of the PKA old master-key register:</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Valid</td>
</tr>
<tr>
<td>8</td>
<td>AES new master-key register state</td>
<td>An ASCII number showing the state of the AES new master-key register:</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Empty</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Partially full</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Full</td>
</tr>
<tr>
<td>9</td>
<td>AES current master-key register state</td>
<td>An ASCII number showing the state of the AES current master-key register:</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Valid</td>
</tr>
<tr>
<td>10</td>
<td>AES old master-key register state</td>
<td>An ASCII number showing the state of the AES old master-key register:</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Valid</td>
</tr>
</tbody>
</table>

Output rule_array for option STATICSB

This keyword also has verb data returned in the verb_data field. See “Verb data returned for CSUACFQ keywords” on page 117.

<table>
<thead>
<tr>
<th>Element number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Card serial number</td>
<td>Eight ASCII characters for the adapter serial number</td>
</tr>
</tbody>
</table>
### Table 22. Cryptographic Facility Query information returned in the rule_array (continued)

<table>
<thead>
<tr>
<th>Element number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>DES new master-key register state</td>
<td>An ASCII number showing the state of the DES new master-key register: Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value 1 Empty</td>
</tr>
<tr>
<td>3</td>
<td>DES current master-key register state</td>
<td>An ASCII number showing the state of the DES current master-key register: Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value 1 Invalid</td>
</tr>
<tr>
<td>4</td>
<td>DES old master-key register state</td>
<td>An ASCII number showing the state of the DES old master-key register: Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value 1 Invalid</td>
</tr>
<tr>
<td>5</td>
<td>PKA new master-key register state</td>
<td>An ASCII number showing the state of the PKA new master-key register: Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value 1 Empty</td>
</tr>
<tr>
<td>6</td>
<td>PKA current master-key register state</td>
<td>An ASCII number showing the state of the PKA current master-key register: Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value 1 Invalid</td>
</tr>
<tr>
<td>7</td>
<td>PKA old master-key register state</td>
<td>An ASCII number showing the state of the PKA old master-key register: Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value 1 Invalid</td>
</tr>
<tr>
<td>8</td>
<td>APKA new master-key register state</td>
<td>An ASCII number showing the state of the APKA new master-key register: Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value 1 Empty</td>
</tr>
<tr>
<td>9</td>
<td>APKA current master-key register state</td>
<td>An ASCII number showing the state of the APKA current master-key register: Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value 1 Invalid</td>
</tr>
<tr>
<td>10</td>
<td>APKA old master-key register state</td>
<td>An ASCII number showing the state of the APKA old master-key register: Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value 1 Invalid</td>
</tr>
</tbody>
</table>

**Output rule_array for option STATICSE**

This keyword also has verb data returned in the verb_data field. See "Verb data returned for CSUACFQ keywords" on page 117.

<table>
<thead>
<tr>
<th>Element number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Card serial number</td>
<td>Eight ASCII characters for the adapter serial number</td>
</tr>
</tbody>
</table>
### Table 22. Cryptographic Facility Query information returned in the rule_array (continued)

<table>
<thead>
<tr>
<th>Element number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>DES new master-key register state</td>
<td>An ASCII number showing the state of the DES new master-key register:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

| 3              | DES current master-key register state | An ASCII number showing the state of the DES current master-key register: |
|                |      | Value | Description |
|                |      | 1     | Invalid      |
|                |      | 2     | Valid        |

| 4              | DES old master-key register state | An ASCII number showing the state of the DES old master-key register: |
|                |      | Value | Description |
|                |      | 1     | Invalid      |
|                |      | 2     | Valid        |

| 5              | PKA new master-key register state | An ASCII number showing the state of the PKA new master-key register: |
|                |      | Value | Description |
|                |      | 1     | Empty        |
|                |      | 2     | Partially full |
|                |      | 3     | Full         |

| 6              | PKA current master-key register state | An ASCII number showing the state of the PKA current master-key register: |
|                |      | Value | Description |
|                |      | 1     | Invalid      |
|                |      | 2     | Valid        |

| 7              | PKA old master-key register state | An ASCII number showing the state of the PKA old master-key register: |
|                |      | Value | Description |
|                |      | 1     | Invalid      |
|                |      | 2     | Valid        |

### Output rule_array for option STATICSF

| 1              | Card serial number | Eight ASCII characters for the adapter serial number |
| 2              | DES new master-key register state | An ASCII number showing the state of the DES new master-key register: |
|                |      | Value | Description |
|                |      | 1     | Empty        |
|                |      | 2     | Partially full |
|                |      | 3     | Full         |

| 3              | DES current master-key register state | An ASCII number showing the state of the DES current master-key register: |
|                |      | Value | Description |
|                |      | 1     | Invalid      |
|                |      | 2     | Valid        |

| 4              | DES old master-key register state | An ASCII number showing the state of the DES old master-key register: |
|                |      | Value | Description |
|                |      | 1     | Invalid      |
|                |      | 2     | Valid        |
Table 22. Cryptographic Facility Query information returned in the rule_array (continued)

<table>
<thead>
<tr>
<th>Element number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>PKA new master-key register state</td>
<td>An ASCII number showing the state of the PKA new master-key register:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> <strong>Description</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Empty</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 Partially full</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 Full</td>
</tr>
<tr>
<td>6</td>
<td>PKA current master-key register state</td>
<td>An ASCII number showing the state of the PKA current master-key register:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> <strong>Description</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 Valid</td>
</tr>
<tr>
<td>7</td>
<td>PKA old master-key register state</td>
<td>An ASCII number showing the state of the PKA old master-key register:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> <strong>Description</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 Valid</td>
</tr>
</tbody>
</table>

Output rule_array for option STATICSX

This keyword also has verb data returned in the verb_data field. See “Verb data returned for CSUACFQ keywords” on page 117.

<p>| 1              | Card serial number                        | Eight ASCII characters for the adapter serial number                        |
| 2              | DES new master-key register state         | An ASCII number showing the state of the DES new master-key register:       |
|                |                                           | <strong>Value</strong> <strong>Description</strong>                                                    |
|                |                                           | 1 Empty                                                                      |
|                |                                           | 2 Partially full                                                            |
|                |                                           | 3 Full                                                                       |
| 3              | DES current master-key register state     | An ASCII number showing the state of the DES current master-key register:   |
|                |                                           | <strong>Value</strong> <strong>Description</strong>                                                    |
|                |                                           | 1 Invalid                                                                    |
|                |                                           | 2 Valid                                                                      |
| 4              | DES old master-key register state         | An ASCII number showing the state of the DES old master-key register:       |
|                |                                           | <strong>Value</strong> <strong>Description</strong>                                                    |
|                |                                           | 1 Invalid                                                                    |
|                |                                           | 2 Valid                                                                      |
| 5              | PKA new master-key register state         | An ASCII number showing the state of the PKA new master-key register:       |
|                |                                           | <strong>Value</strong> <strong>Description</strong>                                                    |
|                |                                           | 1 Empty                                                                      |
|                |                                           | 2 Partially full                                                            |
|                |                                           | 3 Full                                                                       |
| 6              | PKA current master-key register state     | An ASCII number showing the state of the PKA current master-key register:   |
|                |                                           | <strong>Value</strong> <strong>Description</strong>                                                    |
|                |                                           | 1 Invalid                                                                    |
|                |                                           | 2 Valid                                                                      |
| 7              | PKA old master-key register state         | An ASCII number showing the state of the PKA old master-key register:       |
|                |                                           | <strong>Value</strong> <strong>Description</strong>                                                    |
|                |                                           | 1 Invalid                                                                    |
|                |                                           | 2 Valid                                                                      |</p>
<table>
<thead>
<tr>
<th>Element number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Output rule_array for option STATMOFN</strong></td>
<td></td>
</tr>
<tr>
<td>Elements 1 and 2 are treated as a 16-byte string, as are elements 3 and 4, with the high-order 15 bytes containing meaningful information and the 16th byte containing a space character. Each byte provides status information about the i\textsuperscript{th} share, 1 ≤ i ≤ 15, of master-key information.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1, 2</td>
<td>Master-key shares generation</td>
<td>The 15 individual bytes are set to one of these character values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>3, 4</td>
<td>Master-key shares reception</td>
<td>The 15 individual bytes are set to one of these character values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>m</td>
<td>The minimum number of shares required to instantiate a master key through the master-key-shares process. The value is returned in two characters, valued from 01 – 15, followed by six space characters.</td>
</tr>
<tr>
<td>6</td>
<td>n</td>
<td>The maximum number of distinct shares involved in the master-key-shares process. The value is returned in two characters, valued from 01 - 15, followed by six space characters.</td>
</tr>
<tr>
<td></td>
<td><strong>Output rule_array for option TIMEDATE</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Date</td>
<td>The current date is returned as a character string of the form YYYYMMDD, where:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYYY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DD</td>
</tr>
<tr>
<td>2</td>
<td>Time</td>
<td>The current UTC time of day is returned as a character string of the form HHMMSS, where:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SS</td>
</tr>
<tr>
<td>3</td>
<td>Day of the week</td>
<td>The day of the week is returned as a number between 1 (Sunday) and 7 (Saturday).</td>
</tr>
<tr>
<td></td>
<td><strong>Output rule_array for option TKESTATE</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>TKE access enabled</td>
<td>Indicates whether a TKE can be used to administer this CEX*C. Values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TKEPERM</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TKEDENY</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Output rule_array for option WRAPMTHD</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Internal tokens</td>
<td>Default wrapping method for internal tokens.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Table 22. Cryptographic Facility Query information returned in the rule_array (continued)

<table>
<thead>
<tr>
<th>Element number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>External tokens</td>
<td>Default wrapping method for external tokens.</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Keys are wrapped with the original method.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Keys are wrapped with the enhanced X9.24 method.</td>
</tr>
</tbody>
</table>

verb_data_length

Direction: Input/Output
Type: Integer

The verb_data_length parameter is a pointer to an integer variable containing the number of bytes of data in the verb-data variable.

verb_data

Direction: Input/Output
Type: String

A pointer to a string variable containing output data that is returned for some of the keywords specified in the rule_array variable. This output is described in a separate section for each keyword in "Verb data returned for CSUACFQ keywords."

Verb data returned for CSUACFQ keywords

Some keywords return specific data in the verb_data parameter, and update the verb_data_length field with the count of bytes returned.

The verb_data buffer must be large enough to receive the data (see keyword-specific sizes below) and the verb_data_length parameter as passed in to Cryptographic Facility Query (CSUACFQ) must indicate that size (or a larger value). If either the verb_data or verb_data_length fields are not valid, no data is returned at all. In this case, a return code of 8 and a reason code of 72 is returned.

GET-UDX

This rule_array keyword causes a variable length list of 2-byte UDX identifiers to be returned.

The identifiers represent the authorized UDX verb IDs for the adapter. A UDX is a set of one or more custom CCA APIs added to the adapter, using the installable code feature. Unless the programming source has also provided an updated host library, these UDX calls are not accessible from the IBM Z Linux host library. If an updated host library is provided, refer to the accompanying documentation for usage.

The maximum number of names to be returned is 100. Using this number, the maximum size buffer is 6400 bytes.

GETCOMPD

This rule_array keyword causes all data to be returned in one VUD block. The described structure also comprises extra data fields that contains output if the SIGNSTAT keyword is specified.

The VUD block header and data has a layout as described in Table 23 on page 118.
### Table 23. Output data format for the GETCOMPD keyword

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
<th>Subordinate data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Begin outer templating</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>Struct name</td>
<td>Signed_data_t</td>
<td>Structure name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Value: ( \text{mbif} _\text{types}.h:118:#define SIGNED_DATA_T 0x82 )</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>Struct version</td>
<td>Signed_data_t</td>
<td>Structure version</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Value: 0x00</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>Signed data len</td>
<td>Signed_data_t</td>
<td>Length of entire Signed_data_t structure, including appended data and signature</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>Data offset</td>
<td>Signed_data_t</td>
<td>Offset from start of the data sub-struct (here) to the start of the actual data payload.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Value: 0x14</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>Data len</td>
<td>Signed_data_t</td>
<td>Length of the data payload</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>Sig offset</td>
<td>Signed_data_t</td>
<td>Offset from start of the signature sub-structure (here) to the start of the actual signature.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian</td>
</tr>
<tr>
<td>22</td>
<td>4</td>
<td>Sig len</td>
<td>Signed_data_t</td>
<td>Length of the signature</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian</td>
</tr>
<tr>
<td>26</td>
<td>4</td>
<td>Sig type</td>
<td>Signed_data_t</td>
<td>Type of the signature</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Value: 0x04 (ECDSA over SHA-512 hash with ECC P521, raw format)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x00 (no signature)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Begin payload</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>7</td>
<td>VE</td>
<td>xcVpd_t:VE field</td>
<td>Card secure part number field from VPD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: ASCII, not NULL terminated</td>
</tr>
<tr>
<td>37</td>
<td>1</td>
<td>Reserved1</td>
<td>n/a</td>
<td>Reserved field</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer</td>
</tr>
<tr>
<td>38</td>
<td>7</td>
<td>EC</td>
<td>xcVpd_t:EC field</td>
<td>Card EC field from VPD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: ASCII, not NULL terminated</td>
</tr>
<tr>
<td>45</td>
<td>1</td>
<td>Reserved1</td>
<td>n/a</td>
<td>Reserved field</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer</td>
</tr>
<tr>
<td>46</td>
<td>12</td>
<td>SN</td>
<td>xcVpd_t:[sn_hdr | sn] fields</td>
<td>Card serial number header and serial number</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>concatenated; making up a 12 byte quantity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: ASCII, not NULL terminated</td>
</tr>
</tbody>
</table>

**Cryptographic Facility Query (CSUACFQ)**
Table 23. Output data format for the GETCOMPD keyword (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
<th>Subordinate data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>16</td>
<td>Current_clock</td>
<td>Cca_gentime_t structure</td>
<td>The current card clock time maintained by the RTC hardware. Type: ASCII characters for the date in the following layout of 14 characters, with 2 NULL characters at the end since the day-of-week is not returned. YYYYMMDDHHMMSS\0\0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Note:</strong> This is similar to ASN.1 Generalized Time format for <strong>Local time</strong>. Since the adapter reports whatever time it has been set to, and has no external reference for timezone verification, it is not appropriate to have a Z indicating UTC or to specify an offset from UTC.</td>
</tr>
<tr>
<td>74</td>
<td>8</td>
<td>Cca_version</td>
<td></td>
<td>Same value reported for other CSUACFQ calls that report the CCA version. Type: ASCII, not NULL terminated</td>
</tr>
<tr>
<td>82</td>
<td>8</td>
<td>Udx_version1</td>
<td>Ccax_version1</td>
<td>UDX supplied version field (first), should be <em>none</em> for PCI-HSM 2016 capable firmware. Type: ASCII, not NULL terminated</td>
</tr>
<tr>
<td>90</td>
<td>8</td>
<td>Udx_version2</td>
<td>Ccax_version2</td>
<td>UDX supplied version field (second), should be <em>none</em> for PCI-HSM 2016 capable firmware. Type: ASCII, not NULL terminated</td>
</tr>
<tr>
<td>98</td>
<td>16</td>
<td>Build_date</td>
<td>Cca_gentime_t structure</td>
<td>Build time-date: Local Date and time on machine where firmware was built. Type: ASCII characters for the date in the following layout of 14 characters, with 2 NULL characters at the end since the day-of-week is not returned. YYYYMMDDHHMMSS\0\0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Note:</strong> This is similar to ASN.1 Generalized Time format for “Local time”. Since the adapter reports whatever time it has been set to, and has no external reference for timezone verification, it is not appropriate to have a Z indicating UTC or to specify an offset from UTC.</td>
</tr>
</tbody>
</table>
## Table 23. Output data format for the GETCOMPD keyword (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
<th>Subordinate data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>114</td>
<td>4</td>
<td>Card_action</td>
<td>cmp_srdi_hdr_t:action_flags</td>
<td>Card scope action flags</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1. CARD_ZEROIZE_START 0x8000_0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a. Flag Name: Card-wide zeroize started</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b. one bit flag to indicate that card zeroize is starting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>c. Default value is 0b0, which indicates that a card zeroize is NOT in progress</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2. CARD_CLOCK_SET 0x4000_0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a. Flag Name: Card-wide system clock has been set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b. one bit flag to indicate if the card has seen a valid SETCLOCK operation (from the TKE) in the time covered by the current SRDI files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>c. Set once and never un-set or zeroed except for card-scope zeroize events.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>d. Default value: 0b0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3. RESERVED (all bits not defined above)</td>
</tr>
<tr>
<td>118</td>
<td>4</td>
<td>Comp_issues</td>
<td>DRAM variable</td>
<td>Returns any issues with compliance: reasons the card cannot support a compliance setting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CCA_comp_issue_flags</td>
<td>Type: binary integer, big-endian</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Flags are defined:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1. CMPIF_CERT_NO_ISSUES 0x00000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2. CMPIF_FW_UDX 0x80000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This value indicates a UDX was detected</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3. CMPIF_FW_SIM 0x40000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This value indicates code is a simulator</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4. Remaining values: reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Default value: 0b0</td>
</tr>
<tr>
<td>122</td>
<td>4</td>
<td>Sec_log_max</td>
<td>Not in a structure</td>
<td>Maximum count of events for this domain, this is not a byte count. This is the same for the life of a card, but the API will live beyond one card.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian</td>
</tr>
<tr>
<td>126</td>
<td>2</td>
<td>Sec_log_event _size</td>
<td>Not in a structure</td>
<td>Maximum size of one event in bytes. This is the same for the life of a card, but the API will live beyond one card.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian</td>
</tr>
<tr>
<td>128</td>
<td>2</td>
<td>Dmn_kdf</td>
<td>Kdf value from DRAM</td>
<td>KDF value that maps to the compliance flags.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian</td>
</tr>
<tr>
<td>130</td>
<td>4</td>
<td>Dmn_action</td>
<td>cmp_srdi_mbr_t:action_flags</td>
<td>Domain scope action flags.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian</td>
</tr>
</tbody>
</table>

(description continued in next row)
Table 23. Output data format for the GETCOMPD keyword (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
<th>Subordinate data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Description continued for Dmn_action:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1. DOMAIN_ZERO_START 0x8000_0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a. Flag Name: domain-scope zeroize has started</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b. one bit flag to indicate the beginning of work to zero-ize a domain</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>c. Default value is 0b0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2. DOMAIN_IMPRINT_START 0x4000_0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a. Flag Name: This domain has started transition to Imprint mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b. one bit flag to indicate start of internal preparation for imprint mode transition.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>c. Default value is 0b0, which indicates a zero-ize is NOT in progress</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3. DOMAIN_IMPRINT_ACTIVE 0x2000_0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a. Flag Name: This domain is in Imprint mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b. one bit flag to indicate domain is in imprint mode for the compliance mode indicated by the compliance flags</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>c. Default value is 0b0, which indicates the domain is NOT in imprint mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4. DOMAIN_COMP_ACTIVE 0x1000_0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a. Flag Name: This domain has 1 or more compliance modes active</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b. one bit flag to indicate the compliance flags field is in force</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>c. Default value is 0b0, which indicates the compliance flags field is not in force</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5. DOMAIN_COMPREMOVE_START 0x0800_0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a. Flag Name: This domain has started removal of 1 or more compliance modes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b. one bit flag to indicate the beginning of work to un-set a compliance mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>c. Default value is 0b0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6. DOMAIN_COMP_MIGRATION 0x0400_0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a. Flag Name: This domain is in migration mode as a reduced mode of an active compliance mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b. one bit flag to indicate that migration mode is active.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>c. In this mode Comp-tagged keys may be created from qualified non-tagged keys.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>d. If this bit is active then the domain is within the 30 minute inactivity timeout window. After 30 minutes of not receiving any migration work (applying a comp-tag to a non-comp-tag key) then the domain will revert to compliance mode and the flag is automatically turned to 0b0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>e. Default value is 0b0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7. (reserved) 0x0200_0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a. Flag is reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b. Default value is 0b0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8. DOMAIN_SLOG_ENABLE 0x0000_8000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a. Flag Name: This domain has Secure Log enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b. one bit flag to indicate that Secure Log is enabled for this domain</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>c. This flag will be 0b1 for every case where DOMAIN_IMPRINT_ACTIVE flag or DOMAIN_COMP_ACTIVE flag are active, however this flag may be 0b1 when neither flag is active.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This means that a domain can have Secure Log active even if it is not in imprint mode or compliant mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>d. Default value is 0b0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9. DOMAIN_SLOG_NOWRAP 0x0000_4000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a. Flag Name: This domain has Secure Log configured for NOT WRAP when the Log fills</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b. one bit flag to indicate that Secure Log is configured to NOT WRAP for this domain.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>c. This flag is meaningful only if the DOMAIN_SLOG_ENABLE flag is 0b1, all other times it is set to 0b0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>d. Similar to DOMAIN_SLOG_ENABLE, this flag will be 0b1 for every case where DOMAIN_IMPRINT_ACTIVE flag or DOMAIN_COMP_ACTIVE flag are active, however this flag may be 0b1 when neither flag is active. This means that No Wrap is a required feature of the Secure Log if the domain is in Imprint mode or Compliant Mode, and No Wrap is an optional feature of the Secure Log otherwise.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>e. Default value is 0b0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10. (Reserved) all remaining bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Default value 0b0</td>
</tr>
</tbody>
</table>
**Cryptographic Facility Query (CSUACFQ)**

Table 23. Output data format for the GETCOMPD keyword (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
<th>Subordinate data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>134</td>
<td>4</td>
<td>Dmn_compl</td>
<td>cmp_srdi_mbr_t:compl_flags</td>
<td>Domain scope compliance flags</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian (description continued in next row)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Description continued for Dmn_compl:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1. COMPF_PCI_HSM_2016 0x8000_0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a. Flag Name: This domain has the PCI-HSM v3.0 (June 2016) mode active</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b. one bit flag to indicate domain has PCI_HSM_2016 enabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>c. Default value is 0b0, which indicates PHM is <em>disabled</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2. (Reserved) all remaining bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Default value 0b0</td>
</tr>
<tr>
<td>138</td>
<td>4</td>
<td>Sec_log_cnt</td>
<td>Not in a structure</td>
<td>Current count of events for this domain, this is not a byte count.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Will be 0x00000000 if Secure Log is not enabled at domain scope.</td>
</tr>
<tr>
<td>142</td>
<td>2</td>
<td>Owner2</td>
<td>Rom_status_t:owner2</td>
<td>Type: binary integer, big-endian two byte owner-ID field.</td>
</tr>
<tr>
<td>144</td>
<td>2</td>
<td>Owner3</td>
<td>Rom_status_t:owner3</td>
<td>Type: binary integer, big-endian two byte owner-ID field.</td>
</tr>
<tr>
<td>146</td>
<td>4</td>
<td>Miniboot versions</td>
<td>xcMB_Version_t</td>
<td>Type: binary integer, big-endian two bytes for miniboot 0, two bytes for miniboot 1</td>
</tr>
<tr>
<td>150</td>
<td>4</td>
<td>Adapter type</td>
<td>Adapter_type: CardRevId from xcAdapterInfo_t</td>
<td>Type: binary integer, big-endian four byte field from adapter info</td>
</tr>
</tbody>
</table>

Begin signature section sig-section (if SIGNSTAT passed)

<table>
<thead>
<tr>
<th>X</th>
<th>signature</th>
<th>Raw r and s values for</th>
<th>Length X is given in signed_data_tsig len field at the beginning of returned data.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>signature over SHA-512 hash of the payload</td>
<td>X will always be 132 bytes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>r and s are 66 bytes each, r is first.</td>
</tr>
<tr>
<td>64</td>
<td>Payload hash</td>
<td>Raw hash value</td>
<td>Raw SHA-512 hash over the payload. This is the value used for calculating the signature.</td>
</tr>
</tbody>
</table>

**NUM-DECT**

This *rule_array* keyword causes a 4-byte binary number to be returned.

This is the number of bytes required for the *verb_data* variable when the STATDECT rule-array keyword is specified.

**SIZEWPIN**

For the CSUACFQ verb, the SIZEWPIN and STATWPIN keywords are added to the rule array to query the size and to obtain the state information of the weak PIN table. The SIZEWPIN *rule_array* keyword returns the size of the weak PIN table.
Table 24. Output data format for the SIZEWPIN keyword

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length in bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>weak_PIN_table_length</td>
<td>4</td>
<td>Integer value of the number of bytes of data that the verb.data requires when the STATWPIN rule-array keyword is specified.</td>
</tr>
</tbody>
</table>

**STATDECT**

This rule_array keyword causes a table of up to 100 PIN decimalization tables to be returned.

Table 25 shows the data format for a decimalization table.

Table 25. Output data format for the STATDECT keyword

<table>
<thead>
<tr>
<th>Field</th>
<th>Length in bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>3</td>
<td>PIN decimalization table identifier in ASCII digits 001 - 100 (X’303031’ - X’313030’).</td>
</tr>
</tbody>
</table>
| state   | 1               | Table state in ASCII:
|         |                 | Code   Description |
|         |                 | A (X’41’)  Active                                      |
|         |                 | L (X’4C’)  Loaded                                      |
| table   | 16              | PIN decimalization table. Contains ASCII digits 0 - 9 (X’30’ - X’39’), in the clear. |
| Total byte count | Depends on the number of returned decimalization tables. There are 20 byte for each decimalization table. The total byte count is returned for the NUM-DECT keyword. |

**STATICSA**

This rule_array keyword causes the indicated master key hash and verification patterns to be returned for the master keys loaded in the current domain.

The status variables for the various master key registers returned in the rule_array will indicate which of these verification pattern structures returned contain useful data. An empty master key register cannot have a meaningful verification pattern. However, the data structures are returned for all registers indicated, so that interpretation is reliable.

The output data format for STATICSA operational key parts is given in Table 26 on page 124.

**Note:**

1. The fields will be returned in the order given, however the *_ID fields should be used for verification.
2. The verb.data_length parameter will indicate the total size at the bottom of the table describing the verb.data.
3. Multiple byte fields are stored in Big-Endian format, as is typical for CEX*C communication.
## Table 26. Output data format for STATICSA operational key parts

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length in bytes</th>
<th>Field value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYM_OMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>SYM_OMK_MDC4_ID</td>
<td>2</td>
<td>X'0F02'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>SYM_OMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Symmetric Key old master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>SYM_CMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>SYM_CMK_MDC4_ID</td>
<td>2</td>
<td>X'0F01'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>SYM_CMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Symmetric Key current master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>SYM_NMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>SYM_NMK_MDC4_ID</td>
<td>2</td>
<td>X'0F00'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>SYM_NMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Symmetric Key new master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>ASYM_OMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>ASYM_OMK_MDC4_ID</td>
<td>2</td>
<td>X'0F05'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>ASYM_OMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Asymmetric Key old master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>ASYM_CMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>ASYM_CMK_MDC4_ID</td>
<td>2</td>
<td>X'0F04'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>ASYM_CMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Asymmetric Key current master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>ASYM_NMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>ASYM_NMK_MDC4_ID</td>
<td>2</td>
<td>X'0F03'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>ASYM_NMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Asymmetric Key new master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>SYM_OMK_VP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>SYM_OMK_VP_ID</td>
<td>2</td>
<td>X'0F08'</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
</tbody>
</table>
Table 26. Output data format for STATICSA operational key parts (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length in bytes</th>
<th>Field value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYM_OMK_VP</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the Symmetric Key old master-key register calculated using the default algorithm.</td>
</tr>
<tr>
<td>SYM_CMK_VF_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>SYM_CMK_VF_ID</td>
<td>2</td>
<td>X'0F07'</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>SYM_CMK_VF</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the Symmetric Key current master-key register, calculated using the default algorithm.</td>
</tr>
<tr>
<td>SYM_NMK_VF_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>SYM_NMK_VF_ID</td>
<td>2</td>
<td>X'0F06'</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>SYM_NMK_VF</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the Symmetric Key new master-key register, calculated using the default algorithm.</td>
</tr>
<tr>
<td>SYM_NMK_MKAP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Authentication pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Authentication pattern).</td>
</tr>
<tr>
<td>SYM_NMK_MKAP_ID</td>
<td>2</td>
<td>X'0F09'</td>
<td>Hexadecimal identifier indicating the contents of the following Authentication pattern field.</td>
</tr>
<tr>
<td>SYM_NMK_MKAP</td>
<td>8</td>
<td>variable</td>
<td>Authentication pattern over the Symmetric Key new master-key register, calculated using the ICSF specified algorithm.</td>
</tr>
<tr>
<td>AES_OMK_VF_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>AES_OMK_VF_ID</td>
<td>2</td>
<td>X'0F0C'</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>AES_OMK_VF</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the AES Key old master-key register, calculated using the SHA-256 algorithm.</td>
</tr>
<tr>
<td>AES_CMK_VF_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>AES_CMK_VF_ID</td>
<td>2</td>
<td>X'0F0B'</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>AES_CMK_VF</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the AES Key current master-key register calculated using the SHA-256 algorithm.</td>
</tr>
<tr>
<td>AES_NMK_VF_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>AES_NMK_VF_ID</td>
<td>2</td>
<td>X'0F0A'</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>AES_NMK_VF</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the AES Key new master-key register, calculated using the SHA-256 algorithm.</td>
</tr>
<tr>
<td>Total byte count</td>
<td></td>
<td>204</td>
<td></td>
</tr>
</tbody>
</table>
STATICSB
This rule_array keyword causes the indicated master key hash and verification patterns to be returned for the master keys loaded in the current domain.

The status variables for the various master-key registers returned in the rule_array indicate which of these verification pattern structures returned contain useful data. An empty master-key register cannot have a meaningful verification pattern. However, the data structures are returned for all registers indicated, so that interpretation is reliable.

The output data format for STATICSB operational key parts is given in Table 27.

Note:
1. The fields will be returned in the order given, however the *_ID fields should be used for verification.
2. The verb_data_length parameter indicates the total size at the bottom of the table describing the verb_data.
3. Multiple byte fields are stored in Big-Endian format, as is typical for CEX*C communication.

Table 27. Output data format for STATICSB operational key parts

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length in bytes</th>
<th>Field value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYM_OMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>SYM_OMK_MDC4_ID</td>
<td>2</td>
<td>X'0F02'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>SYM_OMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Symmetric Key old master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>SYM_CMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>SYM_CMK_MDC4_ID</td>
<td>2</td>
<td>X'0F01'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>SYM_CMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Symmetric Key current master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>SYM_NMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>SYM_NMK_MDC4_ID</td>
<td>2</td>
<td>X'0F00'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>SYM_NMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Symmetric Key new master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>ASYM_OMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>ASYM_OMK_MDC4_ID</td>
<td>2</td>
<td>X'0F05'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>ASYM_OMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Asymmetric Key old master-key register, calculated using the MDC4 algorithm.</td>
</tr>
</tbody>
</table>
Table 27. Output data format for STATICSB operational key parts (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length in bytes</th>
<th>Field value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASYM_CMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the <code>verb_data</code> (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>ASYM_CMK_MDC4_ID</td>
<td>2</td>
<td>X'0F04'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>ASYM_CMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Asymmetric Key current master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>ASYM_NMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the <code>verb_data</code> (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>ASYM_NMK_MDC4_ID</td>
<td>2</td>
<td>X'0F03'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>ASYM_NMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Asymmetric Key new master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>SYM_OMK_VP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the <code>verb_data</code> (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>SYM_OMK_VP_ID</td>
<td>2</td>
<td>X'0F08'</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>SYM_OMK_VP</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the Symmetric Key old master-key register calculated using the default algorithm.</td>
</tr>
<tr>
<td>SYM_CMK_VP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the <code>verb_data</code> (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>SYM_CMK_VP_ID</td>
<td>2</td>
<td>X'0F07'</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>SYM_CMK_VP</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the Symmetric Key current master-key register, calculated using the default algorithm.</td>
</tr>
<tr>
<td>SYM_NMK_VP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the <code>verb_data</code> (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>SYM_NMK_VP_ID</td>
<td>2</td>
<td>X'0F06'</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>SYM_NMK_VP</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the Symmetric Key new master-key register, calculated using the default algorithm.</td>
</tr>
<tr>
<td>SYM_NMK_MKAP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Authentication pattern block in the <code>verb_data</code> (comprising this length field, the following ID field, and the field for the Authentication pattern).</td>
</tr>
<tr>
<td>SYM_NMK_MKAP_ID</td>
<td>2</td>
<td>X'0F09'</td>
<td>Hexadecimal identifier indicating the contents of the following Authentication pattern field.</td>
</tr>
<tr>
<td>SYM_NMK_MKAP</td>
<td>8</td>
<td>variable</td>
<td>Authentication pattern over the Symmetric Key new master-key register, calculated using the ICSF specified algorithm.</td>
</tr>
<tr>
<td>AES_OMK_VP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the <code>verb_data</code> (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>AES_OMK_VP_ID</td>
<td>2</td>
<td>X'0F0C'</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
</tbody>
</table>
## Table 27. Output data format for STATICSB operational key parts (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length in bytes</th>
<th>Field value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES_OMK_VP</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the Symmetric Key old master-key register, calculated using the SHA-256 algorithm.</td>
</tr>
<tr>
<td>AES_CMK_VP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>AES_CMK_VP_ID</td>
<td>2</td>
<td>X'0F0B'</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>AES_CMK_VP</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the Symmetric Key current master-key register, calculated using the SHA-256 algorithm.</td>
</tr>
<tr>
<td>AES_NMK_VP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Authentication pattern).</td>
</tr>
<tr>
<td>AES_NMK_VP_ID</td>
<td>2</td>
<td>X'0F0A'</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>AES_NMK_VP</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the Symmetric Key new master-key register, calculated using the SHA-256 algorithm.</td>
</tr>
<tr>
<td>APKA_OMK_VP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>APKA_OMK_VP_ID</td>
<td>2</td>
<td>X'0F0F'</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>APKA_OMK_VP</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the Symmetric Key old master-key register, calculated using the SHA-256 algorithm.</td>
</tr>
<tr>
<td>APKA_CMK_VP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>APKA_CMK_VP_ID</td>
<td>2</td>
<td>X'0F0E'</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>APKA_CMK_VP</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the Symmetric Key current master-key register, calculated using the SHA-256 algorithm.</td>
</tr>
<tr>
<td>APKA_NMK_VP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Authentication pattern).</td>
</tr>
<tr>
<td>APKA_NMK_VP_ID</td>
<td>2</td>
<td>X'0F0D'</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>APKA_NMK_VP</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the Symmetric Key new master-key register, calculated using the SHA-256 algorithm.</td>
</tr>
</tbody>
</table>

**Total byte count** 240

This keyword was introduced with CCA 4.1.0.

### STATICSE

This **rule_array** keyword returns the indicated master key hash and verification patterns for the master keys loaded in the current domain.

The status variables for the various master-key registers returned in the **rule_array** indicate which of these verification pattern structures returned contain useful data.
An empty master-key register cannot have a meaningful verification pattern. However, the data structures are returned for all registers indicated, so that interpretation is reliable.

The output data format for STATICSE operational key parts is given in Table 28.

**Note:**
1. The fields will be returned in the order given, however the *_ID fields should be used for verification.
2. The verb_data_length parameter will indicate the total size at the bottom of the table describing the verb_data.
3. Multiple byte fields are stored in Big-Endian format, as is typical for CEX*C communication.

### Table 28. Output data format for STATICSE operational key parts

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length in bytes</th>
<th>Field value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYM_OMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>SYM_OMK_MDC4_ID</td>
<td>2</td>
<td>X'0F02'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>SYM_OMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Symmetric Key old master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>SYM_CMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>SYM_CMK_MDC4_ID</td>
<td>2</td>
<td>X'0F01'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>SYM_CMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Symmetric Key current master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>SYM_NMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>SYM_NMK_MDC4_ID</td>
<td>2</td>
<td>X'0F00'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>SYM_NMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Symmetric Key new master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>ASYM_OMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>ASYM_OMK_MDC4_ID</td>
<td>2</td>
<td>X'0F05'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>ASYM_OMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Asymmetric Key old master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>ASYM_CMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>ASYM_CMK_MDC4_ID</td>
<td>2</td>
<td>X'0F04'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>ASYM_CMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Asymmetric Key current master-key register, calculated using the MDC4 algorithm.</td>
</tr>
</tbody>
</table>
Table 28. Output data format for STATICSE operational key parts (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length in bytes</th>
<th>Field value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASYM_NMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>ASYM_NMK_MDC4_ID</td>
<td>2</td>
<td>X'0F03'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>ASYM_NMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Asymmetric Key new master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>SYM_OMK_VP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>SYM_OMK_VP_ID</td>
<td>2</td>
<td>X'0F08'</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>SYM_OMK_VP</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the Symmetric Key old master-key register calculated using the default algorithm.</td>
</tr>
<tr>
<td>SYM_CMK_VP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>SYM_CMK_VP_ID</td>
<td>2</td>
<td>X'0F07'</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>SYM_CMK_VP</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the Symmetric Key current master-key register, calculated using the default algorithm.</td>
</tr>
<tr>
<td>SYM_NMK_VP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>SYM_NMK_VP_ID</td>
<td>2</td>
<td>X'0F06'</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>SYM_NMK_VP</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the Symmetric Key new master-key register, calculated using the default algorithm.</td>
</tr>
<tr>
<td>SYM_NMK_MKAP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Authentication pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Authentication pattern).</td>
</tr>
<tr>
<td>SYM_NMK_MKAP_ID</td>
<td>2</td>
<td>X'0F09'</td>
<td>Hexadecimal identifier indicating the contents of the following Authentication pattern field.</td>
</tr>
<tr>
<td>SYM_NMK_MKAP</td>
<td>8</td>
<td>variable</td>
<td>Authentication pattern over the Symmetric Key new master-key register, calculated using the ICSF specified algorithm.</td>
</tr>
<tr>
<td>Total byte count</td>
<td></td>
<td>168</td>
<td></td>
</tr>
</tbody>
</table>

**STATICSX**

This rule_array keyword returns the indicated master key hash and verification patterns for the master keys loaded in the current domain.

The status variables for the various master key registers returned in the rule_array will indicate which of these verification pattern structures returned contain useful data. An empty master key register cannot have a meaningful verification pattern. However, the data structures are returned for all registers indicated, so that interpretation is reliable.
The output data format for STATICSX operational key parts is given in Table 29.

Note:
1. The fields will be returned in the order given, however the *_ID fields should be used for verification.
2. The verb_data_length parameter will indicate the total size at the bottom of the table describing the verb_data.
3. Multiple byte fields are stored in Big-Endian format, as is typical for CEX*C communication.

Table 29. Output data format for STATICSX operational key parts

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length in bytes</th>
<th>Field value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYM_OMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>SYM_OMK_MDC4_ID</td>
<td>2</td>
<td>X'0F02'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>SYM_OMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Symmetric Key old master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>SYM_CMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>SYM_CMK_MDC4_ID</td>
<td>2</td>
<td>X'0F01'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>SYM_CMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Symmetric Key current master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>SYM_NMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>SYM_NMK_MDC4_ID</td>
<td>2</td>
<td>X'0F00'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>SYM_NMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Symmetric Key new master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>ASYM_OMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>ASYM_OMK_MDC4_ID</td>
<td>2</td>
<td>X'0F05'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>ASYM_OMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Asymmetric Key old master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>ASYM_CMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
<tr>
<td>ASYM_CMK_MDC4_ID</td>
<td>2</td>
<td>X'0F04'</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>ASYM_CMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Asymmetric Key current master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>ASYM_NMK_MDC4_LEN</td>
<td>2</td>
<td>20</td>
<td>Length in bytes of this Hash pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Hash pattern).</td>
</tr>
</tbody>
</table>
Table 29. Output data format for STATICSS operational key parts (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length in bytes</th>
<th>Field value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASYM_NMK_MDC4_ID</td>
<td>2</td>
<td>X’0F03’</td>
<td>Hexadecimal identifier indicating the contents of the following Hash pattern field.</td>
</tr>
<tr>
<td>ASYM_NMK_MDC4_HP</td>
<td>16</td>
<td>variable</td>
<td>Hash pattern over the Asymmetric Key new master-key register, calculated using the MDC4 algorithm.</td>
</tr>
<tr>
<td>SYM_OMK_VP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>SYM_OMK_VP_ID</td>
<td>2</td>
<td>X’0F08’</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>SYM_OMK_VP</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the Symmetric Key old master-key register calculated using the default algorithm.</td>
</tr>
<tr>
<td>SYM_CMK_VP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>SYM_CMK_VP_ID</td>
<td>2</td>
<td>X’0F07’</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>SYM_CMK_VP</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the Symmetric Key current master-key register, calculated using the default algorithm.</td>
</tr>
<tr>
<td>SYM_NMK_VP_LEN</td>
<td>2</td>
<td>12</td>
<td>Length in bytes of this Verification pattern block in the verb_data (comprising this length field, the following ID field, and the field for the Verification pattern).</td>
</tr>
<tr>
<td>SYM_NMK_VP_ID</td>
<td>2</td>
<td>X’0F06’</td>
<td>Hexadecimal identifier indicating the contents of the following Verification pattern field.</td>
</tr>
<tr>
<td>SYM_NMK_VP</td>
<td>8</td>
<td>variable</td>
<td>Verification pattern over the Symmetric Key new master-key register, calculated using the default algorithm.</td>
</tr>
<tr>
<td><strong>Total byte count</strong></td>
<td><strong>156</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**STATKPR**

This keyword returns non-secret information about a particular named operational key part loaded by the TKE to the user.

The structures for various key types are given under section “STATKPR output data.” An appropriate name for an existing operational key part is expected to be provided as described in section “STATKPRL input data.” If not, the error return code of 8 and a reason code of 1026 is be returned, meaning key name not found.

**STATKPRL input data**

A 64-byte key name must be provided in the verb_data field, while the verb_data_length must be set to a value of 64.

The operational key name must exactly match the name returned by a call to STATKPRL.

**STATKPR output data**

Note:
1. The fields are returned in the order given.
2. Output data overwrites the input data in the verb_data field, and set the verb_data_length field to the output value.

3. The verb_data_length parameter indicates the total size, at the bottom of the table describing the verb_data.
   Notice that the output data is smaller than the input data.

4. Multiple byte fields are stored in Big-Endian format, as is typical for CEX*C communication.

Table 30. Output data format for STATKPR operational key parts

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length in bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>1</td>
<td>State of the key part register:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong>                      <strong>Description</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'11'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'12'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'13'</td>
</tr>
<tr>
<td>reserved</td>
<td>1</td>
<td>Will have a value of X'00'.</td>
</tr>
<tr>
<td>key_length</td>
<td>1</td>
<td>Length of key in bytes. For DES keys, values are: 8, 16, 24. For AES keys,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>values are: 16, 24, 32.</td>
</tr>
<tr>
<td>cv_length</td>
<td>1</td>
<td>Length of Control Vector (CV) for key part, in bytes. The value is 8 or 16,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>indicating how much of the CV field to use. Note that CV is NOT a variable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>length field.</td>
</tr>
<tr>
<td>cv</td>
<td>16</td>
<td>Control Vector for the operational key part.</td>
</tr>
<tr>
<td>reserved_2</td>
<td>8</td>
<td>Has a value of X'00' for the entire length.</td>
</tr>
<tr>
<td>key_part_hash</td>
<td>20</td>
<td>Hash over the key stored in the key part register. For DES keys, the hash</td>
</tr>
<tr>
<td></td>
<td></td>
<td>algorithm is SHA-1. For AES keys, the hash algorithm is SHA-256.</td>
</tr>
<tr>
<td>ver_pattern</td>
<td>4</td>
<td>Verification pattern over the key calculated using the default algorithm.</td>
</tr>
<tr>
<td>Total byte count</td>
<td>52</td>
<td></td>
</tr>
</tbody>
</table>

**STATKPR**
This keyword returns a list of the names of all operational key parts that have been loaded by the TKE into the CEX*C.

Each name has a length of 64 bytes. The maximum number of key slots (and thus the count of labels returned) depends on the firmware loaded to the adapter. This count is currently set to 100. If not enough space has been provided (using the verb_data_length field passed in by the application) to return the available list, a return code of 8 and a reason code of 72 is returned.

**Note:** This keyword can return AES keys as well as HMAC key descriptions.

**STATOAHL**
This rule_array keyword causes all data to be returned in one VUD block. The described structure also comprises extra data fields that contains output if the SIGNSTAT keyword is specified.

The VUD block header and data has a layout as described in Table 31 on page 134.
### Table 31. Output data format for the STATOAHL keyword

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
<th>Subordinate data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Begin outer templating</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>Struct name</td>
<td>Signed_data_t</td>
<td>Name of structure</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Value: <code>mbif_types.h:118:#define SIGNED_DATA_T 0x82</code></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>Struct version</td>
<td>Signed_data_t</td>
<td>Version of structure</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Value: 0x00</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>Signed data len</td>
<td>Signed_data_t</td>
<td>Length of the entire <code>signed_data_t</code> structure, including appended data and signature</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>Data offset</td>
<td>Signed_data_t</td>
<td>Offset from start of the data sub-struct (here) to the start of the actual data payload.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Value: 0x14</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>Data len</td>
<td>Signed_data_t</td>
<td>Length of the data payload</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>Sig offset</td>
<td>Signed_data_t</td>
<td>Offset from start of the sig sub-structure (here) to the start of the actual signature.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian</td>
</tr>
<tr>
<td>22</td>
<td>4</td>
<td>Sig len</td>
<td>Signed_data_t</td>
<td>Length of the signature</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian</td>
</tr>
<tr>
<td>26</td>
<td>4</td>
<td>Sig type</td>
<td>Signed_data_t</td>
<td>Type of signature</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type: binary integer, big-endian</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Value: 0x04 (ECDSA over SHA-512 hash with ECC P521, raw format)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x00 (no signature)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Signed_data_t</strong>: 26 bytes (+ VUD fields = 30 bytes)</td>
</tr>
<tr>
<td>Begin payload</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Data format**: This data matches the output from the secure bootloader of the adapter. Assuming the same nonce is provided and given correct ordering of operations (query the bootloader first) the payload data hashes to the same value as that data provided by the secure bootloader.

The overall structure is called `health_t`, as indicated by the enumerated value in the first `struct_id_t` name field. This contains multiple sub-structures, each reproduced here.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
<th>Subordinate data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>name</td>
<td>struct_id_t</td>
<td>Sub-structure that defines the name and version of the data structure</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Enum value: <code>HEALTH_T (0x90)</code></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>version</td>
<td>struct_id_t</td>
<td><code>HEALTH_T (0x90)</code></td>
</tr>
</tbody>
</table>

**Begin**: Rom_status_t
<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
<th>Subordinate data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>name</td>
<td>Rom_status_tstruct_id_t</td>
<td>Rom status name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Enum value: ROM_STATUS_T (0x00)</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>version</td>
<td>Rom_status_tstruct_id_t</td>
<td>Rom status version (0x00)</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>Rsvd1</td>
<td>Rom_status_t</td>
<td>(0x00)</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>Rom_version</td>
<td>Rom_status_t</td>
<td>Rom_version is a packed representation of the POST0 and MB0 version numbers.</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>Page1_certified</td>
<td>Rom_status_t</td>
<td>Has the value 1 if MB1 has successfully performed an IBM_INIT command to the point of installing the page1 certificate and has value 0 otherwise.</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>Rsvd2</td>
<td>Rom_status_t</td>
<td>0x00</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>Boot_count_right</td>
<td>Rom_status_t</td>
<td>The number of times the card has been booted and SSP is reset. MB0 has run far enough to enforce the state rules.</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>adapterID</td>
<td>Rom_status_t</td>
<td>Byte array of adapter unique numeric serial number, taken from the DS3645 Silicon Serial Number register.</td>
</tr>
</tbody>
</table>

**Begin:** Rom_status_t:xcVpd_t

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
<th>Subordinate data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>1</td>
<td>Ds_tag</td>
<td>Rom_status_t:xcVpd_t</td>
<td>0x82, tag for description field</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>Ds_length</td>
<td>Rom_status_t:xcVpd_t</td>
<td>0x2C, description length</td>
</tr>
<tr>
<td>26</td>
<td>44</td>
<td>Ds</td>
<td>Rom_status_t:xcVpd_t</td>
<td>ASCII description</td>
</tr>
<tr>
<td>70</td>
<td>1</td>
<td>Vpdr_tag</td>
<td>Rom_status_t:xcVpd_t</td>
<td>0x90, tag for VPDR field</td>
</tr>
<tr>
<td>71</td>
<td>2</td>
<td>Vpdr_length</td>
<td>Rom_status_t:xcVpd_t</td>
<td>0x00CD, length of VPDR</td>
</tr>
<tr>
<td>73</td>
<td>2</td>
<td>Ec_tag</td>
<td>Rom_status_t:xcVpd_t</td>
<td>EC, ASCII tag for EC field</td>
</tr>
<tr>
<td>75</td>
<td>1</td>
<td>Ec_length</td>
<td>Rom_status_t:xcVpd_t</td>
<td>0x07, length of EC field</td>
</tr>
<tr>
<td>76</td>
<td>7</td>
<td>Ec</td>
<td>Rom_status_t:xcVpd_t</td>
<td>ASCII field</td>
</tr>
<tr>
<td>83</td>
<td>2</td>
<td>Pn_tag</td>
<td>Rom_status_t:xcVpd_t</td>
<td>Pn, ASCII tag for part number field</td>
</tr>
<tr>
<td>85</td>
<td>1</td>
<td>Pn_length</td>
<td>Rom_status_t:xcVpd_t</td>
<td>0x07, length of PN field</td>
</tr>
<tr>
<td>86</td>
<td>7</td>
<td>Pn</td>
<td>Rom_status_t:xcVpd_t</td>
<td>ASCII field</td>
</tr>
<tr>
<td>93</td>
<td>2</td>
<td>Fn_tag</td>
<td>Rom_status_t:xcVpd_t</td>
<td>Fn, ASCII tag for FRU number field</td>
</tr>
<tr>
<td>95</td>
<td>1</td>
<td>Fn_length</td>
<td>Rom_status_t:xcVpd_t</td>
<td>0x07, length of FN field</td>
</tr>
<tr>
<td>96</td>
<td>7</td>
<td>Fn</td>
<td>Rom_status_t:xcVpd_t</td>
<td>ASCII field</td>
</tr>
<tr>
<td>103</td>
<td>2</td>
<td>Ve_tag</td>
<td>Rom_status_t:xcVpd_t</td>
<td>VE, ASCII tag for Secure part number field</td>
</tr>
<tr>
<td>105</td>
<td>1</td>
<td>Ve_length</td>
<td>Rom_status_t:xcVpd_t</td>
<td>0x07, length of VE field</td>
</tr>
<tr>
<td>106</td>
<td>7</td>
<td>Ve</td>
<td>Rom_status_t:xcVpd_t</td>
<td>ASCII field</td>
</tr>
<tr>
<td>113</td>
<td>2</td>
<td>Mf_tag</td>
<td>Rom_status_t:xcVpd_t</td>
<td>MF, ASCII tag for manufacturing location field</td>
</tr>
<tr>
<td>115</td>
<td>1</td>
<td>Mf_length</td>
<td>Rom_status_t:xcVpd_t</td>
<td>0x02, length of MF field</td>
</tr>
<tr>
<td>116</td>
<td>2</td>
<td>Mf</td>
<td>Rom_status_t:xcVpd_t</td>
<td>ASCII field</td>
</tr>
<tr>
<td>118</td>
<td>2</td>
<td>Sn_tag</td>
<td>Rom_status_t:xcVpd_t</td>
<td>SN, ASCII tag for serial number field</td>
</tr>
<tr>
<td>120</td>
<td>1</td>
<td>Sn_length</td>
<td>Rom_status_t:xcVpd_t</td>
<td>0x0C, length of SN field</td>
</tr>
<tr>
<td>121</td>
<td>4</td>
<td>Sn_hdr</td>
<td>Rom_status_t:xcVpd_t</td>
<td>first part of serial number</td>
</tr>
</tbody>
</table>
### Table 31. Output data format for the STOAHL keyword (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
<th>Subordinate data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>8</td>
<td>Sn</td>
<td>Rom_status_t:xcVpd_t</td>
<td>Rest of serial number</td>
</tr>
<tr>
<td>133</td>
<td>2</td>
<td>Rv_tag</td>
<td>Rom_status_t:xcVpd_t</td>
<td>RV, ASCII tag for checksum field</td>
</tr>
<tr>
<td>135</td>
<td>1</td>
<td>Rv_length</td>
<td>Rom_status_t:xcVpd_t</td>
<td>Length of reserved field</td>
</tr>
<tr>
<td>136</td>
<td>1</td>
<td>Checksum</td>
<td>Rom_status_t:xcVpd_t</td>
<td>1 byte checksum field</td>
</tr>
<tr>
<td>137</td>
<td>141</td>
<td>Reserved</td>
<td>Rom_status_t:xcVpd_t</td>
<td>Reserved field, all 0x00 bytes</td>
</tr>
<tr>
<td>278</td>
<td>1</td>
<td>End_tag</td>
<td>Rom_status_t:xcVpd_t</td>
<td>0x78, end tag</td>
</tr>
</tbody>
</table>

**End: Rom_status_t:xcVpd_t**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
<th>Subordinate data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>279</td>
<td>1</td>
<td>Init_state</td>
<td>Rom_status_t</td>
<td>State of segment 2:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>280</td>
<td>1</td>
<td>Seg2_state</td>
<td>Rom_status_t</td>
<td>State of segment 2 (same possible values as</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Seg2_state</strong></td>
</tr>
<tr>
<td>281</td>
<td>1</td>
<td>Seg3_state</td>
<td>Rom_status_t</td>
<td></td>
</tr>
<tr>
<td>282</td>
<td>2</td>
<td>Owner2</td>
<td>Rom_status_t</td>
<td></td>
</tr>
<tr>
<td>284</td>
<td>2</td>
<td>Owner3</td>
<td>Rom_status_t</td>
<td></td>
</tr>
<tr>
<td>286</td>
<td>1</td>
<td>Active_seg1</td>
<td>Rom_status_t</td>
<td></td>
</tr>
<tr>
<td>287</td>
<td>2</td>
<td>Rsvd3</td>
<td>Rom_status_t</td>
<td></td>
</tr>
<tr>
<td>289</td>
<td>4</td>
<td>usr</td>
<td>Rom_status_t</td>
<td></td>
</tr>
</tbody>
</table>

**End: Rom_status_t + struct_id_t (291 + 2 = 293 bytes)**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
<th>Subordinate data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>4</td>
<td>nonce</td>
<td>Raw nonce</td>
<td>Value passed by user to be signed as part of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>returned payload. This indicates the signature is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>live, not a replayed value that was calculated</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>earlier.</td>
</tr>
</tbody>
</table>

**Begin: var_t array for segment identifiers**

A **var_t** is a simple structure with 2 fields, specifying an offset and a length. These help you find the item the **var_t** describes.

In this section are three **var_t** structures, followed in the next section by three segment identifiers that are pointed to by the **var_t** structures.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
<th>Subordinate data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>Offset</td>
<td>Var_t : for segment 1</td>
<td>Offset to the segment identifier from byte 0 of this</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>length</td>
<td>Var_t : for segment 1</td>
<td>Length of the segment identifier</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>Offset</td>
<td>Var_t : for segment 2</td>
<td>Offset to the segment identifier from byte 0 of this</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>length</td>
<td>Var_t : for segment 2</td>
<td>Length of the segment identifier</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>Offset</td>
<td>Var_t : for segment 3</td>
<td>Offset to the segment identifier from byte 0 of this</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>length</td>
<td>Var_t : for segment 3</td>
<td>Length of the segment identifier</td>
</tr>
</tbody>
</table>

**End: var_t array (24 bytes)**

**Begin: segment identifiers array, struct is mbid_t** **mbid_t** indicates: *Miniboot identifier. Miniboo*’ is the name for the secure boot loader that manages the firmware loaded to the adapter.

**Begin: mbid_t for segment 1**
<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
<th>Subordinate data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>name</td>
<td>mbid_t:struct_id_t</td>
<td>0x81, name for mbid_t,</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>version</td>
<td>mbid_t:struct_id_t</td>
<td>0x02, ECC version of mbid_t: indicates secure bootloader uses ECDSA to verify image for this loaded segment of firmware.</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>type</td>
<td>mbid_t</td>
<td>Type field: Value is: FAM_OWNER 0x03 If type is FAM_OWNER, the adapterID is a bit vector that specifies onto which device families the image identified by the mbid_t parameter may be loaded. An attempt to load an image onto a device belonging to a family whose bit in adapterID is not set, will fail.</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>name</td>
<td>mbid_t:ownerID_t:struct_id_t</td>
<td>0x80, name for ownerID_t structure</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>version</td>
<td>mbid_t:ownerID_t:struct_id_t</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>seg</td>
<td>mbid_t:ownerID_t</td>
<td>Indicates the segment to which this ownerID_t applies, may be 0x01, 0x02 or 0x03.</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>Owner2</td>
<td>mbid_t:ownerID_t</td>
<td>If the seg field holds 0x01, then this field holds 0x0000. Otherwise, this field holds the two-byte owner ID for the applicable segment 2.</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>Owner3</td>
<td>mbid_t:ownerID_t</td>
<td>If the seg field holds 0x01 or 0x02, then this field holds 0x0000. Otherwise, this field holds the two-byte owner ID for the applicable segment 2.</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>trust1</td>
<td>mbid_t</td>
<td>Owner specified value of what to do if a lower level segment is updated. This trust1 field value refers to segment 1, and is 0x00 if the seg value is 0x01. For other cases, the meaning implies to trust or not trust, and is of interest to holders of signing keys.</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>Trust2</td>
<td>mbid_t</td>
<td>Owner specified value of what to do if a lower level segment is updated. This trust2 field value refers to segment 2, and is 0x00 if the seg value is 0x01 or 0x02. For other cases, the meaning implies to trust or not trust, and is of interest to holders of signing keys.</td>
</tr>
<tr>
<td>12</td>
<td>80</td>
<td>name</td>
<td>mbid_t</td>
<td>Byte array for image name</td>
</tr>
<tr>
<td>92</td>
<td>2</td>
<td>rev</td>
<td>mbid_t</td>
<td>Revision assigned to the segment by the owner of the segment that this mbid_t structure is describing</td>
</tr>
<tr>
<td>94</td>
<td>64</td>
<td></td>
<td>mbid_t</td>
<td></td>
</tr>
<tr>
<td>158</td>
<td>8</td>
<td>hash</td>
<td>mbid_t</td>
<td>SHA512 hash of the image identified by the name field</td>
</tr>
<tr>
<td>166</td>
<td>4</td>
<td>reserved</td>
<td>mbid_t:Etc (var_t)</td>
<td>Reserved field. The data has validity only inside the card.</td>
</tr>
<tr>
<td>170</td>
<td>4</td>
<td>reserved</td>
<td>mbid_t:Etc (var_t)</td>
<td>Reserved field. The data has validity only inside the card.</td>
</tr>
</tbody>
</table>
### Table 31. Output data format for the STATOAHL keyword (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
<th>Subordinate data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>174</td>
<td>4</td>
<td>offset</td>
<td>mbid_t:Token (var_t)</td>
<td>Segment key pointer field.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This var_t parameter indicates the offset from byte 0 of this field to the beginning of the public key which describes the segment that this mbid_t represents.</td>
</tr>
<tr>
<td>178</td>
<td>4</td>
<td>len</td>
<td>mbid_t:Token (var_t)</td>
<td>Length of the public key token for the segment</td>
</tr>
</tbody>
</table>

**Begin: mbid_t:token for segment 1**

**Explanation of the public key here**

A firmware load operation is actually a pre-packaged and signed *command*, as recognized by the secure bootloader in the adapter. While it is commonly said that “the firmware is signed”, it is more accurate to say the “firmware load command is signed, and includes the new firmware”. Each firmware load command has been signed by the owner of a signing key, and this owner will become the ‘owner of the segment’ after the owner’s firmware has been loaded. Thus we refer to the ‘segment owner’ as the person who held the private key that signed the command to load the firmware for that segment. Each firmware load command package also includes a signed version of the public key of the ‘segment owner’ for the segment to be loaded. The segment owner public key signature has been computed with the private key of the owner of the lower segment that is expected to have been installed. For example, consider load of a new segment 2:

1. The segment 2 load command is a binary software image, with:
   a. firmware image for new segment 2
   b. command of how to load the segment 2
   c. signature over the firmware image and command, calculated with segment 2 owner private key
   d. segment 2 owner public key
   e. signature over segment 2 owner public key calculated with segment 1 owner private key

2. The secure boot loader
   a. uses segment 1 owner public key to verify signature over segment 2 owner public key
   b. uses segment 2 owner public key to verify signature over firmware and command
   c. executes the command.

Given the above explanation, it is now possible to say that the public key included in the mbid_t is the segment owner public key for the segment that the mbid_t describes. If the mbid_t describes segment 1, then the segment 1 owner public key is included at the end of the mbid_t when returned from the adapter.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
<th>Subordinate data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>name</td>
<td>mbid_t:token:struct_id_t</td>
<td>struct_id_t, sub-structure that defines the name and version of the data structure</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Enum value: ECC_TOKEN_T (0x97)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>version</td>
<td>mbid_t:token:struct_id_t</td>
<td>0x00</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Rsvd1</td>
<td>mbid_t:token</td>
<td>0x00</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>tokenLength</td>
<td>mbid_t:token</td>
<td>Length of token including header.</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>Rsvd2</td>
<td>mbid_t:token</td>
<td>0x00</td>
</tr>
</tbody>
</table>
### Cryptographic Facility Query (CSUACFQ)

**Table 31. Output data format for the STATOAH keyword (continued)**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
<th>Subordinate data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>1</td>
<td>name</td>
<td>mbid_t:token:struct_id_t</td>
<td>struct_id_t defines the name and version of the data structure</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Enum value: ECC_PUBLIC_TOKEN_T (0x99)</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>version</td>
<td>mbid_t:token:struct_id_t</td>
<td>0x00</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>sectionLength</td>
<td>mbid_t:token</td>
<td>Length of this section</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>Rsvd1</td>
<td>mbid_t:token</td>
<td>0x00</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>curveType</td>
<td>mbid_t:token</td>
<td>Type of curve</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Value: 0x00 (Prime/NIST)</td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>Rsvd2</td>
<td>mbid_t:token</td>
<td>0x00</td>
</tr>
<tr>
<td>22</td>
<td>2</td>
<td>pLength</td>
<td>mbid_t:token</td>
<td>Length of curve</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Value: 0x0209 (P521)</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>qLen</td>
<td>mbid_t:token</td>
<td>0x91</td>
</tr>
<tr>
<td>26</td>
<td>1</td>
<td>Pubkey_promise</td>
<td>mbid_t:token</td>
<td>Value: 0x04</td>
</tr>
<tr>
<td>27</td>
<td>72</td>
<td>Ecc_public_x</td>
<td>mbid_t:token</td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>72</td>
<td>Ecc_public_y</td>
<td>mbid_t:token</td>
<td>Y value for the public key</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Length is 66 bytes for ECC P521 key + padding bytes for hardware use of the value.</td>
</tr>
</tbody>
</table>

**End: mbid_t:token for segment 1 (171 bytes)**

**End: mbid_t for segment 1 (182 + 171 = 353 bytes)**

<table>
<thead>
<tr>
<th>353</th>
<th>mbid_t</th>
<th>mbid_t for segment 2</th>
<th>The mbid_t for each segment is the same size and has the same fields as the mbid_t for segment 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>353</td>
<td>mbid_t</td>
<td>mbid_t for segment 3</td>
<td>The mbid_t for each segment is the same size and has the same fields as the mbid_t for segment 1</td>
</tr>
</tbody>
</table>

**End: segment identifiers array, struct is mbid_t (3 * 353 bytes = 1059 bytes)**

**End payload**

**Begin signature section sig-section (if SIGNSTAT passed)**

| 132   | signature | Raw r and s values for signature over SHA-512 hash of the payload | Length X is given in the signed_data_t:sig_len field at the beginning of returned data |
|       |           |                                                                 | X is always be 132 bytes | r and s are 66 bytes each, r is first. |
| 64    | Payload hash | Raw hash value | Raw SHA-512 hash over the payload. This is the value used for calculating the signature. |

**Total structure size:**

- VUD fields + signed_data_t: 30 bytes
- Rom_status_t: 294 bytes
- Nonce: 32 bytes
- var_t array (describing mbid_t): 24 bytes
- mbid_t array: 1059 bytes
- Signature section and hash: 196 bytes
- **Total:** 1634 bytes
Cryptographic Facility Query (CSUACFQ)

**STATVKPL**
This keyword returns a list of the names of all operational key parts that have been loaded by the TKE into the CEX*C for preparation of variable length key tokens.

This function is different from STATKPRL, which describes the list of 64-byte legacy style tokens in preparation.

Each name has a length of 64 bytes. The maximum number of key slots (and thus the count of labels returned) depends on the firmware loaded to the adapter. This count is currently set to 100. If not enough space has been provided (using the `verb_data_length` field passed in by the application) to return the available list, a return code of 8 and a reason code of 72 is returned.

**Note:** This keyword can return AES keys as well as HMAC key descriptions.

**STATVKPR**
This keyword cause non-secret information about a particular named operational key part loaded by the TKE to returned to the user.

This is different from STATKPR in that a register for creating a key in a variable length key token is described. The structures for various key types are given in section **STATVKPR output data.** An appropriate name for an existing operational key part is expected to be provided as described in **STATVKPL input data.** If not, the error return code of 8 and a reason code of 1026 is returned, meaning that the key name is not found.

**STATVKPL input data:**

A 64 byte key name must be provided in the `verb_data` field, while the `verb_data_length` must be set to 64.

The operational key name must match exactly the name returned by a call to **STATVKPL.**

**STATVKPR output data:**

See [Table 32](#) for the output data format.

**Note:**
1. The fields are returned in the order given.
2. Output data overwrites the input data in the `verb_data` field, and set the `verb_data_length` field to the output value.
3. The `verb_data_length` parameter indicates the total size, at the bottom of the table describing the `verb_data`.
   
   Note that the output data is smaller than the input data.
4. Multiple byte fields are stored in Big-Endian format, as is typical for CEX*C communication.

**Table 32. Output data format for STATVKPR operational key parts**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length in bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>1</td>
<td>Version of the structure</td>
</tr>
</tbody>
</table>
Table 32. Output data format for STATVKPR operational key parts (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length in bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>1</td>
<td>State of the key part register:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'00'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'01'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'02'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'03'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'11'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'12'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'13'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'21'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'22'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'23'</strong></td>
</tr>
<tr>
<td>key_length</td>
<td>1</td>
<td>Length of key in bytes. For DES keys, values are: 8, 16, 24. For AES keys, values are: 16, 24, 32.</td>
</tr>
<tr>
<td>key_completeness</td>
<td>1</td>
<td>Number of parts needed to complete key</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'C0'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'80'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'40'</strong></td>
</tr>
<tr>
<td>ver_pattern</td>
<td>4</td>
<td>ENC-ZERO method calculated verification pattern of the key.</td>
</tr>
<tr>
<td>key_part_hash</td>
<td>8</td>
<td>Hash using the SHA-256 algorithm over the key that is currently stored, at the current level of completeness.</td>
</tr>
<tr>
<td>skel_length</td>
<td>2</td>
<td>Skeleton token length.</td>
</tr>
<tr>
<td>pad</td>
<td>2</td>
<td>Pad structure to 4-byte boundary.</td>
</tr>
<tr>
<td>skel</td>
<td>384</td>
<td>Stored key token skeleton, which will hold completed key when operation is complete. No key material is stored or returned here.</td>
</tr>
<tr>
<td>reserved2</td>
<td>108</td>
<td>Extra bytes.</td>
</tr>
<tr>
<td>Total byte count</td>
<td>512</td>
<td></td>
</tr>
</tbody>
</table>

**STATWPIN**

For the CSUACFQ verb, the SIZEWPIN and STATWPIN keywords are added to the rule array to query the size and to obtain the state information of the weak PIN table. The STATWPIN rule_array keyword obtains the state information of the weak PIN table.

Table 33. Output data format for the STATWPIN keyword

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length in bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>weak_PIN_table_state</td>
<td>11 - 460</td>
<td>Returns the WPIN entry structure as described in Table 34</td>
</tr>
</tbody>
</table>

Table 34. CSUACFQ weak PIN entry structure (type X'30')

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length in bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01</td>
<td>Weak PIN structure type (in ASCII): X'30' - weak PIN entry structure</td>
</tr>
<tr>
<td>1</td>
<td>03</td>
<td>Weak PIN entry identifier in ASCII digits 001 - 020 (X'303031' - X'303230')</td>
</tr>
</tbody>
</table>
Table 34. CSUACFQ weak PIN entry structure (type X'30') (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length in bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>01</td>
<td>State of entry (in ASCII):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A (X'41')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L (X'4C')</td>
</tr>
<tr>
<td>5</td>
<td>02</td>
<td>Clear PIN length (in ASCII digits). For DK verbs, valid values are 04 - 12 (X'3034' - X'3132'). Otherwise, valid values are 04 - 16 (X'3034' - X'3136').</td>
</tr>
<tr>
<td>7</td>
<td>04 — 16</td>
<td>Clear weak PIN (in ASCII digits). Valid values are 0 - 9 (X'30' - X'39').</td>
</tr>
</tbody>
</table>

**Restrictions**

You cannot limit the number of returned rule_array elements.

Table 22 on page 106 describes the number and meaning of the information in output rule_array elements.

**Tip:** Allocate a minimum of 30 rule_array elements to allow for extensions of the returned information.

**Required commands**

The CSUACFQ required commands.

None.

**Usage notes**

Usage notes for CSUACFQ.

This verb is not impacted by the AUTOSELECT option. See “Verbs that ignore AUTOSELECT” on page 12 for more information.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSUACFQJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSUACFQJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber verb_data_length,
    byte[] verb_data
);```

Cryptographic Facility Query (CSUACFQ)
Cryptographic Facility Version (CSUACFV)

The Cryptographic Facility Version verb is used to retrieve information about the Security Application Program Interface (SAPI) Version and the Security Application Program Interface build date.

In the same format as the Cryptographic Facility Query (CSUACFQ) verb returns for the CCA application with the STATCCA rule_array option.

This verb returns information elements in the version_data variable.

Format

The format of CSUACFV.

```
CSUACFV(
   return_code,
   reason_code,
   exit_data_length,
   exit_data,
   version_data_length,
   version_data)
```

Parameters

The parameter definitions for CSUACFV.

Note that there is no rule_array keyword.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see "Parameters common to all verbs" on page 22.

version_data_length

- **Direction:** Input/Output
- **Type:** Integer

The version_data_length parameter is a pointer to an integer variable containing the number of bytes in the version data variable. This value must be a minimum of 17 bytes. On input, the version_data_length variable must be set to the total size of the variable pointed to by the version_data parameter. On output, this variable contains the number of bytes of data returned by the verb in the version_data variable.

version_data

- **Direction:** Output
- **Type:** String

The version_data parameter is a pointer to a string variable containing data returned by the verb. An 8-byte character string identifies the version of the Security Application Program Interface (SAPI) library, followed by an 8-byte character string containing the build date for the SAPI library, followed by a null terminating character. The build date is in the format: yyyyymmdd, where yyyy is the year, mm is the month, and dd is the day of the month.

Restrictions

The restrictions for CSUACFV.

None.
Cryptographic Facility Version (CSUACFV)

**Required commands**
The CSUACFV required commands.
None.

**Usage notes**
Usage notes for CSUACFV.
None.

**JNI version**
This verb has a Java Native Interface (JNI) version, which is named CSUACFVJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**
```java
public native void CSUACFVJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber version_data_length,
    byte[] version_data
);
```

Cryptographic Resource Allocate (CSUACRA)
The Cryptographic Resource Allocate verb is used to allocate a specific CCA coprocessor for use by the thread or process, depending on the scope of the verb.

This verb is scoped to a thread. When a thread or process, depending on the scope, allocates a cryptographic resource, requests are routed to that resource. When a cryptographic resource is not allocated, requests are routed to the default cryptographic resource.

You can set the default cryptographic resource. If you take no action, the default assignment is CRP01.

You cannot allocate a cryptographic resource while one is already allocated. Use the Cryptographic Resource Deallocate verb (see “Cryptographic Resource Deallocate (CSUACRD)” on page 147) to deallocate an allocated cryptographic resource.

Be sure to review “Multi-coprocessor selection capabilities” on page 11.
Cryptographic Resource Allocate (CSUACRA)

Format

The format of CSUACRA.

```plaintext
CSUACRA(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    resource_name_length,
    resource_name)
```

Parameters

The parameter definitions for CSUACRA.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see [“Parameters common to all verbs” on page 22](#).

**rule_array_count**

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1.

**rule_array**

Direction: Input
Type: String array

The `rule_array` parameter is a pointer to a string variable containing an array of keywords. The keywords are eight bytes in length and must be left-aligned and padded on the right with space characters. The `rule_array` keyword is described in [Table 35](#).

Table 35. Keywords for Cryptographic Resource Allocate control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cryptographic resource</strong> (Required)</td>
<td></td>
</tr>
<tr>
<td>DEVICE</td>
<td>Specifies a CEX*C coprocessor.</td>
</tr>
<tr>
<td>DEV-ANY</td>
<td>Specifies to enable the AUTOSELECT option, such that the operating system may select the CCA coprocessor to be used from the available resources according to its policy. This selection applies to most verbs, but not all. See <a href="#">“Multi-coprocessor selection capabilities” on page 11</a> for more information.</td>
</tr>
<tr>
<td>HCPUACLR</td>
<td>Specifies the use of host CPU assist for clear keys. This keyword enables clear key use of the CPACF, for clear key AES encryption and decryption with hash algorithms: SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512. This is the default state at the time of the first use of the CCA library by a PID or TID.</td>
</tr>
<tr>
<td>HCPUAPRT</td>
<td>Specifies the use of host CPU assist for protected keys. This keyword enables protected key use of the CPACF for protected key AES and DES, TDES, and MAC. This is not the default state at the time of the first use of the CCA library by a PID or TID.</td>
</tr>
</tbody>
</table>
Cryptographic Resource Allocate (CSUACRA)

There is an environment variable that also impacts the default card:
CSU_DEFAULT_ADAPTER (see “Multi-coprocessor selection capabilities” on page 11). There are also environment variables that influence CPACF support (see “Environment variables that affect CPACF usage” on page 14).

The actual hardware configuration determines what features are available, and CCA uses what exists if the user sets these values as desired, with respect to appropriate defaults.

resource_name_length

Direction: Input
Type: Integer

The resource_name_length parameter is a pointer to an integer variable containing the number of bytes of data in the resource_name variable. The length must be 1 - 64.

resource_name

Direction: Input
Type: String

The resource_name parameter is a pointer to a string variable containing the name of the coprocessor to be allocated.

Restrictions

Restrictions for CSUACRA.

None.

Required commands

The CSUACRA required commands.

None.

Usage notes

Usage notes for CSUACRA.

For optimal performance, ensure that you have enabled CPACF in the thread doing the processing, by making a quick call on the host side at thread startup time, to Cryptographic Resource Allocate, specifying the correct HCPUACL and HCPUAPRT keyword values for your operation. See the Cryptographic Resource Allocate rule array keyword definitions, and see “Access control points that affect CPACF protected key operations” on page 16 for more affected verbs. Note that it is the user’s responsibility to ensure that all adapters accessible to the operating system use the DEV-ANY state. Further use of DEV-ANY as a target adapter for verbs that modify the state of an adapter (such as the generation of retained keys) can lead to unexpected results. See “Multi-coprocessor selection capabilities” on page 11 for more information.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSUACRAJ.

See “Building Java applications using the CCA JNI” on page 28.
Cryptographic Resource Deallocate (CSUACRD)

The Cryptographic Resource Deallocate verb is used to deallocate a specific CCA coprocessor that is allocated by the thread or process, depending on the scope of the verb.

This verb is scoped to a thread. When a thread or process, depending on the scope, de-allocates a cryptographic resource, requests are routed to the default cryptographic resource.

You can set the default cryptographic resource. If you take no action, the default assignment is CRP01.

If a thread with an allocated coprocessor ends without first de-allocating the coprocessor, excess memory consumption results. It is not necessary to deallocate a cryptographic resource if the process itself is ending, only if individual threads end while the process continues to run.

Be sure to review “Multi-coprocessor selection capabilities” on page 11.

Format

The format of CSUACRD.

```
public native void CSUACRD(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber resource_name_length,
    byte[] resource_name);
```

Parameters

The parameter definitions for CSUACRD.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

`rule_array_count`

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1.
Cryptographic Resource Deallocate (CSUACRD)

**rule_array**

- **Direction:** Input
- **Type:** String array

The `rule_array` parameter is a pointer to a string variable containing an array of keywords. The keywords are eight bytes in length and must be left-aligned and padded on the right with space characters. The `rule_array` keyword is described in Table 36.

### Table 36. Keywords for Cryptographic Resource Deallocate control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cryptographic resource (Required)</td>
<td></td>
</tr>
<tr>
<td>DEVICE</td>
<td>Specifies a CEX*C coprocessor.</td>
</tr>
<tr>
<td>DEV-ANY</td>
<td>Specifies to disable the AUTOSELECT option. Verbs will now all use the default adapter or a previously configured &quot;selected&quot; adapter as chosen via CSUCRA. See “Multi-coprocessor selection capabilities” on page 111 for more information.</td>
</tr>
<tr>
<td>HCPUACLR</td>
<td>Specifies the use of host CPU assist for clear keys. This keyword enables clear key use of the CPACF, for clear key AES encryption and decryption with hash algorithms: SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512. This is the default state at the time of the first use of the CCA library by a PID or TID.</td>
</tr>
<tr>
<td>HCPUAPRT</td>
<td>Specifies the use of host CPU assist for protected keys. This keyword disables protected key use of the CPACF for protected key AES and DES, TDES, and MAC. This is the default state at the time of the first use of the CCA library by a PID or TID.</td>
</tr>
</tbody>
</table>

There is an environment variable that also impacts the default card: CSU_DEFAULT_ADAPTER (see “Multi-coprocessor selection capabilities” on page 111). There are also environment variables that influence CPACF support (see “Environment variables that affect CPACF usage” on page 14).

The actual hardware configuration determines what features are available, and CCA uses what exists if the user sets these values as desired, with respect to appropriate defaults.

**resource_name_length**

- **Direction:** Input
- **Type:** Integer

The `resource_name_length` parameter is a pointer to an integer variable containing the number of bytes of data in the `resource_name` variable. The length must be 1 - 64.

**resource_name**

- **Direction:** Input
- **Type:** String

The `resource_name` parameter is a pointer to a string variable containing the name of the coprocessor to be deallocated.

### Restrictions

The restrictions for CSUACRD.

None.
Cryptographic Resource Deallocate (CSUACRD)

Required commands
The CSUACRD required commands.

None.

Usage notes
Usage notes for CSUACRD.

To disable CPACF usage in your processing thread, make a call to Cryptographic Resource Deallocate, specifying the correct HCPUACLR and HCPUAPRT keyword as appropriate. See the Cryptographic Resource Deallocate rule_array keyword definitions, and see “Access control points that affect CPACF protected key operations” on page 16 for more affected verbs.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSUACRDJ.

See “Building Java applications using the CCA JNI” on page 28.

Format
public native void CSUACRDJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber resource_name_length,
    byte[] resource_name);

Key Storage Initialization (CSNBKSI)
The Key Storage Initialization verb initializes a key-storage file using the current symmetric or asymmetric master-key.

The initialized key storage file does not contain any preexisting key records. The key storage data and index files are in the /opt/IBM/CCA/keys directory.

The key storage functions do not work with HMAC keys.

Format
The format of CSNBKSI.

```
CSNBKSI(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_storage_file_name_length,
    key_storage_file_name,
    key_storage_description_length,
    key_storage_description,
    clear_master_key)
```
Key Storage Initialization (CSNBKSI)

Parameters

The parameter definitions for CSNBKSI.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

rule_array_count

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 2.

rule_array

Direction: Input
Type: String array

The rule_array parameter is a pointer to a string variable containing an array of keywords. The keywords are eight bytes in length and must be left-aligned and padded on the right with space characters. The rule_array keywords are described in Table 37.

Table 37. Keywords for Key Storage Initialization control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master-key source (Required)</td>
<td></td>
</tr>
<tr>
<td>CURRENT</td>
<td>Specifies the current symmetric master-key of the default cryptographic facility is to be used for the initialization.</td>
</tr>
<tr>
<td>Key-storage selection (One required)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>Initialize AES key storage (fixed-length and variable-length AES key tokens and HMAC key tokens).</td>
</tr>
<tr>
<td>DES</td>
<td>Initialize DES key storage.</td>
</tr>
<tr>
<td>PKA</td>
<td>Initialize PKA key storage (RSA and ECC key tokens).</td>
</tr>
</tbody>
</table>

key_storage_file_name_length

Direction: Input
Type: Integer

The key_storage_file_name_length parameter is a pointer to an integer variable containing the number of bytes of data in the key_storage_file_name variable. The length must be within the range of 1 - 64.

key_storage_file_name

Direction: Input
Type: String

The key_storage_file_name parameter is a pointer to a string variable containing the fully qualified file name of the key-storage file to be initialized. If the file does not exist, it is created. If the file does exist, it is overwritten and all existing keys are lost.

key_storage_description_length

Direction: Input
Type: Integer
The `key_storage_description_length` parameter is a pointer to an integer variable containing the number of bytes of data in the `key_storage_description` variable.

**key_storage_description**

Direction: Input  
Type: String

The `key_storage_description` parameter is a pointer to a string variable containing the description string stored in the key-storage file when it is initialized.

**clear_master_key**

Direction: Input  
Type: String

The `clear_master_key` parameter is unused, but it must be declared and point to 24 data bytes in application storage.

**Restrictions**

The restrictions for CSNBKSI.

ECC and variable-length symmetric key tokens are not supported in releases before Release 4.1.0.

**Required commands**

The CSNBKSI required commands.

In order to access key storage, the Key Storage Initialization verb requires the `Compute Verification Pattern` command (offset X’001D’) to be enabled in the active role.

**Usage notes**

Usage notes for CSNBKSI.

Using the Key Storage Initialization verb to initialize DES (symmetric) key storage currently requires both the symmetric master key and the asymmetric master key to be loaded, completed and set using the Master Key Process (CSNBMKP) verb or suitable utility, or an interface tool that uses the Master Key Process (CSNBMKP) verb.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBKSIJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBKSIJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber key_storage_file_name_length,
    // other parameters...
);```

Chapter 7. CCA nodes and resource control  151
Log Query (CSUALGQ)

Use the Log Query verb to retrieve system log (SYSLOG) message data and CCA log message data from the coprocessor. Syslog data is available for one of the five latest boot cycles (current boot cycle and up to four previous boot cycles). CCA log message data is optionally available during the current boot cycle. The verb supports service personnel and developers in testing and debugging issues.

The SYSLOG for a boot cycle is initially populated with operating system messages and other startup messages, but more messages are written as CCA encounters errors at log level 12 or above. A log level of 12 means, that if an error at return code value 12 or higher occurs during the processing of an operation or during general system processing, then a descriptive message is sent to the SYSLOG. The log level is configurable, with a default value of 12. If configured to be logged, messages for return code 4 and return code 8 are saved in volatile memory (CCA log) separately from messages in the SYSLOG. This keeps these messages from overwhelming more important errors and system messages. Each new boot cycle wipes out any previous CCA log, while the current SYSLOG and up to four previous boot cycle SYSLOGs are maintained. The CCA log is a circular log. When the log is full, the oldest messages are replaced as needed with any new messages.

Format

The format of CSUALGQ.

```
CSUALGQ(
   return_code,
   reason_code,
   exit_data_length,
   exit_data,
   rule_array_count,
   rule_array,
   log_number_or_level,
   reserved0,
   log_data_length,
   log_data,
   reserved1_length,
   reserved1,
   reserved2_length,
   reserved2)
```

Parameters

The parameter definitions for CSUALGQ.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

`rule_array_count`

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. The value must be 1.
Log Query (CSUALGQ)

**rule_array**

- Direction: Input
- Type: String array

A pointer to a string variable containing an array of keywords. The keywords are eight bytes in length and must be left-aligned and padded on the right with space characters. The **rule_array** keywords are described in Table 38.

Table 38. Keywords for Log Query control information.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCALGLEN</td>
<td>Specifies to return the length of the CCA log data buffer in the <strong>log_data_length</strong> variable (the <strong>log_number_or_level</strong> parameter is ignored). No data is returned in the <strong>log_data</strong> variable.</td>
</tr>
<tr>
<td>GETCCALG</td>
<td>Specifies to get up to <strong>log_data_length</strong> bytes of data from the current CCA log data buffer (the <strong>log_number_or_level</strong> parameter is ignored). This data is returned in the <strong>log_data</strong> variable. The returned log only contains all available CCA messages that had a return code of 4 or 8. All other messages are filtered out. Note: If the log level (see SETLGLVL keyword) has never been set to 4 or 8 in the current boot cycle, then the CCA log is empty and no data is available to return.</td>
</tr>
<tr>
<td>GETLGLVL</td>
<td>Specifies to return the current log level in the <strong>log_number_or_level</strong> variable. The log level is used to determine which return codes the coprocessor uses to generate a log message. A return code greater than or equal to this level causes a log message to be generated. No data is returned in the <strong>log_data</strong> variable.</td>
</tr>
<tr>
<td>GETSYSLG</td>
<td>Specifies to get up to <strong>log_data_length</strong> bytes of data from the SYSLOG (for the given log number as specified by the <strong>log_number_or_level</strong> parameter). This data is returned in the <strong>log_data</strong> variable. The returned log contains all available SYSLOG messages, including boot, major error, and CCA messages.</td>
</tr>
</tbody>
</table>
| SETLGLVL  | Specifies to set the current log level inside the coprocessor to the value specified by the **log_number_or_level** parameter (4, 8, or 12). A CCA return code greater than or equal to the log level causes a log message to be generated. No data is returned in the **log_data** variable. If the log level for the coprocessor has never been set in the current boot cycle, it defaults to 12. Where the coprocessor stores a message, depends on the message return code:  
  - Return code 4 and return code 8 messages are stored in the CCA log. The CCA log is stored in a volatile message data buffer and is only available for the current boot cycle.  
  - Return code 12 and return code 16 messages are stored in the current boot cycle SYSLOG along with non-CCA messages. In addition to the current boot cycle SYSLOG, there are up to four SYSLOGs maintained from previous boot cycles. |
| SYSLGLEN  | Specifies to return the length of the SYSLOG data buffer in the **log_data_length** variable (for the given log number specified by the **log_number_or_level** parameter). No data is returned in the **log_data** variable. |

**log_number_or_level**

- Direction: Input/Output
- Type: Integer

A pointer to an integer variable containing the log number or level for the service requested. This parameter may be unused, but must always be declared. Table 39 describes the meaning of this parameter for the different requested services.

Table 39. Meaning of **log_number_or_level** for the requested service (keyword).

<table>
<thead>
<tr>
<th>Service requested</th>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCALGLEN</td>
<td>Input</td>
<td>The value is ignored.</td>
</tr>
</tbody>
</table>
Log Query (CSUALGQ)

Table 39. Meaning of log_number_or_level for the requested service (keyword) (continued).

<table>
<thead>
<tr>
<th>Service requested</th>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETCCALG</td>
<td>Input</td>
<td>The value is ignored.</td>
</tr>
<tr>
<td>GETLGLVL</td>
<td>Output</td>
<td>This is the current minimum level of CCA return code for which a coprocessor generates a log message. Valid values are 4, 8, or 12.</td>
</tr>
<tr>
<td>GETSYSLOG</td>
<td>Input</td>
<td>Specifies which of five possible groups of system log message data to retrieve from the coprocessor. See the note at end of the table.</td>
</tr>
<tr>
<td>SETLGLVL</td>
<td>Input</td>
<td>Specifies the minimum level of the CCA return code for which a coprocessor is to generate a log message. Valid values are 4, 8, or 12. If the log level is not set, the default is 12. Value: CCA return code that generates a log message</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4: 4, 8, 12, or 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8: 8, 12, or 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12: 12 or 16 (default if log level not set)</td>
</tr>
<tr>
<td>SYSLOGLEN</td>
<td>Input</td>
<td>Specifies which length of five possible groups of SYSLOG message data to retrieve from the coprocessor. See the note at end of the table.</td>
</tr>
</tbody>
</table>

Note: To process the system log message data from the current boot cycle, specify 0 in the log_number_or_level parameter. To process the data from the oldest available boot cycle possible, use a log number of 4. If the log number specifies an empty slot (that is, the coprocessor has not been power-cycled enough times for the given log number to have data yet), no data is returned.

reserved0

Direction: Input/Output
Type: Integer

A pointer to an integer variable. This parameter is reserved for future use. It must be a null pointer or point to a value of 0.

log_data_length

Direction: Input/Output
Type: Integer

A pointer to an integer variable containing the number of bytes in the log_data variable, or the maximum number of data bytes available to retrieve for a particular log inside the coprocessor. This parameter may be unused, but must always be declared. Table 40 describes the meaning of this parameter for the different requested services.

Table 40. Meaning of log_data_length for the requested service (keyword).

<table>
<thead>
<tr>
<th>Service requested</th>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCALGLEN</td>
<td>Output</td>
<td>Specifies the maximum number of CCA log data bytes available to retrieve inside the Coprocessor (the log_number_or_level parameter is ignored). No data is returned in the log_data variable.</td>
</tr>
</tbody>
</table>
Table 40. Meaning of log_data_length for the requested service (keyword) (continued).

<table>
<thead>
<tr>
<th>Service requested</th>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETCCALG</td>
<td>Input/Output</td>
<td>On input, this variable specifies the number of bytes available in the log_data variable. On output, the variable is updated to the number of bytes of CCA log data returned in the log_data variable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. To determine the maximum number of bytes that the current CCA log data buffer contains, use the CCALGLEN keyword. The maximum is less than or equal to 1024.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. The actual amount of data returned is variable and can be less than the maximum, depending on the amount of log messages issued so far.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. On input, it is acceptable to specify a value less than the total size of the available CCA log data buffer. On output, the end of the data buffer is truncated as needed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. On input, it is acceptable to specify a value greater than the total size of the available CCA log data buffer. On output, the length is adjusted to the total size.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5. Interpretation of the data is defined internally by IBM. The data may or may not be human-readable. The data may contain truncated messages or partial messages that in some way conflict with a given editor.</td>
</tr>
</tbody>
</table>

| GETLGLVL          | Input     | The value is ignored. |
| GETSYSLOG         | Input/Output | On input, this variable specifies the number of bytes available in the log_data variable. On output, the variable is updated to the number of bytes of SYSLOG data returned in the log_data variable (for the given log number as specified by the log_number_or_level parameter). |
|                   |           | **Note:** |
|                   |           | 1. To determine the maximum number of bytes that the specified SYSLOG data buffer contains, use the SYSLGLEN keyword. The maximum that can be returned is 16384. |
|                   |           | 2. The actual amount of data returned is variable and can be less than the maximum, depending on the amount of log messages issued so far. |
|                   |           | 3. On input, it is acceptable to specify a value less than the total size of the available SYSLOG data buffer. On output, the end of the data buffer is truncated as needed. |
|                   |           | 4. On input, it is acceptable to specify a value greater than the total size of the available SYSLOG data buffer. On output, the length is adjusted to the total size. |
|                   |           | 5. Interpretation of the data is defined internally by IBM. The data may or may not be human-readable. The data may contain truncated messages or partial messages that in some way conflict with a given editor. |

| SETLGLVL          | Input     | The value is ignored. |
| SYSLGLEN          | Output    | Specifies the maximum number of SYSLOG data bytes available to retrieve inside the coprocessor (for the given log number as specified by the log_number_or_level parameter). No data is returned in the log_data variable. |

**log_data**

**Direction:** Input/Output  
**Type:** String

A pointer to a string variable containing the returned log data when keyword GETCCALG or GETSYSLOG is specified in the rule array. This parameter may be unused, but must always be declared.
Log Query (CSUALGQ)

reserved1_length
- **Direction:** Input/Output
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the reserved1 variable. This parameter must be a null pointer or point to a value of 0.

reserved1
- **Direction:** Input/Output
- **Type:** String

This parameter is a pointer to a string variable. It is reserved for future use.

reserved2_length
- **Direction:** Input/Output
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the reserved2 variable. This parameter must be a null pointer or point to a value of 0.

reserved2
- **Direction:** Input/Output
- **Type:** String

This parameter is a pointer to a string variable. It is reserved for future use.

**Restrictions**

The restrictions for CSUALGQ.

None.

**Required commands**

The CSUALGQ required commands.

The Log Query verb requires the following commands to be enabled in the active role:

*Table 41. Required commands for the Log Query verb.*

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Log number value</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCALGLEN, GETCCALG</td>
<td>Ignored</td>
<td>X’0035’</td>
<td>Log Query: CCA</td>
</tr>
<tr>
<td>GETSYSLG, SYSGLGEN</td>
<td>Ignored</td>
<td>X’0034’</td>
<td>Log Query: System</td>
</tr>
<tr>
<td>SETLGLVL</td>
<td>4</td>
<td>X’0036’</td>
<td>Log Query: Set Log Level -4-</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>X’0037’</td>
<td>Log Query: Set Log Level -8-</td>
</tr>
</tbody>
</table>
Usage notes

Usage notes for CSUALGQ.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSUALGQJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSUALGQ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber log_number_or_level,
    hikmNativeNumber reserved0,
    hikmNativeNumber log_data_length,
    byte[] log_data,
    hikmNativeNumber reserved1_length,
    byte[] reserved1,
    hikmNativeNumber reserved2_length,
    byte[] reserved2);
```

Master Key Process (CSNBMPK)

The Master Key Process verb operates on the three master-key registers: new, current, and old.

Use the verb to perform the following services:

- Clear the new and clear the old master-key registers.
- Generate a random master-key value in the new master-key register.
- XOR a clear value as a key part into the new master-key register.
- Set the master key, which transfers the current master-key to the old master-key register, and the new master-key to the current master-key register. It then clears the new master-key register.

You can choose to process either the symmetric or asymmetric registers by specifying the **SYM-MK** and the **ASYM-MK** `rule_array` keywords.

**Tip:** Before starting to load new master-key information, ensure the new master-key register is cleared. Do this by using the **CLEAR** keyword in the `rule_array`.

To form a master key from key parts in the new master-key register, use the verb several times to complete the following tasks:

- Clear the register, if it is not already clear.
- Load the first key part.
- Load any middle key parts, calling the verb once for each middle key part.
- Load the last key part.
- SET or confirm a master key for which the last key part has been loaded into the new master-key register.
Master Key Process (CSNBMKP)

For the SYM-MK, the low-order bit in each byte of the key is used as parity for the remaining bits in the byte. Each byte of the key part must contain an odd number of one bits. If this is not the case, a warning is issued. The product maintains odd parity on the accumulated symmetric master-key value.

When the last master key part is entered, this additional processing is performed:

- If any two of the 8-byte parts of the new master-key have the same value, a warning is issued. Do not ignore this warning. Do not use a key with this property.
- If any of the 8-byte parts of the new master-key compares equal to one of the weak DES-keys, the verb fails with return code 8, reason code 703. See "Questionable DES keys" on page 160 for a list of these weak keys. A parity-adjusted version of the asymmetric master-key is used to look for weak keys.

If an AES, DES or PKA key storage exists, the header record of each key storage is updated with the verification pattern of the new, current master key.

Format

The format of CSNBMKP.

```c
CSNBMKP(  
  return_code,  
  reason_code,  
  exit_data_length,  
  exit_data,  
  rule_array_count,  
  rule_array,  
  key_part)
```

Parameters

The parameter definitions for CSNBMKP.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

`rule_array_count`

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1, 2, or 3.

`rule_array`

Direction: Input
Type: String array

The `rule_array` parameter is a pointer to a string variable containing an array of keywords. The keywords are eight bytes in length and must be left-aligned and padded on the right with space characters. The `rule_array` keywords are described in Table 42.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cryptographic component</td>
<td>(Optional)</td>
</tr>
</tbody>
</table>

Table 42. Keywords for Master Key Process control information
### Table 42. Keywords for Master Key Process control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAPTER</td>
<td>Specifies the coprocessor. This is the default.</td>
</tr>
<tr>
<td>AES-MK</td>
<td>Specifies operation with the AES master-key registers.</td>
</tr>
<tr>
<td>APKA-MK</td>
<td>Specifies operation with the APKA master-key registers. This keyword was introduced with CCA 4.1.0.</td>
</tr>
<tr>
<td>ASYM-MK</td>
<td>Specifies operation with the asymmetric master-key registers.</td>
</tr>
<tr>
<td>SYM-MK</td>
<td>Specifies operation with the symmetric master-key registers.</td>
</tr>
</tbody>
</table>

**Master key register class** (One, required)

See Note at the end of this table.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR</td>
<td>Specifies to clear the NMK register.</td>
</tr>
<tr>
<td>FIRST</td>
<td>Specifies to load the first <em>key_part</em>.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>Specifies to XOR the second, third, or other intermediate <em>key_part</em> into the NMK register.</td>
</tr>
<tr>
<td>LAST</td>
<td>Specifies to XOR the last <em>key_part</em> into the NMK register.</td>
</tr>
<tr>
<td>SET</td>
<td>Specifies to advance the CMK to the OMK register, to advance the NMK to the CMK register, and to clear the NMK register.</td>
</tr>
</tbody>
</table>

**Note:** The *master-key register class* is not optional for Linux on Z. There is no default for this environment. If a suitable keyword is not specified, return code 8 with reason code 33 will be returned.

### key_part

**Direction:** Input  
**Type:** String

A pointer to a string variable containing a 168-bit or 192-bit clear key-part used when you specify one of the keywords FIRST, MIDDLE, or LAST. If you use the CLEAR or SET keywords, the information in the variable is ignored, but you must declare the variable.

### Restrictions

The restrictions for CSNBMKP.

**General restrictions:**

- You must set up the groups for the users who will be loading the master keys to the cards. Each part of the load process is owned by a different Linux group created by the RPM install procedure, and verified in the host library implementing the API allowing master key processing. To complete a specific step, the user must have membership in the proper group. See [Master key load](#) (Step 7 on page 1113).

- When writing your own application using this verb, you must link it with the /usr/lib64/libcsulccamk.so library.

### Required commands

The CSNBMKP required commands.

This verb requires the following commands to be enabled in the active role based on the master-key class and master-key operation:
### Master Key Process (CSNBMKP)

<table>
<thead>
<tr>
<th>Master-key operation</th>
<th>Master-key class</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR</td>
<td>AES-MK</td>
<td>X'0124'</td>
<td>AES Master Key - Clear new master key register</td>
</tr>
<tr>
<td></td>
<td>SYM-MK</td>
<td>X'0032'</td>
<td>DES Master Key - Clear new master key register</td>
</tr>
<tr>
<td></td>
<td>ASYM-MK</td>
<td>X'0060'</td>
<td>RSA Master Key - Clear new master key register</td>
</tr>
<tr>
<td></td>
<td>APKA-MK</td>
<td>X'031F'</td>
<td>ECC Master Key - Clear new master key register</td>
</tr>
<tr>
<td>FIRST</td>
<td>AES-MK</td>
<td>X'0125'</td>
<td>AES Master Key - Load first key part</td>
</tr>
<tr>
<td></td>
<td>SYM-MK</td>
<td>X'0018'</td>
<td>DES Master Key - Load first key part</td>
</tr>
<tr>
<td></td>
<td>ASYM-MK</td>
<td>X'0053'</td>
<td>RSA Master Key - Load first key part</td>
</tr>
<tr>
<td></td>
<td>APKA-MK</td>
<td>X'0320'</td>
<td>ECC Master Key - Load first key part</td>
</tr>
<tr>
<td>MIDDLE or LAST</td>
<td>AES-MK</td>
<td>X'0126'</td>
<td>AES Master Key - Combine key parts</td>
</tr>
<tr>
<td></td>
<td>SYM-MK</td>
<td>X'0019'</td>
<td>DES Master Key - Combine key parts</td>
</tr>
<tr>
<td></td>
<td>ASYM-MK</td>
<td>X'0054'</td>
<td>RSA Master Key - Combine key parts</td>
</tr>
<tr>
<td></td>
<td>APKA-MK</td>
<td>X'0321'</td>
<td>ECC Master Key - Combine key parts</td>
</tr>
<tr>
<td>SET</td>
<td>AES-MK</td>
<td>X'0128'</td>
<td>AES Master Key - Set master key</td>
</tr>
<tr>
<td></td>
<td>SYM-MK</td>
<td>X'001A'</td>
<td>DES Master Key - Set master key</td>
</tr>
<tr>
<td></td>
<td>ASYM-MK</td>
<td>X'0057'</td>
<td>RSA Master Key - Set master key</td>
</tr>
<tr>
<td></td>
<td>APKA-MK</td>
<td>X'0322'</td>
<td>ECC Master Key - Set master key</td>
</tr>
</tbody>
</table>

### Usage notes

Usage notes for CSNBMKP.

This verb is not impacted by the AUTOSELECT option. See "Verbs that ignore AUTOSELECT" on page 12 for more information.

### Questionable DES keys

These keys are considered questionable DES keys, and so should probably not be used when entering SYM-MK or ASYM-MK master keys.

```
01 01 01 01 01 01 01 01 /* weak */
FE FE FE FE FE FE FE FE /* weak */
1F 1F 1F 1F 0E 0E 0E 0E /* weak */
E0 E0 E0 E0 F1 F1 F1 F1 /* weak */
01 FE 01 FE 01 FE 01 FE /* semi-weak */
FE 01 FE 01 FE 01 FE 01 FE /* semi-weak */
1F E0 1F E0 0E 0E 0E 0E /* semi-weak */
E0 IF 01 E0 0F F1 F1 F1 /* semi-weak */
01 E0 01 E0 01 E0 01 F1 /* semi-weak */
E0 01 E0 01 F1 01 F1 01 /* semi-weak */
1F FE 1F FE 0E FE 0E FE /* semi-weak */
FE 1F FE 1F FE 0E FE 0E /* semi-weak */
01 1F 01 1F 01 OE 01 OE /* semi-weak */
1F 01 IF 01 0E 01 0E /* semi-weak */
E0 FE E0 FE F1 FE F1 /* semi-weak */
FE E0 FE E0 FE FE F1 FE /* semi-weak */
1F IF 01 01 0E 0E 01 01 /* possibly semi-weak */
01 IF 1F 01 01 0E 0E 01 /* possibly semi-weak */
1F 01 1F 01 0E 01 0E /* possibly semi-weak */
01 01 1F 01 01 0E 0E /* possibly semi-weak */
E0 E0 01 01 F1 F1 F1 01 /* possibly semi-weak */
FE FE 01 01 FE FE 01 F1 /* possibly semi-weak */
FE 01 E0 1F FE 01 FE 01 /* possibly semi-weak */
E0 IF 01 01 F1 FE F1 FE /* possibly semi-weak */
01 IF 01 01 FE OE 01 01 /* possibly semi-weak */
E0 IF 01 01 FE 01 FE 01 /* possibly semi-weak */
```
These tests, selected by a cryptographic operational tests. Known-answer tests for periodic testing of cryptographic functions of DES encryption and decryption, RSA encryption and decryption, HMAC and SHA-1 hashing.

Random Number Tests (CSUARNT)

The Random Number Tests verb invokes the USA NIST FIPS PUB 140-1 specified cryptographic operational tests.

These tests, selected by a rule_array keyword, consist of:

- Tests of the random-number generation output using procedures specified in The NIST SP 800-90A Deterministic Random Bit Generator Validation System (DRBGVS).
- Known-answer tests for periodic testing of cryptographic functions of DES encryption and decryption, RSA encryption and decryption, HMAC and SHA-1 hashing.
- Known-answer tests for AES Galois/Counter Mode (GCM).

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBMKPJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

public native void CSNBMKPJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] key_part);
Random Number Tests (CSUARNT)

The tests are performed three times. If there is any test failure, the verb returns return code 4 and reason code 1.

Format

The format of CSUARNT.

```c
CSUARNT(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array)
```

Parameters

The parameter definitions for CSUARNT.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the `rule_array` parameter. This value must be 1.

**rule_array**

Direction: Input
Type: String array

A pointer to a string variable containing an array of keywords. The keywords are eight bytes in length and must be left-aligned and padded on the right with space characters. The `rule_array` keywords are described in Table 43.

### Table 43. Keywords for Random Number Tests control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test selection (One required).</td>
<td></td>
</tr>
<tr>
<td>FIPS-RNT</td>
<td>Perform the FIPS 140-1 specified test on the random number generation output.</td>
</tr>
<tr>
<td>KAT</td>
<td>Perform the FIPS 140-1 specified known-answer tests on DES, RSA, and SHA-1.</td>
</tr>
<tr>
<td>KAT2</td>
<td>Specifies to perform known-answer tests of AES GCM functions.</td>
</tr>
</tbody>
</table>

Restrictions

The restrictions for CSUARNT.

None.
Required commands

The CSUARNT required commands.

None.

Usage notes

Usage notes for CSUARNT.

This verb is not impacted by the AUTOSELECT option. See “Verbs that ignore AUTOSELECT” on page 12 for more information.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSUARNTJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSUARNTJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array);
```
Random Number Tests (CSUARNT)
Chapter 8. Managing AES, DES, and HMAC cryptographic keys

A group of verbs that generate and maintain AES, DES, and HMAC cryptographic keys.

Using CCA, you can generate keys using the Key Generate verb. CCA provides a number of verbs to assist you in managing and distributing AES, DES, and HMAC keys, generating random numbers, and maintaining the key storage files.

The following verbs are described:

- “Clear Key Import (CSNBCKI)” on page 166
- “Multiple Clear Key Import (CSNBCKM)” on page 167
- “Control Vector Generate (CSNBCVG)” on page 170
- “Control Vector Translate (CSNBCVT)” on page 174
- “Cryptographic Variable Encipher (CSNBCVE)” on page 178
- “Data Key Export (CSNBDKX)” on page 180
- “Data Key Import (CSNBDKM)” on page 181
- “Diversified Key Generate (CSNBDKG)” on page 183
- “Diversified Key Generate2 (CSNBDKG2)” on page 189
- “EC Diffie-Hellman (CSNDEDH)” on page 195
- “Key Export (CSNBKEX)” on page 209
- “Key Generate (CSNBKGN)” on page 211
- “Key Generate2 (CSNBKGN2)” on page 221
- “Key Import (CSNBKIM)” on page 234
- “Key Part Import (CSNBKPI)” on page 237
- “Key Part Import2 (CSNBKPI2)” on page 241
- “Key Test (CSNBKTY)” on page 245
- “Key Test2 (CSNBKTY2)” on page 249
- “Key Test Extended (CSNBKTYX)” on page 254
- “Key Token Build (CSNBKTB)” on page 259
- “Key Token Build2 (CSNBKTB2)” on page 264
- “Key Token Change (CSNBKTC)” on page 301
- “Key Token Change2 (CSNBKTC2)” on page 304
- “Key Token Parse (CSNBKTP)” on page 307
- “Key Token Parse2 (CSNBKTP2)” on page 311
- “Key Translate (CSNBKTR)” on page 321
- “Key Translate2 (CSNBKTR2)” on page 323
- “PKA Decrypt (CSNPDKD)” on page 327
- “PKA Encrypt (CSNPDKE)” on page 330
- “Prohibit Export (CSNBPEX)” on page 334
- “Prohibit Export Extended (CSNBPEXX)” on page 335
- “Restrict Key Attribute (CSNBRKA)” on page 337
- “Random Number Generate (CSNBRNG)” on page 341

© Copyright IBM Corp. 2007, 2018
Clear Key Import (CSNBCKI)

Use the Clear Key Import verb to import a clear DATA key that is to be used to encipher or decipher data.

This verb can import only DATA keys. The Clear Key Import verb accepts an 8-byte clear DATA key, enciphers it under the master key, and returns the encrypted DATA key in operational form in an internal key token.

If the clear key value does not have odd parity in the low-order bit of each byte, the verb returns a warning value in the reason_code parameter. This verb does not adjust the parity of the key.

Note: To import 16-byte or 24-byte DATA keys, use the Multiple Clear Key Import verb that is described in “Multiple Clear Key Import (CSNBCKM)” on page 167.

Format

The format of CSNBCKI.

```
CSNBCKI(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    clear_key,
    target_key_identifier)
```

Parameters

The parameter definitions for CSNBCKI.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

**clear_key**

- **Direction:** Input
- **Type:** String

The clear_key specifies the 8-byte clear key value to import.

**target_key_identifier**

- **Direction:** Input/Output
- **Type:** String

A 64-byte string that is to receive the internal key token. “Key tokens, key labels, and key identifiers” on page 23 describes the internal key token.
Restrictions
The restrictions for CSNBCKI.

None.

Required commands
The CSNBCKI required commands.

This verb requires the Clear Key Import/Multiple Clear Key Import - DES command (offset X'00C3') to be enabled in the active role.

Note: A role with offset X'00C3' enabled can also use the Multiple Clear Key Import verb with the DES algorithm.

Usage notes
Usage notes for CSNBCKI.

None.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNBCKIJ.

See “Building Java applications using the CCA JNI” on page 28.

Format
public native void CSNBCKIJ(
  hikmNativeNumber return_code,
  hikmNativeNumber reason_code,
  hikmNativeNumber exit_data_length,
  byte[] exit_data,
  byte[] clear_key,
  byte[] target_key_identifier);

Multiple Clear Key Import (CSNBCKM)

Use the Multiple Clear Key Import verb to import a clear single, double, or triple-length DATA key that is to be used to encipher or decipher data.

This verb can import only DATA keys. Multiple Clear Key Import accepts a clear DATA key, enciphers it under the master key, and returns the encrypted DATA key in operational form in an internal key token.

Format
The format of CSNBCKM.

CSNBCKM(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  clear_key_length,
  clear_key,
  target_key_identifier)
Multiple Clear Key Import (CSNBCKM)

Parameters

The parameter definitions for CSNBCKM.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0, 1, 2, or 3.

**rule_array**

Direction: Input
Type: String array

Zero or one keyword that supplies control information to the verb. The keyword must be in eight bytes of contiguous storage, left-aligned and padded on the right with blanks. The `rule_array` keywords are described in Table 44.

Table 44. Keywords for Multiple Clear Key Import control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm  (On, optional)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>The key should be enciphered under the master key as an AES key.</td>
</tr>
<tr>
<td>DES</td>
<td>The key should be enciphered under the master key as a DES key. This is the default.</td>
</tr>
<tr>
<td>Key-wrapping method (One, optional)</td>
<td></td>
</tr>
<tr>
<td>USECONFIG</td>
<td>This is the default. Specifies to wrap the key using the configuration setting for the default wrapping method. The default wrapping method configuration setting may be changed using the TKE. This keyword is ignored for AES keys.</td>
</tr>
<tr>
<td>WRAP-ENH</td>
<td>Specifies to wrap the key using the legacy wrapping method. This keyword is ignored for AES keys. This keyword was introduced with CCA 4.1.0.</td>
</tr>
<tr>
<td>WRAP-ECB</td>
<td>Specifies to wrap the key using the enhanced wrapping method. Valid only for DES keys. This keyword was introduced with CCA 4.1.0.</td>
</tr>
<tr>
<td>Translation control (Optional). This is valid only with key-wrapping method WRAP-ENH or with USECONFIG when the default wrapping method is WRAP-ENH. This option cannot be used on a key with a control vector valued to binary zeros. This keyword was introduced with CCA 4.1.0.</td>
<td></td>
</tr>
<tr>
<td>ENH-ONLY</td>
<td>Specifies to restrict the key from being wrapped with the legacy wrapping method after it has been wrapped with the enhanced wrapping method. Sets bit 56 (ENH-ONLY) of the control vector to B'1'.</td>
</tr>
</tbody>
</table>

**clear_key_length**

Direction: Input
Type: Integer

The `clear_key_length` specifies the length of the clear key value to import. This length must be 8, 16, or 24.

**clear_key**
Multiple Clear Key Import (CSNBCKM)

**Direction:** Input
**Type:** String

The `clear_key` specifies the clear key value to import.

**target_key_identifier**

**Direction:** Output
**Type:** String

A 64-byte string that is to receive the internal key token. Chapter 19, “Key token formats,” on page 847 describes the key tokens.

**Restrictions**

The restrictions for CSNBCKM.

None.

**Required commands**

The CSNBCKM required commands.

This verb requires the following commands to be enabled in the active role based on the algorithm or key-wrapping method:

<table>
<thead>
<tr>
<th>Algorithm or method</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>X'0129'</td>
<td>Multiple Clear Key Import/Multiple Secure Key Import - AES</td>
</tr>
<tr>
<td>DES</td>
<td>X'00C3'</td>
<td>Clear Key Import/Multiple Clear Key Import - DES</td>
</tr>
<tr>
<td>WRAP-ECB or WRAP-ENH used, and default key-wrapping method setting does not match keyword</td>
<td>X'0141'</td>
<td>Multiple Clear Key Import - Allow wrapping override keywords</td>
</tr>
</tbody>
</table>

**Note:** Note: A role with offset X'00C3' can also use the Clear Key Import verb.

**Usage notes**

Usage notes for CSNBCKM.

This verb produces an internal DATA token with a control vector which is usable on the Cryptographic Coprocessor Feature. If a valid internal token is supplied as input to the verb in the `target_key_identifier` field, that token's control vector will not be used in the encryption of the clear key value.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBCKMJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBCKMJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
)```

Chapter 8. AES, DES, and HMAC cryptographic keys  169
Control Vector Generate (CSNBCVG)

The Control Vector Generate verb builds a control vector from keywords specified by the key_type and rule_array parameters.

Format

The format of CSNBCVG.

```
CSNBCVG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_type,
    rule_array_count,
    rule_array,
    reserved,
    control_vector)
```

Parameters

The parameter definitions for CSNBCVG.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see "Parameters common to all verbs" on page 22.

key_type

Direction: Input  
Type: String

A string variable containing a keyword for the key type. The keyword is eight bytes in length, left-aligned, and padded on the right with space characters. It is taken from the following list:

- CIPHER
- CVARXCL
- DKYGENKY
- MAC
- CIPHERXI
- CVARXCVR
- ENCIPHER
- MACVER
- CIPHERXL
- DATA
- EXPORTER
- OKEYXLAT
- CIPHERXO
- DATAC
- IKEYXLAT
- OPINENC
- CVARDEC
- DATAM
- IMPORTER
- PINGEN
- CVARENC
- DATAMV
- IPINENC
- PINVER
- CVARPINE
- DECIPHER
- KEYGENKY
- SECMSG

For information on the meaning of the key types, see Table 6 on page 44.

rule_array_count

Direction: Input  
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable.

rule_array

- multiple clear key import (CSNBCKM)
Keywords that provide control information to the verb. Each keyword is left-aligned in 8-byte fields, and padded on the right with blanks. All keywords must be in contiguous storage. “Key Token Build (CSNBKTB)” on page 259 illustrates the key type and key usage keywords that can be combined in the Control Vector Generate and Key Token Build verbs to create a control vector.

See Figure 3 on page 48 for the key usage keywords that can be specified for a given key type. The rule_array keywords are described in Table 45.

Table 45. Keywords for Control Vector Generate control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key usage keywords:</strong> These keywords allow the key to be used with a callable service, restrict the key to a single algorithm, or allow the key to be used for a specific function (One, required)</td>
<td></td>
</tr>
<tr>
<td><strong>Key encrypting keys</strong></td>
<td></td>
</tr>
<tr>
<td>OPIM</td>
<td>IMPORTEr keys that have this attribute can be used in the CSNBKGN service when the key form is OPIM.</td>
</tr>
<tr>
<td>IMEX</td>
<td>IMPORTEr and EXPORTER keys that have this attribute can be used in the CSNBKGN service when the key form is IMEX.</td>
</tr>
<tr>
<td>IMIM</td>
<td>IMPORTEr keys that have this attribute can be used in the CSNBKGN service when the key form is IMIM.</td>
</tr>
<tr>
<td>IMPORT</td>
<td>IMPORTEr keys that have this attribute can be used to import a key in the CSNBKIM service.</td>
</tr>
<tr>
<td>OPEX</td>
<td>EXPORTER keys that have this attribute can be used in the CSNBKGN service when the key form is OPEX.</td>
</tr>
<tr>
<td>EXEX</td>
<td>EXPORTER keys that have this attribute can be used in the CSNBKGN service when the key form is EXEX.</td>
</tr>
<tr>
<td>EXPORT</td>
<td>EXPORTER keys that have this attribute can be used to export a key in the CSNBKEX service.</td>
</tr>
<tr>
<td>XLATE</td>
<td>IMPORTEr and EXPORTER keys that have this attribute can be used in the CSNBKTR and CSNBKTR2 services.</td>
</tr>
<tr>
<td>ANY</td>
<td>This key attribute has been discontinued. Its usage is allowed for backward compatibility reasons only.</td>
</tr>
<tr>
<td>NOT-KEK</td>
<td>This key attribute has been discontinued. Its usage is allowed for backward compatibility reasons only.</td>
</tr>
<tr>
<td>DATA</td>
<td>This key attribute has been discontinued. Its usage is allowed for backward compatibility reasons only.</td>
</tr>
<tr>
<td>LMTD-KEK</td>
<td>This key attribute has been discontinued. Its usage is allowed for backward compatibility reasons only.</td>
</tr>
<tr>
<td><strong>MAC keys</strong></td>
<td></td>
</tr>
<tr>
<td>ANY-MAC</td>
<td>A key with this attribute can be used with any service that uses MAC keys.</td>
</tr>
<tr>
<td>CVVKEY-A</td>
<td>Restricts the usage of the key to single-length key-A key or double-length key-A and key-B keys for the CSNBCSG and CSNBCSV services.</td>
</tr>
<tr>
<td>CVVKEY-B</td>
<td>Restricts the usage of the key to single-length key-B key for the CSNBCSG and CSNBCSV services.</td>
</tr>
<tr>
<td>AMEX-CSC</td>
<td>This key attribute has been discontinued. Its usage is allowed for backward compatibility reasons only.</td>
</tr>
<tr>
<td>ANSIX9.9</td>
<td>This key attribute has been discontinued. Its usage is allowed for backward compatibility reasons only.</td>
</tr>
</tbody>
</table>
### Table 45. Keywords for Control Vector Generate control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data operation keys</strong></td>
<td></td>
</tr>
<tr>
<td>When the key type is SECMSG, either SMKEY or SMPIN must be specified in the rule array.</td>
<td></td>
</tr>
<tr>
<td>SMKEY</td>
<td>A key with this attribute can be used to encrypt keys in an EMV secure message.</td>
</tr>
<tr>
<td>SMPIN</td>
<td>A key with this attribute can be used to encrypt PINs in an EMV secure message.</td>
</tr>
<tr>
<td><strong>PIN keys</strong></td>
<td></td>
</tr>
<tr>
<td>NO-SPEC</td>
<td>The key is not restricted to a specific PIN-calculation method.</td>
</tr>
<tr>
<td>IBM-PIN</td>
<td>The key can be used with the IBM 3624 PIN-calculation method.</td>
</tr>
<tr>
<td>IBM-PINO</td>
<td>The key can be used with the IBM 3624 PIN-calculation method with offset processing.</td>
</tr>
<tr>
<td>GBP-PIN</td>
<td>The key can be used with the IBM German Bank Pool PIN-calculation method.</td>
</tr>
<tr>
<td>GBP-PINO</td>
<td>The key can be used with the IBM German Bank Pool PIN-calculation method with institution-PIN input or output.</td>
</tr>
<tr>
<td>VISA-PVV</td>
<td>The key can be used with the Visa PVV PIN-calculation method.</td>
</tr>
<tr>
<td>INBK-PIN</td>
<td>The key can be used with the InterBank PIN-calculation method.</td>
</tr>
<tr>
<td>NOOFFSET</td>
<td>Indicates that a PINGEN or PINVER key cannot be used to generate or verify of a PIN when an offset process is requested.</td>
</tr>
<tr>
<td>CPINGEN</td>
<td>The key can be used with the CSNBPGN service.</td>
</tr>
<tr>
<td>CPINGENA</td>
<td>The key can be used with the CSNBCPA service.</td>
</tr>
<tr>
<td>EPINGEN</td>
<td>The key can be used with the CSNBEPG service.</td>
</tr>
<tr>
<td>EPINVER</td>
<td>The key can be used with the CSNBPVR service.</td>
</tr>
<tr>
<td>CPINENC</td>
<td>The key can be used with the CSNBCPE service.</td>
</tr>
<tr>
<td>REFORMAT</td>
<td>The key can be used with the CSNBPTR service in the REFORMAT mode.</td>
</tr>
<tr>
<td>TRANSLATE</td>
<td>The key can be used with the CSNBPTR service in the TRANSLATE mode.</td>
</tr>
<tr>
<td><strong>Key-generating keys</strong></td>
<td></td>
</tr>
<tr>
<td>When the key type is KEYGENKY, either CLR8-ENC or UKPT must be specified in the rule array.</td>
<td></td>
</tr>
<tr>
<td>CLR8-ENC</td>
<td>The key can be used to multiply-encrypt 8 bytes of clear data with a generating key.</td>
</tr>
<tr>
<td>DALL</td>
<td>The key can be used to generate keys with the following key types: DATA, DATAC, DATAM, DATAMV, DMKEY, DMPIN, EXPORTER, IKEYXLAT, IMPORTER, MAC, MACVER, OKEYXLAT, and PINVER.</td>
</tr>
<tr>
<td>DDATA</td>
<td>The key can be used to generate a single-length or double-length DATA or DATAC key.</td>
</tr>
<tr>
<td>DEXP</td>
<td>The key can be used to generate an EXPORTER or an OKEYXLAT key.</td>
</tr>
<tr>
<td>DIMP</td>
<td>The key can be used to generate an IMPORTER or an IKEYXLAT key.</td>
</tr>
<tr>
<td>DMAC</td>
<td>The key can be used to generate a MAC or DATAM key.</td>
</tr>
<tr>
<td>DMKEY</td>
<td>The key can be used to generate a SECMSG with a SMKEY secure messaging key for encrypting keys.</td>
</tr>
<tr>
<td>DMPIN</td>
<td>The key can be used to generate a SECMSG with a SMPIN secure messaging key for encrypting PINs.</td>
</tr>
<tr>
<td>DMV</td>
<td>The key can be used to generate a MACVER or DATAMV key.</td>
</tr>
<tr>
<td>DPV</td>
<td>The key can be used to generate a PINVER key.</td>
</tr>
<tr>
<td>DKYL0</td>
<td>A DKYGENKY key with this subtype can be used to generate a key based on the key-usage bits.</td>
</tr>
</tbody>
</table>
Table 45. Keywords for Control Vector Generate control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DKYL1</td>
<td>A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL0.</td>
</tr>
<tr>
<td>DKYL2</td>
<td>A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL1.</td>
</tr>
<tr>
<td>DKYL3</td>
<td>A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL2.</td>
</tr>
<tr>
<td>DKYL4</td>
<td>A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL3.</td>
</tr>
<tr>
<td>DKYL5</td>
<td>A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL4.</td>
</tr>
<tr>
<td>DKYL6</td>
<td>A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL5.</td>
</tr>
<tr>
<td>DKYL7</td>
<td>A DKYGENKY key with this subtype can be used to generate a DKYGENKY key with a subtype of DKYL6.</td>
</tr>
<tr>
<td>UKPT</td>
<td>The key can be used to derive operational keys.</td>
</tr>
</tbody>
</table>

Key management keywords

These keywords are valid with all key types. The keywords are used to allow or disallow key management functions.

Key lengths

| MIXED | Indicates that the key can be either a replicated single-length key or a double-length key with two different, random 8-byte values. |
| SINGLE, KEYLN8 | Specifies the key as a single-length key. |
| DOUBLE, KEYLN16 | Specifies the key as a double-length key. |
| DOUBLE-O | Specifies the key as a double-length key with guaranteed different key values. |

Miscellaneous attributes

| COMP-TAG | The key can be used with PCI-HSM compliant applications. |
| ENH-ONLY | Prohibits the key from being wrapped with the legacy method after it has been wrapped with the enhanced method. |
| KEY-PART | Specifies the control vector is for a key part. |
| NO-XPORT | Prohibits the key from being exported by Key_Export or Data_Key_Export. |
| NCOMPTAG | The key cannot be used with PCI-HSM compliant applications. |
| NOT31XPT | Prohibits the key from being exported by the Key_Export_to_TR31 verb. |
| T31XPTOK | Permits the key to be exported by the Key_Export_to_TR31 verb. |
| XPORT-OK | Permits the key to be exported by Key_Export or Data_Key_Export. Also permits the key to be exported by the Key_Export_to_TR31 verb, unless NOT31XPT is enabled. |

Note:

1. When the KEYGENKY key type is coded, either CLR8-ENC or UKPT must be specified in rule_array.
2. When the SECMSG key_type is coded, either SMKEY or SMPIN must be specified in the rule_array.
3. Keyword ENH-ONLY was introduced with CCA 4.1.0.

reserved
Control Vector Generate (CSNBCVG)

Direction: Input
Type: String

The reserved parameter must be a variable of eight bytes of X'00'.

control_vector

Direction: Output
Type: String

A 16-byte string variable in application storage where the verb returns the generated control vector.

Restrictions

The restrictions for CSNBCVG.

None.

Required commands

The CSNBCVG required commands.

None.

Usage notes

Usage notes for CSNBCVG.

See the key_type parameter on page "key_type" on page 260 for an illustration of key type and key usage keywords that can be combined in the Control Vector Generate and Key Token Build verbs to create a control vector.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBCVGJ.

See "Building Java applications using the CCA JNI" on page 28.

Format

public native void CSNBCVGJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] key_type,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] reserved,
    byte[] control_vector);

Control Vector Translate (CSNBCVT)

The Control Vector Translate verb changes the control vector used to encipher an external DES key.

Detailed information about control vectors and how to use this verb can be found in Chapter 21, "Control vectors and changing control vectors with the Control Vector Translate verb," on page 989.
Control Vector Translate (CSNBCVT)

Format
The format of CSNBCVT.

```
CSNBCVT(  
  return_code,  
  reason_code,  
  exit_data_length,  
  exit_data,  
  kek_key_identifier,  
  source_key_token,  
  array_key_left_identifier,  
  mask_array_left,  
  array_key_right_identifier,  
  mask_array_right,  
  rule_array_count,  
  rule_array,  
  target_key_token)
```

Parameters
The parameter definitions for CSNBCVT.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

kek_key_identifier

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing an operational key-token or the key label of an operational key-token record containing the key-encrypting key. The control vector in the key token must specify the key type `IMPORTER`, `EXPORTER`, `IKEYXLAT`, or `OKEYXLAT`.

source_key_token

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing the external DES key-token with the key and control vector to be processed.

array_key_left_identifier

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing an operational DES key-token or a key label of an operational DES key-token record that deciphers the left mask-array. The key token must contain a control vector specifying a `CVARXCVL` key-type. The `CVARXCVL` key must be single length.

mask_array_left

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing the mask array enciphered under the left-array key.

array_key_right_identifier

Chapter 8. AES, DES, and HMAC cryptographic keys 175
Control Vector Translate (CSNBCVT)

**mask_array_right**

Direction: Input
Type: String

A pointer to a string variable containing the mask array enciphered under the right-array key.

**rule_array_count**

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 0, 1, or 2.

**rule_array**

Direction: Input
Type: String array

A pointer to a string variable containing an array of keywords. The keywords are eight bytes in length and must be left-aligned and padded on the right with space characters. The rule_array keywords are described in Table 46.

### Table 46. Keywords for Control Vector Translate control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parity adjustment (One, optional)</td>
<td></td>
</tr>
<tr>
<td>ADJUST</td>
<td>Ensures that all target-key bytes have odd parity. This is the default.</td>
</tr>
<tr>
<td>NOADJUST</td>
<td>Prevents the parity of the target key from being altered.</td>
</tr>
</tbody>
</table>

**Key half processing mode** (One, optional)

<table>
<thead>
<tr>
<th>LEFT</th>
<th>Causes an 8-byte source key, or the left half of a 16-byte source key, to be processed with the result placed into both halves of the target key. This is the default.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIGHT</td>
<td>Causes the right half of a 16-byte source key to be processed with the result placed into only the right half of the target key. The left half of the target key is unchanged.</td>
</tr>
<tr>
<td>BOTH</td>
<td>Causes both halves of a 16-byte source key to be processed with the result placed into corresponding halves of the target key. When you use the BOTH keyword, the mask array must be able to validate the translation of both halves.</td>
</tr>
<tr>
<td>SINGLE</td>
<td>Causes the left half of the source key to be processed with the result placed into only the left half of the target. The right half of the target key is unchanged.</td>
</tr>
</tbody>
</table>

**target_key_token**

Direction: Input/Output
Type: String

A pointer to a string variable containing an external DES key-token with the new control vector. This key token contains the key halves with the new control vector.
Restrictions

The restrictions for CSNBCVT.

None.

Required commands

The CSNBCVT required commands.

This verb requires the Control Vector Translate command (offset X'00D6') to be enabled in the active role.

Usage notes

Usage notes for CSNBCVT.

Consider that Control Vector Translate represents the capability to translate, by definition, the limitations on the operations that a key can be used for, into a different set of limitations. The control vector is the heart of security against the misuse of keys that were defined for a specific purpose. The masks that control what the key can be translated into being able to do (the right and left masks) are themselves single-length (8-byte), and are encrypted with DES. Therefore, the protection against translating the key to have more power (or less power) than it did before are protected with single-DES. This reduces the security (somewhat) of a double-length DES key. You cannot decrypt the double-length key with this approach, or gain access to a key that you did not otherwise have the rights to use. But you can make a key which you already have access to, on a system you already have access to, more powerful than it was before if you can break single-DES.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBCVTJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBCVTJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] kek_key_identifier,
    byte[] source_key_token,
    byte[] array_key_left,
    byte[] mask_array_left,
    byte[] array_key_right,
    byte[] mask_array_right,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] target_key_token);
```
Cryptographic Variable Encipher (CSNBCVE)

Cryptographic Variable Encipher (CSNBCVE)

This verb is used to encrypt plaintext using a CVARENC key to produce ciphertext using the Cipher Block Chaining (CBC) method.

The plaintext must be a multiple of eight bytes in length.

Specify the following parameters to encrypt plaintext:

- An operational DES key-token or a key label of an operational DES key-token record that contains the key to be used to encrypt the plaintext with the c-variable_encrypting_key_identifier parameter. The control vector in the key token must specify the CVARENC key-type.
- The length of the plaintext, which is the same as the length of the returned ciphertext, with the text_length parameter. The plaintext must be a multiple of eight bytes in length.
- The plaintext with the plaintext parameter.
- The initialization vector with the initialization_vector parameter.
- A variable for the returned ciphertext with the ciphertext parameter. The length of this field is specified with the text_length variable.

This verb does the following:
- Uses the CVARENC key and the initialization value with the CBC method to encrypt the plaintext.
- Returns the encrypted plaintext in the variable pointed to by the ciphertext parameter.

Format

The format of CSNBCVE.

```plaintext
CSNBCVE(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    c_variable_encrypting_key_identifier,
    text_length,
    plain_text,
    initialization_vector,
    cipher_text)
```

Parameters

The parameter definitions for CSNBCVE.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see "Parameters common to all verbs" on page 22.

**c_variable_encrypting_key_identifier**

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing an operational DES key-token or a key label of an operational DES key-token record. The key token must contain a control vector that specifies a CVARENC key-type.

**text_length**
Cryptographic Variable Encipher (CSNBCVE)

**Direction:** Input
**Type:** Integer

A pointer to an integer variable containing the length of the `plain_text` variable and the `cipher_text` variable.

**plain_text**

**Direction:** Input
**Type:** String

A pointer to a string variable containing the plaintext to be encrypted.

**initialization_vector**

**Direction:** Input
**Type:** String

A pointer to a string variable containing the 8-byte initialization vector that the verb uses in encrypting the plaintext.

**cipher_text**

**Direction:** Output
**Type:** String

A pointer to a string variable containing the ciphertext returned by the verb.

**Restrictions**

The restrictions for CSNBCVE.

The text length must be a multiple of eight bytes.

The minimum length of text that the security server can process is eight bytes and the maximum is 256 bytes.

**Required commands**

The required commands for CSNBCVE.

This verb requires the **Cryptographic Variable Encipher** command (offset X'00DA') to be enabled in the active role.

**Usage notes**

Usage notes for CSNBCVE.

None.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBCVEJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBCVEJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
```

Chapter 8. AES, DES, and HMAC cryptographic keys  179
Data Key Export (CSNBDKX)

Use the Data Key Export verb to re-encipher a data-encrypting key (key type of DATA only) from encryption under the master key to encryption under an exporter key-encrypting key.

The re-enciphered key is in a form suitable for export to another system.

The Data Key Export verb generates a key token with the same key length as the input token's key.

Format

The format of CSNBDKX.

```
CSNBDKX(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    source_key_identifier,
    exporter_key_identifier,
    target_key_identifier)
```

Parameters

The parameter definitions for CSNBDKX.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see "Parameters common to all verbs" on page 22.

**source_key_identifier**

- **Direction:** Input/Output
- **Type:** String

A 64-byte string for an internal key token or label that contains a data-encrypting key to be re-enciphered. The data-encrypting key is encrypted under the master key.

**exporter_key_identifier**

- **Direction:** Input/Output
- **Type:** String

A 64-byte string for an internal key token or key label that contains the exporter key_encrypting key. The data-encrypting key above will be encrypted under this exporter key_encrypting key.

**target_key_identifier**

- **Direction:** Input/Output
- **Type:** String

A 64-byte field that is to receive the external key token, which contains the re-enciphered key that has been exported. The re-enciphered key can now be
exchanged with another cryptographic system.

**Restrictions**

The restrictions for CSNBDKX.

For security reasons, requests will fail by default if they use an equal key halves exporter to export a key with unequal key halves. You must have access control point 'Data Key Export - Unrestricted' explicitly enabled if you want to export keys in this manner.

**Required commands**

The CSNBDKX required commands.

This verb requires the **Data Key Export** command (offset X'010A') to be enabled in the active role.

By also specifying the **Data Key Export - Unrestricted** command (offset X'0277'), you can permit a less secure mode of operation that enables an equal key-halves EXPORTER key-encrypting-key to export a key having unequal key-halves (key parity bits are ignored).

**Usage notes**

Usage notes for CSNBDKX.

None.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBDKXJ.

See "Building Java applications using the CCA JNI" on page 28.

**Format**

```
public native void CSNBDKXJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] source_key_identifier,
    byte[] exporter_key_identifier,
    byte[] target_key_token);
```

**Data Key Import (CSNBDKM)**

Use the Data Key Import verb to import an encrypted source DES single-length, double-length or triple-length DATA key and create or update a target internal key token with the master key enciphered source key.
Data Key Import (CSNBDKM)

Format

The format of CSNBDKM.

```
CSNBDKM(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    source_key_token,
    importer_key_identifier,
    target_key_identifier)
```

Parameters

The parameter definitions for CSNBDKM.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

**source_key_token**

- **Direction:** Input
- **Type:** String

A pointer to a 64-byte string variable containing the source key to be imported. The source key must be an external token or null token. The external key token must indicate that a control vector is present; however, the control vector is usually valued at zero. A double-length key that should result in a default DATA control vector must be specified in a version X'01' external key token. Otherwise, both single and double-length keys are presented in a version X'00' key token. For the null token, the verb will process this token format as a DATA key encrypted by the importer key and a null (all zero) control vector.

**importer_key_identifier**

- **Direction:** Input/Output
- **Type:** String

A pointer to a 64-byte string variable containing the (IMPORTER) transport key or key label of the transport key used to decipher the source key.

**target_key_identifier**

- **Direction:** Output
- **Type:** String

A pointer to a 64-byte string variable containing a null key token or an internal key token. The key token receives the imported key.

Restrictions

The restrictions for CSNBDKM.

For security reasons, requests will fail by default if they use an equal key halves importer to import a key with unequal key halves. You must have access control point 'Data Key Import - Unrestricted' explicitly enabled if you want to import keys in this manner.

Required commands

The CSNBDKM required commands.
This verb requires the **Data Key Import** command (offset X'0109') to be enabled in the active role.

By also specifying the **Data Key Import - Unrestricted** command (offset X'027C'), you can permit a less secure mode of operation that enables an equal key-halves **IMPORTER** key-encrypting key to import a key having unequal key-halves (key parity bits are ignored).

**Usage notes**

Usage notes for CSNBDKM.

This verb does not adjust the key parity of the source key.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBDKMJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBDKMJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] source_key_token,
    byte[] importer_key_identifier,
    byte[] target_key_identifier);
```

---

**Diversified Key Generate (CSNBDKG)**

Use the Diversified Key Generate verb to generate a key based on the key-generating key, the processing method, and the parameter supplied.

The control vector of the key-generating key also determines the type of target key that can be generated.

To use this verb, specify the following:

- The **rule_array** keyword to select the diversification process.
- The operational key-generating key from which the diversified keys are generated. The control vector associated with this key restricts the use of this key to the key generation process. This control vector also restricts the type of key that can be generated.
- The data and length of data used in the diversification process.
- The generated-key could be an internal token or a skeleton token containing the desired CV of the generated-key. The generated key CV must be one that is permitted by the processing method and the key-generating key. The generated key will be returned in this parameter.
- A key generation method keyword.

This verb generates diversified keys as follows:

- Determines if it can support the process specified in the **rule_array**.
- Recovers the key-generating key and checks the key-generating key class and the specified usage of the key-generating key.
Diversified Key Generate (CSNBDKG)

- Determines that the control vector in the generated-key token is permissible for the specified processing method.
- Determines that the control vector in the generated-key token is permissible by the control vector of the key-generating key.
- Determines the required data length from the processing method and the generated-key CV. Validates the data_length.
- Generates the key appropriate to the specific processing method. Adjusts parity of the key to odd. Creates the internal token and returns the generated diversified key.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.

Format

The format of CSNBDKG.

```
CSNBDKG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    generating_key_identifier,
    data_length,
    data,
    data_decrypting_key_identifier,
    generated_key_identifier)
```

Parameters

The parameter definitions for CSNBDKG.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

rule_array_count

- Direction: Input
- Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 1, 2, or 3.

rule_array

- Direction: Input
- Type: String array

A pointer to a string variable containing an array of keywords that provide control information to the verb. The processing method is the algorithm used to create the generated key. The keyword is left-aligned and padded on the right with blanks. The rule_array keywords are described in Table 47.

Table 47. Keywords for Diversified Key Generate control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Method for generating or updating diversified keys (One required)</td>
<td></td>
</tr>
</tbody>
</table>
## Table 47. Keywords for Diversified Key Generate control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
</table>
| CLR8-ENC | Specifies that eight bytes of clear (not encrypted) data shall be Triple-DES encrypted with the generating key *(generating_key_identifier)* to create a generated key.  
The key selected by the generating_key_identifier must specify a KEYGENKY key-type also with control vector bit 19 set to B’1’.  
The key identified by the data_decrypting_key_identifier must identify a null DES key-token.  
The key token identified by the generated_key_identifier variable must contain a control vector that specifies a single-length key of one of these types: DATA, CIPHER, DECIPHER, MAC, or MACVER. |
| TDES-CBC | Specifies that 16 bytes of clear (not encrypted) data shall be Triple-DES encrypted with the generating key to create the generated key. If the generated_key_identifier variable specifies a double-length key, then 16 bytes of clear data are Triple-DES encrypted in CBC mode with an initial value of binary zeros.  
**Note:** The EMV Card Personalization specification states that CBC encryption mode should be used in the diversification process.  
The key selected by the generating_key_identifier parameter must specify a DKYGENKY key-type that has the appropriate control vector usage bits (bits 19 – 22) set for the desired generated key.  
Control vector bits 12 – 14 binary encode the key-derivation sequence level (DKYL7 down to DKYL0). The final key is derived when bits 12 – 14 are B’000’. The verb verifies the incremental relationship between the value in the control vectors of the key tokens identified by the generated_key_identifier parameter and the generating_key_identifier parameter control vector. In the case when the generated_key_identifier is a null DES token, the appropriate counter value is placed into the output key-token.  
The data_decrypting_key_identifier parameter must identify a null DES key-token.  
A key token identified by the generated_key_identifier parameter that is not a null key-token must contain a control vector that specifies a double-length key having a key type that is consistent with the specification in bits 19 – 22 of the generating key. |
| TDES-DEC | Specifies that 8 or 16 bytes of clear (not encrypted) data shall be Triple-DES decrypted with the generating key to create the generated key. If the generated_key_identifier variable specifies a single-length key, then 8 bytes of clear data are Triple-DES decrypted. If the generated_key_identifier variable specifies a double-length key, then 16 bytes of clear data are Triple-DES decrypted in ECB mode.  
The key selected by the generating_key_identifier must specify a DKYGENKY key-type that has the appropriate control vector usage bits (bits 19 – 22) set for the desired generated key.  
Control vector bits 12 – 14 binary encode the key-derivation sequence level (DKYL7 down to DKYL0). The final key is derived when bits 12 – 14 are B’000’. The verb verifies the incremental relationship between the value in the generated_key_identifier variable control-vector and the generating_key_identifier variable control-vector. Or in the case when the generated_key_identifier variable is a null DES key-token, the appropriate counter value is placed into the output key-token. The data_decrypting_key_identifier parameter must identify a null DES key-token.  
A key token identified by the generated_key_identifier variable that is not a null DES key-token must contain a control vector that specifies a single-length or double-length key having a key type consistent with the specification in bits 19 – 22 of the generating key. |
### Keywords for Diversified Key Generate control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TDES-ENC</strong></td>
<td>Specifies that 16 bytes of clear (not encrypted) data shall be Triple-DES decrypted with the generating key to create the generated key. If the <code>generated_key_identifier</code> variable specifies a double-length key, then 16 bytes of clear data are Triple-DES decrypted in CBC mode with an initial value of binary zeros. <strong>Note:</strong> The EMV Card Personalization specification states that CBC encryption mode should be used in the diversification process. The key selected by the <code>generated_key_identifier</code> must specify a <code>DKYGENKY</code> key-type that has the appropriate control vector usage bits (bits 19 – 22) set for the desired generated key. Control vector bits 12 – 14 binary encode the key-derivation sequence level (<code>DKYL7</code> down to <code>DKYL0</code>). The final key is derived when bits 12 – 14 are <code>B'000'</code>. The verb verifies the incremental relationship between the value in the <code>generated_key_identifier</code> variable control-vector and the <code>generating_key_identifier</code> variable control-vector. Or in the case when the <code>generated_key_identifier</code> variable is a null DES key-token, the appropriate counter value is placed into the output key-token. The <code>data_decrypting_key_identifier</code> parameter must identify a null DES key-token. A key token identified by the <code>generated_key_identifier</code> variable that is not a null DES key-token must contain a control vector that specifies a single-length or double-length key having a key type consistent with the specification in bits 19 – 22 of the generating key.</td>
</tr>
<tr>
<td><strong>TDES-XOR</strong></td>
<td>Specifies that 10 bytes or 18 bytes of clear (not encrypted) data shall be processed as described in &quot;Working with Europay-Mastercard-Visa Smart cards&quot; on page 492 to create the generated key. The data variable contains either 8 bytes or 16 bytes of data to be triple-encrypted to which you append a 2-byte Application Transaction Counter value (previously received from the smart card). Place the counter value in a string construct with the high-order counter bit first in the string. The key selected by the <code>generating_key_identifier</code> parameter must specify a <code>DKYGENKY</code> key-type at level-0 (bits 12 – 14 <code>B'000'</code>) and indicate permission to create one of several key types in bits 19 – 22: <code>B'0001'</code> DDATA, to generate a DATA key; <code>B'0010'</code> DMAC, to generate a MAC key; <code>B'0011'</code> DMV, to generate a MACVER key; <code>B'1000'</code> DMKEY, to generate a SECMSG SMKEY (used in secure messaging, key encryption, see the Secure Messaging for PINs verb); <code>B'1001'</code> DMPIN, to generate a SECMSG SMPIN (used in secure messaging, PIN encryption, see the Secure Messaging for PINs verb). The <code>data_decrypting_key_identifier</code> parameter must identify a null DES key-token. A key token or key-token record identified by the <code>generated_key_identifier</code> parameter that is not a null DES key-token. The token must contain a control vector that specifies a key type conforming to that specified in control-vector bits 19 – 22 for the key-generating key. The control vector must specify a double-length key.</td>
</tr>
<tr>
<td><strong>TDESEMV2</strong></td>
<td>This option supports generation of a session key by the EMV 2000 algorithm (This EMV2000 algorithm uses a branch factor of 2). The generating key must be a level 0 <code>DKYGENKY</code> and cannot have replicated halves. The session key generated must be double length and the allowed key types are DATA, DATAC, DATAM, DATAMV, MAC, MACVER, SMPIN, and SMKEY. Key type must be allowed by the generating key control vector.</td>
</tr>
</tbody>
</table>
Table 47. Keywords for Diversified Key Generate control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDESEMV4</td>
<td>This option supports generation of a session key by the EMV 2000 algorithm (This EMV2000 algorithm uses a branch factor of 4). The generating key must be a level 0 DKYGENKY and cannot have replicated halves. The session key generated must be double length and the allowed key types are DATA, DATAC, DATAM, DATAMV, MAC, MACVER, SMPIN, and SMKEY. Key type must be allowed by the generating key control vector.</td>
</tr>
</tbody>
</table>

**Processing Method for updating a diversified key (optional)**

**SESS-XOR**

Specifies the VISA method for session key generation, namely that 8 bytes or 16 bytes of data shall be exclusive-ORed with the clear value of the session key contained in the key token identified by the `generating_key_identifier` parameter. If the `generating_key_identifier` parameter identifies a single-length key, then 8 bytes of data are exclusive-ORed. If the `generating_key_identifier` parameter identifies a double-length key, then 16 bytes of data are exclusive-ORed.

The key token specified by the `generating_key_identifier` parameter must be of key type DATA, DATAC, DATAM, DATAMV, MAC, MACVER.

The `data_decrypting_key_identifier` parameter must identify a null DES key-token.

On input, the token identified by the `generated_key_identifier` parameter must identify a null DES key-token. The control vector contained in the output key token identified by the `generated_key_identifier` is the same as the control vector contained in the key token identified by the `generating_key_identifier`.

**Key-wrapping method (One, optional)**

**USECONFG**

This is the default. Specifies to wrap the key using the configuration setting for the default wrapping method. The default wrapping method configuration setting may be changed using the TKE. This keyword is ignored for AES keys.

**WRAP-ECB**

Specifies to wrap the key using the legacy wrapping method.

**WRAP-ENH**

Specifies to wrap the key using the enhanced wrapping method.

**Translation control** (Optional). This is valid only with key-wrapping method WRAP-ENH or with USECONFG when the default wrapping method is WRAP-ENH. This option cannot be used on a key with a control vector valued to binary zeros.

**ENH-ONLY**

Specifies to restrict the key from being wrapped with the legacy wrapping method once it has been wrapped with the enhanced wrapping method. Sets bit 56 (ENH-ONLY) of the control vector to B1.'
Diversified Key Generate (CSNBDKG)

Data input to the diversified key or session key generation process. Data depends on the processing method and the generated_key_identifier.

key_identifier

Direction: Input/Output
Type: String

This parameter is currently not used. It must be a 64-byte null token.

generated_key_identifier

Direction: Input/Output
Type: String

The internal token of an operational key, a skeleton token containing the control vector of the key to be generated, or a null token. A null token can be supplied if the generated_key_identifier is a DKYGENKY with a CV derived from the generating_key_identifier. A skeleton token or internal token is required when generated_key_identifier will not be a DKYGENKY key type or the processing method is not SESS-XOR. For SESS-XOR, this must be a null token. On output, this parameter contains the generated key.

Restrictions

The restrictions for CSNBDKG.

None.

Required commands

The CSNBDKG required commands.

This verb requires the following commands to be enabled in the active role based on the keyword specified for the process rule:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR8-ENC</td>
<td>X'0040'</td>
<td>Diversified Key Generate - CLR8-ENC</td>
</tr>
<tr>
<td>SESS-XOR</td>
<td>X'0043'</td>
<td>Diversified Key Generate - SESS-XOR</td>
</tr>
<tr>
<td>TDES-DEC</td>
<td>X'0042'</td>
<td>Diversified Key Generate - TDES-DEC</td>
</tr>
<tr>
<td>TDES-ENC</td>
<td>X'0041'</td>
<td>Diversified Key Generate - TDES-ENC</td>
</tr>
<tr>
<td>TDES-XOR</td>
<td>X'0045'</td>
<td>Diversified Key Generate - TDES-XOR</td>
</tr>
<tr>
<td>TDESEMV2 or TDESEMV4</td>
<td>X'0046'</td>
<td>Diversified Key Generate - Diversified Key Generate - TDESEMV2/TDESEMV4</td>
</tr>
<tr>
<td>WRAP-ECB or WRAP-ENH and default key-wrapping method setting does not match keyword</td>
<td>X'013D'</td>
<td>Diversified Key Generate - Allow wrapping override keywords</td>
</tr>
</tbody>
</table>

When a key-generating key of key type DKYGENKY is specified with control vector bits (19 - 22) of B’1111’, the Diversified Key Generate - DKYGENKY - DALL command (offset X’0290’) must also be enabled in the active role.

Note: A role with offset X’0290’ enabled can also use the PIN Change/Unblock verb with a DALL key.
When using the TDES-ENC or TDES-DEC modes, you can specifically enable generation of a single-length key or a double-length key with equal key-halves (an effective single-length key) by enabling the Diversified Key Generate - Single length or same halves command (offset X’0044’).

Usage notes

Usage notes for CSNBDKG.

Refer to Chapter 21, “Control vectors and changing control vectors with the Control Vector Translate verb,” on page 989 for information on the control vector bits for the DKG key generating key.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBDKGJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBDKGJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] generating_key_identifier,
    hikmNativeNumber data_length,
    byte[] data,
    byte[] dataDecrypting_key_identifier,
    byte[] generated_key_identifier);
```

Diversified Key Generate2 (CSNBDKG2)

The Diversified Key Generate2 service generates an AES key based on a function of a key-generating key, the process rule, and data that you supply.

To use this service, specify:

- the rule array keyword to select the diversification process
- the operational AES key-generating key from which the diversified keys are generated:
  - Key usage field 1 determines the type of key that is generated and restricts the use of this key to the key-diversification process.
  - Key usage field 2 contains a flag to determine how key usage fields 3 through 6 control the key usage fields of the generated key.
    - When the flag is on, the key usage fields of the DKYGENKY must be equal to the key usage fields of the generated key (KUF-MBE, meaning: key usage fields must be equal).
    - When the flag is off, the key usage fields of the DKYGENKY limit the values of the key usage fields of the generated key (KUF-MBP, meaning: key usage fields must be permitted).

For the service to be valid, the generated key cannot have a usage that is not enabled in the DKYGENKY key. The UDX-ONLY bit is always treated as must be equal.

- Key usage fields 3 through 6 in the key generating key indicate the key usage attributes for the key to be generated.
Diversified Key Generate2 (CSNBDKG2)

Note: The only exception to this rule is when the type of key to diversify is D-ALL.

- the data and length of data used in the diversification process
- the AES key token with a suitable key usage field for receiving the diversified key.

Format

The format of CSNBDKG2.

```
csnbdkg2(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  generating_key_identifier_length,
  generating_key_identifier,
  derivation_data_length,
  derivation_data,
  reserved1_length,
  reserved1,
  reserved2_length,
  reserved2,
  generated_key_identifier1_length,
  generated_key_identifier1,
  generated_key_identifier2_length,
  generated_key_identifier2)
```

Parameters

The parameter definitions for CSNBDKG2.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1 or 2.

**rule_array**

- **Direction:** Input
- **Type:** String array

The keyword that provides control information to the verb. The only `rule_array` keyword is described in Table 48.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Diversification Process</em></td>
<td>(One required)</td>
</tr>
</tbody>
</table>
Table 48. Keyword for Diversified Key Generate2 control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KDFM-DK</strong></td>
<td>Specifies to use the DK version of Key Derivation Function (KDF) in Feedback Mode (NIST SP 800-108), as specified in <em>DK Kryptographie Teil 1: Empfohlene kryptographische Algorithmen</em>, to generate a bank specific Issuer Master Key. The generated Issuer Master Key (keying material) can be used to derive an ICC master key.</td>
</tr>
<tr>
<td>(Release 4.4 or later)</td>
<td>This method uses AES CMAC to encipher 16 - 40 bytes of derivation data with the k-bit diversified key generating key (banking association specific master key) to produce a k-bit generated Bank specific Issuer Master Key, where k = 128, 192, or 256.</td>
</tr>
<tr>
<td><strong>MK-OPTC</strong></td>
<td>Specifies to use the EMV Master Key Derivation Option C, as specified in <em>EMV Integrated Circuit Card Specifications for Payments Systems</em>, to generate an ICC master key. The generated ICC master key (keying material) can be used for Application Cryptogram generation or verification, issuer authentication, and secure messaging.</td>
</tr>
<tr>
<td>(Release 4.4 or later)</td>
<td>This method uses AES in ECB mode to encipher the 16 bytes of derivation data with the k-bit diversified key generating key (Issuer Master Key) to produce a k-bit generated ICC master key, where k = 128, 192, or 256.</td>
</tr>
<tr>
<td><strong>SESS-ENC</strong></td>
<td>A session key is created by enciphering a 16-byte diversification value with the k-bit AES key-generating key to produce a k-bit AES session key using the AES algorithm in ECB mode, where k is 128, 192 or 256 bits.</td>
</tr>
<tr>
<td><strong>Bit length of generated key</strong></td>
<td>(One, optional). Release 4.4 or later. Valid only with the KDFM-DK keyword. Default is to use the bit length of the generating key as the bit length of the generated key.</td>
</tr>
<tr>
<td><strong>KLEN128</strong></td>
<td>Specifies the bit length of the generated key to be 128.</td>
</tr>
<tr>
<td><strong>KLEN192</strong></td>
<td>Specifies the bit length of the generated key to be 192, allowed if and only if the bit length of the generating key is greater than or equal to 192. See “Required commands” on page 194.</td>
</tr>
<tr>
<td><strong>KLEN256</strong></td>
<td>Specifies the bit length of the generated key to be 256, allowed if and only if the bit length of the generating key is 256. See “Required commands” on page 194.</td>
</tr>
</tbody>
</table>

**generating_key_identifier_length**

- **Direction:** Input
- **Type:** Integer

Length of the `generating_key_identifier` parameter in bytes. If the `generating_key_identifier` contains a label, the value must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**generating_key_identifier**

- **Direction:** Input/Output
- **Type:** String

The identifier of the key-generating key. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES and the key type must be DKYGENKY. The key usage field indicates the key type of the generated key.

If SESS-ENC is specified, the clear length of the generated key is equal to the clear length of the generating key. Also, beginning with Release 4.4, the sequence level can be set to DKYL0, DKYL1, or DKYL2 in the key usage field 2.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.
Diversified Key Generate2 (CSNBDKG2)

If the rule_array parameter specifies a diversification process of KDFFM-DK, the key-derivation sequence level of the generating key must be DKYL2. Otherwise, if KDFFM-DK is not specified, any sequence level is allowed for the generating key.

derivation_data_length
Direction: Input
Type: Integer

Length of the derivation_data parameter in bytes. The value must be in the range 16 - 40 for the diversification process keyword KDFFM-DK, otherwise the value must be 16.

derivation_data
Direction: Input
Type: String

The derivation data to be used in the key generation process. This data is often referred to as the diversification data. For SESS-ENC, the derivation data is 16 bytes long. Note that if SESS-ENC is specified and the length of the key generating key is 192 bits or 256 bits, the data is manipulated in conformance with the EMV Common Session Key Derivation Option.

reserved1_length
Direction: Input
Type: Integer

Length of the reserved1 parameter in bytes. The value must be 0.

reserved1
Direction: Input
Type: String

This parameter is ignored.

reserved2_length
Direction: Input
Type: Integer

Length of the reserved2 parameter in bytes. The value must be 0.

reserved2
Direction: Input
Type: String

This parameter is ignored.

generated_key_identifier1_length
Direction: Input/Output
Type: Integer

On input, this parameter specifies the length in bytes of the buffer for the generated_key_identifier1 parameter. The maximum value is 725 bytes.

On output, the parameter holds the actual length in bytes of the generated_key_identifier1 parameter.
Direction: Input/Output  
Type: String  

A pointer to a string variable containing an internal variable-length symmetric key-token or the key label of such a record in AES key-storage.

On input, identify a null key token or a skeleton key token that specifies the desired attributes of the key on output. The key token identified by `generating_key_identifier1` determines whether on input the `generating_key_identifier1` can identify a null key token or a skeleton key token. See Table 49.

<table>
<thead>
<tr>
<th>Input generating key token</th>
<th>Input generated key token</th>
<th>Output generated key token</th>
</tr>
</thead>
<tbody>
<tr>
<td>DKYGENKY, DKYL0, type of key to diversify D-ALL</td>
<td>Null AES key token not allowed; AES skeleton key token required.</td>
<td>Key type same as skeleton; diversified key final.</td>
</tr>
<tr>
<td>DKYGENKY, DKYL0, type of key to diversify not D-ALL</td>
<td>Either null AES key token or AES skeleton key token required.</td>
<td>Key type determined by input generated key token type of key to diversify; if null key token on input, output key token will have attributes based on the related generated key usage fields of the input generating key token, otherwise the output key token has attributes of the input skeleton key token.</td>
</tr>
<tr>
<td>DKYGENKY, DKYL1, any type of key to diversify</td>
<td>Null AES key token required; AES skeleton key token not allowed.</td>
<td>Same as input generating key token except DKYL0 and with new level of diversified key.</td>
</tr>
<tr>
<td>DKYGENKY, DKYL2, any type of key to diversify</td>
<td>Null AES key token required; AES skeleton key token not allowed.</td>
<td>Same as input generating key token except DKYL1 and with new level of diversified key.</td>
</tr>
</tbody>
</table>

**Note:**
1. If the supplied generated key-token contains a key, the key value and length are ignored and overwritten.
2. The key type must match what the generating key indicates can be created in the key generating key usage field at offset 45.
3. The key usage fields in the generated key must meet the requirements (KUF must be equal (KUF-MBE) or KUF must be permissible (KUF-MBP)) of the corresponding key usage fields in the generating key unless D-ALL is specified in the generating key. A flag bit in the DKGYGENKY key-usage field 2 determines whether the key-usage field level of control is KUF-MBE or KUF-MBP.
4. If authorized by access control, D-ALL permits the derivation of several different keys. See "Required commands" on page 194.

---

**generated_key_identifier2_length**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Length of the `generated_key_identifier2` parameter in bytes. The value must be 0.
Diversified Key Generate2 (CSNBDKG2)

**generated_key_identifier2**

- **Direction:** Input/Output
- **Type:** String

This parameter is ignored.

**Restrictions**

The restrictions for CSNBDKG2.

None.

**Required commands**

The CSNBDKG2 required commands.

The Diversified Key Generate2 verb requires the following commands to be enabled in the active role:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>KDFFM-DK (Release 4.4 or later)</td>
<td>X'02D3'</td>
<td>Diversified Key Generate2 - KDFFM-DK</td>
</tr>
<tr>
<td>KLEN192 and KLEN256 (Release 4.4 or later)</td>
<td>X'02D4'</td>
<td>Allow DKG2 Generated Key Length Option with KDFFM-DK Keyword</td>
</tr>
<tr>
<td>MK-OPTC (Release 4.4 or later)</td>
<td>X'02D2'</td>
<td>Diversified Key Generate2 - MK-OPTC</td>
</tr>
<tr>
<td>SESS-ENC</td>
<td>X'02CC'</td>
<td>Diversified Key Generate2 - AES EMV1 SESS</td>
</tr>
</tbody>
</table>

An additional command is required when the key usage fields of the key token identified by the `generating_key_identifier` parameter specify a type of key to diversify of D-ALL (any type of key allowed). In this case, the verb requires the **Diversified Key Generate2 - DALL** command (offset X'02CD') to be enabled in the active role.

**Usage notes**

Usage notes for CSNBDKG2.

None.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBDKG2J.

See "Building Java applications using the CCA JNI" on page 28.

**Format**

```java
public native void CSNBDKG2J(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber generating_key_identifier_length,
    byte[] generating_key_identifier,
```
**EC Diffie-Hellman (CSNDEDH)**

Use the EC Diffie-Hellman verb to create symmetric key material from a pair of Elliptic Curve Cryptography (ECC) keys using the Elliptic Curve Diffie-Hellman (ECDH) protocol and “Z” - The “secret” material output from EC Diffie-Hellman process.

For more information, see “EC Diffie-Hellman key agreement models” on page 52.

ECDH is a key-agreement protocol that allows two parties, each having an elliptic curve public-private key pair, to establish a shared secret over an insecure channel. This shared secret is used to derive another symmetric key. The ECDH protocol is a variant of the Diffie-Hellman protocol using elliptic curve cryptography. ECDH derives a shared secret value from a secret key owned by an Entity A and a public key owned by an Entity B, when the keys share the same elliptic curve domain parameters. Entity A can be either the initiator of a key-agreement transaction, or the responder in a scheme. If two entities both correctly perform the same operations with corresponding keys as inputs, the same shared secret value is produced.

Both parties must create an ECC public-private key pair. See “PKA Key Token Build (CSNDPKB)” on page 699 and “PKA Key Generate (CSNDPKG)” on page 689. A key can be internal or external, as well as encrypted or in the clear. Both keys must have the same elliptic curve domain parameters (curve type and key size):

- Brainpool (key size 160, 192, 224, 256, 320, 384, or 512)
- Prime (key size 192, 224, 256, 384, or 521)

In addition to having the same elliptic curve domain parameters, the keys must have their key-usage field set to permit key establishment (either KEY-MGMT or KM-ONLY). See “ECC key token” on page 874.

To use this verb, specify the following:

- One to six rule-array keywords:
  - A required key-agreement keyword
  - An optional transport key-type (required if output KEK_key_identifier is a label) that identifies which key-storage dataset contains the output KEK key-token
  - An optional output key-type (required if output_key_identifier is a label) that identifies which key-storage dataset contains the output key-token
  - When the output is a DES key-token, an optional key-wrapping method and an optional translation control keyword
  - An optional hash type for rule-array keyword DERIV02.
The internal or external ECC key-token containing the private key
(public-private key pair).
If the private key is in an external key-token and is not in the clear, specify the
internal KEK that was used to wrap the private key. Otherwise, specify a private
KEK key-length of 0.

- An internal or external ECC key-token containing the public key (public key
only or public-private key pair) of the other party.
If the public key is in a key token that contains a public-private key pair, only
the public-key portion is used. No attempt is made to decrypt the private key.

- Party information data in parameter party_info:
  - When rule-array keyword DERIV01 is given: From 8 - 64 bytes, specify the
    party information data of the initiator and the responder entities, according to
    NIST SP800-56A Section 5.8
  - When rule-array keyword DERIV02 is given: From 0 - 256 bytes, specify party
    information data, according to section 5.6.3 of ANS X9.63-2011

- The number of bits of key material, from 64 - 256, to derive and place in the
  provided output key-token
- The length in bytes of the buffer for the output key-identifier
- An internal or external skeleton key-token to be used as input for the output
  key-token
  The skeleton key-token must be an AES key or a DES key, as shown in the
  following table:

Table 50. CSNDEDH skeleton key-tokens

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Token version number</th>
<th>Key type (see note)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>X'04' (legacy fixed-length symmetric key-token)</td>
<td>DATA</td>
</tr>
</tbody>
</table>
|           | X'05' (variable-length symmetric key-token) | • CIPHER
  Both parties can provide any combination of encryption or decryption for key-usage field.
  • EXPORTER or IMPORTER
  Both parties can provide any combination of EXPORTER or IMPORTER. |
| DES       | X'00', X'01', X'03' (legacy fixed-length symmetric key-token) | • CIPHER, DECIPHER, ENCIPHER
  Both parties can provide any combination of Encipher or Decipher key-usage bits in the control vector.
  • CIPHER XI, CIPHERXL, CIPHERXO
  Both parties can provide any combination of Encipher or Decipher key-usage bits in the control vector.
  • EXPORTER or IMPORTER
  Both parties can provide any combination of EXPORT or IMPORT key-usage bits in the control vector. |

Note:
1. Except as otherwise noted, both parties must provide identical skeleton key tokens for
   the output key in order to derive identical keys. For legacy skeletons, control vector
   parity bits are not used in the key derivation process.
2. Control vector bits and key-usage fields are not used in the key derivation process
   when rule-array keyword DERIV02 is specified.
If the skeleton key-token is an external key-token, specify the internal KEK to be used to wrap the output key-token. Otherwise, specify an output KEK length of 0.

If the `output_key_identifier` specifies a DES key-token, then the `output KEK_key_token` must identify a legacy DES KEK. Otherwise it must identify a variable-length symmetric AES KEK key-token.

Table 51 provides the format for the concatenation string from which the key is derived when DERIV01 is specified in the rule-array:

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>Initialized to X'00000001'</td>
<td>Counter (four-byte unsigned integer)</td>
</tr>
<tr>
<td>4</td>
<td>xx</td>
<td>Z</td>
<td>A shared secret bit string or octet string</td>
</tr>
<tr>
<td>4 + xx</td>
<td>1</td>
<td>Value</td>
<td>Algorithm identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
<td>DES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'04'</td>
<td>AES</td>
</tr>
<tr>
<td>5 + xx</td>
<td>1</td>
<td>Passed party_info_length variable</td>
<td>Party information length passed by caller, converted to a one-byte unsigned integer</td>
</tr>
<tr>
<td>6 + xx</td>
<td>party_info_length</td>
<td>String identified by party_info parameter</td>
<td>Parties information</td>
</tr>
<tr>
<td>6 + xx + party_info_length</td>
<td>2</td>
<td>Supplied public information length, zz</td>
<td>Two-byte unsigned integer specifying length of supplied public information</td>
</tr>
<tr>
<td>8 + xx + party_info_length</td>
<td>zz</td>
<td>Supplied public information</td>
<td>Token data extracted from the skeleton key token identified by the <code>output_key_identifier</code> parameter (refer to Table 52 on page 198)</td>
</tr>
</tbody>
</table>

Note: All integers are in big-endian format.
Table 52. DERIV01 supplied public information

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Key-token version</th>
<th>Key type</th>
<th>Supplied public information length, zz (bytes)</th>
<th>Supplied public information</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>X’04‘ (legacy fixed-length symmetric key-token)</td>
<td>DATA</td>
<td>8</td>
<td>Control vector (CV) from key token beginning at offset 48 for zz bytes. If flag byte indicates no CV present, information valued to binary zeros.</td>
</tr>
<tr>
<td></td>
<td>X’05‘ (variable-length symmetric key-token)</td>
<td>CIPHER</td>
<td>1 + (key usage fields count * 2) + 1 + (key management fields count * 2)</td>
<td>Key usage and key management information from key token beginning at offset 44 for zz bytes, with ENCRYPT and DECRYPT bits (B’11xx xxxx’ ) of KUF 1 high-order byte masked off to maintain compatibility between keys with different key usage for these bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXPORTER or IMPORTER</td>
<td>1 + (key usage fields count * 2) + 1 + (key management fields count * 2)</td>
<td>Key usage and key management information from key token beginning at offset 44 for zz bytes, with no bits masked off</td>
</tr>
<tr>
<td>DES</td>
<td>X’00’, X’01’, X’03‘ (legacy fixed-length symmetric key token)</td>
<td>CIPHER, DECIPHER, or ENCIPHER</td>
<td>8 for single-length keys, 16 for double-length keys</td>
<td>CV from key token beginning at offset 32 for zz bytes, with CV bits 18, 19, and 23 (X’xx11 xxx1’) masked off (in both CV halves if double-length key) to maintain compatibility between keys with different Encipher, Decipher, and parity bit values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXPORTER or IMPORTER</td>
<td>16</td>
<td>CV from key token beginning at offset 32 for zz bytes, with CV bits 9, 14, and 15 (X’1xx x111’) masked off (in both CV halves) to maintain compatibility between different KEK key types and parity bit values.</td>
</tr>
</tbody>
</table>

Table 53 provides the format for the concatenation string from which the key is derived when DERIV02 is specified in the rule-array:

Table 53. CSNDEDH concatenation string format for DERIV02

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>xx</td>
<td>Z</td>
<td>A shared secret bit string or octet string</td>
</tr>
<tr>
<td>xx</td>
<td>4</td>
<td>Initialized to X‘00000001’</td>
<td>Counter (four-byte unsigned integer)</td>
</tr>
</tbody>
</table>
Table 53. CSNDEDH concatenation string format for DERIV02 (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx + 4</td>
<td>yy</td>
<td>String identified by <strong>party_info</strong> parameter</td>
<td>Parties information; length not an explicit field in concatenation string</td>
</tr>
</tbody>
</table>

Note: All integers are in big-endian format.

The output from this verb can be one of the following formats:

- Internal CCA Token (DES or AES): AES keys are in the variable-length symmetric key token format. DES keys are in the DES external key token format.
- External CCA key token (DES or AES): AES keys are in the variable-length symmetric key token format. DES keys are in the DES external key token format.
- "Z" – The "secret" material output from EC Diffie-Hellman process.

The **PASSTHRU** service is provided as a convenience to users who wish to implement their own key completion process in host application software. While the "Z" derivation process is not reversible (the ECC keys cannot be discovered by obtaining "Z") there is a level of security compromise associated with returning the clear "Z" to the application. Future derivations for CCA key tokens using ECC keys previously used in **PASSTHRU** must be considered to have lower security, and using the same ECC keys for **PASSTHRU** as for **DERIV01** is strongly discouraged. It should also be noted that since "Z" is the secret material, returning it in the clear to the host application reduces security level of the "Z" from the HSM level to the host application level, and keys made from this material should not be regarded as having any HSM protection.

For more information, see "EC Diffie-Hellman key agreement models" on page 52.
EC Diffie-Hellman (CSNDEDH)

Format

The format of CSNDEDH.

```c
CSNDEDH(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    private_key_identifier_length,
    private_key_identifier,
    private_KEY_key_identifier_length,
    private_KEY_key_identifier,
    public_key_identifier_length,
    public_key_identifier,
    chaining_vector_length,
    chaining_vector,
    party_info_length,
    party_info,
    key_bit_length,
    reserved_1_length,
    reserved_1,
    reserved_2_length,
    reserved_2,
    reserved_3_length,
    reserved_3,
    reserved_4_length,
    reserved_4,
    reserved_5_length,
    reserved_5,
    output_KEY_key_identifier_length,
    output_KEY_key_identifier,
    output_key_identifier_length,
    output_key_identifier)
```

Parameters

The parameter definitions for CSNDEDH.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

`rule_array_count`

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. Valid values are 1 - 6.

`rule_array`

- **Direction:** Input
- **Type:** String array

A pointer to a string variable containing an array of keywords. The keywords are 8 bytes in length and must be left-aligned and padded on the right with space characters. The `rule_array` keywords for this verb are shown below:
### Table 54. Keywords for EC Diffie-Hellman control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key agreement</strong> (one required). Initiator and responder must have a sufficient level of trust such that they each derive only one element of any key pair. DERIV01 is designed for CCA-to-CCA interaction.</td>
<td></td>
</tr>
<tr>
<td><strong>DERIV01</strong></td>
<td>Use input skeleton key-token and derive one element of any key pair. Denotes ANS X9.63 protocol static unified model key-agreement scheme (see NIST SP800-56A).</td>
</tr>
<tr>
<td><strong>DERIV02</strong></td>
<td>Use input skeleton key-token and derive one element of any key pair. Denotes key derivation function ANS-X9.63-KDF (refer to ANS X9.63-2011 section 5.6.3).</td>
</tr>
<tr>
<td><strong>PASSTHRU</strong></td>
<td>Skip key derivation step and return raw Z material. <strong>Note:</strong> This keyword is available only for Linux on Z.</td>
</tr>
<tr>
<td><strong>Transport key-type</strong> (one, optional; one required if output KEK_key_identifier is a label)</td>
<td></td>
</tr>
<tr>
<td><strong>OKEK-AES</strong></td>
<td>The output KEK_key_identifier represents an AES key-token.</td>
</tr>
<tr>
<td><strong>OKEK-DES</strong></td>
<td>The output KEK_key_identifier represents a DES key-token.</td>
</tr>
<tr>
<td><strong>Output key-type</strong> (one, optional; required if output key_identifier is a label)</td>
<td></td>
</tr>
<tr>
<td><strong>KEY-AES</strong></td>
<td>The outbound key-encrypting key represents an AES skeleton key-token.</td>
</tr>
<tr>
<td><strong>KEY-DES</strong></td>
<td>The outbound key-encrypting key represents a DES skeleton key-token.</td>
</tr>
<tr>
<td><strong>Key-wrapping method</strong> (one, optional). DES only.</td>
<td></td>
</tr>
<tr>
<td><strong>USECONFIG</strong></td>
<td>This is the default. Specifies to wrap the key using the configuration setting for the default wrapping method. The default wrapping method configuration setting may be changed using the TKE. This keyword is ignored for AES keys.</td>
</tr>
<tr>
<td><strong>WRAP-ECB</strong></td>
<td>Specifies to wrap the key using the legacy wrapping method. This keyword is ignored for AES keys.</td>
</tr>
<tr>
<td><strong>WRAP-ENH</strong></td>
<td>Specifies to wrap the key using the enhanced wrapping method. Valid only for DES keys.</td>
</tr>
<tr>
<td><strong>Translation control</strong> (optional). This is valid only with key-wrapping method WRAP-ENH or with USECONFIG when the default wrapping method is WRAP-ENH. This option cannot be used on a key with a control vector valued to binary zeros.</td>
<td></td>
</tr>
<tr>
<td><strong>ENH-ONLY</strong></td>
<td>Specifies to restrict the key from being wrapped with the legacy wrapping method after it has been wrapped with the enhanced wrapping method. Sets bit 56 (ENH-ONLY) of the control vector to B’1’.</td>
</tr>
<tr>
<td><strong>Hash method</strong> (one, optional). Only valid with DERIV02 key agreement keyword.</td>
<td></td>
</tr>
<tr>
<td><strong>SHA-224</strong></td>
<td>Specifies the use of the SHA-224 method.</td>
</tr>
<tr>
<td><strong>SHA-256</strong></td>
<td>Specifies the use of the SHA-256 method. This is the default.</td>
</tr>
<tr>
<td><strong>SHA-384</strong></td>
<td>Specifies the use of the SHA-384 method.</td>
</tr>
<tr>
<td><strong>SHA-512</strong></td>
<td>Specifies the use of the SHA-512 method.</td>
</tr>
</tbody>
</table>

**private_key_identifier_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes in the **private_key_identifier** variable.

**private_key_identifier**
A pointer to a string variable containing an internal or external ECC key-token, or a key label identifying a PKA key-storage record for such a token. The ECC key-token will contain a public-private key pair. A clear private-key is only allowed when rule-array keyword DERIV01 is specified.

The ECC curve type and size must be the same as the type (Prime or Brainpool) and size of the ECC key-token specified by the public_key_identifier parameter. The key-usage flag of the ECC key-token identified by the private_key_identifier parameter must permit key establishment (either KEY-MGMT or KM-ONLY).

For rule-array keyword DERIV02, the ECC key-token identified by the private_key_identifier parameter must contain an ECC key-derivation information section (section identifier X'23'). Refer to Table 254 on page 879.

private_KEYK_key_identifier_length
Direction: Input
Type: Integer

A pointer to an integer variable containing the number of bytes in the private_KEYK_key_identifier variable. The maximum value is 900. If the private_key_identifier contains an internal ECC token, this value must be a zero.

private_KEYK_key_identifier
Direction: Input
Type: String

A pointer to a string variable containing an internal KEK key-token, or the key label of such a record in AES key storage. The KEK key-token must be present if the key token specified by the private_key_identifier contains an external encrypted ECC key-token.

diffie_Hellman_key
Direction: Input/Output
Type: Integer

A pointer to an integer variable containing an ECC key-token, or a key label identifying an AES key-storage record for such a key token. The ECC curve type and size must be the same as the type and size of the ECC key-token specified by the private_key_identifier parameter. If the public_key_identifier parameter identifies a key token containing a public-private key pair, no attempt to decrypt the private part is made.

Note that even though this variable is not currently updated on output, it is reserved as an output field for future use.

public_key_identifier
Direction: Input/Output
Type: String

A pointer to a string variable containing an ECC key-token, or a key label identifying an AES key-storage record for such a key token. The ECC curve type and size must be the same as the type and size of the ECC key-token specified by the private_key_identifier parameter. If the public_key_identifier parameter identifies a key token containing a public-private key pair, no attempt to decrypt the private part is made.

Note that even though this variable is not currently updated on output, it is reserved as an output field for future use.
chaining_vector_length
Direction: Input/Output
Type: Integer

A pointer to an integer variable containing the number of bytes in the chained_vector variable. This field is currently not used. The value must be 0.

chaining_vector
Direction: Input/Output
Type: String

A pointer to a string variable containing a buffer that is currently reserved.

party_info_length
Direction: Input/Output
Type: Integer

A pointer to an integer variable containing the number of bytes in the party_info variable. The value must be 8 - 64 for DERIV01. For DERIV02, the value must be 0 - 256.

party_info
Direction: Input/Output
Type: String

A pointer to a string variable. For DERIV01, the string contains the combined entity identifier information, including nonce. This information must contain data of both entities according to NIST SP800-56A Section 5.8. For DERIV02, the string contains optional shared data according to ANS X9.63-2011 section 5.6.3.

key_bit_length
Direction: Input
Type: Integer

A pointer to an integer variable containing the number of bits of key material to derive and place in the provided output key-token. The value must be 0 if the PASSTHRU rule-array keyword is specified. The value must be 64 - 256.

reserved_1_length
Direction: Input/Output
Type: Integer

A pointer to an integer variable containing the number of bytes in the reserved_1 variable. The value must be 0.

reserved_1
Direction: Input/Output
Type: String

A pointer to a string variable that is currently not used.

reserved_2_length
Direction: Input/Output
Type: Integer
A pointer to an integer variable containing the number of bytes in the `reserved_2` variable. The value must be 0.

**reserved_2**

- **Direction:** Input/Output
- **Type:** String

A pointer to a string variable that is currently not used.

**reserved_3_length**

- **Direction:** Input/Output
- **Type:** Integer

A pointer to an integer variable containing the number of bytes in the `reserved_3` variable. The value must be 0.

**reserved_3**

- **Direction:** Input/Output
- **Type:** String

A pointer to a string variable that is currently not used.

**reserved_4_length**

- **Direction:** Input/Output
- **Type:** Integer

A pointer to an integer variable containing the number of bytes in the `reserved_4` variable. The value must be 0.

**reserved_4**

- **Direction:** Input/Output
- **Type:** String

A pointer to a string variable that is currently not used.

**reserved_5_length**

- **Direction:** Input/Output
- **Type:** Integer

A pointer to an integer variable containing the number of bytes in the `reserved_5` variable. The value must be 0.

**reserved_5**

- **Direction:** Input/Output
- **Type:** String

A pointer to a string variable that is currently not used.

**output_KEY_key_identifier_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes in the `output_KEY_key_identifier` variable. The maximum value is 900. The `output_KEY_key_identifier_length` must be zero if `output_key_identifier` will contain an internal token or if the PASSTHRU rule-array keyword was specified.
output_KEK_key_identifier

Direction: Input
Type: String

A pointer to a string variable containing an internal KEK key-token, or a key label identifying a key-storage record for such a token. This parameter must identify a KEK key-token whenever the output_key_identifier specifies an external key-token. If the output_key_identifier specifies a DES key-token, then the output_KEK_key_identifier must identify a legacy DES KEK, otherwise it must identify a variable-length symmetric AES KEK key-token.

If this variable contains a key label, specify a transport key-type rule-array keyword (OKEK-DES or OKEK-AES) to identify which key-storage dataset contains the key token. If a transport key-type keyword is specified, it must match the type of key identified by this parameter, whether the key is in key storage or not.

If the output_KEK_key_identifier specifies a legacy DES KEK, then the key token must contain either an EXPORTER control vector with bit 21 on (EXPORT) or an IMPORTER control vector with bit 21 set to B’1’ (IMPORT). The XLATE bit (bit 22) is not checked. Similarly, if the output_KEK_key_identifier identifies a variable-length symmetric AES KEK, then the KEK must be have a key type of EXPORTER or IMPORTER. Key-usage field 1 of the KEK must be set so that the key can be used for EXPORT or IMPORT. In addition, key-usage field 4 must be set so that the key can wrap DERIVATION class keys.

output_key_identifier_length

Direction: Input/Output
Type: Integer

A pointer to an integer variable containing the number of bytes in the output_key_identifier variable. On input, the output_key_identifier_length variable must be set to the total size of the buffer pointed to by the output_key_identifier parameter. On output, this variable contains the number of bytes of data returned by the verb in the output_key_identifier variable. The maximum allowed value is 900 bytes.

output_key_identifier

Direction: Input/Output
Type: String

A pointer to a string variable. On input, it must contain an internal or an external skeleton key-token, or a key label identifying a key-storage record for such a token. The skeleton key-token must be one of the following:

DES (legacy DES key-token)
- CIPHER, DECIPHER, or ENCIIPHER
- EXPORTER or IMPORTER

AES
- DATA (legacy AES key-token)
- CIPHER (variable-length symmetric key-token) with key-usage field set so that the key can be used for decryption, encryption, or both)
- EXPORTER or IMPORTER (variable-length symmetric key-token)

On successful completion, this variable contains either:
EC Diffie-Hellman (CSNDEDH)

- An updated key-token that contains the generated symmetric key material, or the key label of the key-token that has been updated in key storage.
- "Z" data (in the clear) if the PASSTHRU rule-array keyword was specified.

If this variable contains an external key-token on input, then the output KEK_key_identifier is used to securely encrypt it for output. If this variable contains a key label, specify an output key-type rule-array keyword (KEY-DES or KEY-AES) to identify which key-storage dataset contains the key token. If an output key-type keyword is specified, it must match the type of key identified by this parameter, whether the key is in key storage or not.

If this variable identifies an external DES key-token, then the output KEK_key_identifier must identify a DES KEK key-token. If this variable is present and identifies an external key-token other than a DES key-token, then the output KEK_key_identifier must identify an AES KEK key-token.

Restrictions

The restrictions for CSNDEDH.
- The NIST security strength requirements are enforced, with respect to ECC curve type (input) and derived key-length. See "Required commands" about how you can override this enforcement.

<table>
<thead>
<tr>
<th>Output key ID type</th>
<th>Valid key bit lengths</th>
<th>Minimum curve required</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES</td>
<td>64</td>
<td>P160</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>P160</td>
</tr>
<tr>
<td></td>
<td>192</td>
<td>P224</td>
</tr>
<tr>
<td>AES</td>
<td>128</td>
<td>P256</td>
</tr>
<tr>
<td></td>
<td>192</td>
<td>P384</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>P512</td>
</tr>
</tbody>
</table>

- A clear private key is only allowed when rule-array keyword DERIV01 is specified.

Required commands

The CSNDEDH required commands.

This table describes access control points that the EC Diffie-Hellman verb must have enabled in the active role under certain circumstances.

<table>
<thead>
<tr>
<th>Command</th>
<th>Offset</th>
<th>When required</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC Diffie-Hellman</td>
<td>X'0360'</td>
<td>When using the EC Diffie-Hellman verb</td>
</tr>
<tr>
<td>ECC Diffie-Hellman - Allow DERIV02</td>
<td>X'035F'</td>
<td>When using the DERIV02 rule</td>
</tr>
<tr>
<td>ECC Diffie-Hellman - Allow key wrap override</td>
<td>X'0362'</td>
<td>If the output_key_identifier parameter identifies a DES key-token, and the wrapping method specified is WRAP-ECB or WRAP-ENH.</td>
</tr>
</tbody>
</table>
EC Diffie-Hellman (CSNDEDH)

<table>
<thead>
<tr>
<th>Command</th>
<th>Offset</th>
<th>When required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prohibit weak wrapping - Transport keys</td>
<td>X'0328'</td>
<td>To disable the wrapping of a stronger key with a weaker transport key</td>
</tr>
<tr>
<td>_warn when weak wrap - Transport keys</td>
<td>X'032C'</td>
<td>To receive a warning against the wrapping of a stronger key with a weaker transport key</td>
</tr>
<tr>
<td>ECC Diffie-Hellman - Prohibit weak key generate</td>
<td>X'036F'</td>
<td>To disable a weaker key from being used to generate a stronger key</td>
</tr>
<tr>
<td>ECC Diffie-Hellman - Allow PASSTHRU</td>
<td>X'0361'</td>
<td>When specifying the PASSTHRU rule-array keyword.</td>
</tr>
</tbody>
</table>

Depending on curve type, each length of $p$ in bits contained in the ECC private-key section and the ECC public-key section must have the following command enabled in the active role:

<table>
<thead>
<tr>
<th>Curve type</th>
<th>Length of prime $p$ in bits</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brainpool</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>160 (X'00A0')</td>
<td>X'0368'</td>
<td>ECC Diffie-Hellmann - Allow BP Curve 160</td>
</tr>
<tr>
<td></td>
<td>192 (X'00C0')</td>
<td>X'0369'</td>
<td>ECC Diffie-Hellmann - Allow BP Curve 192</td>
</tr>
<tr>
<td></td>
<td>224 (X'00E0')</td>
<td>X'036A'</td>
<td>ECC Diffie-Hellmann - Allow BP Curve 224</td>
</tr>
<tr>
<td></td>
<td>256 (X'0100')</td>
<td>X'036B'</td>
<td>ECC Diffie-Hellmann - Allow BP Curve 256</td>
</tr>
<tr>
<td></td>
<td>320 (X'0140')</td>
<td>X'036C'</td>
<td>ECC Diffie-Hellmann - Allow BP Curve 320</td>
</tr>
<tr>
<td></td>
<td>384 (X'0180')</td>
<td>X'036D'</td>
<td>ECC Diffie-Hellmann - Allow BP Curve 384</td>
</tr>
<tr>
<td></td>
<td>512 (X'0200')</td>
<td>X'036E'</td>
<td>ECC Diffie-Hellmann - Allow BP Curve 512</td>
</tr>
<tr>
<td>Prime</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>192 (X'00C0')</td>
<td>X'0363'</td>
<td>ECC Diffie-Hellmann - Allow Prime Curve 192</td>
</tr>
<tr>
<td></td>
<td>224 (X'00E0')</td>
<td>X'0364'</td>
<td>ECC Diffie-Hellmann - Allow Prime Curve 224</td>
</tr>
<tr>
<td></td>
<td>256 (X'0100')</td>
<td>X'0365'</td>
<td>ECC Diffie-Hellmann - Allow Prime Curve 256</td>
</tr>
<tr>
<td></td>
<td>384 (X'0180')</td>
<td>X'0366'</td>
<td>ECC Diffie-Hellmann - Allow Prime Curve 384</td>
</tr>
<tr>
<td></td>
<td>521 (X'0209')</td>
<td>X'0367'</td>
<td>ECC Diffie-Hellmann - Allow Prime Curve 521</td>
</tr>
</tbody>
</table>
To disable the wrapping of a key with a weaker master key, the Prohibit weak wrapping - Master keys command (offset X'0333') must be enabled in the active role.

To receive a warning when wrapping a key with a weaker master key, enable the Warn when weak wrap - Master keys command (offset X'0332') in the active role. The Prohibit weak wrapping - Master keys command overrides this command.

Usage notes
Usage notes for CSNDEDH.

The PASSTHRU service is provided as a convenience to users who wish to implement their own key completion process in host application software. While the "Z" derivation process is not reversible (the ECC keys cannot be discovered by obtaining "Z") there is a level of security compromise associated with returning the clear "Z" to the application. Future derivations for CCA key tokens using ECC keys previously used in PASSTHRU must be considered to have lower security, and using the same ECC keys for PASSTHRU as for DERIV01 is strongly discouraged. It should also be noted that since "Z" is the secret material, returning it in the clear to the host application reduces security level of the "Z" from the HSM level to the host application level, and keys made from this material should not be regarded as having any HSM protection.

For more information, see “EC Diffie-Hellman key agreement models” on page 52.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNDEDHJ.

See “Building Java applications using the CCA JNI” on page 28.

Format
public native void CSNDEDHJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber private_key_identifier_length,
    byte[] private_key_identifier,
    hikmNativeNumber private_KEK_key_identifier_length,
    byte[] private_KEK_key_identifier,
    hikmNativeNumber public_key_identifier_length,
    byte[] public_key_identifier,
    hikmNativeNumber chaining_vector_length,
    byte[] chaining_vector,
    hikmNativeNumber party_info_length,
    byte[] party_info,
    hikmNativeNumber key_bit_length,
    hikmNativeNumber reserved_1_length,
    byte[] reserved_1,
    hikmNativeNumber reserved_2_length,
    byte[] reserved_2,
    hikmNativeNumber reserved_3_length,
    byte[] reserved_3,
    hikmNativeNumber reserved_4_length,
    byte[] reserved_4,
    hikmNativeNumber reserved_5_length,
    byte[] reserved_5,
Key Export (CSNBKEX)

Use the Key Export verb to re-encipher any type of key (except an IMP-PKA) from encryption under a master key variant to encryption under the same variant of an exporter key-encrypting key. The format of .

The re-enciphered key can be exported to another system.

If the key to be exported is a DATA key, the Key Export verb generates a key token with the same key length as the input token’s key.

This verb supports the no-export bit that the Prohibit Export verb sets in the internal token.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.

Format

The format of CSNBKEX.

```
CSNBKEX(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_type,
    source_key_identifier,
    exporter_key_identifier,
    target_key_identifier)
```

Parameters

The parameter definitions for CSNBKEX.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

`key_type`

Direction: Input
Type: String

The parameter is an 8-byte field that contains either a key type value or the keyword TOKEN. The keyword is left-aligned and padded on the right with blanks.

If the key type is TOKEN, CCA determines the key type from the control vector (CV) field in the internal key token provided in the `source_key_identifier` parameter.

Key type values for the Key Export verb are:

- CIPHER
- DATAMV
- MAC
- CIPHERXI
- DECIPHER
- MACVER
- CIPHERXL
- ENCIPHER
- OKEYXLAT
Key Export (CSNBKEX)

For information about the meaning of the key types, see Table 6 on page 44.

**source_key_identifier**

- **Direction:** Input
- **Type:** String

A 64-byte string of the internal key token that contains the key to be re-enciphered. This parameter must identify an internal key token in application storage, or a label of an existing key in the DES key storage file.

If you supply TOKEN for the key_type parameter, CCA looks at the control vector in the internal key token and determines the key type from this information. If you supply TOKEN for the key_type parameter and supply a label for this parameter, the label must be unique in the DES key storage file.

**exporter_key_identifier**

- **Direction:** Input/Output
- **Type:** String

A 64-byte string of the internal key token or key label that contains the exporter key-encrypting key. This parameter must identify an internal key token in application storage, or a label of an existing key in the key storage file.

If the NOCV bit is on in the internal key token containing the key-encrypting key, the key-encrypting key itself (not the key-encrypting key variant) is used to encipher the generated key.

Control vectors are explained in Control vector on page 40 and the NOCV bit is shown in Table 233 on page 849.

**target_key_identifier**

- **Direction:** Input/Output
- **Type:** String

The 64-byte field external key token that contains the re-enciphered key. The re-enciphered key can be exchanged with another cryptographic system.

**Restrictions**

The restrictions for CSNBKEX.

For security reasons, requests will fail by default if they use an equal key halves exporter to export a key with unequal key halves. You must have access control point 'Key Export - Unrestricted' explicitly enabled if you want to export keys in this manner.

**Required commands**

The CSNBKEX required commands.

This verb requires the Key Export command (offset X'0013') to be enabled in the active role.
By also specifying the **Key Export - Unrestricted** command (offset X'0276'), you can permit a less secure mode of operation that enables an equal key-halves EXPORTER key-encrypting-key to export a key having unequal key-halves (key parity bits are ignored).

**Usage notes**

Usage notes for CSNBKEX.

For Key Export, you can use the following combinations of parameters:

- A valid key type in the `key_type` parameter and an internal key token in the `source_key_identifier` parameter. The key type must be equivalent to the control vector specified in the internal key token.
- A `key_type` parameter of `TOKEN` and an internal key token in the `source_key_identifier` parameter. The `source_key_identifier` can be a label with `TOKEN` only if the label name is unique in the key storage. The key type is extracted from the control vector contained in the internal key token.
- A valid key type in the `key_type` parameter, and a label in the `source_key_identifier` parameter.

If internal key tokens are supplied in the `source_key_identifier` or `exporter_key_identifier` parameters, the key in one or both tokens can be re-enciphered. This occurs if the master key was changed since the internal key token was last used. The return and reason codes that indicate this do *not* indicate which key was re-enciphered. Therefore, assume both keys have been re-enciphered.

Existing internal tokens created with key type `MACD` must be exported with either a `TOKEN` or `DATAM` key type. The external CV will be `DATAM` CV. The `MACD` key type is not supported.

To export a double-length MAC generation or MAC verification key, it is recommended that a key type of `TOKEN` be used.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBKEXJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBKEXJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] key_type,
    byte[] source_key_identifier,
    byte[] exporter_key_identifier,
    byte[] target_key_identifier);
```

**Key Generate (CSNBKKG)**

Use the Key Generate verb to generate an AES key of type DATA, or either one or two odd parity DES keys of any type.
Key Generate (CSNBKGN)

The DES keys can be single-length (8-byte), double-length (16-byte), or, in the case of DATA keys, triple-length (24-byte). The AES keys can be 16, 24 or 32 bytes in length. The Key Generate verb does not produce keys in clear form; all keys are returned in encrypted form. When two keys are generated (DES only), each key has the same clear value, although this clear value is not exposed outside the secure cryptographic feature.

For AES, the verb returns only one copy of the key, enciphered under the AES master key. For DES, the verb selectively returns one copy of the key or two, with each copy enciphered under a user-specified DES key-encrypting key.

This verb returns the key to the application program that called it and the application program can then use the CCA key storage verbs to store the key in the key storage file.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.

Format

The format of CSNBKGN.

```plaintext
CSNBKGN(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_form,
    key_length,
    key_type_1,
    key_type_2,
    kek_key_identifier_1,
    kek_key_identifier_2,
    generated_key_identifier_1,
    generated_key_identifier_2)
```

Parameters

The parameter definitions for CSNBKGN.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

key_form

Direction: Input
Type: String

A 4-byte keyword that defines the type of key you want generated. This parameter also specifies if each key should be returned for either operational, importable, or exportable use. The keyword must be in a 4-byte field, left-aligned, and padded with blanks.

The possible key forms are:

**Operational (OP)**

The key is used for cryptographic operations on the local system. Operational keys are protected by master key variants and can be stored in the CCA key storage file or held by applications in internal key tokens.
Importable (IM)

The key is stored with a file or sent to another system. Importable keys are protected by importer key-encrypting keys.

Exportable (EX)

The key is transported or exported to another system and imported there for use. Exportable keys are protected by exporter key-encrypting keys and cannot be used by CCA verb.

Importable and exportable keys are contained in external key tokens. For more information on key tokens, refer to "Key token" on page 35.

The first two characters refer to key_type_1. The next two characters refer to key_type_2.

The following keywords are allowed: OP, IM, EX, OPIIM, OPEX, IMEX, EXEX, OPOP, and IMIM. See Table 56 for their meanings.

Table 56. Keywords for the Key Generate verb key_form parameter

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EX</td>
<td>Return one copy of the key enciphered under an exporter KEK with key usage EXEX.</td>
</tr>
<tr>
<td>EXEX</td>
<td>Return two copies of the key, both enciphered under exporter key-encrypting keys with key usage EXEX.</td>
</tr>
<tr>
<td>IM</td>
<td>Return one copy of the key enciphered under an importer KEK with key usage IMEX.</td>
</tr>
<tr>
<td>IMEX</td>
<td>Return two copies of the key, the first enciphered under an importer KEK with key usage IMEX, and the second under an exporter KEK with key usage IMEX.</td>
</tr>
<tr>
<td>IMIM</td>
<td>Return two copies of the key, both enciphered under importer KEKs with key usage IMIM.</td>
</tr>
<tr>
<td>OP</td>
<td>Return one copy of the key enciphered under the DES master key.</td>
</tr>
<tr>
<td>OPEX</td>
<td>Return two copies of the key, the first enciphered key under the DES master key and the second under an exporter KEK with key usage OPEX.</td>
</tr>
<tr>
<td>OPIM</td>
<td>Return two copies of the key, the first enciphered key under the DES master key and the second under an importer KEK with key usage OPIM.</td>
</tr>
<tr>
<td>OPOP</td>
<td>Return two copies of the key, both enciphered under the DES master key.</td>
</tr>
</tbody>
</table>

The key forms are defined as follows:

Operational (OP)

The key value is enciphered under a master key. The result is placed into an internal key token. The key is then operational at the local system.

Importable (IM)

The key value is enciphered under an importer key-encrypting key. The result is placed into an external key token. The key can then be imported later to the local node. This key form cannot be used by any CCA verb.

Exportable (EX)

The key value is enciphered under an exporter key-encrypting key. The result is placed into an external key token. The key can then be transported or exported to another system and imported there for use. This key form cannot be used by any CCA verb.

The keys are placed into tokens that the generated_key_identifier_1 and generated_key_identifier_2 parameters identify.
Valid key type combinations depend on the key form. See Table 60 on page 219 for valid key combinations.

**key_length**

**Direction:** Input  
**Type:** String

An 8-byte value that defines the length of the key as being 8, 16, 24 or 32 bytes. The keyword must be left-aligned and padded on the right with blanks. You must supply one of the key length values in the `key_length` parameter. Table 57 lists the key lengths used for various key types.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>SINGLE, SINGLE-R, or KEYLN8</td>
<td>Single length (8-byte or 64-bit) key</td>
<td>DES</td>
</tr>
<tr>
<td>DOUBLE or KEYLN16</td>
<td>Double length (16-byte or 128-bit) key</td>
<td>AES or DES</td>
</tr>
<tr>
<td>DOUBLE-O</td>
<td>Double length (16-byte or 128-bit) key</td>
<td>DES</td>
</tr>
<tr>
<td>KEYLN24</td>
<td>Triple length (24-byte or 192-bit) key</td>
<td>AES or DES</td>
</tr>
<tr>
<td>KEYLN32</td>
<td>32-byte (256-bit) key</td>
<td>AES</td>
</tr>
</tbody>
</table>

AES keys allow only KEYLN16, KEYLN24, and KEYLN32. To generate a 128-bit AES key, specify `key_length` as KEYLN16. For 192-bit AES keys specify `key_length` as KEYLN24. A 256-bit AES key requires a `key_length` of KEYLN32. All AES keys are DATA keys.

Keys with a length of 32 bytes have four 8-byte key parts. This key length is valid only for AES keys. To generate a 32-byte AES key with four different values to be the basis of each key part, specify `key_length` as KEYLN32.

To generate a single-length key, specify `key_length` as SINGLE or KEYLN8.

Double-length (16-byte) keys have an 8-byte left half and an 8-byte right half. Both halves can have identical clear values or not. If you want the same value to be used in both key halves (called replicated key values), specify a `key_length` of SINGLE, SINGLE-R, or KEYLN8. If you want different values to be the basis of each key half, specify a `key_length` of DOUBLE or KEYLN16.

Triple-length (24-byte) keys have three 8-byte key parts. This key length is valid for DATA keys only. To generate a triple-length DATA key with three different values to be the basis of each key part, specify a `key_length` of KEYLN24.

Use SINGLE or SINGLE-R if you want to create a DES transport key that you would use to exchange DATA keys with a PCF system. Because PCF does not use double-length transport keys, specify SINGLE so that the effects of multiple encipherment are nullified.

When generating an AKEK, the `key_length` parameter is ignored. The AKEK key length (8-byte or 16-byte) is determined by the skeleton token created by the Key Token Build verb and provided in the `generated_key_identifier_1` parameter.

The key length specified must be consistent with the key length indicated by the token you supply. For DES keys, this length is a field in the control vector.
For AES keys, the length is an explicit field in the token. Table 58 shows the valid key lengths for each key type. An X indicates that a key length is permitted for a key type. A Y indicates that the key generated will be a double-length key with replicated key values. It is preferred that SINGLE-R be used for this result.

Table 58. Key Generate - key lengths for each key type

<table>
<thead>
<tr>
<th>Key Type</th>
<th>SINGLE KEYLN8</th>
<th>SINGLE-R</th>
<th>DOUBLE KEYLN16</th>
<th>DOUBLE-O</th>
<th>Triple (KEYLN24)</th>
<th>Triple (KEYLN32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AESDATA</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACVER</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATAM</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATAMV</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXPORTER</td>
<td>Y</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMPORTER</td>
<td>Y</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IKEYXLAT</td>
<td>Y</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OKEYXLAT</td>
<td>Y</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CIPHER</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECIIPHER</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENCIIPHER</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PINENC</td>
<td>Y</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPINENC</td>
<td>Y</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PINGEN</td>
<td>Y</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PINVER</td>
<td>Y</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARDEC*</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARENC*</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARPINE*</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARXCVL*</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARXCVR*</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DKYGENKY*</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEYGENKY*</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CIPHERXI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>CIPHERXL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>CIPHERXO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

**Note:** Key types marked with an asterisk (*) are requested through the use of the TOKEN keyword and specifying a proper control vector in a key token.

**key_type_1**

- **Direction:** Input
- **Type:** String

An 8-byte keyword from the following group:

<table>
<thead>
<tr>
<th>AESDATA</th>
<th>DATA</th>
<th>ENCIIPHER</th>
<th>MACVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>AESTOKEN</td>
<td>DATAC</td>
<td>EXPORTER</td>
<td>OKEYXLAT</td>
</tr>
<tr>
<td>CIPHER</td>
<td>DATAM</td>
<td>IKEYXLAT</td>
<td>OPINENC</td>
</tr>
<tr>
<td>CIPHERXI</td>
<td>DATAMV</td>
<td>IMPORTER</td>
<td>PINGEN</td>
</tr>
<tr>
<td>CIPHERXL</td>
<td>DATAXLAT</td>
<td>PINENC</td>
<td>PINVER</td>
</tr>
<tr>
<td>CIPHERXO</td>
<td>DECIPHER</td>
<td>MAC</td>
<td></td>
</tr>
</tbody>
</table>

or the keyword TOKEN.
Key Generate (CSNBKGN)

For information on the meaning of the key types, see Table 6 on page 44.

Use the key_type_1 parameter for the first, or only key, that you want generated. The keyword must be left-aligned and padded with blanks. Valid type combinations depend on the key form.

If key_type_1 is TOKEN, CCA examines the control vector (CV) field in the generated_key_identifier_1 parameter to derive the key type. When key_type_1 is TOKEN, CCA does not check for the length of the key for DATA keys. Instead, it uses the key_length parameter to determine the length of the key.

Use the AESTOKEN keyword for AES keys, or the TOKEN keyword for DES keys to indicate that the verb should determine the key type from the key token that you supply. For AES, all keys are type AESDATA. For DES, the key type is determined from the control vector in the key tokens. Alternatively, you can specify the key type using keywords shown in Table 59 on page 219 and Table 60 on page 219.

Key types can have mandatory key forms. For example, CVARENC keys must be generated in pairs with CVARDEC keys. The reason is that a CVARENC key can only be used for encryption, and without a CVARDEC key you cannot decrypt the data. See Table 59 on page 219 and Table 60 on page 219 for valid key type and key form combinations.

key_type_2

Direction: Input
Type: String

An 8-byte keyword from the following group:

<table>
<thead>
<tr>
<th>AESDATA</th>
<th>DATA</th>
<th>ENCIIPHER</th>
<th>MACVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>AESTOKEN</td>
<td>DATAC</td>
<td>EXPORTER</td>
<td>OKEYXLAT</td>
</tr>
<tr>
<td>CIPHER</td>
<td>DATAM</td>
<td>IKEYXLAT</td>
<td>OPINENC</td>
</tr>
<tr>
<td>CIPHERXI</td>
<td>DATAMV</td>
<td>IMPORTER</td>
<td>PINGEN</td>
</tr>
<tr>
<td>CIPHERXL</td>
<td>DATAXLAT</td>
<td>IPINENC</td>
<td>PINVER</td>
</tr>
<tr>
<td>CIPHERXO</td>
<td>DECIPHER</td>
<td>MAC</td>
<td></td>
</tr>
</tbody>
</table>

or the keyword TOKEN.

For information on the meaning of the key types, see Table 6 on page 44.

Use the key_type_2 parameter for a key pair, which is shown in Table 60 on page 219. The keyword must be left-aligned and padded with blanks. Valid type combinations depend on the key form.

If key_type_2 is TOKEN, CCA examines the control vector (CV) field in the generated_key_identifier_2 parameter to derive the key type. When key_type_2 is TOKEN, CCA does not check for the length of the key for DATA keys. Instead, it uses the key_length parameter to determine the length of the key.

If you want only one key to be generated, specify the key_type_2 and KEK_key_identifier_2 as binary zeros.

See Table 59 on page 219 and Table 60 on page 219 for valid key type and key form combinations.

KEK_key_identifier_1

Direction: Input/Output
Type: String

A 64-byte string of an internal key token containing the importer or exporter.
key-encrypting key, or a key label. If you supply a key label that is less than 64-bytes, it must be left-aligned and padded with blanks. KEK_key_identifier_1 is required for a key_form of IM, EX, IMEX, EXEX, or IMIM.

If the key_form is OP, OPEX, OPIM, or OPOP, the KEK_key_identifier_1 is null.

If the NOCV bit is on in the internal key token containing the key-encrypting key, the key-encrypting key itself (not the key-encrypting key variant) is used to encipher the generated key.

Control vectors are explained in “Control vector” on page 40 and the NOCV bit is shown in Table 233 on page 849.

This parameter is not used when generating AES keys, and should point to null key-tokens.

 KEK_key_identifier_2

Direction:       Input/Output
Type:           String

A 64-byte string of an internal key token containing the importer or exporter key-encrypting key, or a key label of an internal token. If you supply a key label that is less than 64-bytes, it must be left-aligned and padded with blanks. KEK_key_identifier_2 is required for a key_form of OPIM, OPEX, IMEX, IMIM, or EXEX. This field is ignored for key_form keywords OP, IM and EX.

If the NOCV bit is on in the internal key token containing the key-encrypting key, the key-encrypting key itself (not the key-encrypting key variant) is used to encipher the generated key.

Control vectors are explained in “Control vector” on page 40 and the NOCV bit is shown in Table 233 on page 849.

This parameter is not used when generating AES keys, and should point to null key-tokens.

generated_key_identifier_1

Direction:       Input/Output
Type:           String

This parameter specifies either a generated:
• Internal key token for an operational key form, or
• External key token containing a key enciphered under the kek_key_identifier_1 parameter.

When key_type_1 parameter is AESDATA, the generated_key_identifier_1 parameter is ignored. In this case, it is recommended that the parameter be initialized to 64-bytes of X’00’.

If you specify a key_type_1 of TOKEN, then this field contains a valid token of the key type you want to generate. Otherwise, on input, this parameter must be binary zeros. See key_type_1 for a list of valid key types.

If you specify a key_type_1 of IMPORTER or EXPORTER and a key_form of OPEX, and if the generated_key_identifier_1 parameter contains a valid internal token of the same type, the NOCV bit, if on, is propagated to the generated key token.

Using the AESTOKEN or TOKEN keyword in the key type parameters requires that the key tokens already exist when the verb is called, so the information in those tokens can be used to determine the key type:
Key Generate (CSNBKGN)

- The `key_type_1` parameter overrides the type in the token.
- The `key_length` parameter overrides the length value in the generated key token.

In general, unless you are using the `AESTOKEN` or `TOKEN` keyword, you must identify a null key token in the generated key identifier parameters on input.

`generated_key_identifier_2`

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>String</td>
</tr>
</tbody>
</table>

This parameter specifies a generated external key token containing a key enciphered under the `kek_key_identifier_2` parameter.

If you specify a `key_type_2` of `TOKEN`, then this field contains a valid token of the key type you want to generate. Otherwise, on input, this parameter must be binary zeros. See `key_type_1` for a list of valid key types.

The token can be an internal or external token.

Using the `AESTOKEN` or `TOKEN` keyword in the key type parameters requires that the key tokens already exist when the verb is called, so the information in those tokens can be used to determine the key type. In general, unless you are using the `AESTOKEN` or `TOKEN` keyword, you must identify a null key token in the generated key identifier parameters on input.

Restrictions

The restrictions for CSNBKGN.

None.

Required commands

The CSNBKGN required commands.

Depending on the `key_type` and `key_form` parameters selected, the verb could require one or more of these commands to be enabled in the active role:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'008C'</td>
<td>Key Generate - Key set</td>
</tr>
<tr>
<td>X'008E'</td>
<td>Key Generate - OP</td>
</tr>
<tr>
<td>X'00D7'</td>
<td>Key Generate - Key set extended</td>
</tr>
<tr>
<td>X'00DB'</td>
<td>Key Generate - SINGLE-R</td>
</tr>
</tbody>
</table>

Note: A role with offset X'00DB' enabled can also possibly use the Remote Key Export verb to replicate a single-length source key.

Usage notes

The usage notes for CSNBKGN.

Table 59 on page 219 shows the valid key type and key form combinations for a single key. Key types marked with an "*" must be requested through the
specification of a proper control vector in a key token and through the use of the \texttt{TOKEN} keyword. See also Chapter 20, “Key forms and types used in the Key Generate verb,” on page 985.

Note: Not all key types are valid on all hardware. See Table 6 on page 44.

For AES keys, only key form \texttt{OP} is supported. AES keys cannot be generated in pairs.

\textit{Table 59. Keywords for Key Generate, valid key types and key forms for a single key}

<table>
<thead>
<tr>
<th>Key Type 1</th>
<th>Key Type 2</th>
<th>OP</th>
<th>IM</th>
<th>EX</th>
</tr>
</thead>
<tbody>
<tr>
<td>AESDATA</td>
<td>Not applicable</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AESTOKEN</td>
<td>Not applicable</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>Not applicable</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DATAC*</td>
<td>Not applicable</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DATAM</td>
<td>Not applicable</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DKYGENKY*</td>
<td>Not applicable</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>KEYGENKY*</td>
<td>Not applicable</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>MAC</td>
<td>Not applicable</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PINGEN</td>
<td>Not applicable</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

\textit{Table 60} shows the valid key type and key form combinations for a key pair.

\textit{Table 60. Keywords for Key Generate, valid key types and key forms for a key pair}

<table>
<thead>
<tr>
<th>Key Type 1</th>
<th>Key Type 2</th>
<th>OPEX</th>
<th>EXEX</th>
<th>OPIM, OPOP, IMIM</th>
<th>IMEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIPHER</td>
<td>CIPHER</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CIPHER</td>
<td>DECPHER</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CIPHER</td>
<td>ENCIPHER</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CVARDEC*</td>
<td>CVARENC*</td>
<td>E</td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARDECC*</td>
<td>CVARPINE*</td>
<td>E</td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARENC*</td>
<td>CVARDEC*</td>
<td>E</td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARENC*</td>
<td>CVARXCVL*</td>
<td>E</td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARENC*</td>
<td>CVARXCVR*</td>
<td>E</td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARXCVL*</td>
<td>CVARENC*</td>
<td>E</td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARXCVR*</td>
<td>CVARENC*</td>
<td>E</td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARPINE*</td>
<td>CVARDEC*</td>
<td>E</td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>DATA</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DATA</td>
<td>DATAXLAT</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DATAC*</td>
<td>DATAC*</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DATAM</td>
<td>DATAM</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DATAM</td>
<td>DATAMV</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DATAXLAT</td>
<td>DATAXLAT</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>DECPHER</td>
<td>CIPHER</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DECPHER</td>
<td>ENCIPHER</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
### Table 60. Keywords for Key Generate, valid key types and key forms for a key pair (continued)

<table>
<thead>
<tr>
<th>Key Type 1</th>
<th>Key Type 2</th>
<th>OPEX</th>
<th>EXEX</th>
<th>OPIM, OPOP, IMIM</th>
<th>IMEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>DKYGENKY*</td>
<td>DKYGENKY*</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
</tr>
<tr>
<td>ENCIPHER</td>
<td>CIPHER</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
</tr>
<tr>
<td>ENCIPHER</td>
<td>DECIPHER</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
</tr>
<tr>
<td>EXPORTER</td>
<td>IKEYXLAT</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>EXPORTER</td>
<td>IMPORTER</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>IKEYXLAT</td>
<td>EXPORTER</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>IKEYXLAT</td>
<td>OKEYXLAT</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>IMPORTER</td>
<td>EXPORTER</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>IMPORTER</td>
<td>OKEYXLAT</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>IPINENC</td>
<td>OPINENC</td>
<td>(X)</td>
<td>(X)</td>
<td>(E)</td>
<td>(X)</td>
</tr>
<tr>
<td>KEYGENKY*</td>
<td>KEYGENKY*</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
</tr>
<tr>
<td>MAC</td>
<td>MAC</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
</tr>
<tr>
<td>MAC</td>
<td>MACVER</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
</tr>
<tr>
<td>OKEYXLAT</td>
<td>IKEYXLAT</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>OKEYXLAT</td>
<td>IMPORTER</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>OPINENC</td>
<td>IPINENC</td>
<td>(X)</td>
<td>(X)</td>
<td>(E)</td>
<td>(X)</td>
</tr>
<tr>
<td>OPINENC</td>
<td>OPINENC</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>PINVER</td>
<td>PINGEN</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>PINGEN</td>
<td>PINVER</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td></td>
</tr>
</tbody>
</table>

**Note:**
1. AES keys cannot be generated in pairs.
2. An 'X' indicates a permissible key type combination for a given key form. An 'E' indicates that a special (Extended) command is required as those keys require special handling.
3. The key types marked with an '*' must be requested through the specification of a proper control-vector in a key token and the use of the TOKEN keyword.

### JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNBKGNJ.

See [Building Java applications using the CCA JNI](#) on page 28.

**Format**

```java
class CSNBKGNJ{
    public native void CSNBKGNJ(
        hikmNativeNumber return_code,
        hikmNativeNumber reason_code,
        hikmNativeNumber exit_data_length,
        byte[] exit_data,
        byte[] key_form,
        byte[] key_length,
        byte[] key_type_1,
        byte[] key_type_2,
        byte[] KEK_key_identifier_1,
    )
}
```
Key Generate2 (CSNBKGN2)

Use the Key Generate2 verb to randomly generate a keyed hash message authentication code (HMAC) key or an AES key. Depending on the key form specified, the verb returns either one or two enciphered copies of the key, each in a variable-length symmetric key-token. Key tokens can be returned to either application storage or AES key storage.

To generate keys that are returned in a fixed-length symmetric key-token, see “Key Generate (CSNBKGN)” on page 211.

The CSNBKGN2 verb selectively returns one or two copies of an AES or an HMAC key enciphered under the AES master key or an AES key-encrypting key. Keys enciphered under the master key are immediately usable at the local node.

The verb can create default key tokens, update the key in existing key tokens, or complete skeleton key tokens. You can use the Key Token Build2 verb to build a skeleton key token (see “Key Token Build2 (CSNBKTB2)” on page 264.

**Note:** Variable-length symmetric key tokens have an associated data section that contains clear data. This section is hashed and the hash value is cryptographically bound to its enciphered payload. See “Variable-length symmetric key tokens” on page 889.

To use this verb, specify the following:

- Two required rule array keywords:
  - a required token algorithm keyword that selects the type of algorithm that the key can be used for (either AES or HMAC)
  - a required key form keyword that selects the number of keys to return, either one or two, and the token type for each key, either internal or external.
- The number of bits of clear-key data to randomly generate and return encrypted in the generated key or keys
  - AES keys can be 128, 192, or 256 bits
  - HMAC keys can be 80 - 2048 bits
Any generated key will have the specified key length.
- The key types of each AES or HMAC key to be returned. A key type of TOKEN indicates that the generated key token provided as input is to be updated.

**Note:**

1. When generating only one copy of the key, use eight space characters for the second key-type variable.
2. To update an existing key token with a copy of the randomly generated key, specify keyword TOKEN as the key type and identify the key token to be updated using the generated_key_identifier_1 parameter.
3. If the key type is not TOKEN, the generated_key_identifier_1 parameter must have a length of zero or point to a null key-token. This results in a key token with default key-usage and key-management fields.
Key Generate2 (CSNBKGN2)

- The optional key name (label), 64 bytes, of one or both keys that is to be placed in the associated data of the key token; if provided, this data overrides any label in the input generated key token.
- The optional user-defined associated data, up to 255 bytes, of one or both keys that is to be placed in the associated data of the key token; if provided, this data overrides any user-defined associated data in the input generated key token.

**Note:** A `user_associated_data_n` variable that contains data overrides any user-defined associated data contained in a key token to be updated.

- A key-encrypting key (KEK) identifier of key type EXPORTER or IMPORTER, contained in an internal variable-length symmetric key-token, for wrapping each external key to be returned.
- The key identifier for each key to be generated.

**Format**

The format of CSNBKGN2.

```c
CSNBKGN2(
    return_code, 
    reason_code, 
    exit_data_length, 
    exit_data, 
    rule_array_count, 
    rule_array, 
    clear_key_bit_length, 
    key_type_1, 
    key_type_2, 
    key_name_1_length, 
    key_name_1, 
    key_name_2_length, 
    key_name_2, 
    user_associated_data_1_length, 
    user_associated_data_1, 
    user_associated_data_2_length, 
    user_associated_data_2, 
    key_encrypting_key_identifier_1_length, 
    key_encrypting_key_identifier_1, 
    key_encrypting_key_identifier_2_length, 
    key_encrypting_key_identifier_2, 
    generated_key_identifier_1_length, 
    generated_key_identifier_1, 
    generated_key_identifier_2_length, 
    generated_key_identifier_2)
```

**Parameters**

The parameter definitions for CSNBKGN2.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

The number of keywords you supplied in the `rule_array` parameter. This value must be 2, 3, or 4.

**rule_array**
Key Generate2 (CSNBKGN2)

The `rule_array` contains keywords that provide control information to the verb. The keywords must be in contiguous storage with each of the keywords left-aligned in its own 8-byte location and padded on the right with blanks. The `rule_array` keywords are described in Table 61.

### Table 61. Keywords for Key Generate2 control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Token algorithm</strong> (One required)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>Specifies to generate an AES key token.</td>
</tr>
<tr>
<td>HMAC</td>
<td>Specifies to generate an HMAC key token.</td>
</tr>
<tr>
<td><strong>Key form</strong> (One, required) The first two characters refer to <code>key_type_1</code>. The next two characters refer to <code>key_type_2</code>. See “Usage notes” on page 228 for details.</td>
<td></td>
</tr>
<tr>
<td>EX</td>
<td>Return one copy of the key enciphered under an exporter KEK with key usage GEN-EXEX.</td>
</tr>
<tr>
<td>EXEX</td>
<td>Return two copies of the key, both enciphered under exporter key-encrypting keys with key usage GEN-EXEX.</td>
</tr>
<tr>
<td>IM</td>
<td>Return one copy of the key enciphered under an importer KEK with key usage GEN-IMEX.</td>
</tr>
<tr>
<td>IMEX</td>
<td>Return two copies of the key, the first enciphered under an importer KEK with key usage GEN-IMEX, and the second under an exporter KEK with key usage GEN-IMEX.</td>
</tr>
<tr>
<td>IMIM</td>
<td>Return two copies of the key, both enciphered under importer KEKs with key usage GEN-IMIM.</td>
</tr>
<tr>
<td>OP</td>
<td>Return one copy of the key enciphered under the AES master key.</td>
</tr>
<tr>
<td>OPEX</td>
<td>Return two copies of the key, the first enciphered under the AES master key and the second under an exporter KEK with key usage GEN-OPEX.</td>
</tr>
<tr>
<td>OPIM</td>
<td>Return two copies of the key, the first enciphered under the AES master key and the second under an importer KEK with key usage GEN-OPIM.</td>
</tr>
<tr>
<td>OPOP</td>
<td>Return two copies of the key, both enciphered under the AES master key.</td>
</tr>
</tbody>
</table>

### Payload Version for generated_key_identifier_1 (one, optional)

Note: If `TOKEN` is specified for `key_type_1`, the payload format version is determined by the information in the key token identified by the `generated_key_identifier_1` parameter unless specifically overridden by one of the following keywords.

| V0PYLDK1 | Return a key token identified by the `generated_key_identifier_1` parameter with a payload formatted using the less secure legacy variable-length version 0 format. This is the default if the `key_type_1` variable is not valued to `TOKEN` and the key type is AES CIPHER, AES EXPORTER, AES IMPORTER, or HMAC MAC. Only valid with those key types. |

| V1PYLDK1 | Return a key token using the `generated_key_identifier_1` parameter with a payload formatted using the more secure fixed-length version 1 format. This is the default if the `key_type_1` variable is not valued to `TOKEN` and the key type is not AES CIPHER, AES EXPORTER, AES IMPORTER, or HMAC MAC. Not valid with HMAC MAC. Note: This option produces a key token that is not compatible with releases before Release 4.4. |

### Payload Version for generated_key_identifier_2 (one, optional when generating a key pair, otherwise not allowed)

Note: If `TOKEN` is specified for `key_type_2` when generating a key pair, the payload format version is determined by the information in the key token identified by the `generated_key_identifier_2` parameter unless specifically overridden by one of the following keywords.

| V0PYLDK2 | Return a key token identified by the `generated_key_identifier_2` parameter with a payload formatted using the original variable-length version 0 format. This is the default if the `key_type_2` variable is not valued to `TOKEN` and the key type is AES CIPHER, AES EXPORTER, AES IMPORTER, or HMAC MAC. Only valid with those key types. |
Table 61. Keywords for Key Generate2 control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1PYLDK2</td>
<td>Return a key token using the <code>generated_key_identifier_2</code> parameter with a payload formatted using the more secure fixed-length version 1 format. This is the default if the <code>key_type_2</code> variable is not valued to <code>TOKEN</code> and the key type is not AES CIPHER, AES EXPORTER, AES IMPORTER, or HMAC MAC. Not valid with HMAC MAC. Note: This option produces a key token that is not compatible with releases before Release 4.4.</td>
</tr>
</tbody>
</table>

**clear_key_bit_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of clear-key bits to randomly generate and return encrypted in the generated key or keys. If a generated key token has a key type of `TOKEN`, this value overrides any key length contained in the key token. The value can be 128, 192, and 256 for AES keys, and 80 - 2048 for HMAC keys.

**key_type_1, key_type_2**

- **Direction:** Input
- **Type:** String

The `key_type_1` and `key_type_2` parameters are pointers to 8-byte string variables, each containing a keyword that is left aligned and padded on the right with space characters. The keyword specifies the key type of the key being generated. If a single copy of the key is being generated, set the `key_type_2` variable to eight space characters.

The verb returns each copy of the generated key in a default key token that it builds, or updates a key token that is provided. Keyword `TOKEN` indicates that the verb is to return an updated key token that contains the key-usage and key-management fields of the key token that is provided by the corresponding `key_identifier_1` or `key_identifier_2` parameter. A keyword other than `TOKEN` indicates that a null key-token is provided and that the verb is to build and return a default key-token for the specified key type (AES key types CIPHER, EXPORTER, or IMPORTER only).

Valid type combinations depend on the key form, and are documented in [Table 64 on page 229](#) and [Table 65 on page 229](#).

The 8-byte keyword for the `key_type_1` or `key_type_2` parameters can be one of the following:

**Table 62. Keywords and associated algorithms for key_type_1/2 parameter**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIPHER</td>
<td>AES</td>
</tr>
<tr>
<td>EXPORTER</td>
<td>AES</td>
</tr>
<tr>
<td>IMPORTER</td>
<td>AES</td>
</tr>
<tr>
<td>MAC</td>
<td>AES or HMAC</td>
</tr>
<tr>
<td>MACVER</td>
<td>HMAC</td>
</tr>
</tbody>
</table>

Specify the keyword `TOKEN` when supplying a key token in the `generated_key_identifier_1/2` parameter.
If `key_type_1` or `key_type_2` is **TOKEN**, the associated data in the `generated_key_identifier_1` or `generated_key_identifier_2` parameter is used to derive the key type.

**key_name_1_length**

**Direction:** Input  
**Type:** Integer

The length of the `key_name` parameter for `generated_key_identifier_1`. Valid values are 0 and 64.

**key_name_1**

**Direction:** Input  
**Type:** String

A pointer to a string variable containing the optional key label that is placed in the associated data of the key token identified by the `generated_key_identifier_1` variable. If present, it must be a valid key label. This data is cryptographically bound to the first copy of the key.

**key_name_2_length**

**Direction:** Input  
**Type:** Integer

The length of the `key_name` parameter for `generated_key_identifier_2`. Valid values are 0 and 64. When only one key is being generated, set this value to 0.

**key_name_2**

**Direction:** Input  
**Type:** String

A pointer to a string variable containing the optional key label that is placed in the associated data of the key token identified by the `generated_key_identifier_2` variable. If present, it must be a valid key label. This data is cryptographically bound to the first copy of the key.

When only one key is being generated, this parameter is ignored.

**user_associated_data_1_length**

**Direction:** Input  
**Type:** Integer

The length of the user-associated data parameter for `generated_key_identifier_1`. The valid values are 0 - 255 bytes.

**user_associated_data_1**

**Direction:** Input  
**Type:** String

User-associated data to be stored in the associated data structure for `generated_key_identifier_1`.

**user_associated_data_2_length**

**Direction:** Input  
**Type:** Integer

The length of the user-associated data parameter for `generated_key_identifier_2`. 

---

Key Generate2 (CSNBKGN2)
The valid values are 0 - 255 bytes. When only one key is being generated, this parameter is ignored.

**user-associated-data_2**

**Direction:** Input  
**Type:** String

User associated data to be stored in the associated data structure for `generated_key_identifier_2`.

When only one key is being generated, this parameter is ignored.

**key-encrypting-key-identifier_1_length**

**Direction:** Input  
**Type:** Integer

The length of the buffer for `key-encrypting-key-identifier_1` in bytes. When the key form rule is OP, OPOP, OPIM, or OPEX, this length must be zero. When the key form rule is EX, EXEX, IM, IMEX, or IMIM, the value must be between the actual length of the token and 725 when `key-encrypting-key-identifier_1` is a token.

The value must be 64 bytes when `key-encrypting-key-identifier_1` is a label.

**key-encrypting-key-identifier_1**

**Direction:** Input  
**Type:** String

When `key-encrypting-key-identifier_1_length` is zero, this parameter is ignored. Otherwise, `key-encrypting-key-identifier_1` contains an internal key token containing the AES importer or exporter key-encrypting key, or a key label.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

**key-encrypting-key-identifier_2_length**

**Direction:** Input  
**Type:** Integer

The length of the buffer for `key-encrypting-key-identifier_2` in bytes. When the key form rule is OPOP, this length must be zero. When the key form rule is EXEX, IMEX, IMIM, OPIM, or OPEX, the value must be between the actual length of the token and 725 when `key-encrypting-key-identifier_2` is a token. The value must be 64 when `key-encrypting-key-identifier_2` is a label.

When only one key is being generated, this parameter is ignored.

**key-encrypting-key-identifier_2**

**Direction:** Input/Output  
**Type:** String

When `key-encrypting-key-identifier_2_length` is zero, this parameter is ignored. Otherwise, `key-encrypting-key-identifier_2` contains an internal key token containing the AES importer or exporter key-encrypting key, or a key label.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

When only one key is being generated, this parameter is ignored.

**generated_key_identifier_1_length**
Key Generate2 (CSNBKGN2)

Direction: Input/Output
Type: Integer

On input, the length of the buffer for the generated_key_identifier_1 parameter in bytes. The maximum value is 900 bytes.

On output, the parameter will hold the actual length of the generated_key_identifier_1.

generated_key_identifier_1

Direction: Input/Output
Type: String

The buffer for the first generated key token.

On input, if you specify a key_type_1 of TOKEN, then the buffer contains a valid key token of the key type you want to generate. The key token must be left-aligned in the buffer. Otherwise, this parameter must be binary zeros. See "key_type_1, key_type_2" on page 224 for valid key types.

On output, the buffer contains the generated key token.

generated_key_identifier_2_length

Direction: Input/Output
Type: Integer

On input, the length of the buffer for the generated_key_identifier_2 in bytes. The minimum value is 120 bytes and the maximum value is 725 bytes. The maximum value is 900 bytes.

On output, the parameter will hold the actual length of the generated_key_identifier_2.

When only one key is being generated, this parameter is ignored.

generated_key_identifier_2

Direction: Input/Output
Type: String

The buffer for the second generated key token.

On input, if you specify a key_type_2 of TOKEN, then the buffer contains a valid key token of the key type you want to generate. The key token must be left-aligned in the buffer. Otherwise, this parameter must be binary zeros. See "key_type_1, key_type_2" on page 224 for valid key types.

On output, the buffer contains the generated key token.

When only one key is being generated, this parameter is ignored

Restrictions

The restrictions for CSNBKGN2.

None.

Required commands

The CSNBKGN2 required commands.
Depending on your specification of key form and key type, different commands are required to enable the processing of the Key Generate2 verb.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00EA'</td>
<td><strong>Key Generate2 - OP</strong></td>
</tr>
<tr>
<td></td>
<td>This command is required, if key form and key type combinations are specified that are shown with an X in Table 64 on page 229.</td>
</tr>
<tr>
<td>X'00EB'</td>
<td><strong>Key Generate2 - Key set</strong></td>
</tr>
<tr>
<td></td>
<td>This command is required, if key form and key type combinations are specified that are shown with an X in Table 65 on page 229.</td>
</tr>
<tr>
<td>X'00EC'</td>
<td><strong>Key Generate2 - Key set extended</strong></td>
</tr>
<tr>
<td></td>
<td>This command is required, if key form and key type combinations are specified that are shown with an E in Table 65 on page 229.</td>
</tr>
</tbody>
</table>

To disallow the wrapping of a key with a weaker key-encrypting key, enable the **Prohibit weak wrapping - Transport keys** command (offset X'0328') in the active role. This command affects multiple verbs. See Chapter 24, “Access control points and verbs,” on page 1047.

To receive a warning when wrapping a key with a weaker key-encrypting key, enable the **Warn when weak wrap - Transport keys** command (offset X'032C') in the active role. The **Prohibit weak wrapping - Transport keys** command (offset X'0328') overrides this command.

To disable the wrapping of a key with a weaker master key, the **Prohibit weak wrapping - Master keys** command (offset X'0333') must be enabled in the active role.

To receive a warning when wrapping a key with a weaker master key, enable the **Warn when weak wrap - Master keys** command (offset X'0332') in the active role. The **Prohibit weak wrapping - Master keys** command overrides this command.

The following access-control points support DK keys (DK enabled AES key types MAC, PINCALC, PINPROT, and PINPRW):  

<table>
<thead>
<tr>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'02BB'</td>
<td><strong>Key Generate2 - DK PIN key set</strong></td>
</tr>
<tr>
<td>X'02BC'</td>
<td><strong>Key Generate2 - DK PIN print key</strong></td>
</tr>
<tr>
<td>X'02BD'</td>
<td><strong>Key Generate2 - DK PIN admin1 key set PINPROT</strong></td>
</tr>
<tr>
<td>X'02BE'</td>
<td><strong>Key Generate2 - DK PIN admin1 key set MAC</strong></td>
</tr>
<tr>
<td>X'02BF'</td>
<td><strong>Key Generate2 - DK PIN admin2 key set MAC</strong></td>
</tr>
</tbody>
</table>

**Usage notes**

Read the contained usage notes and related information for the CSNBKGN2 verb, especially about the key type and key form specifications.
The key forms are defined as follows:

**Operational (OP)**
The key value is enciphered under a master key. The result is placed into an internal key token. The key is then operational at the local system.

**Importable (IM)**
The key value is enciphered under an importer key-encrypting key. The result is placed into an external key token. The corresponding `key_encrypting_key_identifier_n` parameter must contain an AES IMPORTER key token or label.

**Exportable (EX)**
The key value is enciphered under an exporter key-encrypting key. The result is placed into an external key token. The corresponding `key_encrypting_key_identifier_n` parameter must contain an AES EXPORTER key token or label.

**Key type specifications:** Generated AES and HMAC keys returned in an internal key token are enciphered with the AES master key, while generated keys returned in an external key token are enciphered under an AES key-encrypting key.

There are two methods for specifying the type of keys to be generated:
- One or two key type keywords are examined depending on the value of the key form rule-array keyword. Table 64 shows the permissible key type and key form keyword combinations to generate a single copy of a key. Table 65 shows the permissible key type and key form keyword combinations to generate two copies of a key.
- Use the TOKEN keyword and provide a key token to be updated or a skeleton key-token to be completed.

**Table 64. Key Generate2 key_type and key_form keywords for one AES or HMAC key**

<table>
<thead>
<tr>
<th>key_type_1</th>
<th>Required key usage</th>
<th>Key form OP</th>
<th>Key form IM or EX</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIPHER</td>
<td>DECRYPT and ENCRYPT</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DKEYGENKY</td>
<td>D-ALL</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DKEYGENKY</td>
<td>D-CIPHER</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DKEYGENKY</td>
<td>D-MAC</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>MAC</td>
<td>GENERATE and VERIFY</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PINCALC</td>
<td>GENONLY</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Note:**
1. An X indicates a permissible key type and key usage combination for a given key form.
2. The key types marked with an asterisk must be requested through the specification of a proper key usage in a key token and the use of the TOKEN keyword. These key types are not recognized by the verb as key type keywords.

**Table 65. Key Generate2 key_type and key_form keywords for a pair of AES or HMAC keys**

<table>
<thead>
<tr>
<th>key_type_1 (usage)</th>
<th>key_type_2 (usage)</th>
<th>key_form</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>OPOP, OPIM, IMIM</td>
</tr>
<tr>
<td>CIPHER</td>
<td>CIPHER</td>
<td>X</td>
</tr>
<tr>
<td>CIPHER</td>
<td>CIPHER (DECRYPT)</td>
<td>X</td>
</tr>
</tbody>
</table>
### Key Generate2 (CSNBKGN2)

Table 65. Key Generate2 key_type and key_form keywords for a pair of AES or HMAC keys (continued)

<table>
<thead>
<tr>
<th>key_type_1 (usage)</th>
<th>key_type_2 (usage)</th>
<th>key_form</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>OPOP, OPIM, IMIM</td>
</tr>
<tr>
<td>CIPHER</td>
<td>CIPHER (ENCRIPT)</td>
<td>X</td>
</tr>
<tr>
<td>CIPHER</td>
<td>CIPHER (DECRIPT C-XLATE)</td>
<td>X</td>
</tr>
<tr>
<td>CIPHER</td>
<td>CIPHER (ENCRIPT C-XLATE)</td>
<td>X</td>
</tr>
<tr>
<td>CIPHER</td>
<td>CIPHER (DECRIPT C-XLATE)</td>
<td>X</td>
</tr>
<tr>
<td>CIPHER (DECRIPT)</td>
<td>CIPHER (ENCRIPT)</td>
<td>X</td>
</tr>
<tr>
<td>CIPHER (DECRIPT)</td>
<td>CIPHER (DECRIPT)</td>
<td></td>
</tr>
<tr>
<td>CIPHER (DECRIPT)</td>
<td>CIPHER (ENCRIPT)</td>
<td>X</td>
</tr>
<tr>
<td>CIPHER (ENCRIPT)</td>
<td>CIPHER (DECRIPT C-XLATE)</td>
<td>X</td>
</tr>
<tr>
<td>CIPHER (ENCRIPT)</td>
<td>CIPHER (ENCRIPT C-XLATE)</td>
<td></td>
</tr>
<tr>
<td>CIPHER (DECRIPT C-XLATE)</td>
<td>CIPHER (ENCRIPT C-XLATE)</td>
<td>X</td>
</tr>
<tr>
<td>CIPHER (DECRIPT C-XLATE)</td>
<td>CIPHER (DECRIPT)</td>
<td></td>
</tr>
<tr>
<td>CIPHER (DECRIPT C-XLATE)</td>
<td>CIPHER (ENCRIPT)</td>
<td>X</td>
</tr>
<tr>
<td>CIPHER (DECRIPT C-XLATE)</td>
<td>CIPHER (DECRIPT C-XLATE)</td>
<td>E</td>
</tr>
<tr>
<td>CIPHER (ENCRIPT C-XLATE)</td>
<td>CIPHER (ENCRIPT C-XLATE)</td>
<td>X</td>
</tr>
<tr>
<td>CIPHER (ENCRIPT C-XLATE)</td>
<td>CIPHER (DECRIPT)</td>
<td>X</td>
</tr>
</tbody>
</table>
Table 65. Key Generate2 key_type and key_form keywords for a pair of AES or HMAC keys (continued)

<table>
<thead>
<tr>
<th>key_type_1 (usage)</th>
<th>key_type_2 (usage)</th>
<th>key_form</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIPHER (ENCRYPT C-XLATE)</td>
<td>CIPHER (ENCRYPT)</td>
<td></td>
</tr>
<tr>
<td>CIPHER (ENCRYPT C-XLATE)</td>
<td>CIPHER (DECRYPT C-XLATE)</td>
<td>E X E</td>
</tr>
<tr>
<td>CIPHER (ENCRYPT C-XLATE)</td>
<td>CIPHER (ENCRYPT C-XLATE)</td>
<td></td>
</tr>
<tr>
<td>CIPHER (ENCRYPT C-XLATE)</td>
<td>CIPHER (DECRYPT C-XLATE)</td>
<td></td>
</tr>
<tr>
<td>CIPHER (ENCRYPT C-XLATE)</td>
<td>CIPHER (DECRYPT C-XLATE)</td>
<td></td>
</tr>
<tr>
<td>CIPHER (DECRYPT C-XLATE)</td>
<td>CIPHER (ENCRYPT C-XLATE)</td>
<td>X E X E</td>
</tr>
<tr>
<td>CIPHER (DECRYPT C-XLATE)</td>
<td>CIPHER (ENCRYPT)</td>
<td></td>
</tr>
<tr>
<td>CIPHER (DECRYPT C-XLATE)</td>
<td>CIPHER (ENCRYPT)</td>
<td></td>
</tr>
<tr>
<td>CIPHER (DECRYPT C-XLATE)</td>
<td>CIPHER (ENCRYPT)</td>
<td></td>
</tr>
<tr>
<td>CIPHER (DECRYPT C-XLATE)</td>
<td>CIPHER (ENCRYPT)</td>
<td></td>
</tr>
<tr>
<td>CIPHER (DECRYPT C-XLATE)</td>
<td>CIPHER (ENCRYPT)</td>
<td></td>
</tr>
<tr>
<td>CIPHER (DECRYPT C-XLATE)</td>
<td>CIPHER (ENCRYPT C-XLATE)</td>
<td>E X E</td>
</tr>
<tr>
<td>MAC' (GENERATE)</td>
<td>MAC' (GENERATE)</td>
<td>X X X X X</td>
</tr>
<tr>
<td>MAC' (GENERATE)</td>
<td>MAC' (VERIFY)</td>
<td>X X X</td>
</tr>
<tr>
<td>MAC' (VERIFY)</td>
<td>MAC' (GENERATE)</td>
<td>X X X</td>
</tr>
<tr>
<td>MAC' (GENONLY)</td>
<td>MAC' (VERIFY)</td>
<td>X X X</td>
</tr>
<tr>
<td>MAC' (VERIFY)</td>
<td>MAC' (GENONLY)</td>
<td>X X X</td>
</tr>
<tr>
<td>MAC' (GENONLY)</td>
<td>MAC' (GENERATE)</td>
<td>X X X</td>
</tr>
<tr>
<td>IMPORTER</td>
<td>IMPORTER</td>
<td>X X</td>
</tr>
<tr>
<td>EXPORTER</td>
<td>EXPORTER</td>
<td>X X X</td>
</tr>
<tr>
<td>DKYGENKY'</td>
<td>DKYGENKY'</td>
<td>X X X</td>
</tr>
</tbody>
</table>
Table 65. Key Generate2 key_type and key_form keywords for a pair of AES or HMAC keys (continued)

<table>
<thead>
<tr>
<th>key_type_1 (usage)</th>
<th>key_type_2 (usage)</th>
<th>key_form</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>OPOP, OPIM, IMIM</td>
</tr>
</tbody>
</table>

Note:
1. An X indicates a permissible key type and key usage combination for a given key form. An E indicates that a special (Extended) command is required as those keys require special handling. See the "Required commands" section for the commands required to enable these permissible combinations.
2. The key types marked with an asterisk (*) must be requested through the specification of a proper key usage in a key token and the use of the TOKEN keyword. These key types are not recognized by the verb as key type keywords.
3. A pair of DKYGENKY keys can be used to diversify a pair of keys with different key types and key usage attributes. The combination of key types and key usage attributes that can be diversified must meet the requirements of using the Key_Generate2 verb to generate those same keys. A DKYGENKY key with D-ALL usage can only be paired with a DKYGENKY key with D-ALL usage.
4. Refer to Table 67

For AES keys, the AES KEK must be at least as strong as the key being generated to be considered sufficient strength.

For HMAC keys, the AES KEK must be sufficient strength as described in the following table:

Table 66. AES KEK strength required for generating an HMAC key under an AES KEK

<table>
<thead>
<tr>
<th>Key-usage field 2 in the HMAC key contains</th>
<th>Minimum strength of AES KEK to adequately protect the HMAC key</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-256, SHA-384, or SHA-512</td>
<td>256 bits</td>
</tr>
<tr>
<td>SHA-224</td>
<td>192 bits</td>
</tr>
<tr>
<td>SHA-1</td>
<td>128 bits</td>
</tr>
</tbody>
</table>

Table 67 describes the key generation processing for DK keys (DK enabled AES key types MAC, PINCALC, PINPROT, and PINPRW). They have special rules related to which keys can exist on which system.

Table 67. CSNKGN2 access control requirements for DK enabled keys

<table>
<thead>
<tr>
<th>generated_key_identifier_1</th>
<th>generated_key_identifier_2</th>
<th>key_form</th>
</tr>
</thead>
<tbody>
<tr>
<td>key_type_1</td>
<td>KUF 1 high-order byte</td>
<td></td>
</tr>
<tr>
<td>KUF 2 high-order byte</td>
<td>KUF 3 high-order byte</td>
<td></td>
</tr>
<tr>
<td>kيك_type_2</td>
<td>KUF 1 high-order byte</td>
<td></td>
</tr>
<tr>
<td>KUF 2 high-order byte</td>
<td>KUF 3 high-order byte</td>
<td></td>
</tr>
<tr>
<td>OPOP, OPIM, IMIM</td>
<td>OPEX</td>
<td>EXEX</td>
</tr>
</tbody>
</table>

When using Key Generate2 to generate one or two DK keys that have DKPINOP, DKPINOPP, DKPINAD1, or DKPINAD2 on in key-usage field 3 of at least one skeleton key-token, the following table rows show the valid key usage for each DK key and the required access control command required for each key_form keyword.
### Table 67. CSBKGN2 access control requirements for DK enabled keys (continued)

<table>
<thead>
<tr>
<th>generated_key_identifier_1</th>
<th>generated_key_identifier_2</th>
<th>key_form</th>
</tr>
</thead>
<tbody>
<tr>
<td>PINPROT</td>
<td>ENCRYPT</td>
<td>Any usage</td>
</tr>
<tr>
<td>CIPHER</td>
<td>DECRYPT</td>
<td>Any usage</td>
</tr>
<tr>
<td>PINPROT</td>
<td>ENCRYPT</td>
<td>Any usage</td>
</tr>
<tr>
<td>PINPROT</td>
<td>DECRYPT</td>
<td>Any usage</td>
</tr>
<tr>
<td>MAC</td>
<td>GENONLY</td>
<td>Any usage</td>
</tr>
<tr>
<td>MAC</td>
<td>VERIFY</td>
<td>Any usage</td>
</tr>
<tr>
<td>MAC</td>
<td>GENONLY</td>
<td>Any usage</td>
</tr>
<tr>
<td>MAC</td>
<td>VERIFY</td>
<td>Any usage</td>
</tr>
<tr>
<td>MAC</td>
<td>VERIFY</td>
<td>Any usage</td>
</tr>
<tr>
<td>PINPRW</td>
<td>GENONLY</td>
<td>Any usage</td>
</tr>
<tr>
<td>PINPRW</td>
<td>VERIFY</td>
<td>Any usage</td>
</tr>
<tr>
<td>PINCALC</td>
<td>GENONLY</td>
<td>Any usage</td>
</tr>
</tbody>
</table>

The symbols in the key_form columns are as follows:
- `%`  Generate DK Set Locally (OPOP, OPIM, IMIM), offset X'02BB'
- `*`  Generate DK PIN Print Pair, offset X'02BC'
- `&`  Generate DK PIN Admin 1 PINPROT Pair, offset X'02BD'
- `~`  Generate DK PIN Admin 1 MAC Pair, offset X'02BE'
- `$`  Generate DK PIN Admin 2 MAC Pair, offset X'02BF'
- `#`  Generate2 Key, offset X'00EA'
- `x`  Generate2 Key Set, offset X'00EB'
- `none`  Error 8/155

The DKPINAD1 MAC keys are special. They are the only keys listed as needing no special permission to be generated as OPOP, OPIM, or IMIM pairs. Those keys are needed for generating and verifying the value EPB_M_FUS in some DK functions. Those functions run on the same system (Generating Unit) and are the only functions that use the DKPINAD1 keys. Since the entire key pair is required on the same system, special permission should not be needed in order to generate a complete key pair on the same system.

DKPINAD2 MAC key pairs can be generated with GENONLY and VERIFY key usage in key form OPOP, OPIM, and IMIM (those formats that render possible the existence of the entire key pair on one system) only with an appropriate access control point (%) set.
Key Generate2 (CSNBKGN2)

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBKGN2J.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBKGN2J(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] clear_key_bit_length,
    byte[] key_type_1,
    byte[] key_type_2,
    byte[] key_name_1_length,
    byte[] key_name_1,
    byte[] key_name_2_length,
    byte[] key_name_2,
    byte[] user_associated_data_1_length,
    byte[] user_associated_data_1,
    byte[] user_associated_data_2_length,
    byte[] user_associated_data_2,
    byte[] key_encrypting_key_identifier_1_length,
    byte[] key_encrypting_key_identifier_1,
    byte[] key_encrypting_key_identifier_2_length,
    byte[] key_encrypting_key_identifier_2,
    byte[] generated_key_identifier_1_length,
    byte[] generated_key_identifier_1,
    byte[] generated_key_identifier_2_length,
    byte[] generated_key_identifier_2);
```

Key Import (CSNBKIM)

Use the Key Import verb to re-encipher a key from encryption under an importer key-encrypting key to encryption under the master key.

The re-enciphered key is in operational form.

Choose one of the following options:

- Specify the `key_type` parameter as `TOKEN` and specify the external key token in the `source_key_identifier` parameter. The key type information is determined from the control vector in the external key token.

- Specify a key type in the `key_type` parameter and specify an external key token in the `source_key_identifier` parameter. The specified key type must be compatible with the control vector in the external key token.

- Specify a valid key type in the `key_type` parameter and a null key token in the `source_key_identifier` parameter. The default control vector for the `key_type` specified will be used to process the key.

For DATA keys, this verb generates a key of the same length as that contained in the input token.

**Note:** This verb supports PCI-HSM 2016 compliant-tagged key tokens.
Key Import (CSNBKIM)

Format
The format of CSNBKIM.

```c
CSNBKIM(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_type,
    source_key_identifier,
    importer_key_identifier,
    target_key_identifier)
```

Parameters
The parameter definitions for CSNBKIM.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

**key_type**

**Direction:** Input  
**Type:** String

The type of key you want to re-encipher under the master key. Specify an 8-byte keyword or the keyword TOKEN. The keyword must be left-aligned and padded on the right with blanks.

If the key type is TOKEN, CCA determines the key type from the control vector (CV) field in the external key token provided in the `source_key_identifier` parameter.

The key type of TOKEN is not allowed when the `importer_key_identifier` parameter is NOCV.

Key type values for the Key Import verb are:

CIPHER  DATAC  EXPORTER  MACVER  PINVER  
CIPHERXL DATAM  IMPORTER  MACD  TOKEN  
CIPHERXI DATAMV  IKEYXLAT  OKEYXLAT  
CIPHERXO DECRYPTER  IPINENC  OPINENC  
DATA  ENCIPHER  MAC  PINGEN

For information on the meaning of the key types, see Table 6 on page 44.

We recommend using key type of TOKEN when importing double-length MAC and MACVER keys.

**source_key_identifier**

**Direction:** Input  
**Type:** String

The key you want to re-encipher under the master key. The parameter is a 64-byte field for the enciphered key to be imported containing either an external key token or a null key token. If you specify a null token, the token is all binary zeros, except for a key in bytes 16-23 or 16-31, or in bytes 16-31 and 48-55 for triple-length DATA keys. Refer to Table 236 on page 852.

If key type is TOKEN, this field might not specify a null token.

This verb supports the no-export function in the CV.
Key Import (CSNBKIM)

**importer_key_identifier**

**Direction:** Input/Output  
**Type:** String

The importer key-encrypting key that the key is currently encrypted under. The parameter is a 64-byte area containing either the key label of the key in the cryptographic key data set or the internal key token for the key. If you supply a key label that is less than 64-bytes, it must be left-aligned and padded with blanks.

**Note:** If you specify a NOCV importer in the `importer_key_identifier` parameter, the key to be imported must be enciphered under the importer key itself.

**target_key_identifier**

**Direction:** Input/Output  
**Type:** String

This parameter is the generated re-enciphered key. The parameter is a 64-byte area that receives the internal key token for the imported key.

If the imported key type is IMPORTER or EXPORTER and the token key type is the same, the `target_key_identifier` parameter changes direction to both input and output. If the application passes a valid internal key token for an IMPORTER or EXPORTER key in this parameter, the NOCV bit is propagated to the imported key token.

**Restrictions**

The restrictions for CSNBKIM.

For security reasons, a request fails by default if it uses an equal-key-halves importer to import a key with unequal key halves. You must have access control point **Key Import - Unrestricted** (offset X'027B') explicitly enabled if you want to import keys in this manner.

**Required commands**

The required commands for CSNBKIM.

This verb requires the **Key Import** command (offset X'0012') to be enabled in the active role.

By also enabling the **Key Import - Unrestricted** command (offset X'027B'), you can permit a less secure mode of operation that enables an equal key-halves IMPORTER key-encrypting key to import a key having unequal key-halves (key parity bits are ignored).

To disable the wrapping of a key with a weaker master key, the **Prohibit weak wrapping - Master keys** command (offset X'0333') must be enabled in the active role.

To receive a warning when wrapping a key with a weaker master key, enable the **Warn when weak wrap - Master keys** command (offset X'0332') in the active role. The **Prohibit weak wrapping - Master keys** command overrides this command.

**Usage notes**

Usage notes for CSNBKIM.
Use of NOCV keys are controlled by an access control point in the CEX*C. Creation of NOCV key-encrypting keys is available only for standard IMPORTERs and EXPORTERs.

This verb will mark an imported KEK as a NOCV-KEK KEK:

- If a token is supplied in the target token field, it must be a valid importer or exporter token. If the token fails token validation, processing continues, but the NOCV flag will not be copied.
- The source token (key to be imported) must be a importer or exporter with the default control vector.
- If the target token is valid and the NOCV flag is on and the source token is valid and the control vector of the target token is exactly the same as the source token, the imported token will have the NOCV flag set on.
- If the target token is valid and the NOCV flag is on and the source token is valid and the control vector of the target token is NOT exactly the same as the source token, a return code will be given.
- All other scenarios will complete successfully, but the NOCV flag will not be copied.

The software bit used to mark the imported token with export prohibited is not supported on a CEX*C. The internal token for an export prohibited key will have the appropriate control vector that prohibits export.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBKIMJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBKIMJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] key_type,
    byte[] source_key_identifier,
    byte[] importer_key_identifier,
    byte[] target_key_identifier);
```

**Key Part Import (CSNBKPI)**

Use the Key Part Import verb to combine, by XORing, the clear key parts of any key type and return the combined key value either in an internal token or as an update to the key storage file.

Before you use the Key Part Import verb for the first key part, you must use the Key Token Build or Key Token Build2 verb to create the internal key token into which the key will be imported. Subsequent key parts are combined with the first part in internal token form or as a label from the key storage file.

The preferred way to specify key parts is **FIRST, ADD-PART, and COMPLETE** in the `rule_array`. Only when the combined key parts have been marked as complete can the key token be used in any cryptographic operation. The partial key can be passed to the Key Token Change or Key Token Change2 verb for re-encipherment, in case building the key was started during a master key change operation.
Key Part Import (CSNBKPI)

partial key can be passed to the Key Token Parse verb, in order to discover how the key token was originally specified, if researching an old partial key. Partial keys can also be passed to the Key Test, Key Test2, and Key Test Extended verbs.

Key parts can also be specified as FIRST, MIDDLE, or LAST in the rule_array. ADD-PART or MIDDLE can be executed multiple times for as many key parts as necessary. Only when the LAST part has been combined can the key token be used in any other service.

New applications should employ the ADD-PART and COMPLETE keywords in lieu of the MIDDLE and LAST keywords in order to ensure a separation of responsibilities between someone who can add key-part information and someone who can declare that appropriate information has been accumulated in a key.

The Key Part Import verb can also be used to import a key without using key parts. Call the Key Part Import verb FIRST with key part value X'0000...' then call the Key Part Import verb LAST with the complete value.

Keys created using this service have odd parity. The FIRST key part is adjusted to odd parity. All subsequent key parts are adjusted to even parity before being combined.

Format

The format of CSNBKPI.

```
CSNBKPI(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_part,
    key_identifier)
```

Parameters

The parameter definitions for CSNBKPI.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

rule_array_count

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 1 or 2.

rule_array

Direction: Input
Type: String array

The keyword that provides control information to the verb. The keywords must be eight bytes of contiguous storage with the keyword left-aligned in its 8-byte location and padded on the right with blanks. The rule_array keywords are described in Table 68 on page 239.
Table 68. Keywords for Key Part Import control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key part</td>
<td>(One, required)</td>
</tr>
<tr>
<td>FIRST</td>
<td>This keyword specifies that an initial key part is being entered. This verb returns this key-part encrypted by the master key in the key token that you supplied.</td>
</tr>
<tr>
<td>ADD-PART</td>
<td>This keyword specifies that additional key-part information is provided.</td>
</tr>
<tr>
<td>COMPLETE</td>
<td>This keyword specifies that the key-part bit shall be turned off in the control vector of the key rendering the key fully operational. Note that no key-part information is added to the key with this keyword.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>This keyword specifies that an intermediate key part, which is neither the first key part nor the last key part, is being entered. Note that the command control point for this keyword is the same as that for the LAST keyword and different from that for the ADD-PART keyword.</td>
</tr>
<tr>
<td>LAST</td>
<td>This keyword specifies that the last key part is being entered. The key-part bit is turned off in the control vector.</td>
</tr>
<tr>
<td>RETRKPR</td>
<td>A key label must be passed as the key_identifier. This key label corresponds to a key stored in a KPIT register inside the crypto-card (not in host key storage). The key in that register has been loaded by label and key part using the KPIT verb by the TKE. This keyword for KPI allows the user to tell the card to wrap that key (it must be in the complete state) using the master key, place it in an internal token, and return that token to the user.</td>
</tr>
<tr>
<td></td>
<td>This keyword applies only when using IBM Z.</td>
</tr>
</tbody>
</table>

**Key-wrapping method** (One, optional)

<table>
<thead>
<tr>
<th>USECONFIG</th>
<th>This is the default. Specifies to wrap the key using the configuration setting for the default wrapping method. The default wrapping method configuration setting may be changed using the TKE. This keyword is ignored for AES keys.</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRAP-ENH</td>
<td>Specifies to wrap the key using the legacy wrapping method. This keyword is ignored for AES keys. This keyword was introduced with CCA 4.1.0.</td>
</tr>
<tr>
<td>WRAP-ECB</td>
<td>Specifies to wrap the key using the enhanced wrapping method. Valid only for DES keys. This keyword was introduced with CCA 4.1.0.</td>
</tr>
</tbody>
</table>

key_part

Direction: Input
Type: String

A 16-byte field containing the clear key part to be entered. If the key is a single-length key, the key part must be left-aligned and padded on the right with zeros. This field is ignored if COMPLETE is specified.

key_identifier

Direction: Input/Output
Type: String

A 64-byte field containing an internal token or a label of an existing key in the key storage file. If rule_array is FIRST, this field is the skeleton of an internal token of a single- or double-length key with the KEY-PART marking. If rule_array is MIDDLE or LAST, this is an internal token or key label of a partially combined key. Depending on the input format, the accumulated partial or complete key is returned as an internal token or as an updated key storage file record. The returned key_identifier will be encrypted under the current master key.

Restrictions

The restrictions for CSNBKPI.
Key Part Import (CSNBKPI)

If a label is specified on key_identifier, the label must be unique. If more than one record is found, the verb fails.

You must have access control point 'Key Part Import - Unrestricted' explicitly enabled. Otherwise, current applications will fail with either of the following conditions:

- The first eight bytes of key identifier is different than the second eight bytes AND the first eight bytes of the combined key are the same as the last second eight bytes
- The first eight bytes of key identifier is the same as the second eight bytes AND the first eight bytes of the combined key are different than the second eight bytes.

Required commands
The required commands for CSNBKPI.

This verb requires the following commands to be enabled in the active role:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIRST</td>
<td>X'001B'</td>
<td>Key Part Import - first key part</td>
</tr>
<tr>
<td>ADD-PART</td>
<td>X'0278'</td>
<td>Key Part Import - ADD-PART</td>
</tr>
<tr>
<td>COMPLETE</td>
<td>X'0279'</td>
<td>Key Part Import - COMPLETE</td>
</tr>
<tr>
<td>MIDDLE or LAST</td>
<td>X'001C'</td>
<td>Key Part Import - middle and last</td>
</tr>
<tr>
<td>MIDDLE or LAST</td>
<td>X'001C'</td>
<td>Key Part Import - middle and last</td>
</tr>
<tr>
<td>WRAP-ECB or WRAP-ENH used, and default key-wrapping method setting does not match keyword</td>
<td>X'0140'</td>
<td>Key Part Import - Allow wrapping override keywords</td>
</tr>
</tbody>
</table>

The Key Part Import verb enforces the key-halves restriction when the Key Part Import - Unrestricted command (offset X'027A') is disabled in the active role. Enabling this command results in less secure operation and is not recommended.

To disable the wrapping of a key with a weaker master key, the Prohibit weak wrapping - Master keys command (offset X'0333') must be enabled in the active role.

To receive a warning when wrapping a key with a weaker master key, enable the Warn when weak wrap - Master keys command (offset X'0332') in the active role. The Prohibit weak wrapping - Master keys command overrides this command.

Usage notes
The usage notes for CSNBKPI.

None.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNBKPIJ.

See "Building Java applications using the CCA JNI" on page 28.
Key Part Import2 (CSNBKPI2)

Use the Key Part Import2 verb to combine, by XORing, the clear key parts of any key type and return the combined key value either in a variable-length internal key token or as an update to the key storage file.

Before you use the Key Part Import2 verb for the first key part, you must use the Key Token Build2 verb to create the variable-length internal key token into which the key will be imported. Subsequent key parts are combined with the first part in variable-length internal key token form, or as a label from the key storage file.

The preferred way to specify key parts is FIRST, ADD-PART, and COMPLETE in the rule_array. Only when the combined key parts have been marked as complete can the key token be used in any cryptographic operation. The partial key can be passed to the Key Token Change2 verb for re-encipherment, in case building the key was started during a master key change operation. The partial key can be passed to the Key Token Parse verb, in order to discover how the key token was originally specified, if researching an old partial key. Partial keys can also be passed to the Key Test, Key Test2, and Key Test Extended verbs.

Key parts can also be specified as FIRST, MIDDLE, or LAST in the rule_array. ADD-PART or MIDDLE can be executed multiple times for as many key parts as necessary. Only when the LAST part has been combined can the key token be used by any other verb.

New applications should employ the ADD-PART and COMPLETE keywords in lieu of the MIDDLE and LAST keywords in order to ensure a separation of responsibilities between someone who can add key-part information and someone who can declare that appropriate information has been accumulated in a key.

On each call to Key Part Import2 (except with the COMPLETE keyword), specify the number of bits to use for the clear key part. Place the clear key part in the key_part parameter, and specify the number of bits using the key_part_length variable. Any extraneous bits of key_part data will be ignored.

Consider using the Key Test2 verb to ensure a correct key value has been accumulated prior to using the COMPLETE option to mark the key as fully operational.
Key Part Import2 (CSNBKPI2)

Format

The format of CSNBKPI2.

```c
CSNBKPI2(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_part_bit_length,
    key_part,
    key_identifier_length,
    key_identifier)
```

Parameters

The parameters for CSNBKPI2.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

`rule_array_count`

Direction: Input
Type: Integer

The number of keywords you supplied in the `rule_array` parameter. This value must be 2 or 3.

`rule_array`

Direction: Input
Type: String array

The `rule_array` contains keywords that provide control information to the verb. The keywords must be in contiguous storage with each of the keywords left-aligned in its own 8-byte location and padded on the right with blanks. The `rule_array` keywords are described in Table 69.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token algorithm (Required)</td>
<td></td>
</tr>
<tr>
<td>HMAC</td>
<td>Specifies to import an HMAC key token.</td>
</tr>
<tr>
<td>AES</td>
<td>Specifies to import an AES key token.</td>
</tr>
<tr>
<td>Key part (One required)</td>
<td></td>
</tr>
<tr>
<td>FIRST</td>
<td>This keyword specifies that an initial key part is being entered. This verb returns this key-part encrypted by the master key in the key token that you supplied.</td>
</tr>
<tr>
<td>ADD-PART</td>
<td>This keyword specifies that additional key-part information is provided.</td>
</tr>
<tr>
<td>COMPLETE</td>
<td>This keyword specifies that the key-part bit shall be turned off in the control vector of the key rendering the key fully operational. Note that no key-part information is added to the key with this keyword.</td>
</tr>
</tbody>
</table>
Table 69. Keywords for Key Part Import2 control information  (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETRKPR</td>
<td>A key label must be passed as the key_identifier. This key label corresponds to a key stored in an internal register inside the cryptographic coprocessor (not in host key storage). The key in that register has been loaded by label and key part securely form the TKE. The RETRKPR keyword for CSNBKPI allows the user to tell the card to wrap that key loaded from the TKE (it must be in the complete state) using the master key, place it in an internal token, and return that token to the user. This keyword applies only when using IBM Z .</td>
</tr>
</tbody>
</table>

**Split knowledge** (Optional, required when keyword FIRST is used)

| MIN3PART | Specifies that the key must be entered in at least three parts. |
| MIN2PART | Specifies that the key must be entered in at least two parts. |
| MIN1PART | Specifies that the key must be entered in at least one part. |

**key_part_bit_length**

- Direction: Input
- Type: Integer

The length of the clear key in bits. This indicates the bit length of the key supplied in the key_part field. For FIRST and ADD-PART keywords, valid values are 80 - 2048 for HMAC keys, or 128, 192, or 256 for AES keys. The value must be 0 for the COMPLETE keyword.

**key_part**

- Direction: Input
- Type: String

This parameter is the clear key value to be applied. The key part must be left-aligned. This parameter is ignored if COMPLETE is specified.

**key_identifier_length**

- Direction: Input/Output
- Type: Integer

On input, the length of the buffer for the key_identifier parameter. For labels, the value is 64. The key_identifier must be left-aligned in the buffer. The buffer must be large enough to receive the updated token. The maximum value is 725. The output token will be longer when the first key part is imported.

On output, the actual length of the token returned to the caller. For labels, the value will be 64.

**key_identifier**

- Direction: Input/Output
- Type: String

The parameter containing an internal token or a 64-byte label of an existing key storage file record. If rule_array is FIRST, the key is a skeleton token. If rule_array is ADD-PART, this is an internal token or the label of a key storage file record of a partially combined key. Depending on the input format, the accumulated partial or complete key is returned as an internal token or as an updated record in a key storage file. The returned key_identifier will be encrypted under the current master key.
Restrictions

The restrictions for CSNBKPI2.

None.

Required commands

The required commands for CSNBKPI2.

This verb requires the following commands to be enabled in the active role:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIRST and MIN3PART</td>
<td>X'0297'</td>
<td>Key Part Import2 - Load first key part, require 3 key parts</td>
</tr>
<tr>
<td>FIRST and MIN2PART</td>
<td>X'0298'</td>
<td>Key Part Import2 - Load first key part, require 2 key parts</td>
</tr>
<tr>
<td>FIRST and MIN1PART</td>
<td>X'0299'</td>
<td>Key Part Import2 - Load first key part, require 1 key parts</td>
</tr>
<tr>
<td>ADD-PART</td>
<td>X'029A'</td>
<td>Key Part Import2 - Add second of 3 or more key parts</td>
</tr>
<tr>
<td></td>
<td>X'029B'</td>
<td>Key Part Import2 - Add last required key part</td>
</tr>
<tr>
<td></td>
<td>X'029C'</td>
<td>Key Part Import2 - Add optional key part</td>
</tr>
<tr>
<td>COMPLETE</td>
<td>X'029D'</td>
<td>Key Part Import2 - Complete key</td>
</tr>
</tbody>
</table>

To disable the wrapping of a key with a weaker master key, the Prohibit weak wrapping - Master keys command (offset X'0333') must be enabled in the active role.

To receive a warning when wrapping a key with a weaker master key, enable the Warn when weak wrap - Master keys command (offset X'0332') in the active role. The Prohibit weak wrapping - Master keys command overrides this command.

Usage notes

The usage notes for CSNBKPI2.

On each call to Key Part Import2, also specify a rule-array keyword to define the service action: FIRST, ADD-PART, or COMPLETE.

- With the FIRST keyword, the input key-token must be a skeleton token (no key material). Use of the FIRST keyword requires that the Load First Key Part2 access control point be enabled in the default role.
- With the ADD-PART keyword, the service XORs the clear key-part with the key value in the input key-token. Use of the ADD-PART keyword requires that an Add Key Part2 access control point be enabled in the default role. The key remains incomplete in the updated key token returned from the service.
- With the COMPLETE keyword, the KEY-PART bit is set off in the updated key token that is returned from the service. Use of the COMPLETE keyword requires that the Complete Key Part2 access control point be enabled in the default role. The key_part_bit_length parameter must be set to zero.
JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBKPI2J.

See “Building Java applications using the CCA JNI” on page 28.

Format

\[
\text{public native void CSNBKPI2J(}
\text{hikmNativeNumber return_code,}
\text{hikmNativeNumber reason_code,}
\text{hikmNativeNumber exit_data_length,}
\text{byte[] exit_data,}
\text{hikmNativeNumber rule_array_count,}
\text{byte[] rule_array,}
\text{hikmNativeNumber key_part_bit_length,}
\text{hikmNativeNumber key_part,}
\text{hikmNativeNumber key_identifier_length,}
\text{hikmNativeNumber key_identifier);}
\]

Key Test (CSNBKTYT)

Use the Key Test verb to generate or verify the value of either a master key, an internal AES key or key-part, or an internal DES key or key-part.

A key to test can be in the clear or encrypted under the master key. Keywords in the rule_array parameter specify whether the verb generates or verifies a verification pattern.

This algorithm is supported for clear and encrypted single and double length keys. Single, double and triple length keys are also supported with the ENC-ZERO algorithm. Clear triple length keys are not supported. See “Cryptographic key-verification techniques” on page 1021.

With the default method, the verb generates a verification pattern and it creates and cryptographically processes a random number. This verb returns the random number with the verification pattern.

For historical reasons, the verification information is passed in two 8-byte variables pointed to by the value_1 and value_2 parameters. The GENERATE option uses these variables for output, and the VERIFY option uses these variables as input. For VERIFY, the verb returns a warning of return code 4, reason code 1 if the information provided in these variables does not match the calculated values.

Table 71 on page 246 describes the use of the value_1 and value_2 variables for each of the available verification-process rule keywords.

This document uses new names for two of the parameters. The former names were misleading because they no longer reflected the use of these parameters. The header file, csulincl.h, continues to use the former names. See Table 70.

**Table 70. Key Test parameter changes**

<table>
<thead>
<tr>
<th>Current name (used in this document)</th>
<th>Former name (used in header file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>value_1</td>
<td>random_number</td>
</tr>
<tr>
<td>value_2</td>
<td>verification_pattern</td>
</tr>
</tbody>
</table>
Table 71. Key Test GENERATE outputs and VERIFY inputs

<table>
<thead>
<tr>
<th>Verification-process rule</th>
<th>GENERATE outputs and VERIFY inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>value_1 variable</td>
</tr>
<tr>
<td>ENC-ZERO</td>
<td>Unused</td>
</tr>
<tr>
<td>MDC-4</td>
<td>Contains the 8-byte KVP taken from the high-order 8 bytes of the MDC-4 hash value.</td>
</tr>
<tr>
<td>SHA-1</td>
<td>Contains the 8-byte KVP taken from the high-order 8 bytes of the SHA-1 hash value.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Unused</td>
</tr>
<tr>
<td>No keyword, and first and third parts of the master key have different values</td>
<td>Same as SHA-1</td>
</tr>
<tr>
<td>No keyword, and first and third parts of the master key have the same value</td>
<td>Contains the 8-byte KVP taken from the result of the z/OS-based master-key verification method.</td>
</tr>
</tbody>
</table>

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.

Format

The format of CSNBKYT.

```
CSNBKYT(  
  return_code,  
  reason_code,  
  exit_data_length,  
  exit_data,  
  rule_array_count,  
  rule_array,  
  key_identifier,  
  value_1,  
  value_2)  
```

This document uses new names for two of the parameters. The former names were misleading because they no longer reflected the use of these parameters. The header file, csulincl.h, continues to use the former names. See Table 70 on page 245.

Parameters

The parameters for CSNBKYT.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

rule_array_count

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 2, 3, 4, or 5.

rule_array

Direction: Input
Type: String array

Two to five keywords provide control information to the verb. The keywords must be in contiguous storage with each of the keywords left-aligned in its own 8-byte location and padded on the right with blanks. The rule_array keywords are described in Table 72.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY-CLE</td>
<td>Specifies the key supplied in key_identifier is a single-length clear key.</td>
</tr>
<tr>
<td>KEY-CLE</td>
<td>Specifies the key supplied in key_identifier is a double-length clear key.</td>
</tr>
<tr>
<td>KEY-ENC</td>
<td>Specifies the key supplied in key_identifier is a single-length encrypted key.</td>
</tr>
<tr>
<td>KEY-ENC</td>
<td>Specifies the key supplied in key_identifier is a double-length encrypted key.</td>
</tr>
<tr>
<td>KEY-KM</td>
<td>Specifies that the target is the master key register.</td>
</tr>
<tr>
<td>KEY-NKM</td>
<td>Specifies that the target is the new master-key register.</td>
</tr>
<tr>
<td>KEY-OKM</td>
<td>Specifies that the target is the old master-key register.</td>
</tr>
<tr>
<td>CLR-A128</td>
<td>Process a 128-bit AES clear-key or clear-key part.</td>
</tr>
<tr>
<td>CLR-A192</td>
<td>Process a 192-bit AES clear-key or clear-key part.</td>
</tr>
<tr>
<td>CLR-A256</td>
<td>Process a 256-bit AES clear-key or clear-key part.</td>
</tr>
<tr>
<td>TOKEN</td>
<td>Process an AES clear or encrypted key contained in an AES key-token.</td>
</tr>
<tr>
<td>AES-MK</td>
<td>Process one of the AES master-key registers.</td>
</tr>
<tr>
<td>APKA-MK</td>
<td>Process one of the APKA master-key registers. This keyword was introduced with CCA 4.1.0.</td>
</tr>
<tr>
<td>ASYM-MK</td>
<td>Specifies use of only the asymmetric master-key registers.</td>
</tr>
<tr>
<td>SYM-MK</td>
<td>Specifies use of only the symmetric master-key registers.</td>
</tr>
<tr>
<td>GENERATE</td>
<td>Generate a verification pattern for the key supplied in key_identifier.</td>
</tr>
<tr>
<td>VERIFY</td>
<td>Verify a verification pattern for the key supplied in key_identifier.</td>
</tr>
<tr>
<td>ADJUST</td>
<td>Adjust the parity of test key to odd before generating or verifying the verification pattern. The key_identifier field itself is not adjusted.</td>
</tr>
<tr>
<td>NOADJUST</td>
<td>Do not adjust the parity of test key to odd before generating or verifying the verification pattern. This is the default.</td>
</tr>
<tr>
<td>ENC-ZERO</td>
<td>Specifies use of the “encrypted zeros” method. Use only with KEY-CLE, KEY-CLE, KEY-ENC, or KEY-ENC keywords.</td>
</tr>
<tr>
<td>MDC-4</td>
<td>Specifies use of the MDC-4 master key verification method. Use only with the KEY-KM, KEY-NKM, or KEY-OKM keywords. You must specify one master-key selector keyword to use this keyword.</td>
</tr>
</tbody>
</table>

Table 72. Keywords for Key Test control information

Master-key selector (One, optional). Use only with KEY-KM, KEY-NKM, or KEY-OKM keywords.

Process rule (One, required)

Chapter 8. AES, DES, and HMAC cryptographic keys 247
Key Test (CSNBKYT)

Table 72. Keywords for Key Test control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>Specifies use of the SHA-1 master-key-verification method. Use only with KEY-KM, KEY-NKM, or KEY-OKM keywords. You must specify one master-key selector keyword to use this keyword.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Specifies use of the SHA-256 master-key-verification method. Defaults to the use of the SHA-1 master-key verification method when the ASYM-MK or SYM-MK master-key selector keyword is specified.</td>
</tr>
</tbody>
</table>

No keyword, and first and third parts of the master key have different values. Defaults to the use of the IBM z/OS-based master-key verification method when the ASYM-MK or SYM-MK master-key selector keyword is specified.

**key_identifier**

- **Direction:** Input/Output
- **Type:** String

The key for which to generate or verify the verification pattern. The parameter is a 64-byte string of an internal token, key label, or a clear key value left-aligned.

**Note:** If you supply a key label for this parameter, it must be unique in the key storage file.

**value_1**

- **Direction:** Input/Output
- **Type:** String

A pointer to a string variable. See Table 71 on page 246 for how this variable is used. For process rule GENERATE this parameter is output only, and for process rule VERIFY it is input only. This variable must be specified, even if it is not used. With the ENC-ZERO method, this parameter is not used.

**value_2**

- **Direction:** Input/Output
- **Type:** String

A pointer to a string variable. See Table 71 on page 246 for how this variable is used. For process rule GENERATE this parameter is output only, and for process rule VERIFY it is input only. This variable must be specified, even if it is not used. With the ENC-ZERO method, the high-order four bytes contain the verification data. For more detail, see “Cryptographic key-verification techniques” on page 1021.

**Restrictions**

The restrictions for CSNBKYT.

None.

**Required commands**

The required commands for CSNBKYT.
In order to access key storage, this verb requires the Compute Verification Pattern command (offset X'001D') to be enabled in the active role.

Usage notes

The usage notes for CSNBKYT.

You can generate the verification pattern for a key when you generate the key. You can distribute the pattern with the key and it can be verified at the receiving node. In this way, users can ensure using the same key at the sending and receiving locations. You can generate and verify keys of any combination of key forms, that is, clear, operational or external.

The parity of the key is not tested.

For triple-length keys, use KEY-ENC or KEY-ENCD with ENC-ZERO. Clear triple-length keys are not supported.

In the Transaction Security System, KEY-ENC or KEY-ENCD both support enciphered single-length and double-length keys. They use the key-form bits in byte 5 of CV to determine the length of the key. To be consistent, in this implementation of CCA, both KEY-ENC and KEY-ENCD handle single- and double-length keys. Both products effectively ignore the keywords, which are supplied only for compatibility reasons.

This document uses new names for two of the parameters. The former names were misleading because they no longer reflected the use of these parameters. The header file, csulincl.h, continues to use the former names. See Table 70 on page 245.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBKYTJ.

See Building Java applications using the CCA JNI on page 28.

Format

public native void CSNBKYTJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] key_identifier,
    byte[] value_1,
    byte[] value_2);

Key Test2 (CSNBKYT2)

Use the Key Test2 verb to generate or verify a secure, cryptographic verification pattern for keys contained in a variable-length symmetric key-token.

A key to test can be in the clear or encrypted under the master key. In addition, the verb permits you to test the CCA master keys. Keywords in the rule_array parameter specify whether the verb generates or verifies a verification pattern. See Cryptographic key-verification techniques on page 1021.
Key Test2 (CSNBKYT2)

When the verb tests a verification pattern against a key, you must supply the verification pattern from a previous call to Key Test2. This verb returns the verification result in the return code and reason code.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.

Format
The format of CSNBKYT2.

```
CSNBKYT2(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_length,
    key_identifier,
    key_encrypting_key_identifier_length,
    key_encrypting_key_identifier,
    reserved_length,
    reserved,
    verification_pattern_length,
    verification_pattern)
```

Parameters
The parameters for CSNBKYT2.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**
- Direction: Input
- Type: Integer

The number of keywords you supplied in the `rule_array` parameter. This value must be 2, 3, 4, or 5.

**rule_array**
- Direction: Input
- Type: String array

The `rule_array` contains keywords that provide control information to the verb. The keywords must be in contiguous storage with each of the keywords left-aligned in its own 8-byte location and padded on the right with blanks. The `rule_array` keywords are described in Table 73.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Token algorithm</strong></td>
<td>(Required)</td>
</tr>
<tr>
<td>AES</td>
<td>Specifies that the key token is an AES key token.</td>
</tr>
<tr>
<td>DES</td>
<td>Specifies that the key token is a DES token. CCA internal, CCA external, and TR-31 token types are supported. Clear keys are not supported for this rule.</td>
</tr>
<tr>
<td>HMAC</td>
<td>Specifies that the key token is an HMAC key token.</td>
</tr>
</tbody>
</table>
### Table 73. Keywords for Key Test2 control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Process rule</strong> (One required)</td>
<td></td>
</tr>
<tr>
<td>GENERATE</td>
<td>Generate a verification pattern and an associated random number for the input key or key part for the specified key.</td>
</tr>
<tr>
<td>VERIFY</td>
<td>Verify that a verification pattern matches the specified key.</td>
</tr>
<tr>
<td><strong>Key verification pattern (KVP) calculation algorithm</strong> (One optional). See “Cryptographic key-verification techniques” on page 1021.</td>
<td></td>
</tr>
<tr>
<td>CMACZERO</td>
<td>Specifies to use the CMAC-ZERO method for the key verification pattern (KVP) calculation for AES and DES keys. Calculates the KVP by performing the NIST SP 800-38B block cipher-based MAC (CMAC) algorithm on a data block filled with bytes valued to X'00'. For AES, the data block is 16 bytes, and for DES the data block is 8 bytes. Not valid with HMAC.</td>
</tr>
<tr>
<td>ENC-ZERO</td>
<td>KVP calculation using the ENC-ZERO method for AES and DES keys: the KVP is calculated by encrypting a data block filled with X'00' bytes.</td>
</tr>
<tr>
<td></td>
<td>This is the default for DES keys. Not valid with HMAC. This method is only available for AES if the Key Test2 - AES, ENC-ZERO access control point (offset X'0021) is enabled.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Verification pattern will be calculated for an AES token using the same method as the Key Test verb, with the SHA-256 rule.</td>
</tr>
<tr>
<td></td>
<td>This rule can be used to verify that the same key value is present in a version X'04' DATA token and version X'05' AES CIPHER token or to verify that the same key value is present in a version X'05' AES IMPORTER/EXPORTER pair.</td>
</tr>
<tr>
<td>SHA2VP1</td>
<td>Specifies to use the SHA-256 based verification pattern calculation algorithm. Valid only with HMAC. This is the default for HMAC. For more information, see “SHA2VP1 algorithm” on page 1024.</td>
</tr>
<tr>
<td><strong>Token type rule</strong> (Required if TR-31 token passed and token algorithm DES is specified. Not valid otherwise.)</td>
<td></td>
</tr>
<tr>
<td>TR-31</td>
<td>Specifies that key_identifier contains a TR-31 key block.</td>
</tr>
<tr>
<td>AESKWCV</td>
<td>Specifies that the key_identifier contains an external variable length symmetric key token whose type is DESUSECV. The IKEK-AES keyword must be specified for the KEK identifier rule.</td>
</tr>
<tr>
<td><strong>KEK identifier rules</strong> (Optional - see defaults. One required if the AESKWCV token type is specified.)</td>
<td></td>
</tr>
<tr>
<td>IKEK-AES</td>
<td>The wrapping KEK for the key to test is an AES KEK. This is the default for AES and HMAC token algorithms, and is not allowed with DES.</td>
</tr>
<tr>
<td>IKEK-DES</td>
<td>The wrapping KEK for the key to test is a DES KEK. This is the default for DES token algorithm, and is only allowed with DES token algorithm.</td>
</tr>
<tr>
<td>IKEK-PKA</td>
<td>The wrapping KEK for the key to test is an RSA or (other key stored in PKA key storage.) This is not the default for any token algorithm, and must be specified if an RSA KEK is used. This rule is not allowed with DES token algorithm.</td>
</tr>
<tr>
<td><strong>Verification pattern length rule</strong> (Optional - May only be used with both rules DES and CMACZERO specified, otherwise not allowed.)</td>
<td></td>
</tr>
<tr>
<td>VPLEN3</td>
<td>Specifies that a 3-byte verification pattern is calculated. This is the default.</td>
</tr>
</tbody>
</table>
### Table 73. Keywords for Key Test2 control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPLEN5</td>
<td>Specifies that a 5-byte verification pattern is calculated.</td>
</tr>
</tbody>
</table>

#### key_identifier_length
- **Direction:** Input
- **Type:** Integer

The length of the `key_identifier` in bytes. The maximum value is 9992.

#### key_identifier
- **Direction:** Input
- **Type:** String

A pointer to the key for which to generate or verify the verification pattern. The parameter is a variable length string of an internal token or the 64-byte label of a key in key storage. This token may be a DES internal or external token, AES internal version X'04' token, internal or external variable-length symmetric token, or a TR-31 key block. Clear DES tokens are not supported. If an internal token was supplied and was encrypted under the old master key, the token will be returned encrypted under the current master key.

#### key_encrypting_key_identifier_length
- **Direction:** Input
- **Type:** Integer

The byte length of the `key_encrypting_key_identifier` parameter. When `key_identifier` is an internal token, the value must be zero.

If `key_encrypting_key_identifier` is a label for a record in key storage, the value must be 64. If the `key_encrypting_key_identifier` is an AES KEK, the value must be between the actual length of the token and 725. If the `key_encrypting_key_identifier` is a DES KEK, the value must be 64. If `key_encrypting_key_identifier` is an RSA KEK, the maximum length is 3500.

#### key_encrypting_key_identifier
- **Direction:** Input/Output
- **Type:** String

When `key_encrypting_key_identifier_length` is non-zero, the `key_encrypting_key_identifier` contains an internal key token containing the key-encrypting key, or a key label. If the key identifier supplied was an AES or DES token encrypted under the old master key, the token will be returned encrypted under the current master key.

#### reserved_length
- **Direction:** Input
- **Type:** Integer

The byte length of the `reserved` parameter. This value must be 0.

#### reserved
- **Direction:** Input/Output
- **Type:** String

This parameter is ignored.
verification_pattern_length

Direction: Input/Output
Type: Integer

The length in bytes of the verification_pattern parameter.

On input: for GENERATE the length must be at least 8 bytes; for VERIFY the length must be 8 bytes.

On output for GENERATE the length of the verification pattern returned.

verification_pattern

Direction: Input/Output
Type: String

For GENERATE, the verification pattern generated for the key.

For VERIFY, the supplied verification pattern to be verified.

Restrictions

The restrictions for CSNBKYT2.

The key_identifier parameter must not identify a key label when the input key is in a TR-31 key block.

Required commands

The required commands for CSNBKYT2.

In order to access key storage, this verb also requires the Compute Verification Pattern command (offset X'001D') to be enabled in the active role.

The commands shown in [Table 74] must be enabled in the active role for each combination of KVP calculation algorithm and token algorithms:

Table 74. Required commands for CSNBKYT2

<table>
<thead>
<tr>
<th>Key verification pattern calculation</th>
<th>Token algorithm</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENC-ZERO</td>
<td>AES</td>
<td>X'0021'</td>
<td>Key Test2 - AES, ENC-ZERO (not required for algorithm keyword DES)</td>
</tr>
<tr>
<td>CMACZERO</td>
<td>AES</td>
<td>X'0022'</td>
<td>Key Test2 - AES, CMAC-ZERO</td>
</tr>
<tr>
<td>CMACZERO</td>
<td>DES</td>
<td>X'0023'</td>
<td>Key Test2 - DES, CMAC-ZERO</td>
</tr>
</tbody>
</table>

Usage notes

The usage notes for CSNBKYT2.

You can generate the verification pattern for a key when you generate the key. You can distribute the pattern with the key and it can be verified at the receiving node. In this way, users can ensure using the same key at the sending and receiving locations. You can generate and verify keys of any combination of key forms, that is, clear, operational or external.
JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBKYT2J.

See "Building Java applications using the CCA JNI" on page 28.

Format

```java
public native void CSNBKYT2J(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber key_identifier_length,
    byte[] key_identifier,
    hikmNativeNumber key_encrypting_key_identifier_length,
    byte[] key_encrypting_key_identifier,
    hikmNativeNumber reserved_length,
    byte[] reserved,
    hikmNativeNumber verification_pattern_length,
    byte[] verification_pattern);
```

Key Test Extended (CSNBKYTX)

This verb is essentially the same as Key Test (CSNBKYT).

For further information, see "Key Test (CSNBKYT)" on page 245. The differences are:

- In addition to operating on internal keys and key parts, this verb also operates on external keys and key parts.
- This verb does not operate on clear keys, and does not accept `rule_array` keywords CLR-A128, CLR-A192, CLR-A256, KEY-CLR, and KEY-CLRD.

See also "Key Test (CSNBKYT)" on page 245 for operating only on internal keys.

Use this verb to verify the value of a key or key part in an external or internal key token. This verb supports two options:

**GENERATE**

To compute and return a verification pattern for a specified key.

**VERIFY**

To verify that a passed verification pattern is correct for the specified key.

The verification pattern and the verification process do not reveal any information about the value of the tested key, other than equivalency of two key values. Several verification algorithms are supported.

This verb supports testing of AES, DES, and PKA master keys, and enciphered keys or key parts. `rule_array` keywords are used to specify information about the target key that is not implicit from other verb parameters.

When testing the master keys, there are two sets of `rule_array` keywords to indicate what key to test:

1. The SYM-MK, ASYM-MK, and AES-MK master-key selector keywords indicate whether to test the DES (symmetric) master key, the PKA (asymmetric) master key, or the AES master key.
2. The KEY-KM, KEY-NKM, and KEY-OKM key or key-part rule_array keywords choose among the current-master-key register, the new-master-key register, and the old-master-key register.

Not specifying a master-key selector keyword (SYM-MK, ASYM-MK, or AES-MK) means that the DES (symmetric) and PKA (asymmetric) master keys have the same value, and that you want to test that value.

Several key test algorithms are supported by the verb. See “Cryptographic key-verification techniques” on page 1021. Some are implicitly selected based on the type of key you are testing, while others are optional and selected by specifying a verification process rule keyword. You can specify one of the following:

1. The ENC-ZERO keyword to encrypt a block of binary zeros with the specified key. This verb returns the leftmost 32 bits of the encryption result as the verification pattern. The encrypted block consists of 16 bytes of binary zeros for AES, and eight bytes for DES and Triple-DES keys. This method is valid only with the TOKEN keyword for AES, and KEY-ENC and KEY-ENCD keywords for DES.

2. The MDC-4 keyword to compute a 16-byte verification pattern using the MDC-4 algorithm. This keyword is valid only when computing the verification pattern for a DES (symmetric) or PKA (asymmetric) master key.

3. The SHA-1 keyword to compute the verification pattern using the SHA-1 hashing method. This keyword is valid only when computing the verification pattern for the DES (symmetric) or PKA (asymmetric) master key.

4. The SHA-256 keyword to compute the verification pattern using the SHA-256 hashing method. This keyword is valid only when computing the verification pattern for an AES key.

Table 71 on page 246 describes the use of the random_number and verification_pattern fields for each of the available verification methods.

Note: For historical reasons, the verification information is passed in two 8-byte variables pointed to by the random_number and verification_pattern parameters. The GENERATE option returns information in these two variables, and the VERIFY option uses the information provided in these two variables. If the verb cannot verify the information provided, it returns a return code of 4 and a reason code of 1. For simplicity, these two variables can be two 8-byte elements of a 16-byte array, which is processed by your application program as a single quantity. Both parameters must be coded when calling the API.

DES and Triple-DES keys reserve the low-order bit of each byte for parity. If parity is used, the low-order bit is set so that the total number of B’1’ bits in the byte is odd. These parity adjustment keywords allow you to control how the Key Test Extended verb handles the parity bits:

NOADJUST
   Specifies not to alter the parity bit values in any way. This is the default.

ADJUST
   Specifies to modify the low-order bit of each byte as necessary for odd parity.
Key Test Extended (CSNBKYTX)

This is done on the cleartext value of the key before the verification pattern is computed. The parity adjustment is performed only on a temporary copy of the key within the card, and does not affect the key value in the key_identifier parameter.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.

Format

The format of CSNBKYTX.

```plaintext
CSNBKYTX(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  key_identifier,
  random_number,
  verification_pattern
  kek_key_identifier)
```

Parameters

The parameters for CSNBKYTX.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 2, 3, 4, or 5.

**rule_array**

- **Direction:** Input
- **Type:** String array

Between two and five keywords provide control information to the verb. The keywords must be in contiguous storage with each of the keywords left-aligned in its own 8-byte location and padded on the right with blanks. The rule_array keywords are described in Table 75.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process rule (One required)</td>
<td></td>
</tr>
<tr>
<td>GENERATE</td>
<td>Generate a verification pattern for the key supplied in key_identifier.</td>
</tr>
<tr>
<td>VERIFY</td>
<td>Verify a verification pattern for the key supplied in key_identifier.</td>
</tr>
<tr>
<td>Key or key-part rule (One required)</td>
<td></td>
</tr>
<tr>
<td>KEY-ENC</td>
<td>Specifies that the key supplied in key_identifier is a single-length encrypted key.</td>
</tr>
<tr>
<td>KEY-ENCD</td>
<td>Specifies that the key supplied in key_identifier is a double-length encrypted key.</td>
</tr>
<tr>
<td>KEY-KM</td>
<td>Specifies that the target is the master key register.</td>
</tr>
</tbody>
</table>
Table 75. Keywords for Key Test Extended control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY-NKM</td>
<td>Specifies that the target is the new master-key register.</td>
</tr>
<tr>
<td>KEY-OKM</td>
<td>Specifies that the target is the old master-key register.</td>
</tr>
<tr>
<td>TOKEN</td>
<td>Process an AES clear or encrypted key contained in an AES key-token.</td>
</tr>
</tbody>
</table>

*Master-key selector* (One, optional). Use only with **KEY-KM**, **KEY-NKM**, or **KEY-OKM** keywords. The default is to process the **ASYM-MK** and **SYM-MK** key registers, which must have the same key for the default to be valid.

<table>
<thead>
<tr>
<th>AES-MK</th>
<th>Process one of the AES master-key registers.</th>
</tr>
</thead>
<tbody>
<tr>
<td>APKA-MK</td>
<td>Process one of the APKA master-key registers. This keyword was introduced with CCA 4.1.0.</td>
</tr>
<tr>
<td>ASYM-MK</td>
<td>Specifies use of only the asymmetric master-key registers.</td>
</tr>
<tr>
<td>SYM-MK</td>
<td>Specifies use of only the symmetric master-key registers.</td>
</tr>
</tbody>
</table>

*Parity adjustment* (One, optional) Not valid with the AES-MK *Master-key selector* keyword.

<table>
<thead>
<tr>
<th>ADJUST</th>
<th>Adjust the parity of test key to odd before generating or verifying the verification pattern.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOADJUST</td>
<td>Do not adjust the parity of test key to odd before generating or verifying the verification pattern. This is the default.</td>
</tr>
</tbody>
</table>

*Verification process rule* (One, optional) For the AES master key, **SHA-256** is the default. For the DES or PKA master keys, the default is **SHA-1** if the first and third parts of the key are different, or the IBM z/OS method if the first and third parts of the key are the same.

<table>
<thead>
<tr>
<th>ENC-ZERO</th>
<th>Specifies use of the &quot;encrypted zeros&quot; method. Use only with the <strong>KEY-CLR</strong>, <strong>KEY-CLRD</strong>, <strong>KEY-ENC</strong>, or <strong>KEY-ENCD</strong> keywords.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDC-4</td>
<td>Specifies use of the MDC-4 master key verification method. Use only with the <strong>KEY-KM</strong>, <strong>KEY-NKM</strong>, or <strong>KEY-OKM</strong> keywords. You must specify one master-key selector keyword to use this keyword.</td>
</tr>
<tr>
<td>SHA-1</td>
<td>Specifies use of the SHA-1 master-key-verification method. Use only with the <strong>KEY-KM</strong>, <strong>KEY-NKM</strong>, or <strong>KEY-OKM</strong> keywords. You must specify one master-key selector keyword to use this keyword.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Specifies use of the SHA-256 master-key-verification method.</td>
</tr>
</tbody>
</table>

**key_identifier**

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing an internal or external key-token, a key label that identifies an internal or external key-token record, or a clear key. The key token contains the key or the key part used to generate or verify the verification pattern.

**random_number**

- **Direction:** Input/Output
- **Type:** String

A pointer to a string variable containing a number the verb might use in the verification process. When you specify the **GENERATE** keyword, the verb returns the random number. When you specify the **VERIFY** keyword, you must supply the number. With the **ENC-ZERO** method, the random_number variable is not used but must be specified.

**verification_pattern**

- **Direction:** Input/Output
- **Type:** String
Key Test Extended (CSNBKYTX)

A pointer to a string variable containing the binary verification pattern. When you specify the GENERATE keyword, the verb returns the verification pattern. When you specify the VERIFY keyword, you must supply the verification pattern. With the ENC-ZERO method, the verification data occupies the high-order four bytes, while the low-order four bytes are unspecified (the data is passed between your application and the cryptographic engine but is otherwise unused). For more detail, see “Cryptographic key-verification techniques” on page 1021.

kek_key_identifier

Direction: Input
Type: String

A pointer to a string variable containing an operational key-token or the key label of an operational key-token record containing an IMPORTER or EXPORTER key-encrypting key. If the key_identifier parameter does not identify an external key-token, the contents of the kek_key_identifier variable should contain a null DES key-token.

Restrictions

The restrictions for CSNBKYTX.

None.

Required commands

The required commands for CSNBKYTX.

In order to access key storage, this verb requires the Compute Verification Pattern command (offset X’001D’) to be enabled in the active role.

To enable warning for the case when the rule-array keyword is inconsistent with the key length, enable the command Key Test - Warn when keyword inconsistent with key length (offset X’01CB’).

Usage notes

The usage notes for CSNBKYTX.

You can generate the verification pattern for a key when you generate the key. You can distribute the pattern with the key and it can be verified at the receiving node. In this way, users can ensure using the same key at the sending and receiving locations. You can generate and verify keys of any combination of key forms: clear, operational, or external.

The parity of the key is not tested.

For triple-length keys, use KEY-ENC or KEY-ENCD with ENC-ZERO. Clear triple-length keys are not supported.

In the Transaction Security System, KEY-ENC and KEY-ENCD both support enciphered single-length and double-length keys. They use the key-form bits in byte 5 of the control vector (CV) to determine the length of the key. To be consistent, in this implementation of CCA, both KEY-ENC and KEY-ENCD handle single- and double-length keys. Both products effectively ignore the keywords, which are supplied only for compatibility reasons.
JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBKTYTXJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBKTYTXJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] key_identifier,
    byte[] random_number,
    byte[] verification_pattern,
    byte[] kek_key_identifier);
```

Key Token Build (CSNBKTB)

The Key Token Build verb assembles a fixed-length symmetric key-token in application storage from information you supply, either as an internal fixed-length AES or DES key-token, or as an external fixed-length DES token. CCA does not support fixed-length external AES key tokens.

This verb can include a control vector that you supply or can build a control vector based on the key type and the control vector related keywords in the `rule_array`. The Key Token Build verb does not perform cryptographic services on any key value. You cannot use this verb to change a key or to change the control vector related to a key.

Format

The format of CSNBKTB.

```java
CSNBKTB(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_token,
    key_type,
    rule_array_count,
    rule_array,
    key_value,
    reserved_1,
    reserved_2,
    token_data,
    control_vector,
    reserved_4,
    reserved_5,
    reserved_6,
    master_key_verification_pattern)
```

Note: Previous implementations used the `reserved_1` parameter to point to a four-byte integer or string that represented the master key verification pattern. In current versions, CCA requires this parameter to point to a four-byte value equal to binary zero.
Key Token Build (CSNBKTB)

Parameters

The parameters for CSNBKTB.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**key_token**

Direction: Input/Output
Type: String

The `key_token` parameter is a pointer to a string variable containing the assembled `key_token`.

**Note:** This variable cannot contain a key label.

**key_type**

Direction: Input
Type: String

The `key_type` parameter is a pointer to a string variable containing a keyword that defines the key type. The keyword is eight bytes in length and must be left-aligned and padded on the right with space characters.

Valid AES key type keywords are:
- CLRAES
- DATA

Valid DES key type keywords are:
- CIPHER
- CVARXCVL
- DKYGENKY
- MAC
- USE-CV
- CIPHERXI
- CVARXCVR
- ENCIPHER
- MACVER
- CIPHERXL
- DATA
- EXPORTER
- OKEYXLAT
- CIPHERXO
- DATAC
- IKEYXLAT
- OPINENC
- CVARDEC
- DATAM
- IMPORTER
- PINGEN
- CVARENC
- DATAMV
- IPINENC
- PINVER
- CVARPINE
- DECIPHER
- KEYGENKY
- SECMSG

Specify the `USE-CV` keyword to indicate that the key type should be obtained from the `control_vector` variable.

**rule_array_count**

Direction: Input/Output
Type: Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1, 2, 3, 4, 5, or 6.

**rule_array**

Direction: Output
Type: String array

One to four keywords that provide control information to the verb. The keywords must be in contiguous storage with each of the keywords left-aligned in its own 8-byte location and padded on the right with blanks. For any key type, there are no more than four valid `rule_array` values. The `rule_array` keywords are described in Table 76.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token type</td>
<td>(One required)</td>
</tr>
</tbody>
</table>
### Table 76. Keywords for Key Token Build control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTERNAL</td>
<td>An external key token. Valid only for DES keys.</td>
</tr>
<tr>
<td>INTERNAL</td>
<td>An internal key token. Valid for both AES and DES keys.</td>
</tr>
<tr>
<td><strong>Token algorithm</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>An AES key. Only valid for CLRAES or DATA. If CLRAES is specified, this is the default token algorithm.</td>
</tr>
<tr>
<td>DES</td>
<td>A DES key. Not valid for CLRAES. If CLRAES is not specified, this is the default token algorithm.</td>
</tr>
<tr>
<td><strong>Key status</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>KEY</td>
<td>The key token to build will contain an encrypted key. The key_value parameter identifies the field that contains the key.</td>
</tr>
<tr>
<td>NO-KEY</td>
<td>The key token to build will not contain a key. This is the default key status.</td>
</tr>
<tr>
<td><strong>Key length</strong> (one keyword required for AES keys, one optional for DES keys)</td>
<td></td>
</tr>
<tr>
<td>KEYLN8</td>
<td>Single-length or 8-byte key. Valid only for DES keys.</td>
</tr>
<tr>
<td>KEYLN16</td>
<td>Specifies that the key is 16 bytes long.</td>
</tr>
<tr>
<td>KEYLN24</td>
<td>Specifies that the key is 24 bytes long.</td>
</tr>
<tr>
<td>KEYLN32</td>
<td>Specifies that the key is 32 bytes long. Valid only for AES keys.</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>Double-length or 16-byte key. Synonymous with KEYLN16. Valid only for DES keys.</td>
</tr>
<tr>
<td>DOUBLE-O</td>
<td>Double-length key with guaranteed unique 8-byte key halves. The key is 16 bytes long. Valid only for DES keys.</td>
</tr>
<tr>
<td>MIXED</td>
<td>Double-length key. Indicates that the key can either be a replicated single-length key (both key halves equal), or a double-length key with two different 8-byte values. Valid only for DES keys.</td>
</tr>
<tr>
<td>SINGLE</td>
<td>Single-length or 8-byte key. Synonymous with KEYLN8. Valid only for DES keys.</td>
</tr>
<tr>
<td><strong>Key Part Indicator</strong> (optional). Valid only for DES keys.</td>
<td></td>
</tr>
<tr>
<td>KEY-PART</td>
<td>This token is to be used as input to the Key Part Import service.</td>
</tr>
<tr>
<td><strong>CV source</strong> (One, optional). Valid only for DES keys.</td>
<td></td>
</tr>
<tr>
<td>CV</td>
<td>The verb is to obtain the control vector from the variable identified by the control_vector parameter.</td>
</tr>
<tr>
<td>NO-CV</td>
<td>The control vector is to be supplied based on the key type and the control vector related keywords. This is the default.</td>
</tr>
<tr>
<td><strong>Control vector on the link specification</strong> (optional). Valid only for IMPORTER and EXPORTER.</td>
<td></td>
</tr>
<tr>
<td>CV-KEK</td>
<td>This keyword indicates marking the KEK as a CV KEK. The control vector is applied to the KEK prior to its use in encrypting other keys. This is the default.</td>
</tr>
<tr>
<td>NOCV-KEK</td>
<td>This keyword indicates marking the KEK as a NOCV KEK. The control vector is not applied to the KEK prior to its use in encrypting other keys.</td>
</tr>
<tr>
<td><strong>Key-wrapping method</strong> (One, optional). Valid only for DES keys.</td>
<td></td>
</tr>
<tr>
<td>WRAP-ENH</td>
<td>Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.</td>
</tr>
</tbody>
</table>
### Table 76. Keywords for Key Token Build control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRAP-ECB</td>
<td>Use original key wrapping method, which uses ECB wrapping for DES key tokens and CBC wrapping for AES key tokens.</td>
</tr>
<tr>
<td><strong>Translation control</strong> (Optional). Valid only for DES keys.</td>
<td></td>
</tr>
<tr>
<td>ENH-ONLY</td>
<td>Restrict re-wrapping of the output_key_token. After the token has been wrapped with the enhanced method, it cannot be re-wrapped using the original method.</td>
</tr>
<tr>
<td><strong>Compliance</strong> (Optional)</td>
<td></td>
</tr>
<tr>
<td>COMP-TAG</td>
<td>Generate a compliant-tagged key. While a skeleton key token with the compliance-tag can be created at any time, the skeleton must be passed to an adapter domain that is in PCI-HSM 2016 compliance mode to be provisioned with key material (either generated or imported).</td>
</tr>
<tr>
<td>NCOMPTAG</td>
<td>Do not generate a compliant-tagged key. This is the default.</td>
</tr>
</tbody>
</table>

See Figure 3 on page 48 for the key usage keywords that can be specified for a given key type.

The difference between Key Token Parse (CSNBKTP) and Control Vector Generate (CSNBCVG) is that Key Token Parse returns the rule_array keywords that apply to a parsed token, such as **EXTERNAL, INTERNAL**, and so forth. These rule_array parameters are returned in addition to the key_type parameter.

```
| AMEX-CSC  | DKL0  | EPINGEN | KEYN16 | UKPT   |
| ANSIX9.9  | DKL1  | EPINGEN  | LMTD-KEK | VISA-PVV |
| ANY       | DKL2  | EPINVER  | MIXED  | WRAP-ECB |
| ANY-MAC   | DKL3  | EXEX     | NO-SPEC | WRAP-ENH |
| CLR8-ENC  | DKL4  | EXPORT   | NO-XPORT | XLATE |
| CPINENC   | DKL5  | GBP-PIN  | NOOFFSET | XPORT-OK |
| CPINGEN   | DKL6  | GBP-PINO  | NOT-KEK |        |
| CPINGENA  | DKL7  | IBM-PIN  | OPEX   |        |
| CVKEY-A   | DMC   | IBM-PINO  | OPIM   |        |
| CVKEY-B   | DMAC  | IMEX     | PIN     |        |
| DALL      | DMPIN | IMIM     | REFORMAT |       |
| DATA      | DMV   | IMPORT   | SINGLE  |       |
| DDATA     | DOUBLE | INBK-PIN | SMKEY   |       |
| DEXP      | DPVR  | KEY-PART | SMPIN   |       |
| DIMP      | ENH-ONLY | KEYLN8  | TRANSLAT |       |
```

**key_value**

- **Direction:** Output
- **Type:** String

This parameter is a pointer to a string variable containing the enciphered key or AES clear-key value which is placed into the key field of the key token when you use the **KEY** rule_array keyword. If the **KEY** keyword is not specified, this parameter is ignored.

The length of this variable depends on the type of key that is provided. The length is 16 bytes for DES keys. A single-length DES key must be left-aligned and padded on the right with eight bytes of X'00'. For a clear AES key, the length is 16 bytes for **KEYLN16**, 24 bytes for **KEYLN24**, and 32 bytes for **KEYLN32**. An enciphered AES key is 32 bytes.

**reserved_1**

- **Direction:** Output
- **Type:** Integer
This parameter is a pointer to an integer variable or a 4-byte string variable. The value must be equal to an integer valued 0.

reserved_2
Direction: Output
Type: Integer

This parameter is a pointer to an integer variable. The value must be 0 or a null pointer.

token_data
Direction: Input
Type: String

This parameter is unused for DES keys and cleartext AES keys. In either of those cases it must be a null pointer or point to a string variable containing eight bytes of binary zeros. For encrypted AES keys, this parameter is a pointer to a one-byte string variable containing the LRC value for the key passed in the key_value parameter. For more information on LRC values, see IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors.

control_vector
Direction: Output
Type: String

A parameter is a pointer to a string variable. If you specify the CV keyword in the rule_array, the contents of this variable are copied to the control vector field of the fixed-length DES key token. If the CV keyword is not specified, this keyword is ignored.

reserved_4
Direction: Output
Type: String

This parameter is a pointer to a string variable. The value must be binary zeros or a null pointer.

reserved_5
Direction: Output
Type: Integer

This parameter is a pointer to an integer variable. The value must be 0 or a null pointer.

reserved_6
Direction: Output
Type: String

This parameter is a pointer to an 8-byte string variable. The value must eight space characters or a null pointer.

master_key_verification_pattern
Direction: Output
Type: String
Key Token Build (CSNBKTB)

This parameter is a pointer to a string variable containing the master-key verification pattern of the master key used to encipher the key in the internal key-token. The contents of the variable are copied into the MKVP field of the key token when keywords INTERNAL and KEY are specified, and key_type keyword CLRAES is not specified.

Restrictions
The restrictions for CSNBKTB.
None.

Required commands
The required commands for CSNBKTB.
None.

Usage notes
The usage notes for CSNBKTB.

Because 24-byte (TRIPLE) DES keys can only be generated as DATA keys, capability to create 24-byte DES tokens (with keywords TRIPLE or KEYLN24 has not been added to Key Token Build (CSNBKTB). Instead, call Key Generate (CSNBKGN) directly.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNBKTBJ.

See “Building Java applications using the CCA JNI” on page 28.

Format
public native void CSNBKTBJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] key_token,
    byte[] key_type,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] key_value,
    byte[] reserved_1,
    hikmNativeNumber reserved_2,
    byte[] token_data,
    byte[] control_vector,
    hikmNativeNumber reserved_4,
    byte[] reserved_5,
    byte[] reserved_6,
    byte[] master_key_verification_pattern);

Key Token Build2 (CSNBKTB2)

Use the Key Token Build2 verb to assemble an external or internal AES or HMAC variable-length symmetric key-token in application storage from information that you supply.
This verb assembles the information as a skeleton HMAC or AES key token. This skeleton token can be supplied to the Key Generate2 verb, which then provides a completed key token with the attributes of the skeleton along with a randomly generated key. These attributes become cryptographically bound to the key when it is enciphered.

The Key Token Build2 verb cannot assemble a usable key-token that contains an enciphered key. It can assemble an internal HMAC or AES key token that has either a clear key, usable for a limited number of services, or no key, which is only usable for passing to the Key Generate2 verb in order to receive an enciphered key.

The Key Token Build2 verb is a host-only verb and it does not use the cryptographic coprocessor. This verb does not perform cryptographic services on any key value. You cannot use this verb to change a key or to change the control vector related to a key.

Format

The format of CSNBKTB2.

```
CSNBKTB2(  return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    clear_key_bit_length,  
    clear_key_value,  
    key_name_length,  
    key_name,  
    user_associated_data_length,  
    user_associated_data,  
    token_data_length,  
    token_data,  
    verb_data_length,  
    verb_data,  
    target_key_token_length,  
    target_key_token)
```

Parameters

The parameters for CSNBKTB2.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

`rule_array_count`

Direction: Input  
Type: Integer  

The number of keywords you supplied in the `rule_array` parameter. The minimum value is 4.

`rule_array`

Direction: Input  
Type: String array  

The `rule_array` contains keywords that provide control information to the verb.
Key Token Build2 (CSNBKTB2)

The keywords must be in contiguous storage with each of the keywords left-aligned in its own 8-byte location and padded on the right with blanks. The rule_array keywords are described in Table 77.

Table 77. Keywords for Key Token Build2

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header section</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Token identifier</strong> (one required)</td>
<td></td>
</tr>
<tr>
<td>EXTERNAL</td>
<td>Specifies to build an external variable-length symmetric key-token.</td>
</tr>
<tr>
<td>INTERNAL</td>
<td>Specifies to build an internal variable-length symmetric key-token.</td>
</tr>
<tr>
<td><strong>Wrapping information section</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Key status</strong> (one, optional)</td>
<td></td>
</tr>
<tr>
<td>NO-KEY</td>
<td>Build the key token without a key value. This creates a skeleton key token that can later be supplied to the Key Generate2 (CSNBKGN2) verb. This is the default.</td>
</tr>
<tr>
<td>KEY-CLR</td>
<td>Build the key token with a clear key, AES CIPHER and HMAC MAC keys only.</td>
</tr>
<tr>
<td><strong>Associated data section</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Type of algorithm for which the key can be used</strong> (one required)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>Specifies to build an AES key token.</td>
</tr>
<tr>
<td>HMAC</td>
<td>Specifies to build an HMAC key token.</td>
</tr>
<tr>
<td><strong>CPACF export control</strong></td>
<td></td>
</tr>
<tr>
<td>XPRTCPAC</td>
<td>Allow this key token to be exported as a CPACF protected key. When CSU_HCPUAPRT is enabled, and this key token (or a label referring to this key token) is used, the key token is translated to CPACF by the CCA library and the handle is used with the CPACF.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Only usable with AES keys of key type CIPHER.</td>
</tr>
<tr>
<td>NOEXCPAC</td>
<td>Do not allow this key token to be exported as a CPACF protected key. This is the default.</td>
</tr>
<tr>
<td><strong>Key type</strong> (one required)</td>
<td></td>
</tr>
<tr>
<td>CIPHER</td>
<td>Build a CIPHER key for encryption, decryption, and translation operations. AES algorithm only.</td>
</tr>
<tr>
<td></td>
<td>• Refer to Figure 5 on page 277 for valid rule array keyword combinations.</td>
</tr>
<tr>
<td></td>
<td>• Refer to Table 79 on page 270 for token offsets, offset values, and meanings of these keywords.</td>
</tr>
<tr>
<td></td>
<td>• Refer to Table 271 on page 904 for the format of the key token.</td>
</tr>
<tr>
<td>DKYGENKY</td>
<td>Build a diversifying key generating key. AES algorithm only.</td>
</tr>
<tr>
<td></td>
<td>• Refer to Figure 6 on page 273 for valid rule array keyword combinations.</td>
</tr>
<tr>
<td></td>
<td>• Refer to Table 80 on page 274 for token offsets, offset values, and meanings of these keywords.</td>
</tr>
<tr>
<td></td>
<td>• Refer to &quot;AES DKYGENKY variable-length symmetric key token&quot; on page 951 for the format of the key token.</td>
</tr>
<tr>
<td>EXPORTER</td>
<td>Build an EXPORTER key-encrypting key. AES algorithm only.</td>
</tr>
<tr>
<td></td>
<td>• Refer to Figure 7 on page 277 for valid rule array keyword combinations.</td>
</tr>
<tr>
<td></td>
<td>• Refer to Table 81 on page 278 for token offsets, offset values, and meanings of these keywords.</td>
</tr>
<tr>
<td></td>
<td>• Refer to &quot;AES EXPORTER and IMPORTER variable-length symmetric key token” on page 927 for the format of the key token.</td>
</tr>
<tr>
<td>IMPORTER</td>
<td>Build an IMPORTER key-encrypting key. AES algorithm only.</td>
</tr>
<tr>
<td></td>
<td>• Refer to Figure 7 on page 277 for valid rule array keyword combinations.</td>
</tr>
<tr>
<td></td>
<td>• Refer to Table 81 on page 278 for token offsets, offset values, and meanings of these keywords.</td>
</tr>
<tr>
<td></td>
<td>• Refer to &quot;AES EXPORTER and IMPORTER variable-length symmetric key token” on page 927 for the format of the key token.</td>
</tr>
</tbody>
</table>
Table 77. Keywords for Key Token Build2 (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
</table>
| MAC     | Build a MAC key for message authentication code operations. AES and HMAC algorithms only. For AES:  
  - Refer to Figure 9 on page 285 for valid rule array keyword combinations.  
  - Refer to Table 83 on page 285 for token offsets, offset values, and meanings of these keywords.  
  - Refer to “AES MAC variable-length symmetric key token” on page 911 for the format of the key token.  
  For HMAC:  
  - Refer to Figure 10 on page 288 for valid rule array keyword combinations.  
  - Refer to Table 84 on page 288 for token offsets, offset values, and meanings of these keywords.  
  - Refer to “HMAC MAC variable-length symmetric key token” on page 920 for the format of the key token. |
| PINCALC | Build a DK PIN calculating key. AES algorithm only.  
  - Refer to Figure 11 on page 291 for valid rule array keyword combinations.  
  - Refer to Table 85 on page 291 for token offsets, offset values, and meanings of these keywords.  
  - Refer to “AES PINPROT, PINCALC, and PINPRW variable-length symmetric key token” on page 938 for the format of the key token. |
| PINPROT | Build a DK PIN protection key. AES algorithm only.  
  - Refer to Figure 12 on page 293 for valid rule array keyword combinations.  
  - Refer to Table 86 on page 294 for token offsets, offset values, and meanings of these keywords.  
  - Refer to “AES PINPROT, PINCALC, and PINPRW variable-length symmetric key token” on page 938 for the format of the key token. |
| PINPRW  | Build a DK PIN PRW key. AES algorithm only.  
  - Refer to Figure 13 on page 296 for valid rule array keyword combinations.  
  - Refer to Table 87 on page 296 for token offsets, offset values, and meanings of these keywords.  
  - Refer to “AES PINPROT, PINCALC, and PINPRW variable-length symmetric key token” on page 938 for the format of the key token. |
| SECMG  | Build a secure messaging key. AES algorithm only.  
  - Refer to Figure 14 on page 298 for valid rule array keyword combinations.  
  - Refer to Table 88 on page 299 for token offsets, offset values, and meanings of these keywords.  
  - Refer to “AES SECMG variable-length symmetric key token” on page 960 for the format of the key token. |

**clear_key_bit_length**

  Direction: Input  
  Type: Integer  

  The length of the clear key in bits. Specify 0 when no key value is supplied or a valid HMAC key bit length, between 80 and 2048.

**clear_key_value**

  Direction: Input  
  Type: String  

  This parameter is used when the KEY-CLR keyword is specified. This parameter is the clear key value to be put into the token being built.

**key_name_length**

  Direction: Input  
  Type: Integer  

  The length of the key name parameter. Valid values are 0 and 64.
Key Token Build2 (CSNBKTB2)

key_name
Direction: Input
Type: String

A 64-byte key store label to be stored in the associated data structure of the token.

user_associated_data_length
Direction: Input
Type: Integer

The length of the user-associated data. The valid values are 0 - 255 bytes.

user_associated_data
Direction: Input
Type: String

User-associated data to be stored in the associated data structure.

token_data_length
Direction: Input
Type: Integer

This parameter is reserved. This value must be 0.

token_data
Direction: n/a
Type: String

This parameter is ignored.

verb_data_length
Direction: Input
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the verb_data variable. The value must be 0.

verb_data
Direction: Input
Type: String

A pointer to a string variable containing key-usage field keywords that are related to the type of key to diversify.

DKYUSAGE specifies that the verb_data variable contains all of the keywords necessary to define the key usage attributes related to the type of key to diversify. Based on the verb_data keywords, CSNBKTB2 appends the key usage attributes of the type of key to diversify to the key usage fields of the DKYGENKY key. The related key usage fields control which key usage attributes are permissible for the finally generated diversified key.

DKYUSAGE is not valid with D-ALL, because the type of key to diversify is unspecified. DKYUSAGE is optional with D-CIPHER, D-EXP, and D-IMP. For these key types, if DKYUSAGE is not specified, CSNBKTB2 assigns default key
usage attributes to the related KUF fields. DKYUSAGE is required for the remaining values of type of key to diversify, because those key types do not have default key usage attributes.

Table 78. Related key usage fields when Key Token Build2 builds a DKGENCEY key-token

<table>
<thead>
<tr>
<th>Type of key to diversify</th>
<th>DKYUSAGE usage</th>
<th>Related key usage fields for key type DKGENCEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-ALL</td>
<td>Invalid</td>
<td>None.</td>
</tr>
<tr>
<td>D-CIPHER</td>
<td>Optional</td>
<td>If keyword DKYUSAGE is specified, the verb_data variable must contain key usage fields keywords related to an AES CIPHER key. If not specified, the related key usage fields are those of a default AES CIPHER key.</td>
</tr>
<tr>
<td>D-EXP</td>
<td>Optional</td>
<td>If keyword DKYUSAGE is specified, the verb_data variable must contain key usage fields keywords related to an AES EXPORTER key. If not specified, the related key usage fields will be that of a default AES EXPORTER key.</td>
</tr>
<tr>
<td>D-IMP</td>
<td>Optional</td>
<td>If keyword DKYUSAGE is specified, the verb_data variable must contain key usage fields keywords related to an AES IMPORTER key. If not specified, the related key usage fields will be that of a default AES IMPORTER key.</td>
</tr>
<tr>
<td>D-MAC</td>
<td>Required</td>
<td>The verb_data variable must contain key usage fields keywords related to an AES MAC key.</td>
</tr>
<tr>
<td>D-PCALC</td>
<td>Required</td>
<td>The verb_data variable must contain key usage fields keywords related to an AES PINCALC key.</td>
</tr>
<tr>
<td>D-PPROT</td>
<td>Required</td>
<td>The verb_data variable must contain key usage fields keywords related to an AES PINPROT key.</td>
</tr>
<tr>
<td>D-PPRW</td>
<td>Required</td>
<td>The verb_data variable must contain key usage fields keywords related to an AES PINPRW key.</td>
</tr>
<tr>
<td>D-SECMSG</td>
<td>Required</td>
<td>The verb_data variable must contain key usage fields keywords related to an AES SECMSG key.</td>
</tr>
</tbody>
</table>

**target_key_token_length**

Direction: Input/Output  
Type: Integer

On input, the length of the target_key_token parameter supplied to receive the token. On output, the actual length of the token returned to the caller. Maximum length is 725 bytes.

**target_key_token**

Direction: Output  
Type: String

The key token built by this verb.

**Keywords reference**

Here you find a keywords reference for the CSNBKTB2 rule array parameter.

Figure 5 on page 270 shows all the valid keyword combinations and their defaults for AES key type CIPHER. For a description of these keywords, refer to Table 79 on page 270.
Key Token Build2 (CSNBKTB2)

Figure 5. Key Token Build2 keyword combinations for AES CIPHER keys

Note:
1. V0PYLD is the default. V1PYLD is recommended because it provides improved security.
2. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of a high-order byte (HOB) and a low-order byte (LOB).
3. DECRIPT and ENCRYPT are defaults if neither of these keywords is specified, regardless of whether C-XLATE is specified or not. C-XLATE is for Release 4.3 or later.
4. ANY-MODE is Release 4.4 or later.
5. Choose any number of keywords in this group. No keywords in this group are defaults.
6. NOEX-RAW and XPRT-RAW are defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRT-RAW.

Table 79. Key Token Build2 rule array keywords for AES CIPHER keys

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Key-token header section</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 79. Key Token Build2 rule array keywords for AES CIPHER keys (continued)

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token identifier (one required)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>EXTERNAL</td>
<td>X'02'</td>
<td>Build a key token that is not to be used locally.</td>
</tr>
<tr>
<td></td>
<td>INTERNAL</td>
<td>X'01'</td>
<td>Build a key token that is to be used locally.</td>
</tr>
</tbody>
</table>

Wrapping-information section

| Key status (one, optional) | | | |
|---------------------------|-------------|---------|
| 008 | KEY-CLR | X'01' | Build a key token that contains a clear key. |
| | NO-KEY | X'00' | Build a key token that does not contain a key value. This is the default. |

Payload format version (one, optional). Identifies format of the payload.

| Payload format version (one, optional) | | | |
|---------------------------------------|-------------|---------|
| 000 | V0PYLD | X'00' | Build a key token with a version 0 payload format. This format has a variable length and the key length can be inferred from the size of the payload. This format is compatible with all releases. This is the default. |
| | V1PYLD | X'01' | Build the key token with a version 1 payload format. This format has a fixed length and the key length cannot be inferred by the size of the payload. An obscured key length is considered more secure. This format is not compatible in releases before Release 4.4. |

Associated data section

| Algorithm type (one required) | | | |
|------------------------------|-------------|---------|
| 041 | AES | X'02' | Key can be used for AES algorithm. |

Key type (one required)

| Key type (one required) | | | |
|-------------------------|-------------|---------|
| 042 | CIPHER | X'0001' | Key can be used for encryption, decryption, and translation of data. |

Encryption and translation control (one or more, optional). Key-usage field 1, high-order byte. Keywords DECRIPT and ENCRYPT are defaults unless one or more keywords in the group are specified.

| Encryption and translation control (one or more, optional) | | | |
|-------------------------------------------------|-------------|---------|
| 045 | C-XLATE (Release 4.3 or later) | B'xx1x xxxx' | Key can only be used for Cipher Text Translate2 (CSNBCTT2) operations. This is only valid with AES CIPHER keys. |
| | DECRIPT | B'x1xx xxxx' | Key can be used for decryption. Symmetric_Algorithm_Decipher. |
| | ENCRYPT | B'1xxx xxxx' | Key can be used for encryption. Symmetric_Algorithm_Encipher. |

User-defined extension (UDX) control (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.

| User-defined extension (UDX) control (one or more, optional) | | | |
|---------------------------------------------------|-------------|---------|
| 046 | UDX-ONLY | B'xxxx 1xxx' | Key can only be used in UDXs. |
| | UDX-100 | B'xxxx x1uu' | Leftmost user-defined UDX bit is set on. |
| | UDX-010 | B'xxxx xu1u' | Middle user-defined UDX bit is set on. |
| | UDX-001 | B'xxxx xuu1' | Rightmost user-defined UDX bit is set on. |

Encryption mode (one, optional). Key-usage field 2, high-order byte.
### Table 79. Key Token Build2 rule array keywords for AES CIPHER keys (continued)

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>047</td>
<td>ANY-MODE (Release 4.4 or later)</td>
<td>X'FF'</td>
<td>Key can be used for any encryption mode.</td>
</tr>
<tr>
<td></td>
<td>CBC</td>
<td>X'00'</td>
<td>Key can be used for Cipher Block Chaining. This is the default.</td>
</tr>
<tr>
<td></td>
<td>CFB</td>
<td>X'02'</td>
<td>Key can be used for Cipher Feedback.</td>
</tr>
<tr>
<td></td>
<td>ECB</td>
<td>X'01'</td>
<td>Key can be used for Electronic Code Book.</td>
</tr>
<tr>
<td></td>
<td>GCM</td>
<td>X'04'</td>
<td>Key can be used for Galois/Counter Mode.</td>
</tr>
<tr>
<td></td>
<td>OFB</td>
<td>X'03'</td>
<td>Key can be used for Output Feedback.</td>
</tr>
<tr>
<td></td>
<td>XTS</td>
<td>X'05'</td>
<td>Key can be used for Xor-Encrypt-Xor-based Tweaked Stealing.</td>
</tr>
</tbody>
</table>

**Symmetric-key export control (one, optional).** Key-management field 1, high-order byte.

<table>
<thead>
<tr>
<th>050</th>
<th>NOEX-SYM</th>
<th>B'0xxx xxxx'</th>
<th>Prohibit export using symmetric key.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>XPRT-SYM</td>
<td>B'1xxx xxxx'</td>
<td>Allow export using symmetric key. This is the default.</td>
</tr>
</tbody>
</table>

**Unauthenticated asymmetric-key export control (one, optional).** Key-management field 1, high-order byte.

<table>
<thead>
<tr>
<th>050</th>
<th>NOEXUASY</th>
<th>B'x0xx xxxx'</th>
<th>Prohibit export using an unauthenticated asymmetric key (not a trusted block).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>XPRTUASY</td>
<td>B'x1xx xxxx'</td>
<td>Allow export using an unauthenticated asymmetric key (not a trusted block). This is the default.</td>
</tr>
</tbody>
</table>

**Authenticated asymmetric-key export control (one, optional).** Key-management field 1, high-order byte.

<table>
<thead>
<tr>
<th>050</th>
<th>NOEXAAASY</th>
<th>B'xx0x xxxx'</th>
<th>Prohibit export using an authenticated asymmetric key (trusted block).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>XPRTAAASY</td>
<td>B'xx1x xxxx'</td>
<td>Allow export using an authenticated asymmetric key (trusted block). This is the default.</td>
</tr>
</tbody>
</table>

**Raw-key export control (one, optional).** Key-management field 1, high-order byte.

<table>
<thead>
<tr>
<th>050</th>
<th>NOEX-RAW</th>
<th>B'xxx0 xxxx'</th>
<th>Prohibit export using raw key. Defined for future use. Currently ignored. This is the default.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>XPRT-RAW</td>
<td>B'xxx1 xxxx'</td>
<td>Allow export using raw key. Defined for future use. Currently ignored.</td>
</tr>
</tbody>
</table>

**CPACF export control (one, optional).** Key-management field 1, high-order byte.

<table>
<thead>
<tr>
<th>050</th>
<th>XPRTCPAC</th>
<th>B'xxxx 1xxx'</th>
<th>Allow export of key to CPACF.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NOEXCPAC</td>
<td>B'xxxx 0xxx'</td>
<td>Prohibit export of key to CPACF. This is the default.</td>
</tr>
</tbody>
</table>

**DES-key export control (one, optional).** Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>051</th>
<th>NOEX-DES</th>
<th>B'1xxx xxxx'</th>
<th>Prohibit export using a DES key.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>XPRT-DES</td>
<td>B'0xxx xxxx'</td>
<td>Allow export using a DES key. This is the default.</td>
</tr>
</tbody>
</table>

**AES-key export control (one, optional).** Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>051</th>
<th>NOEX-AES</th>
<th>B'x1xx xxxx'</th>
<th>Prohibit export using an AES key.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>XPRT-AES</td>
<td>B'x0xx xxxx'</td>
<td>Allow export using an AES key.</td>
</tr>
</tbody>
</table>

**RSA-key export control (one, optional).** Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>051</th>
<th>NOEX-RSA</th>
<th>B'xxxx 1xxx'</th>
<th>Prohibit export using an RSA key.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>XPRT-RSA</td>
<td>B'xxxx 0xxx'</td>
<td>Allow export using an RSA key. This is the default.</td>
</tr>
</tbody>
</table>

[Figure 6 on page 273](#) shows all the valid keyword combinations and their defaults for AES key type DKYGENKY. For a description of these keywords, refer to
Table 80 on page 274

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Key type</th>
<th>Token identifier</th>
<th>Key material state</th>
<th>Payload format ver.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>DKYGENKY</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **KUF1 HOB**
  - Note: One required.
  - >D-ALL
  - >D-CIPHER
  - >D-EXP
  - >D-IMP
  - >D-MAC
  - >D-PPROT
  - >D-PPRW
  - >D-SECMSG

- **KUF1 LOB**
  - See Note 6.
  - >UDX-ONLY
  - >UDX-001
  - >UDX-010
  - >UDX-100

- **KMF1 HOB**
  - Note: XPRT-SYM is default.
  - >NOEX-SYM
  - >XPRT-SYM

- **KMF1 LOB**
  - Note: XPRT-DES is default.
  - >NOEX-DES
  - >XPRT-DES

- **KUF2 LOB**
  - Note: One required.
  - >DKYLO
  - >DKYL1
  - >DKYL2

- **KUF2 HOB**
  - Note: KUF-MBE is default.
  - See Note 5.
  - >KUF-MBE
  - >KUF-MBP

- **KUF3 HOB**
  - Note: No default.
  - >DKYUSAGE
  - >DKYUSAGE

- **KUF3 LOB**
  - >V
  - >UDX-ONLY
  - >UDX-001
  - >UDX-010
  - >UDX-100

**Note:**

1. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of two bytes: a high-order byte (HOB) and a low-order byte (LOB).

2. D-PCALC, D-PPROT, and D-PPRW are Release 4.4 or later. D-SECMSG, DKYL1, and DKYL2 are Release 4.4 or later.

3. NOEX-RAW and XPRT-RAW are defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRT-RAW.

4. DKYUSAGE specifies that the verb_data variable contains all of the keywords necessary to define the key usage attributes related to the type of key to diversify. Based on the verb_data keywords, CSNBKTBE2 appends the key usage attributes of the type of key to diversify to the key usage fields of the DKYGENKY key. The related key usage fields control which key usage attributes are permissible for the final generated diversified key. DKYUSAGE is
not valid with D-ALL because the type of key to diversify is unspecified.

DKYUSAGE is optional with D-CIPHER, D-EXP, and D-IMP because key types
CIPHER, EXPORTER and IMPORTER have default key usage attributes. For
these key types, if DKYUSAGE is not specified, CSNBKTB2 assigns default key
usage attributes to the related KUF fields. DKYUSAGE is required for the
remaining values of type of key to diversify because those key types do not
have default key usage attributes.

5. KUF-MBP is not valid if DKADMIN1, DKADMIN2, DPINOP, or DPINOPP
is specified in the verb_data variable (that is, the type of key to diversify is DK
enabled).

6. Choose any number of keywords in this group. No keywords in this group are
defaults.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Key-token header section</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Token identifier (one required).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>EXTERNAL</td>
<td>X'02'</td>
<td>Build a key token that is not to be used locally.</td>
</tr>
<tr>
<td></td>
<td>INTERNAL</td>
<td>X'01'</td>
<td>Build a key token that is to be used locally.</td>
</tr>
<tr>
<td></td>
<td>Wrapping-information section</td>
<td></td>
<td></td>
</tr>
<tr>
<td>008</td>
<td>NO-KEY</td>
<td>X'00'</td>
<td>Build a key token that does not contain a key value. This is the default.</td>
</tr>
<tr>
<td></td>
<td>Payload format version (one, optional). Identifies format of the payload. Release 4.4 or later.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>028</td>
<td>V1PYLD</td>
<td>X'01'</td>
<td>Build the key token with a version 1 payload format. This format has a fixed length and the key length cannot be inferred by the size of the payload. An obscured key length is considered more secure. This is the default.</td>
</tr>
<tr>
<td></td>
<td>Associated data section</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Algorithm type (one required).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>041</td>
<td>AES</td>
<td>X'02'</td>
<td>Key can be used for AES algorithm.</td>
</tr>
<tr>
<td></td>
<td>Key type (one required).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>042</td>
<td>DKYGENKY</td>
<td>X'0009'</td>
<td>Key can be used for generating a diversified key.</td>
</tr>
<tr>
<td></td>
<td>Type of key to diversify (one required). Key-usage field 1, high-order byte.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>045</td>
<td>D-ALL</td>
<td>X'00'</td>
<td>Key can generate a diversified key for any key type listed herafter.</td>
</tr>
<tr>
<td></td>
<td>D-CIPHER</td>
<td>X'01'</td>
<td>Key can generate a diversified CIPHER key.</td>
</tr>
<tr>
<td></td>
<td>D-EXP</td>
<td>X'03'</td>
<td>Key can generate a diversified EXPORTER key.</td>
</tr>
<tr>
<td></td>
<td>D-IMP</td>
<td>X'04'</td>
<td>Key can generate a diversified IMPORTER key.</td>
</tr>
<tr>
<td></td>
<td>D-MAC</td>
<td>X'02'</td>
<td>Key can generate a diversified MAC key.</td>
</tr>
<tr>
<td></td>
<td>D-PCALC</td>
<td>X'06'</td>
<td>Key can generate a diversified PINCALC key.</td>
</tr>
<tr>
<td></td>
<td>D-PPROT</td>
<td>X'05'</td>
<td>Key can generate a diversified PINPROT key.</td>
</tr>
<tr>
<td></td>
<td>D-PPRW</td>
<td>X'07'</td>
<td>Key can generate a diversified PINPRW key.</td>
</tr>
<tr>
<td></td>
<td>D-SECMSG</td>
<td>X'08'</td>
<td>Key can generate a diversified SECMSG key.</td>
</tr>
</tbody>
</table>

User-defined extension (UIDX) control (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.
### Table 80. Key Token Build2 rule array keywords for AES DKYGENKY keys (continued)

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>046</td>
<td>UDX-ONLY</td>
<td>B'xxxx 1xx'</td>
<td>Key can only be used in UDXs.</td>
</tr>
<tr>
<td></td>
<td>UDX-100</td>
<td>B'xxxx x1uu'</td>
<td>Leftmost user-defined UDX bit is set on.</td>
</tr>
<tr>
<td></td>
<td>UDX-010</td>
<td>B'xxxx xu1u'</td>
<td>Middle user-defined UDX bit is set on.</td>
</tr>
<tr>
<td></td>
<td>UDX-001</td>
<td>B'xxxx xu1u'</td>
<td>Rightmost user-defined UDX bit is set on.</td>
</tr>
</tbody>
</table>

**Related generated key-usage field level of control (one required).** Key-usage field 2, high-order byte. If **D-ALL** is specified, **KUF-MBE** and **KUF-MBP** are not valid and there is no default. Otherwise, the default is **KUF-MBE.** **KUF-MBP** is not valid if **DKYUSAGE** is specified and **DKPINOP, DKPINOPP, DKADMIN1,** or **DKADMIN2** is specified in the verb_data variable (that is, the type of key to diversify is DK enabled).

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>047</td>
<td>KUF-MBE</td>
<td>B'1xxx xxxx'</td>
<td>The key usage fields of the key to be generated must be equal to the related generated key-usage fields that start with key usage field 3.</td>
</tr>
<tr>
<td></td>
<td>KUF-MBP</td>
<td>B'0xxx xxxx'</td>
<td>The key usage fields of the key to be generated must be permissible based on the related generated key usage fields that start with key-usage field 3. A key to be diversified is not permitted to have a higher level of usage than any of the related key usage fields permit. The key to be diversified is only permitted to have key usage that is less than or equal to the related key usage fields. One exception is the UDX-ONLY setting in the generated key usage fields. The UDX-ONLY setting must always be equal to the UDX-ONLY setting in the related key usage fields.</td>
</tr>
</tbody>
</table>

**Key-derivation sequence level (one required).** Key-usage field 2, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>048</td>
<td>DKYL0</td>
<td>X'00'</td>
<td>Use this diversifying key to generate a Level 0 diversified key. The type of key to diversify (value at offset 45) determines the key type of the generated key. Level 0 is a completed key.</td>
</tr>
<tr>
<td></td>
<td>DKYL1</td>
<td>X'01'</td>
<td>Use this diversifying key to generate a Level 1 diversified key.</td>
</tr>
<tr>
<td></td>
<td>DKYL2</td>
<td>X'02'</td>
<td>Use this diversifying key to generate a Level 2 diversified key.</td>
</tr>
</tbody>
</table>

**Related generated key usage fields (not allowed for **D-ALL, one, optional, for **D-CIPHER, D-EXP, and D-IMP,** otherwise one required).** Key-usage field 3, high-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>049</td>
<td>DKYUSAGE</td>
<td>Based on verb_data keywords.</td>
<td>The verb_data variable contains key-usage field keywords related to the type of key to diversify. These related attributes become part of the key usage fields of the DKYGENKY diversifying key, beginning with key-usage field 3, high-order byte. They are related because they are used to control which key usage attributes are permissible in the generated diversified key. To generate a diversified key, use the Diversified Key Generate2 (CSNBDKG2) verb.</td>
</tr>
</tbody>
</table>

**Symmetric-key export control (one, optional).** Key-management field 1, high-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>052</td>
<td>NOEX-SYM</td>
<td>B'0xxx xxxx'</td>
<td>Prohibit export using symmetric key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-SYM</td>
<td>B'1xxx xxxx'</td>
<td>Allow export using symmetric key. This is the default.</td>
</tr>
</tbody>
</table>

**Unauthenticated asymmetric-key export control (one, optional).** Key-management field 1, high-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>052</td>
<td>NOEXUASY</td>
<td>B'x0xx xxxx'</td>
<td>Prohibit export using an unauthenticated asymmetric key (not a trusted block).</td>
</tr>
<tr>
<td></td>
<td>XPRTUASY</td>
<td>B'x1xx xxxx'</td>
<td>Allow export using an unauthenticated asymmetric key (not a trusted block). This is the default.</td>
</tr>
</tbody>
</table>

**Authenticated asymmetric-key export control (one, optional).** Key-management field 1, high-order byte.
### Table 80. Key Token Build2 rule array keywords for AES DKYGENKY keys (continued)

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>052</td>
<td>NOEXAASY</td>
<td>B’xx0 x xxxx’</td>
<td>Prohibit export using an authenticated asymmetric key (trusted block).</td>
</tr>
<tr>
<td></td>
<td>XPRTAASY</td>
<td>B’xx1 x xxxx’</td>
<td>Allow export using an authenticated asymmetric key (trusted block). This is the default.</td>
</tr>
</tbody>
</table>

**RAW-key export control (one, optional).** Key-management field 1, high-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>052</td>
<td>NOEX-RAW</td>
<td>B’xxx0 xxxx’</td>
<td>Prohibit export using raw key. Defined for future use. Currently ignored. This is the default.</td>
</tr>
<tr>
<td></td>
<td>XPRT-RAW</td>
<td>B’xxx1 xxxx’</td>
<td>Allow export using raw key. Defined for future use. Currently ignored.</td>
</tr>
</tbody>
</table>

**DES-key export control (one, optional).** Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>053</td>
<td>NOEX-DES</td>
<td>B’1xxx xxxx’</td>
<td>Prohibit export using a DES key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-DES</td>
<td>B’0xxx xxxx’</td>
<td>Allow export using a DES key. This is the default.</td>
</tr>
</tbody>
</table>

**AES-key export control (one, optional).** Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>053</td>
<td>NOEX-AES</td>
<td>B’x1xx xxxx’</td>
<td>Prohibit export using an AES key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-AES</td>
<td>B’x0xx xxxx’</td>
<td>Allow export using an AES key.</td>
</tr>
</tbody>
</table>

**RSA-key export control (one, optional).** Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>053</td>
<td>NOEX-RSA</td>
<td>B’xxxx 1xxx’</td>
<td>Prohibit export using an RSA key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-RSA</td>
<td>B’xxxx 0xxx’</td>
<td>Allow export using an RSA key. This is the default.</td>
</tr>
</tbody>
</table>

Figure 7 on page 277 shows all the valid keyword combinations and their defaults for AES key type EXPORTER. For a description of these keywords, refer to Table 81 on page 278.
Figure 7. Key Token Build2 keyword combinations for AES EXPORTER keys

Note:

1. **V0PYLD** and **V1PYLD** are for Release 4.4 or later. **V0PYLD** is the default for compatibility reasons. **V1PYLD** is recommended because it provides improved security.

2. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of a high-order byte (HOB) and a low-order byte (LOB).

3. All keywords in this group are defaults unless one or more keywords in this group are specified.

4. There is no default. This keyword is defined for future use and its meaning is currently undefined. To avoid this restriction in the future when the meaning is defined, specify this keyword.

5. **WR-AES, WR-DES, and WR-HMAC** are defaults unless one or more keywords in this group are specified.

6. Choose any number of keywords in this group. No keywords in this group are defaults.
7. WR-CARD, WR-DATA, WR-KEK, WR-PIN, and WRDERIVE are defaults unless one or more keywords in this group are specified. WR-CVAR is for Release 4.4 or later.

8. NOEX-RAW and XPRT-RAW are defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRT-RAW.

Table 81. Key Token Build2 rule array keywords for AES EXPORTER keys

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Key-token header section</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Token identifier (one required).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>EXTERNAL</td>
<td>X'02'</td>
<td>Build a key token that is not to be used locally.</td>
</tr>
<tr>
<td>000</td>
<td>INTERNAL</td>
<td>X'01'</td>
<td>Build a key token that is to be used locally.</td>
</tr>
<tr>
<td></td>
<td>Wrapping-information section</td>
<td></td>
<td></td>
</tr>
<tr>
<td>008</td>
<td>NO-KEY</td>
<td>X'00'</td>
<td>Build a key token that does not contain a key value. This is the default.</td>
</tr>
<tr>
<td></td>
<td>Payload format version (one, optional). Identifies format of the payload. Release 4.4 or later; otherwise, undefined.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>028</td>
<td>V0PYLD</td>
<td>X'00'</td>
<td>Build a key token with a version 0 payload format. This format has a variable length and the key length can be inferred from the size of the payload. This format is compatible with all releases. This is the default.</td>
</tr>
<tr>
<td>028</td>
<td>V1PYLD</td>
<td>X'01'</td>
<td>Build the key token with a version 1 payload format. This format has a fixed length and the key length cannot be inferred by the size of the payload. An obscured key length is considered more secure.</td>
</tr>
<tr>
<td></td>
<td>Associated data section</td>
<td></td>
<td></td>
</tr>
<tr>
<td>041</td>
<td>AES</td>
<td>X'02'</td>
<td>Key can be used for AES algorithm.</td>
</tr>
<tr>
<td></td>
<td>Key type (one required)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>042</td>
<td>EXPORT</td>
<td>X'0003'</td>
<td>Key can be used to wrap an external key to be taken from this local node or to wrap an output key in the Key_Translate2 verb.</td>
</tr>
<tr>
<td></td>
<td>KEK control (one or more, optional). Key-usage field 1, high-order byte. All keywords in the group are defaults unless one or more keywords in the group are specified.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>045</td>
<td>EXPORT</td>
<td>B'1xxx xxxx'</td>
<td>Key can be used to wrap a key taken from this local node. Symmetric_Key_Export.</td>
</tr>
<tr>
<td></td>
<td>GEN-EXEX</td>
<td>B'xxxx 1xxx'</td>
<td>Key can be used to wrap the first or the second key that is generated by the CSNBKGN2 verb as part of an EXEX key pair.</td>
</tr>
<tr>
<td></td>
<td>GEN-IMEX</td>
<td>B'xxx1 xxxx'</td>
<td>Key can be used to wrap the second key that is generated by the CSNBKGN2 verb as part of an IMEX key pair.</td>
</tr>
<tr>
<td></td>
<td>GEN-OPEX</td>
<td>B'xx1x xxxx'</td>
<td>Key can be used to wrap the second key that is generated by the CSNBKGN2 verb as part of an OPEX key pair.</td>
</tr>
<tr>
<td></td>
<td>GEN-PUB</td>
<td>B'xxxx x1xx'</td>
<td>Key can be used to wrap the private key (to be used at another node) generated by the CSNDPKG verb as part of an ECC public-private key pair.</td>
</tr>
<tr>
<td></td>
<td>TRANSLAT</td>
<td>B'x1xx xxxx'</td>
<td>Key can be used to wrap an output key in the Key_Translate2 verb. Key_Translate2.</td>
</tr>
</tbody>
</table>
### Table 81. Key Token Build2 rule array keywords for AES EXPORTER keys (continued)

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User-defined extension (UDX) control</strong> (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>046</td>
<td>UDX-ONLY</td>
<td>B'xxxx 1xxx'</td>
<td>Key can only be used in UDXs.</td>
</tr>
<tr>
<td></td>
<td>UDX-100</td>
<td>B'xxxx x1uu'</td>
<td>Leftmost user-defined UDX bit is set on.</td>
</tr>
<tr>
<td></td>
<td>UDX-010</td>
<td>B'xxxx xu1u'</td>
<td>Middle user-defined UDX bit is set on.</td>
</tr>
<tr>
<td></td>
<td>UDX-001</td>
<td>B'xxxx xu1u'</td>
<td>Rightmost user-defined UDX bit is set on.</td>
</tr>
<tr>
<td><strong>TR-31 wrap control</strong> (one, optional). Key-usage field 2, high-order byte.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>047</td>
<td>WR-TR31</td>
<td>B'1xxx xxxx'</td>
<td>Key can wrap a TR-31 key. Defined for future use. Currently ignored.</td>
</tr>
<tr>
<td><strong>Raw key wrap control</strong> (one, optional). Key-usage field 2, low-order byte.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>048</td>
<td>KEK-RAW</td>
<td>B'xxxx xxx1'</td>
<td>Key can wrap a raw key. Defined for future use. Currently ignored.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Algorithm wrap control</strong> (one or more, optional). Key-usage field 3, high-order byte. Keywords WR-AES, WR-DES, and WR-HMAC are defaults unless one or more keywords in the group are specified.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>WR-AES</td>
<td>B'x1xx xxxx'</td>
<td>Key can wrap AES keys.</td>
</tr>
<tr>
<td></td>
<td>WR-DES</td>
<td>B'1xxx xxxx'</td>
<td>Key can wrap DES keys.</td>
</tr>
<tr>
<td></td>
<td>WR-ECC</td>
<td>B'xxxx 1xxx'</td>
<td>Key can wrap ECC keys.</td>
</tr>
<tr>
<td></td>
<td>WR-HMAC</td>
<td>B'xx1x xxxx'</td>
<td>Key can wrap HMAC keys.</td>
</tr>
<tr>
<td></td>
<td>WR- RSA</td>
<td>B'xx1x xxxx'</td>
<td>Key can wrap RSA keys.</td>
</tr>
<tr>
<td><strong>Class wrap control</strong> (one or more, optional). Key-usage field 4, high-order byte. Keywords WR-CARD, WR-DATA, WR-KEK, WR-PIN, and WRDERIVE are defaults unless one or more keywords in the group are specified.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>051</td>
<td>WR-CARD</td>
<td>B'xxxx 1xxx'</td>
<td>Key can wrap card class keys.</td>
</tr>
<tr>
<td></td>
<td>WR-CVAR (Release 4.4 or later)</td>
<td>B'xxxx x1xx'</td>
<td>Key can wrap cryptovariable class keys.</td>
</tr>
<tr>
<td></td>
<td>WR-DATA</td>
<td>B'1xxx xxxx'</td>
<td>Key can wrap data class keys.</td>
</tr>
<tr>
<td></td>
<td>WR-KEK</td>
<td>B'x1xx xxxx'</td>
<td>Key can wrap KEK class keys.</td>
</tr>
<tr>
<td></td>
<td>WR-PIN</td>
<td>B'xx1x xxxx'</td>
<td>Key can wrap PIN class keys.</td>
</tr>
<tr>
<td></td>
<td>WRDERIVE</td>
<td>B'xx1x xxxx'</td>
<td>Key can wrap derivation class keys.</td>
</tr>
<tr>
<td><strong>Symmetric-key export control</strong> (one, optional). Key-management field 1, high-order byte.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>054</td>
<td>NOEX-SYM</td>
<td>B'0xxx xxxx'</td>
<td>Prohibit export using symmetric key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-SYM</td>
<td>B'1xxx xxxx'</td>
<td>Allow export using symmetric key. This is the default.</td>
</tr>
<tr>
<td><strong>Unauthenticated asymmetric-key export control</strong> (one, optional). Key-management field 1, high-order byte.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>054</td>
<td>NOEXUASY</td>
<td>B'x0xx xxxx'</td>
<td>Prohibit export using an unauthenticated asymmetric key (not a trusted block).</td>
</tr>
<tr>
<td></td>
<td>XPRTUASY</td>
<td>B'x1xx xxxx'</td>
<td>Allow export using an unauthenticated asymmetric key (not a trusted block). This is the default.</td>
</tr>
<tr>
<td><strong>Authenticated asymmetric-key export control</strong> (one, optional). Key-management field 1, high-order byte.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>054</td>
<td>NOEXAAASY</td>
<td>B'xx0x xxxx'</td>
<td>Prohibit export using an authenticated asymmetric key (trusted block).</td>
</tr>
<tr>
<td></td>
<td>XPRTAAASY</td>
<td>B'xx1x xxxx'</td>
<td>Allow export using an authenticated asymmetric key (trusted block). This is the default.</td>
</tr>
<tr>
<td><strong>RAW-key export control</strong> (one, optional). Key-management field 1, high-order byte.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 81. Key Token Build2 rule array keywords for AES EXPORTER keys (continued)

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>054</td>
<td>NOEX-RAW</td>
<td>B’xxx0 xxxx’</td>
<td>Prohibit export using raw key. Defined for future use. Currently ignored. This is the default.</td>
</tr>
<tr>
<td></td>
<td>XPRT-RAW</td>
<td>B’xxx1 xxxx’</td>
<td>Allow export using raw key. Defined for future use. Currently ignored.</td>
</tr>
</tbody>
</table>

**DES-key export control** (one, optional). Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>055</td>
<td>NOEX-DES</td>
<td>B’1xxx xxxx’</td>
<td>Prohibit export using a DES key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-DES</td>
<td>B’0xxx xxxx’</td>
<td>Allow export using a DES key. This is the default.</td>
</tr>
</tbody>
</table>

**AES-key export control** (one, optional). Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>055</td>
<td>NOEX-AES</td>
<td>B’x1xx xxxx’</td>
<td>Prohibit export using an AES key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-AES</td>
<td>B’x0xx xxxx’</td>
<td>Allow export using an AES key.</td>
</tr>
</tbody>
</table>

**RSA-key export control** (one, optional). Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>055</td>
<td>NOEX-RSA</td>
<td>B’xxxx 1xxx’</td>
<td>Prohibit export using an RSA key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-RSA</td>
<td>B’xxxx 0xxx’</td>
<td>Allow export using an RSA key. This is the default.</td>
</tr>
</tbody>
</table>

Figure 8 on page 281 shows all the valid keyword combinations and their defaults for AES key type IMPORTER. For a description of these keywords, refer to Table 82 on page 282.
Figure 8. Key Token Build2 keyword combinations for AES IMPORTER keys

Note:

1. V0PYLD and V1PYLD are for Release 4.4 or later. V0PYLD is the default for compatibility reasons. V1PYLD is recommended because it provides improved security.

2. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of a high-order byte (HOB) and a low-order byte (LOB).

3. All keywords in this group are defaults unless one or more keywords in this group are specified.

4. This keyword is defined for future use and its meaning is currently undefined. To avoid this restriction in the future when the meaning is defined, specify this keyword.

5. WR-AES, WR-DES, and WR-HMAC are defaults unless one or more keywords in this group are specified.

6. Choose any number of keywords in this group. No keywords in this group are defaults.
Key Token Build2 (CSNBKTB2)

7. WR-CARD, WR-DATA, WR-KEK, WR-PIN, and WRDERIVE are defaults unless one or more keywords in this group are specified. WR-CVAR is for Release 4.4 or later.

8. NOEX-RAW and XPRT-RAW are defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRT-RAW.

<table>
<thead>
<tr>
<th>Table 82. Key Token Build2 rule array keywords for AES IMPORTER keys</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Token offset</strong></td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td><strong>Key-token header section</strong></td>
</tr>
<tr>
<td>000</td>
</tr>
<tr>
<td>000</td>
</tr>
<tr>
<td><strong>Wrapping-information section</strong></td>
</tr>
<tr>
<td>008</td>
</tr>
<tr>
<td><strong>Payload format version (one, optional). Identifies format of the payload.</strong></td>
</tr>
<tr>
<td>028</td>
</tr>
<tr>
<td>028</td>
</tr>
<tr>
<td><strong>Associated data section</strong></td>
</tr>
<tr>
<td>041</td>
</tr>
<tr>
<td><strong>Key type (one required)</strong></td>
</tr>
<tr>
<td>042</td>
</tr>
</tbody>
</table>

KEK control (one or more, optional). Key-usage field 1, high-order byte. All keywords in the group are defaults unless one or more keywords in the group are specified.
### Table 82. Key Token Build2 rule array keywords for AES IMPORTER keys (continued)

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>045</td>
<td>GEN-IMEX</td>
<td>B’xxx1 xxxx’</td>
<td>Key can be used to wrap the first key that is generated by the CSNBKGN2 verb as part of an IMEX key pair.</td>
</tr>
<tr>
<td></td>
<td>GEN-IMIM</td>
<td>B’xxxx 1xx’</td>
<td>Key can be used to wrap the first or second key that is generated by the CSNBKGN2 verb as part of an IMIM key pair.</td>
</tr>
<tr>
<td></td>
<td>GEN-OPIM</td>
<td>B’xx1x xxxx’</td>
<td>Key can be used to wrap the second key that is generated by the CSNBKGN2 verb as part of an OPIM key pair.</td>
</tr>
<tr>
<td></td>
<td>GEN-PUB</td>
<td>B’xxxx x1xx’</td>
<td>Key can be used to wrap the private key (to be used at the local node) generated by the CSNDPKG verb as part of an ECC public-private key pair.</td>
</tr>
<tr>
<td></td>
<td>IMPORT</td>
<td>B’1xxx xxxx’</td>
<td>Key can be used to unwrap a key brought to this local node. Symmetric_Key_Import.</td>
</tr>
<tr>
<td></td>
<td>TRANSLAT</td>
<td>B’x1xx xxxx’</td>
<td>Key can be used to unwrap an input key in the Key_Translate2 verb. Key_Translate2.</td>
</tr>
<tr>
<td></td>
<td>UDX-ONLY</td>
<td>B’xxxx 1xx’</td>
<td>Key can only be used in UDXs.</td>
</tr>
<tr>
<td></td>
<td>UDX-100</td>
<td>B’xxxx x1uu’</td>
<td>Leftmost user-defined UDX bit is set on.</td>
</tr>
<tr>
<td></td>
<td>UDX-010</td>
<td>B’xxxx xu1u’</td>
<td>Middle user-defined UDX bit is set on.</td>
</tr>
<tr>
<td></td>
<td>UDX-001</td>
<td>B’xxxx xu1u’</td>
<td>Rightmost user-defined UDX bit is set on.</td>
</tr>
<tr>
<td></td>
<td>TR-31 wrap control</td>
<td>B’1xxx xxxx’</td>
<td>Key can unwrap a TR-31 key. Defined for future use. Currently ignored.</td>
</tr>
<tr>
<td></td>
<td>WR-TR31</td>
<td>B’1xxx xxxx’</td>
<td>Key can unwrap a TR-31 key. Defined for future use. Currently ignored.</td>
</tr>
<tr>
<td></td>
<td>KEK-RAW</td>
<td>B’xxxx xxx1’</td>
<td>Key can unwrap a raw key. Defined for future use. Currently ignored.</td>
</tr>
<tr>
<td></td>
<td>WR-AES</td>
<td>B’x1xx xxxx’</td>
<td>Key can unwrap AES keys.</td>
</tr>
<tr>
<td></td>
<td>WR-DES</td>
<td>B’1xxx xxxx’</td>
<td>Key can unwrap DES keys.</td>
</tr>
<tr>
<td></td>
<td>WR-ECC</td>
<td>B’xxxx 1xx’</td>
<td>Key can unwrap ECC keys.</td>
</tr>
<tr>
<td></td>
<td>WR-HMAC</td>
<td>B’xx1x xxxx’</td>
<td>Key can unwrap HMAC keys.</td>
</tr>
<tr>
<td></td>
<td>WR-RSA</td>
<td>B’xx1x xxxx’</td>
<td>Key can unwrap RSA keys.</td>
</tr>
<tr>
<td></td>
<td>WR-CARD</td>
<td>B’xxx1 xxx’</td>
<td>Key can unwrap card class keys.</td>
</tr>
<tr>
<td></td>
<td>WR-CVAR</td>
<td>B’xxx x1xx’</td>
<td>Key can unwrap cryptovariable class keys. (Release 4.4 or later)</td>
</tr>
<tr>
<td></td>
<td>WR-DATA</td>
<td>B’1xxx xxxx’</td>
<td>Key can unwrap data class keys.</td>
</tr>
<tr>
<td></td>
<td>WR-KEK</td>
<td>B’x1xx xxxx’</td>
<td>Key can unwrap KEK class keys.</td>
</tr>
<tr>
<td></td>
<td>WR-PIN</td>
<td>B’xx1x xxxx’</td>
<td>Key can unwrap PIN class keys.</td>
</tr>
<tr>
<td></td>
<td>WRDERIVE</td>
<td>B’xxx1 xxxx’</td>
<td>Key can wrap derivation class keys.</td>
</tr>
</tbody>
</table>

**User-defined extension (UDX) control** (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.

**TR-31 wrap control** (one, optional). Key-usage field 2, high-order byte.

**Raw key wrap control** (one, optional). Key-usage field 2, low-order byte.

**Algorithm wrap control** (one or more, optional). Key-usage field 3, high-order byte. Keywords WR-AES, WR-DES, and WR-HMAC are defaults unless one or more keywords in the group are specified.

**Class wrap control** (one or more, optional). Key-usage field 4, high-order byte. Keywords WR-CARD, WR-DATA, WR-KEK, WR-PIN, and WRDERIVE are defaults unless one or more keywords in the group are specified.

**Symmetric-key export control** (one, optional). Key-management field 1, high-order byte.
### Table 82. Key Token Build2 rule array keywords for AES IMPORTER keys (continued)

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>054</td>
<td>NOEX-SYM</td>
<td>B'0xxx xxxx'</td>
<td>Prohibit export using symmetric key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-SYM</td>
<td>B'1xxx xxxx'</td>
<td>Allow export using symmetric key. This is the default.</td>
</tr>
</tbody>
</table>

*Unauthenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.*

| 054          | NOEXUASY         | B'x0xx xxxx' | Prohibit export using an unauthenticated asymmetric key (not a trusted block). |
|              | XPRTUASY         | B'x1xx xxxx' | Allow export using an unauthenticated asymmetric key (not a trusted block). This is the default. |

*Authenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.*

| 054          | NOEXAASY         | B'xx0x xxxx' | Prohibit export using an authenticated asymmetric key (trusted block). |
|              | XPRTAASY         | B'xx1x xxxx' | Allow export using an authenticated asymmetric key (trusted block). This is the default. |

*RAW-key export control (one, optional). Key-management field 1, high-order byte.*

| 054          | NOEX-RAW         | B'xxx0 xxxx' | Prohibit export using raw key. Defined for future use. Currently ignored. This is the default. |
|              | XPRT-RAW         | B'xxx1 xxxx' | Allow export using raw key. Defined for future use. Currently ignored. |

*DES-key export control (one, optional). Key-management field 1, low-order byte.*

| 055          | NOEX-DES         | B'1xxx xxxx' | Prohibit export using a DES key. |
|              | XPRT-DES         | B'0xxx xxxx' | Allow export using a DES key. This is the default. |

*AES-key export control (one, optional). Key-management field 1, low-order byte.*

| 055          | NOEX-AES         | B'x1xx xxxx' | Prohibit export using an AES key. |
|              | XPRT-AES         | B'x0xx xxxx' | Allow export using an AES key. |

*RSA-key export control (one, optional). Key-management field 1, low-order byte.*

| 055          | NOEX-RSA         | B'xxxx 1xxx' | Prohibit export using an RSA key. |
|              | XPRT-RSA         | B'xxxx 0xxx' | Allow export using an RSA key. This is the default. |

Figure 9 on page 285 shows all the valid keyword combinations and their defaults for AES key type MAC. For a description of these keywords, refer to Table 83 on page 285.
Figure 9. Key Token Build2 keyword combinations for AES MAC keys

Note:
1. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of two bytes: a high-order byte (HOB) and a low-order byte (LOB).
2. NOEX-RAW and XPRT-RAW are defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRT-RAW.
3. Choose any number of keywords in this group. No keywords in this group are defaults.

Table 83. Key Token Build2 rule array keywords for AES MAC keys

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Key-token header section</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>EXTERNAL</td>
<td>X'02'</td>
<td>Build a key token that is not to be used locally.</td>
</tr>
<tr>
<td></td>
<td>INTERNAL</td>
<td>X'01'</td>
<td>Build a key token that is to be used locally.</td>
</tr>
<tr>
<td></td>
<td>Wrapping-information section</td>
<td></td>
<td></td>
</tr>
<tr>
<td>008</td>
<td>NO-KEY</td>
<td>X'00'</td>
<td>Build a key token that does not contain a key value. This is the default.</td>
</tr>
</tbody>
</table>

Payload format version (one, optional). Identifies format of the payload.
### Table 83. Key Token Build2 rule array keywords for AES MAC keys (continued)

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>028</td>
<td>V1PYLD</td>
<td>X’01’</td>
<td>Build the key token with a version 1 payload format. This format has a fixed length and the key length cannot be inferred by the size of the payload. An obscured key length is considered more secure. This is the default.</td>
</tr>
</tbody>
</table>

**Associated data section**

**Algorithm type** (one required).

| 041          | AES               | X’02’       | Key can be used for AES algorithm. |

**Key type** (one required)

| 042          | MAC               | X’0002’     | Key can be used for generation and verification of message authentication codes. |

**MAC operation** (one required). Key-usage field 1, high-order byte.

| 045          | GENERATE          | B’11xx xxxx’ | Key can be used for generate; key can be used for verify. Not valid with keywords DKPINOP, DKPINAD1, and DKPINAD2. MAC_Generate2 and MAC_Verify2. |
|             | GENONLY           | B’10xx xxxx’ | Key can be used for generate; key cannot be used for verify. |
|             | VERIFY            | B’01xx xxxx’ | Key cannot be used for generate; key can be used for verify. MAC_Verify2. |

**User-defined extension (UDX) control** (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.

| 046          | UDX-ONLY          | B’xxxx 1xxx’ | Key can only be used in UDXs. |
|             | UDX-100           | B’xxxx x1uu’ | Leftmost user-defined UDX bit is set on. |
|             | UDX-010           | B’xxxx xu1u’ | Middle user-defined UDX bit is set on. |
|             | UDX-001           | B’xxxx xuu1’ | Rightmost user-defined UDX bit is set on. |

**MAC mode** (one required). Key-usage field 2, high-order byte.

| 047          | CMAC              | X’01’       | Key can be used for block cipher-based MAC algorithm, called CMAC (NIST SP 800-38B). |

**Common control** (one, optional). Key-usage field 3, high-order byte. Use of a common control keyword causes key-usage field 3, low-order byte (field format identifier at token offset 050) to be set to X’01’ (DK enabled).

| 049          | DKPINOP           | X’01’       | PIN_OP |
|             | DKPINAD1         | X’03’       | PIN_AD1 |
|             | DKPINAD2         | X’04’       | PIN_AD2 |

**Symmetric-key export control** (one, optional). Key-management field 1, high-order byte.

| 052 if DK enabled, else 050 | NOEX-SYM | B’0xxx xxxx’ | Prohibit export using symmetric key. |
|                            | XPR-T-SYM | B’1xxx xxxx’ | Allow export using symmetric key. This is the default. |

**Unauthenticated asymmetric-key export control** (one, optional). Key-management field 1, high-order byte.

| 052 if DK enabled, else 050 | NOEXUASY | B’x0xx xxxx’ | Prohibit export using an unauthenticated asymmetric key (not a trusted block). |
|                            | XPR-TUASY | B’x1xx xxxx’ | Allow export using an unauthenticated asymmetric key (not a trusted block). This is the default. |

**Authenticated asymmetric-key export control** (one, optional). Key-management field 1, high-order byte.
Table 83. Key Token Build2 rule array keywords for AES MAC keys (continued)

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>052 if DK enabled, else 050</td>
<td>NOEXAASY</td>
<td>B’xx0x xxxx’</td>
<td>Prohibit export using an authenticated asymmetric key (trusted block).</td>
</tr>
<tr>
<td></td>
<td>XPRTAASY</td>
<td>B’xx1x xxxx’</td>
<td>Allow export using an authenticated asymmetric key (trusted block). This is the default.</td>
</tr>
</tbody>
</table>

RAW-key export control (one, optional). Key-management field 1, high-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>052 if DK enabled, else 050</td>
<td>NOEX-RAW</td>
<td>B’xxx0 xxxx’</td>
<td>Prohibit export using raw key. Defined for future use. Currently ignored. This is the default.</td>
</tr>
<tr>
<td></td>
<td>XPRT-RAW</td>
<td>B’xxx1 xxxx’</td>
<td>Allow export using raw key. Defined for future use. Currently ignored.</td>
</tr>
</tbody>
</table>

DES-key export control (one, optional). Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>053 if DK enabled, else 051</td>
<td>NOEX-DES</td>
<td>B’1xxx xxxx’</td>
<td>Prohibit export using a DES key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-DES</td>
<td>B’0xxx xxxx’</td>
<td>Allow export using a DES key. This is the default.</td>
</tr>
</tbody>
</table>

AES-key export control (one, optional). Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>053 if DK enabled, else 051</td>
<td>NOEX-AES</td>
<td>B’x1xx xxxx’</td>
<td>Prohibit export using an AES key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-AES</td>
<td>B’x0xx xxxx’</td>
<td>Allow export using an AES key.</td>
</tr>
</tbody>
</table>

RSA-key export control (one, optional). Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>053 if DK enabled, else 051</td>
<td>NOEX-RSA</td>
<td>B’xxxx 1xxx’</td>
<td>Prohibit export using an RSA key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-RSA</td>
<td>B’xxxx 0xxx’</td>
<td>Allow export using an RSA key. This is the default.</td>
</tr>
</tbody>
</table>

Figure 10 on page 288 shows all the valid keyword combinations and their defaults for HMAC key type MAC. For a description of these keywords, refer to Table 84 on page 288.
Note:
1. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of two bytes: a high-order byte (HOB) and a low-order byte (LOB).
2. NOEX-RAW and XPRT-RAW are defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRT-RAW.
3. All keywords in this group are defaults unless one or more keywords in this group are specified.
4. Choose any number of keywords in this group. No keywords in this group are defaults.

Table 84. Key Token Build2 rule array keywords for HMAC MAC keys

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Key-token header section</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>EXTERNAL</td>
<td>X'02'</td>
<td>Build a key token that is not to be used locally.</td>
</tr>
<tr>
<td></td>
<td>INTERNAL</td>
<td>X'01'</td>
<td>Build a key token that is to be used locally.</td>
</tr>
<tr>
<td></td>
<td>Wrapping-information section</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Key status (one, optional)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 84. Key Token Build2 rule array keywords for HMAC MAC keys  (continued)

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>008</td>
<td>KEY-CLR</td>
<td>X'01'</td>
<td>Build a key token that contains a clear key.</td>
</tr>
<tr>
<td></td>
<td>NO-KEY</td>
<td>X'00'</td>
<td>Build a key token that does not contain a key value. This is the default.</td>
</tr>
</tbody>
</table>

Payload format version (one, optional). Identifies format of the payload. Release 4.4 or later; otherwise, undefined.

| 028          | V0PYLD            | X'00'        | Build a key token with a version 0 payload format. This format has a variable length and the key length can be inferred from the size of the payload. This is the default. This format is compatible with all releases. |

Associated data section

Algorithm type (one required).

| 041          | HMAC              | X'03'        | Key can be used for HMAC algorithm. |

Key type (one required)

| 042          | MAC               | X'0002'      | Key can be used for generation or verification of message authentication codes. |

MAC operation (one required). Key-usage field 1, high-order byte.

| 045          | GENERATE          | B'11xx xxxx' | Key can be used for generate; key can be used for verify. HMAC_Generate and HMAC_Verify. |
|              | VERIFY            | B'01xx xxxx' | Key cannot be used for generate; key can be used for verify. HMAC_Verify. |

User-defined extension (UDX) control (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.

| 046          | UDX-ONLY          | B'xxxx 1xxx' | Key can only be used in UDXs. |
|              | UDX-100           | B'xxxx x1uu' | Leftmost user-defined UDX bit is set on. |
|              | UDX-010           | B'xxxx xu1u' | Middle user-defined UDX bit is set on. |
|              | UDX-001           | B'xxxx xuu1' | Rightmost user-defined UDX bit is set on. |

Hash method (one, optional). Key-usage field 2, high-order byte. All keywords in the group are defaults unless one or more keywords in the group are specified.

| 047          | SHA-1             | B'1xxx xxxx' | SHA-1 hash method is allowed for the key. |
|              | SHA-224           | B'x1xx xxxx' | SHA-224 hash method is allowed for the key. |
|              | SHA-256           | B'xx1x xxxx' | SHA-256 hash method is allowed for the key. |
|              | SHA-384           | B'xxx1 xxxx' | SHA-384 hash method is allowed for the key. |
|              | SHA-512           | B'xxx 1xxx'  | SHA-512 hash method is allowed for the key. |

Symmetric-key export control (one, optional). Key-management field 1, high-order byte.

| 050          | NOEX-SYM          | B'0xxx xxxx' | Prohibit export using symmetric key. |
|              | XPRT-SYM          | B'1xxx xxxx' | Allow export using symmetric key. This is the default. |

Unauthenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.

| 050          | NOEXUASY          | B'x0xx xxxx' | Prohibit export using an unauthenticated asymmetric key (not a trusted block). |
|              | XPRTUASY          | B'x1xx xxxx' | Allow export using an unauthenticated asymmetric key (not a trusted block). This is the default. |

Authenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.
### Key Token Build2 (CSNBKT2)

**Table 84. Key Token Build2 rule array keywords for HMAC MAC keys (continued)**

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>050</td>
<td>NOEXAASY</td>
<td>B’xx0 xxxx’</td>
<td>Prohibit export using an authenticated asymmetric key (trusted block).</td>
</tr>
<tr>
<td></td>
<td>XPRTAASY</td>
<td>B’xx1 xxxx’</td>
<td>Allow export using an authenticated asymmetric key (trusted block). This is the default.</td>
</tr>
</tbody>
</table>

**RAW-key export control (one, optional).** Key-management field 1, high-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>050</td>
<td>NOEX-RAW</td>
<td>B’xxx0 xxxx’</td>
<td>Prohibit export using raw key. Defined for future use. Currently ignored. This is the default.</td>
</tr>
<tr>
<td></td>
<td>XPRT-RAW</td>
<td>B’xxx1 xxxx’</td>
<td>Allow export using raw key. Defined for future use. Currently ignored.</td>
</tr>
</tbody>
</table>

**DES-key export control (one, optional).** Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>051</td>
<td>NOEX-DES</td>
<td>B’1xxx xxxx’</td>
<td>Prohibit export using a DES key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-DES</td>
<td>B’0xxx xxxx’</td>
<td>Allow export using a DES key. This is the default.</td>
</tr>
</tbody>
</table>

**AES-key export control (one, optional).** Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>051</td>
<td>NOEX-AES</td>
<td>B’x1xx xxxx’</td>
<td>Prohibit export using an AES key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-AES</td>
<td>B’x0xx xxxx’</td>
<td>Allow export using an AES key.</td>
</tr>
</tbody>
</table>

**RSA-key export control (one, optional).** Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>051</td>
<td>NOEX-RSA</td>
<td>B’xxxx 1xxx’</td>
<td>Prohibit export using an RSA key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-RSA</td>
<td>B’xxxx 0xxx’</td>
<td>Allow export using an RSA key. This is the default.</td>
</tr>
</tbody>
</table>

Figure 11 on page 291 shows all the valid keyword combinations and their defaults for AES key type PINCALC. For a description of these keywords, refer to Table 85 on page 291.
Note:
1. Each key-usage field (KUF) and key-management field (KMF) of a version X’05’ variable-length symmetric key-token consists of two bytes: a high-order byte (HOB) and a low-order byte (LOB).

2. NOEX-RAW and XPRU-RAW were defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRU-RAW.

3. Choose any number of keywords in this group. No keywords in the group are defaults.

Table 85. Key Token Build2 rule array keywords for AES PINCALC keys

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token identifier (one required)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>EXTERNAL</td>
<td>X’02’</td>
<td>Build a key token that is not to be used locally.</td>
</tr>
<tr>
<td>000</td>
<td>INTERNAL</td>
<td>X’01’</td>
<td>Build a key token that is to be used locally.</td>
</tr>
</tbody>
</table>

Wrapping-information section

| Key status (one, optional) | | | |
| 008 | NO-KEY | X’00’ | Build a key token that does not contain a key value. This is the default. |

Payload format version (one, optional). Identifies format of the payload.

Figure 11. Key Token Build2 keyword combinations for AES PINCALC keys
Table 85. Key Token Build2 rule array keywords for AES PINCALC keys (continued)

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>028</td>
<td>V1PYLD</td>
<td>X'01'</td>
<td>Build the key token with a version 1 payload format. This format has a fixed length and the key length cannot be inferred by the size of the payload. An obscured key length is considered more secure. This is the default.</td>
</tr>
</tbody>
</table>

**Associated data section**

*Algorithm type* (one required).

| 041 | AES       | X'02'       | Key can be used for AES algorithm. |

*Key type* (one required)

| 042 | PINCALC   | X'0006'     | Key can be used for generation and verification of message authentication codes. |

*MAC operation* (one required). Key-usage field 1, high-order byte.

| 045 | GENONLY   | B'1xxx xxxx' | Key can only be used for generate. |

*User-defined extension (UDX) control* (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.

| 046 | UDX-ONLY  | B'xxxx 1xxx' | Key can only be used in UDXs. |
|     | UDX-100   | B'xxxx x1uu' | Leftmost user-defined UDX bit is set on. |
|     | UDX-010   | B'xxxx xu1u' | Middle user-defined UDX bit is set on. |
|     | UDX-001   | B'xxxx xuu1' | Rightmost user-defined UDX bit is set on. |

*Encryption mode* (one required). Key-usage field 2, high-order byte.

| 047 | CBC       | X'00'       | Key can be used for Cipher Block Chaining. |

*Common control* (one required). Key-usage field 3, high-order byte. Use of a common control keyword causes key-usage field 3, low-order byte (field format identifier at token offset 050) to be set to X'01' (DK enabled).

| 049 | DPKINOP   | X'01'       | PIN_OP |

*Symmetric-key export control* (one, optional). Key-management field 1, high-order byte.

| 052 | NOEX-SYM  | B'0xxx xxxx' | Prohibit export using symmetric key. |
|     | XPRT-SYM  | B'1xxx xxxx' | Allow export using symmetric key. This is the default. |

*Unauthenticated asymmetric-key export control* (one, optional). Key-management field 1, high-order byte.

| 052 | NOEXUASY  | B'x0xx xxxx' | Prohibit export using an unauthenticated asymmetric key (not a trusted block). |
|     | XPRTUASY  | B'x1xx xxxx' | Allow export using an unauthenticated asymmetric key (not a trusted block). This is the default. |

*Authenticated asymmetric-key export control* (one, optional). Key-management field 1, high-order byte.

| 052 | NOEXAASY  | B'xx0x xxxx' | Prohibit export using an authenticated asymmetric key (trusted block). |
|     | XPRTAASY  | B'xx1x xxxx' | Allow export using an authenticated asymmetric key (trusted block). This is the default. |

*RAW-key export control* (one, optional). Key-management field 1, high-order byte.

| 052 | NOEX-RAW  | B'xxx0 xxxx' | Prohibit export using raw key. Defined for future use. Currently ignored. This is the default. |
|     | XPRT-RAW  | B'xxx1 xxxx' | Allow export using raw key. Defined for future use. Currently ignored. |

*DES-key export control* (one, optional). Key-management field 1, low-order byte.
Table 85. Key Token Build2 rule array keywords for AES PINCALC keys (continued)

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>053</td>
<td>NOEX-DES</td>
<td>B'1xxx xxxx'</td>
<td>Prohibit export using a DES key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-DES</td>
<td>B'0xxx xxxx'</td>
<td>Allow export using a DES key. This is the default.</td>
</tr>
</tbody>
</table>

AES-key export control (one, optional). Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>053</td>
<td>NOEX-AES</td>
<td>B'x1xx xxxx'</td>
<td>Prohibit export using an AES key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-AES</td>
<td>B'x0xx xxxx'</td>
<td>Allow export using an AES key.</td>
</tr>
</tbody>
</table>

RSA-key export control (one, optional). Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>053</td>
<td>NOEX-RSA</td>
<td>B'xxxx 1xxx'</td>
<td>Prohibit export using an RSA key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-RSA</td>
<td>B'xxxx 0xxx'</td>
<td>Allow export using an RSA key. This is the default.</td>
</tr>
</tbody>
</table>

Figure 12 shows all the valid keyword combinations and their defaults for AES key type PINPROT. For a description of these keywords, refer to Table 86 on page 294.

Figure 12. Key Token Build2 keyword combinations for AES PINPROT keys

Note:
1. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of two bytes: a high-order byte (HOB) and a low-order byte (LOB).
2. NOEX-RAW and XPRT-RAW are defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRT-RAW.

3. Choose any number of keywords in this group. No keywords in the group are defaults.

Table 86. Key Token Build2 rule array keywords for AES PINPROT keys

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key-token header section</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Token identifier (one required)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>EXTERNAL</td>
<td>X’02’</td>
<td>Build a key token that is not to be used locally.</td>
</tr>
<tr>
<td></td>
<td>INTERNAL</td>
<td>X’01’</td>
<td>Build a key token that is to be used locally.</td>
</tr>
<tr>
<td><strong>Wrapping-information section</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key status (one, optional)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>008</td>
<td>NO-KEY</td>
<td>X’00’</td>
<td>Build a key token that does not contain a key value. This is the default.</td>
</tr>
<tr>
<td><strong>Payload format version</strong> (one, optional). Identifies format of the payload.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>028</td>
<td>V1PYLD</td>
<td>X’01’</td>
<td>Build the key token with a version 1 payload format. This format has a fixed length and the key length cannot be inferred by the size of the payload. An obscured key length is considered more secure. This is the default.</td>
</tr>
<tr>
<td><strong>Associated data section</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithm type (one required)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>041</td>
<td>AES</td>
<td>X’02’</td>
<td>Key can be used for AES algorithm.</td>
</tr>
<tr>
<td>Key type (one required).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>042</td>
<td>PINPROT</td>
<td>X’0005’</td>
<td>Key can be used for encrypting PIN blocks.</td>
</tr>
<tr>
<td>Encryption operation (one required). Key-usage field 1, high-order byte.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>045</td>
<td>DECRYPT</td>
<td>B’01xx xxxx’</td>
<td>Key cannot be used for encryption; key can be used for decryption.</td>
</tr>
<tr>
<td></td>
<td>ENCRYPT</td>
<td>B’10xx xxxx’</td>
<td>Key can be used for encryption; key cannot be used for decryption.</td>
</tr>
<tr>
<td>User-defined extension (UDX) control** (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>046</td>
<td>UDX-ONLY</td>
<td>B’xxxx 1xxx’</td>
<td>Key can only be used in UDXs.</td>
</tr>
<tr>
<td></td>
<td>UDX-100</td>
<td>B’xxxx x1uu’</td>
<td>Leftmost user-defined UDX bit is set on.</td>
</tr>
<tr>
<td></td>
<td>UDX-010</td>
<td>B’xxxx xu1u’</td>
<td>Middle user-defined UDX bit is set on.</td>
</tr>
<tr>
<td></td>
<td>UDX-001</td>
<td>B’xxxx xuu1’</td>
<td>Rightmost user-defined UDX bit is set on.</td>
</tr>
<tr>
<td>Encryption mode** (one required). Key-usage field 2, high-order byte.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>047</td>
<td>CBC</td>
<td>X’00’</td>
<td>Key can be used for Cipher Block Chaining.</td>
</tr>
<tr>
<td>Common control** (one required). Key-usage field 3, high-order byte. Use of a common control keyword causes key-usage field 3, low-order byte (field format identifier at token offset 050) to be set to X’01’ (DK enabled).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>049</td>
<td>DKPINOP</td>
<td>X’01’</td>
<td>PIN_OP</td>
</tr>
<tr>
<td></td>
<td>DKPINOPP</td>
<td>X’02’</td>
<td>PIN_OPP</td>
</tr>
<tr>
<td></td>
<td>DKPINAD1</td>
<td>X’03’</td>
<td>PIN_AD1</td>
</tr>
<tr>
<td>Symmetric-key export control** (one, optional). Key-management field 1, high-order byte.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 86. Key Token Build2 rule array keywords for AES PINPROT keys (continued)

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>052</td>
<td>NOEX-SYM</td>
<td>B'0xxx xxxx'</td>
<td>Prohibit export using symmetric key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-SYM</td>
<td>B'1xxx xxxx'</td>
<td>Allow export using symmetric key. This is the default.</td>
</tr>
</tbody>
</table>

_Unauthenticated asymmetric-key export control_ (one, optional). Key-management field 1, high-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>052</td>
<td>NOEXUASY</td>
<td>B'x0xx xxxx'</td>
<td>Prohibit export using an unauthenticated asymmetric key (not a trusted block).</td>
</tr>
<tr>
<td></td>
<td>XPRTUASY</td>
<td>B'x1xx xxxx'</td>
<td>Allow export using an unauthenticated asymmetric key (not a trusted block). This is the default.</td>
</tr>
</tbody>
</table>

_Authenticated asymmetric-key export control_ (one, optional). Key-management field 1, high-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>052</td>
<td>NOEXAASY</td>
<td>B'xx0x xxxx'</td>
<td>Prohibit export using an authenticated asymmetric key (trusted block).</td>
</tr>
<tr>
<td></td>
<td>XPRTAASY</td>
<td>B'xx1x xxxx'</td>
<td>Allow export using an authenticated asymmetric key (trusted block). This is the default.</td>
</tr>
</tbody>
</table>

_RAW-key export control_ (one, optional). Key-management field 1, high-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>052</td>
<td>NOEX-RAW</td>
<td>B'xxx0 xxxx'</td>
<td>Prohibit export using raw key. Defined for future use. Currently ignored. This is the default.</td>
</tr>
<tr>
<td></td>
<td>XPRT-RAW</td>
<td>B'xxx1 xxxx'</td>
<td>Allow export using raw key. Defined for future use. Currently ignored.</td>
</tr>
</tbody>
</table>

_DES-key export control_ (one, optional). Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>053</td>
<td>NOEX-DES</td>
<td>B'1xxx xxxx'</td>
<td>Prohibit export using a DES key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-DES</td>
<td>B'0xxx xxxx'</td>
<td>Allow export using a DES key. This is the default.</td>
</tr>
</tbody>
</table>

_AES-key export control_ (one, optional). Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>053</td>
<td>NOEX-AES</td>
<td>B'x1xx xxxx'</td>
<td>Prohibit export using an AES key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-AES</td>
<td>B'x0xx xxxx'</td>
<td>Allow export using an AES key.</td>
</tr>
</tbody>
</table>

_RSA-key export control_ (one, optional). Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>053</td>
<td>NOEX-RSA</td>
<td>B'xxxx 1xxx'</td>
<td>Prohibit export using an RSA key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-RSA</td>
<td>B'xxxx 0xxx'</td>
<td>Allow export using an RSA key. This is the default.</td>
</tr>
</tbody>
</table>

Figure 13 on page 296 shows all the valid keyword combinations and their defaults for AES key type PINPRW. For a description of these keywords, refer to Table 87 on page 296.
Key Token Build2 (CSNBKTB2)

![Figure 13. Key Token Build2 keyword combinations for AES PINPRW keys]

Note:
1. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of two bytes: a high-order byte (HOB) and a low-order byte (LOB).
2. NOEX-RAW and XPRT-RAW are defined for future use and their meanings are currently undefined. To avoid this export restriction in the future when the meaning is defined, specify XPRT-RAW.
3. Choose any number of keywords in this group. No keywords in the group are defaults.

Table 87. Key Token Build2 PINPRW related key words

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key-token header section</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>EXTERNAL</td>
<td>X'02'</td>
<td>Build a key token that is not to be used locally.</td>
</tr>
<tr>
<td></td>
<td>INTERNAL</td>
<td>X'01'</td>
<td>Build a key token that is to be used locally.</td>
</tr>
<tr>
<td><strong>Wrapping-information section</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>008</td>
<td>NO-KEY</td>
<td>X'00'</td>
<td>Build a key token that does not contain a key value. This is the default.</td>
</tr>
</tbody>
</table>

Payload format version (one, optional). Identifies format of the payload.
## Table 87. Key Token Build2 PINPRW related key words (continued)

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>028</td>
<td>V1PYLD</td>
<td>X'01'</td>
<td>Build the key token with a version 1 payload format. This format has a fixed length and the key length cannot be inferred by the size of the payload. An obscured key length is considered more secure. This is the default.</td>
</tr>
</tbody>
</table>

### Associated data section

#### Algorithm type (one required).

| 041 | AES | X'02' | Key can be used for AES algorithm. |

#### Key type (one required)

| 042 | PINPRW | X'0007' | Key can be used for generation and verification of message authentication codes. |

#### MAC operation (one required). Key-usage field 1, high-order byte.

| 045 | GENONLY | B'10xx xxxx' | Key can be used for generate; key cannot be used for verify. |
|     | VERIFY  | B'01xx xxxx' | Key cannot be used for generate; key can be used for verify. |

#### User-defined extension (UDX) control (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.

| 046 | UDX-ONLY | B'xxxx 1xxx' | Key can only be used in UDXs. |
|     | UDX-100  | B'xxxx x1uu' | Leftmost user-defined UDX bit is set on. |
|     | UDX-010  | B'xxxx xu1u' | Middle user-defined UDX bit is set on. |
|     | UDX-001  | B'xxxx xu1'  | Rightmost user-defined UDX bit is set on. |

#### MAC mode (one required). Key-usage field 2, high-order byte.

| 047 | CMAC | X'01' | Key can be used for block cipher-based MAC algorithm, called CMAC (NIST SP 800-38B). |

#### Common control (one, required). Key-usage field 3, high-order byte. Use of a common control keyword causes key-usage field 3, low-order byte (field format identifier at token offset 050) to be set to X'01' (DK enabled).

| 049 | DPKINOP | X'01' | PIN_OP |

#### Symmetric-key export control (one, optional). Key-management field 1, high-order byte.

| 052 | NOEX-SYM | B'0xxx xxxx' | Prohibit export using symmetric key. |
|     | XPR-SYM   | B'1xxx xxxx' | Allow export using symmetric key. This is the default. |

#### Unauthenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.

| 052 | NOEXUASY | B'x0xx xxxx' | Prohibit export using an unauthenticated asymmetric key (not a trusted block). |
|     | XPRTUASY  | B'x1xx xxxx' | Allow export using an unauthenticated asymmetric key (not a trusted block). This is the default. |

#### Authenticated asymmetric-key export control (one, optional). Key-management field 1, high-order byte.

| 052 | NOEXAASY | B'xx0x xxxx' | Prohibit export using an authenticated asymmetric key (trusted block). |
|     | XPRTAASY  | B'xx1x xxxx' | Allow export using an authenticated asymmetric key (trusted block). This is the default. |

#### RAW-key export control (one, optional). Key-management field 1, high-order byte.

| 052 | NOEX-RAW | B'xxx0 xxxx' | Prohibit export using raw key. Defined for future use. Currently ignored. This is the default. |
|     | XPR-RAW   | B'xxx1 xxxx' | Allow export using raw key. Defined for future use. Currently ignored. |
Table 87. Key Token Build2 PINPRW related key words (continued)

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES-key export control (one, optional). Key-management field 1, low-order byte.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>053</td>
<td>NOEX-DES</td>
<td>B'1xxx xxxx'</td>
<td>Prohibit export using a DES key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-DES</td>
<td>B'0xxx xxxx'</td>
<td>Allow export using a DES key. This is the default.</td>
</tr>
<tr>
<td>AES-key export control (one, optional). Key-management field 1, low-order byte.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>053</td>
<td>NOEX-AES</td>
<td>B'x1xx xxxx'</td>
<td>Prohibit export using an AES key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-AES</td>
<td>B'x0xx xxxx'</td>
<td>Allow export using an AES key.</td>
</tr>
<tr>
<td>RSA-key export control (one, optional). Key-management field 1, low-order byte.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>053</td>
<td>NOEX-RSA</td>
<td>B'xxxx 1xxx'</td>
<td>Prohibit export using an RSA key.</td>
</tr>
<tr>
<td></td>
<td>XPRT-RSA</td>
<td>B'xxxx 0xxx'</td>
<td>Allow export using an RSA key. This is the default.</td>
</tr>
</tbody>
</table>

Figure 14 shows all the valid keyword combinations and their defaults for AES key type SECMSG.

Algorithm | Key type | Token identifier | Key material state | Payload format ver. |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;&gt;AES</td>
<td>SECMSG</td>
<td>INTERNAL</td>
<td>NO-KEY</td>
<td>V1PYLD</td>
</tr>
</tbody>
</table>

KU1 HOB
Note: One required.
| SMPIN |
KU1 LOB
See Note 4.
| >UDX-ONLY |
| >UDX-001 |
| >UDX-010 |
| >UDX-100 |
KM1 HOB
Note: One required.
| NOEX-SYM |
| NOEXUASY |
| NOEXAASY |
| NOEX-RAW |

Note: NO-KEY is default.
| V1PYLD |

KMF1 HOB
Note: One required.
| NOEX-DES |
| NOEX-AES |
| NOEX-RSA |

Note: V1PYLD is default.
| NOEX-RAW |

KMF1 LOB and KMF1 HOB
See Note 6.
| NOEXPORT |

Note:
1. AES SECMSG is Release 4.4 or later.
2. An AES SECMSG key is always derived. The derived key is the result of a key derivation function (KDF) applied to a fixed diversified key generating key (DKYGENKY) and derivation data. The final derived key is used as a session key and is typically used to encipher and decipher PIN information between...
devices. An AES SECMNG key can only be wrapped by an AES master key and cannot be stored in an external key-token.

3. Each key-usage field (KUF) and key-management field (KMF) of a version X'05' variable-length symmetric key-token consists of two bytes: a high-order byte (HOB) and a low-order byte (LOB).

4. Choose any number of keywords in this group. No keywords in this group are defaults.

5. **NOEX-RAW** is defined for future use.

6. There is no default. Specifying **NOEXPORT** is equivalent to specifying all of the export control keywords (*NOEX-SYM, NOEXUASY, NOEXAASY, NOEX-RAW, NOEX-DES, NOEX-AES, and NOEX-RSA*). Do not specify any export control keywords together with NOEXPORT. If NOEXPORT is not specified, **NOEX-SYM, NOEXUASY, NOEXAASY, NOEX-DES, NOEX-AES, and NOEX-RSA** all must be specified, and **NOEX-RAW** is optional.

Table 88. **Key Token Build2 SECMNG related key words**

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key-token header section</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Token identifier (one required).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>INTERNAL</td>
<td>X'01'</td>
<td>Build a key token that is to be used locally.</td>
</tr>
<tr>
<td>Key status (one, optional).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>008</td>
<td>NO-KEY</td>
<td>X'00'</td>
<td>Build a key token that does not contain a key value. This is the default.</td>
</tr>
<tr>
<td>Payload format version (one, optional). Identifies format of the payload.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>028</td>
<td>V1PYLD</td>
<td>X'01'</td>
<td>Build the key token with a version 1 payload format. This format has a fixed length and the key length cannot be inferred by the size of the payload. An obscured key length is considered more secure. This is the default.</td>
</tr>
<tr>
<td><strong>Associated data section</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithm type (one required)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>041</td>
<td>AES</td>
<td>X'02'</td>
<td>Key can be used for AES algorithm.</td>
</tr>
<tr>
<td>Key type (one required)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>042</td>
<td>SECMNG</td>
<td>X'000A'</td>
<td>Key can be used as an EMV secure messaging key for encrypting PINs or for encrypting keys.</td>
</tr>
<tr>
<td>Secure message encryption enablement (one required). Key-usage field 1, high-order byte</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>045</td>
<td>SMPIN</td>
<td>X'00'</td>
<td>Enable the encryption of PINs in an EMV secure message.</td>
</tr>
<tr>
<td>User-defined extension (UDX) control (one or more, optional). Key-usage field 1, low-order byte. No keywords in the group are defaults.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>046</td>
<td>UDX-ONLY</td>
<td>B'xxxx 1xxx'</td>
<td>Key can only be used in UDXs.</td>
</tr>
<tr>
<td></td>
<td>UDX-100</td>
<td>B'xxxx x1uu'</td>
<td>Leftmost user-defined UDX bit is set on.</td>
</tr>
<tr>
<td></td>
<td>UDX-010</td>
<td>B'xxxx xu1u'</td>
<td>Middle user-defined UDX bit is set on.</td>
</tr>
<tr>
<td></td>
<td>UDX-001</td>
<td>B'xxxx xuu1'</td>
<td>Rightmost user-defined UDX bit is set on.</td>
</tr>
<tr>
<td>Verb restriction (one, optional)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Key Token Build2 (CSNBKTB2)

**Table 88. Key Token Build2 SECMMSG related key words (continued)**

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>047</td>
<td>ANY-USE</td>
<td>X'00'</td>
<td>Any verb (service) can use this key. This is the default.</td>
</tr>
<tr>
<td></td>
<td>DPC-ONLY</td>
<td>X'01'</td>
<td>Only CSNBPCDPC can use this key.</td>
</tr>
</tbody>
</table>

**General export control** (one, optional). Equivalent to specifying all export control keywords (NOEX-SYM, NOEXUASY, NOEXAASY, NOEX-RAW, NOEX-DES, NOEX-AES, and NOEX-RSA). Not valid with any other export control keyword. There is no default. Key-management field 1, high-order byte and low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>050</td>
<td>NOEXPORT</td>
<td>B'0000 xxxx'</td>
<td>Prohibits the export of this key in all cases. Equivalent to specifying NOEX-SYM, NOEXUASY, NOEXAASY, NOEX-RAW, NOEX-DES, NOEX-AES, and NOEX-RSA.</td>
</tr>
<tr>
<td>051</td>
<td></td>
<td>B'11xx 1xxx'</td>
<td></td>
</tr>
</tbody>
</table>

**Symmetric-key export control** (one required if NOEXPORT not specified, otherwise not valid). Key-management field 1, high-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>050</td>
<td>NOEX-SYM</td>
<td>B'0xxx xxxx'</td>
<td>Prohibit export using symmetric key.</td>
</tr>
</tbody>
</table>

**Unauthenticated asymmetric-key export control** (one required if NOEXPORT not specified, otherwise not valid). Key-management field 1, high-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>050</td>
<td>NOEXUASY</td>
<td>B'0xxx xxxx'</td>
<td>Prohibit export using an unauthenticated asymmetric key (not a trusted block).</td>
</tr>
</tbody>
</table>

**Authenticated asymmetric-key export control** (one required if NOEXPORT not specified, otherwise not valid). Key-management field 1, high-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>050</td>
<td>NOEXAASY</td>
<td>B'xx0x xxxx'</td>
<td>Prohibit export using an authenticated asymmetric key (trusted block).</td>
</tr>
</tbody>
</table>

**RAW-key export control** (one optional if NOEXPORT not specified; otherwise not valid). Key-management field 1, high-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>050</td>
<td>NOEX-RAW</td>
<td>B'xxx0 xxxx'</td>
<td>Prohibit export using RAW key. Defined for future use. Currently ignored. This is the default.</td>
</tr>
</tbody>
</table>

**DES-key export control** (one required if NOEXPORT not specified, otherwise not valid). Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>051</td>
<td>NOEX-DES</td>
<td>B'1xxx xxxx'</td>
<td>Prohibit export using a DES key.</td>
</tr>
</tbody>
</table>

**AES-key export control** (one required if NOEXPORT not specified, otherwise not valid). Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>051</td>
<td>NOEX-AES</td>
<td>B'1xx0 xxxx'</td>
<td>Prohibit export using an AES key.</td>
</tr>
</tbody>
</table>

**RSA-key export control** (one required if NOEXPORT not specified, otherwise not valid). Key-management field 1, low-order byte.

<table>
<thead>
<tr>
<th>Token offset</th>
<th>Rule-array keyword</th>
<th>Offset value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>051</td>
<td>NOEX-RSA</td>
<td>B'xxxx 1xxx'</td>
<td>Prohibit export using an RSA key.</td>
</tr>
</tbody>
</table>

### Restrictions

The restrictions for CSNBKTB2.

None.

### Required commands

The required commands for CSNBKTB2.

None.
Usage notes

The usage notes for CSNBKTB2.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBKTB2J.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBKTB2J(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber clear_key_bit_length,
    byte[] clear_key_value,
    hikmNativeNumber key_name_length,
    byte[] key_name,
    hikmNativeNumber user_associated_data_length,
    byte[] user_associated_data,
    hikmNativeNumber token_data_length,
    byte[] token_data,
    hikmNativeNumber verb_data_length,
    byte[] verb_data,
    hikmNativeNumber target_key_token_length,
    byte[] target_key_token);
```

Key Token Change (CSNBKTC)

Use the Key Token Change verb to re-encipher a DES or AES key from encryption under the old master-key to encryption under the current master-key and to update the keys in internal DES or AES key-tokens.

Note:

1. An application system is responsible for keeping all of its keys in a usable form. When the master key is changed, the CEX*C implementations can use an internal key that is enciphered by either the current or the old master-key. Before the master key is changed a second time, it is important to have a key re-enciphered under the current master-key for continued use of the key. Use the Key Token Change verb to re-encipher such a keys.

2. Previous implementations of IBM CCA products had additional capabilities with this verb such as deleting key records and key tokens in key storage. Also, use of a wild card (*) was supported in those implementations.
Key Token Change (CSNBKTC)

Format
The format of CSNBKTC.

```
CSNBKTC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier)
```

Parameters
The parameters for CSNBKTC.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

`rule_array_count`

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 1, 2, 3, or 4.

`rule_array`

- **Direction:** Input
- **Type:** String array

The rule_array parameter is a pointer to a string variable containing an array of keywords. The keywords are eight bytes in length and must be left-aligned and padded on the right with space characters. The rule_array keywords are described in Table 89.

Table 89. Keywords for Key Token Change control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Re-encipherment method (Required)</td>
<td></td>
</tr>
<tr>
<td>RTCMK</td>
<td>Re-enciphers a DES or AES key to the current master-key in an internal key-token in application storage or in key storage. If the supplied key is already enciphered under the current master-key, the verb returns a positive response (return code 0, reason code 0). If the supplied key is enciphered under the old master-key, the key is updated to encipherment by the current master-key and the verb returns a positive response (return code 0, reason code 0). Other cases return some form of abnormal response.</td>
</tr>
</tbody>
</table>
### Table 89. Keywords for Key Token Change control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
</table>
| RTNMK   | Re-enciphers an internal DES or AES key to the new master-key.  
A key enciphered under the new master key is not usable. It is expected that the user will use this keyword (RTNMK) to take a preparatory step in re-encrypting an external key store that they manage themselves to a new master-key, before the set operation has occurred. Note also that the new master-key register must be full; it must have had the last key part loaded and therefore not be empty or partially full (partially full means that one or more key parts have been loaded but not the last key part).  
The SET operation makes the new master-key operational, moving it to the current master-key register, and the current master-key is displaced into the old master-key register. When this happens, all the keys that were re-enciphered to the new master-key are now usable, because the new master-key is not ‘new’ any more, it is ‘current’.  
Because the RTNMK keyword is added primarily for support of externally managed key storage (see “Key Storage on z/OS (RTNMK-focused)” on page 453, it is not valid to pass a key_identifier when the RTNMK keyword is used. Only a full internal key token (encrypted under the current master-key) can be passed for re-encipherment with the RTNMK keyword. When a key label is passed along with the RTNMK keyword, the error return code 8 with reason code 181 will be returned.  
For more information, see “Key storage with Linux on Z, in contrast to z/OS” on page 451. |
| VALIDATE | Validate an internal key token as described in the key_identifier which is enciphered under the current master key. That is, the same processing as RTNMK is applied. However, after a successful checking of the token, no re-enciphering of the token to the new master key takes place. There is just a return code for a successful validation. |
| REFORMAT | Rewrap the input_key_token with the key wrapping method specified. Only the input_KEYK_identifier will be used. The output_KEYK_identifier is ignored. This keyword was introduced with CCA 4.1.0. |

**Algorithm (Optional)**

- **AES** Specifies that the key token is for an AES key.
- **DES** Specifies that the key token is for a DES key. This is the default.

**Key wrapping method (Optional)**

- **USECONFG** This is the default. Specifies to wrap the key using the configuration setting for the default wrapping method. The default wrapping method configuration setting may be changed using the TKE. This keyword is ignored for AES keys.
- **WRAP-ENH** Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.
- **WRAP-ECB** Use original key wrapping method, which uses ECB wrapping for DES key tokens and CBC wrapping for AES key tokens.

**Translation control (Optional)**

- **ENH-ONLY** Restrict rewrapping of the output_key_token. After the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method.

**key_identifier**

- **Direction:** Input/Output
- **Type:** String

The key_identifier parameter is a pointer to a string variable containing the DES internal key-token or the key label of an internal key-token record in key storage.
Key Token Change (CSNBKTC)

Restrictions
The restrictions for CSNBKTC.
None.

Required commands
The required commands for CSNBKTC.
If you specify the RTCMK keyword, the Key Token Change verb requires the Symmetric Key Token Change - RTCMK command (offset X'0090') to be enabled in the active role.
If you specify the REFORMAT keyword, the Key Token Change verb requires the CKDS Conversion2 - Allow use of REFORMAT command (offset X'014C') to be enabled in the active role.
If you specify the WRAP-ECB or WRAP-ENH key wrapping method, and the default key-wrapping method setting does not match this keyword, the CKDS Conversion2 - Allow wrapping override keywords command (offset X'0146') must be enabled in the active role.
If the WRAP-ECB translation-control keyword is specified, and the key in the input key token is wrapped by the enhanced wrapping method (WRAP-ENH), the verb requires the CKDS Conversion2 - Convert from enhanced to original command (offset X'0147') to be enabled. An active role with offset X'0147' enabled can also use the Key Translate2 verb to translate a key from the enhanced key-wrapping method to the less-secure legacy method.

Usage notes
The usage notes for CSNBKTC.
None.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNBKTCJ.
See "Building Java applications using the CCA JNI" on page 28.

Format
public native void CSNBKTCJ(
      hikmNativeNumber return_code,
      hikmNativeNumber reason_code,
      hikmNativeNumber exit_data_length,
      byte[] exit_data,
      hikmNativeNumber rule_array_count,
      byte[] rule_array,
      byte[] key_identifier);

Key Token Change2 (CSNBKTC2)
Use the Key Token Change2 verb to re-encipher a variable-length HMAC or AES key from encryption under the old master-key to encryption under the current master-key and to update the keys in internal HMAC or AES key tokens.

Note:
1. An application system is responsible for keeping all of its keys in a usable form. When the master key is changed, the CEX*C implementations can use an internal key that is enciphered by either the current or the old master-key. Before the master key is changed a second time, it is important to have a key re-enciphered under the current master-key for continued use of the key. Use the Key Token Change2 verb to re-encipher such a keys.

2. Previous implementations of IBM CCA products had additional capabilities with this verb such as deleting key records and key tokens in key storage. Also, use of a wild card (*) was supported in those implementations.

**Format**

The format of CSNBKTC2.

```c
CSNBKTC2(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_length,
    key_identifier)
```

**Parameters**

The parameters for CSNBKTC2.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 2.

**rule_array**

- **Direction:** Input
- **Type:** String array

The `rule_array` parameter is a pointer to a string variable containing an array of keywords. The keywords are eight bytes in length and must be left-aligned and padded on the right with space characters. The `rule_array` keywords are described in Table 90.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm (One, required)</td>
<td></td>
</tr>
<tr>
<td>HMAC</td>
<td>Specifies that the key token is for an HMAC key.</td>
</tr>
<tr>
<td>AES</td>
<td>Specifies that the key token is for an AES key in a variable-length symmetric key token.</td>
</tr>
</tbody>
</table>

**Table 90. Keywords for Key Token Change2 control information**
### Table 90. Keywords for Key Token Change2 control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTCMK</td>
<td>Re-enciphers the internal key provided by the <code>key_identifier</code> to the current master-key in an internal key-token in application storage or in key storage. If the supplied key is already enciphered under the current master-key the verb returns a positive response (return code 0, reason code 0). If the supplied key is enciphered under the old master-key, the key is updated to encipherment by the current master-key and the verb returns a positive response (return code 0, reason code 0). Other cases return some form of abnormal response.</td>
</tr>
<tr>
<td>RTNMK</td>
<td>Re-enciphers the internal key provided by the <code>key_identifier</code> to the new master-key. A key enciphered under the new master key is not usable. It is expected that the user will use this keyword (RTNMK) to take a preparatory step in re-enciphering an external key store that they manage themselves to a new master-key, before the SET operation has occurred. Note also that the new master-key register must be full. It must have had the last key part loaded and therefore not be empty or partially full (partially full means that one or more key parts have been loaded, but not the last key part). The SET operation makes the new master-key operational, moving it to the current master-key register, and the current master-key is displaced into the old master-key register. When this happens, all the keys that were re-enciphered to the new master-key are now usable, because the new master-key is not new anymore, it is current. Because the RTNMK keyword is added primarily for support of externally managed key storage (see “Key Storage on z/OS (RTNMK-focused)” on page 453), it is not valid to pass a <code>key_identifier</code> when the RTNMK keyword is used. Only a full internal key token (enciphered under the current master-key) can be passed for re-encipherment with the RTNMK keyword. When a key label is passed along with the RTNMK keyword, the error return code 8 with reason code 181 will be returned. For more information, see “Key storage with Linux on Z, in contrast to z/OS” on page 451.</td>
</tr>
<tr>
<td>VALIDATE</td>
<td>Validate an internal key token as described in the <code>key_identifier</code> which is enciphered under the current master key. That is, the same processing as RTNMK is applied. However, after a successful checking of the token, no re-enciphering of the token to the new master key takes place. There is just a return code for a successful validation.</td>
</tr>
</tbody>
</table>

**key_identifier_length**

- **Direction:** Input/Output
- **Type:** Integer

The `key_identifier_length` parameter is a pointer to a string variable containing the length in bytes of the `key_identifier` parameter. On input, this variable contains the number of bytes for the `key_identifier` buffer, and must be large enough to hold the key token or key label. On output, this variable contains the number of bytes of data returned in the `key_identifier` variable.

**key_identifier**

- **Direction:** Input/Output
- **Type:** String

A pointer to a string variable containing an internal variable-length symmetric key-token, or a key label of such a key in AES key-storage. The key token referred to is processed according to the rule-array keywords.

### Restrictions

The restrictions for CSNBKTC2.

This verb was introduced with CCA 4.1.0.
**Required commands**

The required commands for CSNBKTC2.

If you specify the RTNMK keyword, this verb requires the Reencipher CKDS2 command (offset X'00F0') to be enabled in the active role.

If you specify the RTCMK keyword, this verb requires the Symmetric Key Token Change2 - RTCMK command (offset X'00F1') to be enabled in the active role.

**Usage notes**

The usage notes for CSNBKTC2.

None.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBKTC2J.

See "Building Java applications using the CCA JNI" on page 28.

**Format**

```java
public native void CSNBKTC2J(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber key_identifier_length,
    byte[] key_identifier);
```

**Key Token Parse (CSNBKTP)**

The Key Token Parse verb disassembles a key token into separate pieces of information.

This verb can disassemble an external key token or an internal key token in application storage.

Use the `key_token` parameter to specify the key token to disassemble.

This verb returns some of the key token information in a set of variables identified by individual parameters and the remaining key token information as keywords in the `rule_array`.

Control vector information is returned in keywords found in the `rule_array` when the verb can fully parse the control vector. Otherwise, the verb returns return code 4, reason code 2039.

The Key Token Parse verb performs no cryptographic services.
Key Token Parse (CSNBKTP)

Format

The format of CSNBKTP:

```c
CSNBKTP(
    return_code,
    reason_code,
    exit_data_length,
    edit_data,
    key_token,
    key_type,
    rule_array_count,
    rule_array,
    key_value,
    masterkey_verification_pattern_v03,
    reserved_2,
    reserved_3,
    control_vector,
    reserved_4,
    reserved_5,
    reserved_6,
    master_key_verification_pattern_v00)
```

Parameters

The parameters for CSNBKTP.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `edit_data` parameters, see "Parameters common to all verbs" on page 22.

**key_token**

- **Direction:** Input
- **Type:** String

The `key_token` parameter is a pointer to a string variable in application storage containing an external or internal key-token to be disassembled.

**Note:** You cannot use a key label for a key-token record in key storage. The key token must be in application storage.

**key_type**

- **Direction:** Output
- **Type:** String

The `key_type` parameter is a pointer to a string variable containing a keyword defining the key type. The keyword is eight bytes in length and must be left-aligned and padded on the right with space characters. Valid `key_type` keywords are shown here:

- CIPHER
- CVARXCVL
- DKYGENKY
- MAC
- CIPHERX1
- CVARXCVR
- ENCIIPHER
- MACVER
- CIPHERXL
- DATA
- EXPORTER
- OKEYXLAT
- CIPHERXO
- DATAC
- IKEYXLAT
- OPINENC
- CVARDEC
- DATAM
- IMPORTER
- PINENC
- CVARENCE
- DATAMV
- IPINENC
- PINVER
- CVARPINE
- DECIIPHER
- KEYGENKY
- SECMSG

Key types are described in "Types of keys" on page 41.

**rule_array_count**

- **Direction:** Input/Output
Key Token Parse (CSNBKTP)

Type: Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be a minimum of 4.

On input, specify the maximum number of usable array elements that are allocated. On output, the verb sets the value to the number of keywords returned to the application.

`rule_array`

Direction: Output

Type: String array

The `rule_array` parameter is a pointer to a string variable containing an array of keywords that expresses the contents of the key token. The keywords are eight bytes in length and are left-aligned and padded on the right with space characters. The `rule_array` keywords are described in Table 91.

Table 91. Keywords for Key Token Parse control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Token type</strong> (One returned)</td>
<td></td>
</tr>
<tr>
<td>INTERNAL</td>
<td>Specifies an internal key-token.</td>
</tr>
<tr>
<td>EXTERNAL</td>
<td>Specifies an external key-token.</td>
</tr>
<tr>
<td><strong>Key status</strong> (One returned)</td>
<td></td>
</tr>
<tr>
<td>KEY</td>
<td>Indicates the key token contains a key. The <code>key_value</code> parameter contains the key.</td>
</tr>
<tr>
<td>NO-KEY</td>
<td>Indicates the key token does not contain a key.</td>
</tr>
<tr>
<td><strong>Key-wrapping method</strong> (One returned)</td>
<td></td>
</tr>
<tr>
<td>WRAP-ECB</td>
<td>The wrapping method for this key is legacy. This keyword was introduced with CCA 4.1.0.</td>
</tr>
<tr>
<td>WRAP-ENH</td>
<td>The wrapping method for this key is enhanced. This keyword was introduced with CCA 4.1.0.</td>
</tr>
<tr>
<td><strong>Control-vector (CV) status</strong> (One returned)</td>
<td></td>
</tr>
<tr>
<td>CV</td>
<td>The key token specifies that a control vector is present. The verb sets the control vector variable with the value of the control vector found in the key token.</td>
</tr>
<tr>
<td>NO-CV</td>
<td>The key token does not specify the presence of a control vector. The verb sets the control vector variable with the value of the control vector variable found in the key token.</td>
</tr>
</tbody>
</table>

The difference between Key Token Parse (CSNBKTP) and Control Vector Generate (CSNBCVG) is that Key Token Parse returns the `rule_array` keywords that apply to a parsed token, such as `EXTERNAL`, `INTERNAL`, and so forth. These `rule_array` parameters are returned in addition to `key_type` parameter. Refer to “Key Token Build (CSNBKTB)” on page 259 and “Key Token Build2 (CSNBKTB2)” on page 264 for keyword discussion.

AMEX-CSC   DKL0    EPINGEN    KEYLN16  UKPT
ANSIX9.9   DKL1    EPINGENA   LMTD-KEK VISA-PVW
ANY        DKL2    EPINVER    MIXED    WRAP-ECB
ANY-MAC    DKL3    EXEK       NO-SPEC  WRAP-ENH
CLR8-ENC   DKL4    EXPORT     NO-XPORT XLATE
CPINENC    DKL5    GBP-PIN    NOOFFSET XPORT-OK
CPINGEN    DKL6    GBP-PINO   NOT-KEK
CPINGENA   DKL7    IBM-PIN    OPEX
CVVKEY-A    DMAC    IBM-PINO   OPIM
CVVKEY-B    DMKEY    IMEX      PIN
DALL       DMPIN    IMIM      reformat
DATA       DMV      IMPORT    SINGLE
Key Token Parse (CSNBKTP)

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key_value</td>
<td>Input</td>
<td>The key_value parameter is a pointer to a string variable. If the verb returns the KEY keyword in the rule_array, the key_value parameter contains the 16-byte enciphered key.</td>
</tr>
<tr>
<td>masterkey_verification_pattern_v03</td>
<td>Output</td>
<td>The masterkey_verification_pattern_v03 parameter is a pointer to an integer variable. The verb writes zero into the variable except when parsing a version X'03' internal key-token.</td>
</tr>
<tr>
<td>reserved_2/5</td>
<td>Output</td>
<td>The reserved_2 and reserved_5 parameters are either null pointers or pointers to integer variables. If the parameter is not a null pointer, the verb writes zero into the reserved variable.</td>
</tr>
<tr>
<td>reserved_3/4</td>
<td>Output</td>
<td>The reserved_3 and reserved_4 parameters are either null pointers or pointers to string variables. If the parameter is not a null pointer, the verb writes eight bytes of X'00' into the reserved variable.</td>
</tr>
<tr>
<td>reserved_6</td>
<td>Output</td>
<td>The reserved_6 parameter is either a null pointer or a pointer to a string variable. If the parameter is not a null pointer, the verb writes eight space characters into the reserved variable.</td>
</tr>
<tr>
<td>control_vector</td>
<td>Output</td>
<td>The control_vector parameter is a pointer to a string variable in application storage. If the verb returns the NO-CV keyword in the rule_array, the key token did not contain a control-vector value and the control vector variable is filled with 16 space characters.</td>
</tr>
<tr>
<td>master_key_verification_pattern_v00</td>
<td>Output</td>
<td>The master_key_verification_pattern_v00 parameter is a pointer to a string variable in application storage. If the verb returns the NO-CV keyword in the rule_array, the key token did not contain a control-vector value and the control vector variable is filled with 16 space characters.</td>
</tr>
</tbody>
</table>
The `master_key_verification_pattern_v00` parameter is a pointer to a string variable in application storage. For version 0 key-tokens that contain a key, the 8-byte master key version number is copied to the variable. Otherwise the variable is filled with eight space characters.

**Restrictions**

The restrictions for CSNBKTP.

None.

**Required commands**

The required commands for CSNBKTP.

None.

**Usage notes**

The usage notes for CSNBKTP.

Be aware that Key Token Parse (CSNBKTP) will fail (return code 8, reason code 49) when given a DES INTERNAL key token that is version X'01'. These tokens are DOUBLE and TRIPLE length DES INTERNAL DATA key tokens.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBKTPJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBKTPJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] key_token,
    byte[] key_type,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] key_value,
    hikmNativeNumber master_key_verification_pattern_v03,
    hikmNativeNumber reserved_2,
    byte[] reserved_3,
    byte[] control_vector,
    byte[] reserved_4,
    hikmNativeNumber reserved_5,
    byte[] reserved_6,
    byte[] master_key_verification_pattern_v00);
```

**Key Token Parse2 (CSNBKTP2)**

Use the Key Token Parse2 verb to disassemble a variable-length symmetric key-token into separate pieces of information.

The verb can disassemble an external or internal variable-length symmetric key-token in application storage into separate pieces of information. To parse a fixed-length symmetric key-token, see “Key Token Parse (CSNBKTP)” on page 307.
The **key_token** input parameter specifies the external or internal key token to disassemble. The verb returns some of the key-token information in a set of variables identified by individual parameters, and returns the remaining information as keywords in the rule array.

The key-usage field and key-management field information is returned in keywords found in the rule array when the verb can fully parse the fields. See the rule_array parameter on page "rule_array" on page 314 for a table of supported keywords. If the token cannot be parsed successfully, the verb returns a warning using reason code 2039 X'7F7'). If a warning or error occurs during processing, the verb updates all of the count and length variables with a value of zero.

The Key Token Parse2 verb performs no cryptographic services.

To use this verb, specify the following:

- An external or internal variable-length symmetric key-token (version X'05') to be parsed.
  
  This parameter does not accept a key label. The key token must be provided from application storage. If a key token located in key storage needs to be parsed, use the AES Key Record Read verb to retrieve it into application storage before calling this verb.
  
  See "HMAC key token" on page 888 for the format of this key token. A review of this format information will greatly assist in understanding the output variables of this verb.

- A rule-array-count value large enough for the verb to return keywords about the input key-token in the rule-array buffer.
  
  To determine the exact count required, and also the required lengths of the other string variables, specify a value of zero. This causes the verb to return all count and length values without updating any string variables.

- Adequate buffer sizes for all of the output variables using the length parameters.
Key Token Parse2 (CSNKTP2)

Format

The format of CSNKTP2.

```
CSNKTP2(  
  return_code,  
  reason_code,  
  exit_data_length,  
  exit_data,  
  key_Token_length,  
  key_token,  
  key_type,  
  rule_array_count,  
  rule_array,  
  key_material_state,  
  payload_bit_length,  
  payload,  
  key_verification_pattern_type,  
  key_verification_pattern_length,  
  key_verification_pattern,  
  key_wrapping_method,  
  key_hash_algorithm,  
  key_name_length,  
  key_name,  
  TLV_data_length,  
  TLV_data,  
  user_associated_data_length,  
  user_associated_data,  
  verb_data_length,  
  verb_data)
```

Parameters

The parameters for CSNKTP2.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**key_token_length**

- **Direction:** Input  
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the `key_token` variable.

**key_token**

- **Direction:** Input  
- **Type:** String

A pointer to a string variable containing an external or internal variable-length symmetric key-token to be disassembled. This parameter must not point to a key label.

**key_type**

- **Direction:** Output  
- **Type:** String

A pointer to a string variable containing a keyword for the key type of the input key. The keyword is 8 bytes in length and is left-aligned and padded on the right with space characters. Valid `key_type` keywords are shown here:
Key Token Parse2 (CSNBKTP2)

Key types are described in “Types of keys” on page 41.

**rule_array_count**
- **Direction:** Input/Output
- **Type:** Integer

A pointer to an integer variable containing the number of 8-byte elements in the `rule_array` variable. The minimum returned value is 3, and the maximum returned value is approximately 50. To determine the exact count required, and also the required lengths of the other string variables, specify a value of zero. This causes the verb to return all count and length values without updating any string variables.

On output, the variable is updated with the actual count of the rule-array keywords. An error is returned if a key token cannot be parsed or any of the output buffers are too small.

**rule_array**
- **Direction:** Output
- **Type:** String array

A pointer to a string variable containing an array of keywords. The keywords are 8 bytes in length, and are left-aligned and padded on the right with space characters. The returned rule array keywords express the contents of the token.

While Key Token Build2 (CSNBKTB2) assembles an internal variable-length symmetric key-token in application storage from information that you supply via the key words in the verb’s `rule_array` parameter, Key Token Parse2 disassembles an input key-token into separate pieces of information which are stored in this verb’s `rule_array` as output. Some of the keywords of Key Token Build2 and Key Token Parse2 therefore have the same meaning.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header section</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Token identifier</strong> (one required)</td>
<td></td>
</tr>
<tr>
<td>EXTERNAL</td>
<td>Specifies to disassemble an external variable-length symmetric key-token.</td>
</tr>
<tr>
<td>INTERNAL</td>
<td>Specifies to disassemble an internal variable-length symmetric key-token.</td>
</tr>
</tbody>
</table>

| **Wrapping information section** | |
| **Key status** (one returned). Refer to the `key_material_state` variable for additional details. | |
| NO-KEY | Key token does not contain a key. The payload variable is empty. |
| KEY | Key token contains a partial or complete key. The payload variable contains the clear or encrypted key. |

**Key verification pattern (KVP) type**
| Note: Not a keyword. Value returned in `key_verification_pattern_type` variable. | |
| **KVP** | Note: Not a keyword. Value returned in `key_verification_pattern` variable. |

**Encrypted section key-wrapping method**
| Note: Not a keyword. Value returned in `key_wrapping_method` variable. | |
| **Hash algorithm used for wrapping** | Note: Not a keyword. Value returned in `key_hash_algorithm` variable. |
### Table 92. Keywords for Key Token Parse2 (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Associated data section</td>
<td></td>
</tr>
<tr>
<td><strong>Type of algorithm for which the key can be used</strong> (one returned)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>Specifies the AES algorithm.</td>
</tr>
<tr>
<td>DES</td>
<td>Specifies the DES algorithm.</td>
</tr>
<tr>
<td>HMAC</td>
<td>Specifies the HMAC algorithm.</td>
</tr>
<tr>
<td><strong>Key type</strong></td>
<td></td>
</tr>
<tr>
<td>Note: Not a keyword. Value returned in <code>key_type</code> variable.</td>
<td></td>
</tr>
<tr>
<td>Key-usage field keywords depend on key type:</td>
<td></td>
</tr>
<tr>
<td><strong>Key type</strong></td>
<td></td>
</tr>
<tr>
<td>CIPHER</td>
<td>Refer to Table 79 on page 270.</td>
</tr>
<tr>
<td>DESUESCV</td>
<td>No key-usage field keywords.</td>
</tr>
<tr>
<td>DKYGENKY</td>
<td>Refer to Table 80 on page 274.</td>
</tr>
<tr>
<td>EXPORTER</td>
<td>Refer to Table 81 on page 278.</td>
</tr>
<tr>
<td>IMPORTER</td>
<td>Refer to Table 82 on page 278.</td>
</tr>
<tr>
<td>MAC</td>
<td>Refer to Table 83 on page 285. For HMAC token algorithm, refer to Table 84 on page 288.</td>
</tr>
<tr>
<td>PINCALC</td>
<td>Refer to Table 85 on page 291.</td>
</tr>
<tr>
<td>PINPROT</td>
<td>Refer to Table 86 on page 294.</td>
</tr>
<tr>
<td>PINPRW</td>
<td>Refer to Table 87 on page 296.</td>
</tr>
<tr>
<td>SECMSG</td>
<td>Refer to Table 88 on page 299.</td>
</tr>
<tr>
<td><strong>Key-management field 1, high order byte</strong></td>
<td></td>
</tr>
<tr>
<td>Symmetric-key export control (one returned, all key types)</td>
<td></td>
</tr>
<tr>
<td>NOEX-SYM</td>
<td>Prohibit export using symmetric key.</td>
</tr>
<tr>
<td>XPRT-SYM</td>
<td>Allow export using symmetric key.</td>
</tr>
<tr>
<td>Unauthenticated asymmetric-key export control (one returned, all key types)</td>
<td></td>
</tr>
<tr>
<td>NOEXUASY</td>
<td>Prohibit export using an unauthenticated asymmetric key.</td>
</tr>
<tr>
<td>XPRTUASY</td>
<td>Allow export using unauthenticated asymmetric key.</td>
</tr>
<tr>
<td>Authenticated asymmetric-key export control (one returned, all key types)</td>
<td></td>
</tr>
<tr>
<td>NOEXAASY</td>
<td>Prohibit export using authenticated asymmetric key.</td>
</tr>
<tr>
<td>XPRTAASY</td>
<td>Allow export using authenticated asymmetric key.</td>
</tr>
<tr>
<td><strong>Key-management field 1, low-order byte</strong></td>
<td></td>
</tr>
<tr>
<td>DES-key export control (one returned, AES algorithm only)</td>
<td></td>
</tr>
<tr>
<td>NOEX-DES</td>
<td>Prohibit export using DES key</td>
</tr>
</tbody>
</table>

Chapter 8. AES, DES, and HMAC cryptographic keys
### Table 92. Keywords for Key Token Parse2 (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPRT-DES</td>
<td>Allow export using DES key.</td>
</tr>
<tr>
<td>NOEX-AES</td>
<td>Prohibit export using AES key.</td>
</tr>
<tr>
<td>XPRT-AES</td>
<td>Allow export using AES key.</td>
</tr>
<tr>
<td>NOEX-RSA</td>
<td>Prohibit export using RSA key.</td>
</tr>
<tr>
<td>XPRT-RSA</td>
<td>Allow export using RSA key.</td>
</tr>
<tr>
<td>NOEX-RAW</td>
<td>Prohibit export using raw key.</td>
</tr>
<tr>
<td>XPRT-RAW</td>
<td>Allow export using raw key.</td>
</tr>
<tr>
<td><strong>Key-management field 2, high order byte</strong></td>
<td></td>
</tr>
<tr>
<td>MIN3PART</td>
<td>Key if present is incomplete. Key requires at least 2 more parts.</td>
</tr>
<tr>
<td>MIN2PART</td>
<td>Key if present is incomplete. Key requires at least 1 more part.</td>
</tr>
<tr>
<td>MIN1PART</td>
<td>Key if present is incomplete. Key can be completed or have more parts added.</td>
</tr>
<tr>
<td>KEYCMPLT</td>
<td>Key if present is complete. No more parts can be added.</td>
</tr>
<tr>
<td><strong>Key-management field 2, low-order byte</strong></td>
<td></td>
</tr>
<tr>
<td>UNTRUSTD</td>
<td>Key was encrypted with an untrusted KEK.</td>
</tr>
<tr>
<td>WOTUATTR</td>
<td>Key was in a format without type/usage attributes.</td>
</tr>
<tr>
<td>WWEAKKEY</td>
<td>Key was encrypted with key weaker than itself.</td>
</tr>
<tr>
<td>NOTCCAFM</td>
<td>Key was in a non-CCA format.</td>
</tr>
<tr>
<td>WECBMODE</td>
<td>Key was encrypted in ECB mode.</td>
</tr>
<tr>
<td><strong>Key-management field 3, high order byte</strong></td>
<td></td>
</tr>
<tr>
<td>POUNKNWN</td>
<td>Unknown.</td>
</tr>
<tr>
<td>POOTHER</td>
<td>Other. Method other than those defined here, probably used in UDX.</td>
</tr>
<tr>
<td>PORANDOM</td>
<td>Randomly generated.</td>
</tr>
<tr>
<td>POKEYAGR</td>
<td>Established by key agreement such as Diffie-Hellman.</td>
</tr>
<tr>
<td>POCLRKC</td>
<td>Created from cleartext key components.</td>
</tr>
<tr>
<td>POCLRKV</td>
<td>Entered as a cleartext key value.</td>
</tr>
<tr>
<td>PDERV</td>
<td>Derived from another key.</td>
</tr>
<tr>
<td>POKPSEC</td>
<td>Cleartext keys or key parts that were entered at TKE and secured from there to the target card.</td>
</tr>
<tr>
<td><strong>Key-management field 3, low-order byte</strong></td>
<td></td>
</tr>
<tr>
<td>POUNKNWN</td>
<td>Unknown.</td>
</tr>
<tr>
<td>PCOTHER</td>
<td>Other. Method other than those defined here, probably used in UDX.</td>
</tr>
<tr>
<td>PCRANDOM</td>
<td>Randomly generated.</td>
</tr>
<tr>
<td>PCKEYAGR</td>
<td>Established by key agreement such as Diffie-Hellman.</td>
</tr>
<tr>
<td>PCCLCOMP</td>
<td>Created from cleartext key components.</td>
</tr>
<tr>
<td>PCCLVAL</td>
<td>Entered as a cleartext key value.</td>
</tr>
</tbody>
</table>
Table 92. Keywords for Key Token Parse2 (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCDERV</td>
<td>Derived from another key.</td>
</tr>
<tr>
<td>PCMVARNP</td>
<td>Imported from CCA version X'05' variable-length symmetric key-token with no pedigree field.</td>
</tr>
<tr>
<td>PCMWCV</td>
<td>Imported from CCA key-token that contained a nonzero control vector.</td>
</tr>
<tr>
<td>PCMNOCV</td>
<td>Imported from CCA key-token that had no control vector or contained a zero control vector.</td>
</tr>
<tr>
<td>PCMT31WC</td>
<td>Imported from a TR-31 key block that contained a control vector (ATTR-CV option).</td>
</tr>
<tr>
<td>PCMT31NC</td>
<td>Imported from a TR-31 key block that did not contain a control vector.</td>
</tr>
<tr>
<td>PCMPK1-2</td>
<td>Imported using PKCS 1.2 RSA encryption.</td>
</tr>
<tr>
<td>PCMOAEP</td>
<td>Imported using PKCS OAEP encryption.</td>
</tr>
<tr>
<td>PCMPKA92</td>
<td>Imported using PKA92 RSA encryption.</td>
</tr>
<tr>
<td>PCMZ-PAD</td>
<td>Imported using RSA ZERO-PAD encryption.</td>
</tr>
<tr>
<td>PCCNVTWC</td>
<td>Converted from a CCA key-token that contained a nonzero control vector.</td>
</tr>
<tr>
<td>PCCNVTNC</td>
<td>Converted from a CCA key-token that had no control vector or contained a zero control vector.</td>
</tr>
<tr>
<td>PCKPSEC</td>
<td>Cleartext keys or key parts that were entered at TKE and secured from there to the target card.</td>
</tr>
<tr>
<td>PCXVARWP</td>
<td>Exported from CCA version X'05' variable-length symmetric key-token with pedigree field.</td>
</tr>
<tr>
<td>PCXVARNP</td>
<td>Exported from CCA version X'05' variable-length symmetric key-token with no pedigree field.</td>
</tr>
<tr>
<td>PCXOAEP</td>
<td>Exported using PKCS OAEP encryption.</td>
</tr>
</tbody>
</table>

Optional clear key or encrypted AESKW payload section

**Payload**

*Note:* Not a keyword. Value returned in the payload variable.

<table>
<thead>
<tr>
<th>key_material_state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction: Output</td>
</tr>
<tr>
<td>Type: Integer</td>
</tr>
<tr>
<td>A pointer to an integer variable containing the indicator for the current state of the key material. The valid values are:</td>
</tr>
<tr>
<td>0 No key present (internal or external)</td>
</tr>
<tr>
<td>1 Key is clear (internal), payload bit length is clear-key bit length</td>
</tr>
<tr>
<td>2 Key is encrypted under a KEK (external)</td>
</tr>
<tr>
<td>3 Key is encrypted under the master key (internal)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>payload_bit_length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction: Input/Output</td>
</tr>
<tr>
<td>Type: Integer</td>
</tr>
<tr>
<td>A pointer to an integer variable containing the number of bits in the token payload. If no key is present, the returned value is 0.</td>
</tr>
<tr>
<td>If a clear key is present, the returned value is in the following range:</td>
</tr>
<tr>
<td>AES 128, 192, or 256</td>
</tr>
<tr>
<td>HMAC 80 - 2048</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction: Output</td>
</tr>
</tbody>
</table>

---

Chapter 8. AES, DES, and HMAC cryptographic keys 317
Key Token Parse2 (CSNBKTP2)

Type: String

A pointer to a string variable containing the key material payload. The `payload` parameter must be addressable up to the nearest byte boundary above the `payload_bit_length` if the `payload_bit_length` is not a multiple of 8. This field will contain the clear key or the encrypted key material.

**key_verification_pattern_type**

**Direction:** Output  
**Type:** Integer

A pointer to an integer variable containing the indicator for the type of key verification pattern used. The valid values are:
- 0: No KVP
- 1: AESMK (8 left-most bytes of SHA-256 hash(X'01' || clear AES MK))
- 2: KEK VP (8 left-most bytes of SHA-256 hash(X'01' || clear KEK))

**key_verification_pattern_length**

**Direction:** Input/Output  
**Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the `key_verification_pattern` parameter. The valid values are 0, 8, or 16. The value 16 is reserved.

**key_verification_pattern**

**Direction:** Output  
**Type:** String

A pointer to a string variable containing the key verification pattern (KVP) of the key-encrypting key used to wrap this key. If the `key_verification_pattern_type` value indicates that a key verification pattern is present, the pattern will be copied from the token, otherwise this variable is empty.

**key_wrapping_method**

**Direction:** Output  
**Type:** Integer

A pointer to an integer variable containing the indicator for the encrypted section key-wrapping method used to protect the key payload. The valid values are:
- 0: NONE (for clear keys or no key)
- 2: AESKW (for external or internal key wrapped with an AES KEK)
- 3: PKOAEP2 (for external tokens wrapped with an RSA public key)

**key_hash_algorithm**

**Direction:** Output  
**Type:** Integer

A pointer to an integer variable containing the indicator for the hash algorithm used for wrapping in the key token. The valid values are as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Hash algorithm</th>
<th>Key-wrapping method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X'00' (clear key)</td>
</tr>
<tr>
<td>0</td>
<td>No hash</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SHA-1</td>
<td></td>
</tr>
</tbody>
</table>
### Key Token Parse2 (CSNBKTP2)

<table>
<thead>
<tr>
<th>Value</th>
<th>Hash algorithm</th>
<th>Key-wrapping method</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>SHA-256</td>
<td>X'00' (clear key)</td>
</tr>
<tr>
<td>4</td>
<td>SHA-384</td>
<td>X'02' (AESKW)</td>
</tr>
<tr>
<td>8</td>
<td>SHA-512</td>
<td>X'03' (PKOAEP2)</td>
</tr>
</tbody>
</table>

**key_name_length**
- **Direction:** Input/Output
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the `key_name` variable. The returned value can be 0 or 64.

**key_name**
- **Direction:** Output
- **Type:** String

A pointer to a string variable containing the optional key label to be stored in the associated data structure of the key token. If there is no key name, then this variable is empty.

**TLV_data_length**
- **Direction:** Input/Output
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the `TLV_data` variable. The returned value is currently always zero.

**TLV_data**
- **Direction:** Output
- **Type:** String

A pointer to a string variable containing the optional tag-length-value (TLV) section. This field is currently unused.

**user_associated_data_length**
- **Direction:** Input/Output
- **Type:** Integer

The `user_associated_data_length` parameter is a pointer to an integer variable containing the number of bytes of data in the `user_associated_data` variable. The returned value is 0 - 255.

**user_associated_data**
- **Direction:** Output
- **Type:** String

A pointer to a string variable containing the user-associated data to be stored in the key token. This user-definable data is cryptographically bound to the key if it is encrypted. If there is no user-defined associated data, this variable is empty.

**verb_data_length**
- **Direction:** Input/Output
Key Token Parse2 (CSNBKTP2)

Type: Integer

A pointer to an integer variable containing the number of bytes of data in the `verb_data` variable. The returned value is zero if the returned `key_type` variable is not DKYGENKY. Otherwise the value can be greater than zero and is a multiple of 8.

**verb_data**

**Direction:** Output

**Type:** String

A pointer to a string variable containing any related key-usage field keywords of an AES DKYGENKY key for the for the type of key to be diversified:

<table>
<thead>
<tr>
<th>DKYGENKY type of key to diversify</th>
<th>Meaning of verb_data keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-CIPHER</td>
<td>Key usage fields for AES CIPHER key.</td>
</tr>
<tr>
<td>D-EXP</td>
<td>Key usage fields for AES EXPORTER key.</td>
</tr>
<tr>
<td>D-IMP</td>
<td>Key usage fields for AES IMPORTER key.</td>
</tr>
<tr>
<td>D-MAC</td>
<td>Key usage fields for AES MAC key.</td>
</tr>
<tr>
<td>D-PPROT</td>
<td>Key-usage fields for AES PINPROT key.</td>
</tr>
<tr>
<td>D-PCALC</td>
<td>Key-usage fields for AES PINCALC key.</td>
</tr>
<tr>
<td>D-PPRW</td>
<td>Key-usage fields for AES PINPRW key.</td>
</tr>
<tr>
<td>D-SECMSG</td>
<td>Key-usage fields for AES SECMSG key.</td>
</tr>
</tbody>
</table>

**Required commands**

The required commands for CSNBKTP2.

None.

**Usage notes**

The usage notes for CSNBKTP2.

If an error occurs while processing the input token, all output lengths are updated to zero and an error is returned.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBKTP2J.

See "Building Java applications using the CCA JNI" on page 28.

**Format**

```java
public native void CSNBKTP2J(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber key_token_length,
    byte[] key_token,
    byte[] key_type,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber key_material_state,
    hikmNativeNumber payload_bit_length,
```
Key Translate (CSNBKTR)

The Key Translate verb uses one key-encrypting key to decipher an input key and then enciphers this key using another key-encrypting key within the secure environment.

Note:

- All key labels must be unique.
- This verb supports PCI-HSM 2016 compliant-tagged key tokens.

Format

The format of CSNBKTR.

```c
CSNBKTR(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    input_key_token,
    input_KEK_key_identifier,
    output_KEK_key_identifier,
    output_key_token)
```

Parameters

The parameters for CSNBKTR.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

**input_key_token**

Direction: Input
Type: String

A 64-byte string variable containing an external key token. The external key token contains the key to be re-enciphered (translated).

**input_KEK_key_identifier**

Direction: Input/Output
Type: String

A 64-byte string variable containing the internal key token or the key label of an internal key token record in the DES key storage file. The internal key token contains the key-encrypting key used to decipher the key. The internal key
Key Translate (CSNBKTR)

token must contain a control vector that specifies an IMPORTER or IKEYXLAT key type. The control vector for an IMPORTER key must have the XLATE bit set to B'1'.

**output KEK_key_identifier**

**Direction:** Input/Output  
**Type:** String

A 64-byte string variable containing the internal key token or the key label of an internal key token record in the DES key storage file. The internal key token contains the key-encrypting key used to encipher the key. The internal key token must contain a control vector that specifies an EXPORTER or OKEYXLAT key type. The control vector for an EXPORTER key must have the XLATE bit set to B'1'.

**output_key_token**

**Direction:** Output  
**Type:** String

A 64-byte string variable containing an external key token. The external key token contains the re-enciphered key.

**Restrictions**

The restrictions for CSNBKTR.

Triple length DATA key tokens are not supported.

**Required commands**

The required commands for CSNBKTR.

This verb requires the **Key Translate** command (offset X'001F') to be enabled in the active role.

**Usage notes**

The usage notes for CSNBKTR.

None.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBKTRJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBKTRJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] input_key_token,
    byte[] input KEK_key_identifier,
    byte[] output KEK_key_identifier,
    byte[] output_key_token);
```
The Key Translate2 verb uses one key-encrypting key to decipher an input key and then enciphers this key using another key-encrypting key within the secure environment.

It can also be used to change the wrapping method of the key with a single key-encrypting key.

To reencipher a key token, specify the external key token, input and output key-encrypting keys. You can specify which key wrapping method to use. If no wrapping method is specified, the wrapping method of the input_key_token will be used.

To change the wrapping method of an external key token, specify the REFORMAT rule array keyword, the wrapping method to use, the external key token, and the input key-encrypting key. If no wrapping method is specified, the wrapping method of the input_key_token will be used. Note that the output KEK_identifier will be ignored.

Note:
- All key labels must be unique.
- This verb supports PCI-HSM 2016 compliant-tagged key tokens.

**Format**

The format of CSNBKTR2.

```
CSNBKTR2( return_code, reason_code, exit_data_length, exit_data, rule_array_count, rule_array, input_key_token_length, input_key_token, input_KEY_key_identifier_length, input_KEY_key_identifier, output_KEY_key_identifier_length, output_KEY_key_identifier, output_key_token_length, output_key_token )
```

**Parameters**

The parameters for CSNBKTR2.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see the Parameters common to all verbs on page 22.

**rule_array_count**

Direction: Input  
Type: Integer  

The number of keywords you supplied in the rule_array parameter. This value must be 0, 1, 2, or 3.

**rule_array**
Key Translate2 (CSNBKTR2)

Direction:  Input
Type:  String array

Keywords that provide control information to the verb. The keywords must be 8 bytes of contiguous storage with the keyword left-aligned in its 8-byte location and padded on the right with blanks. The `rule_array` keywords are described in Table 93.

Table 93. Keywords for Key Translate2 control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Encipherment</strong></td>
<td><em>(Optional)</em></td>
</tr>
<tr>
<td>COMP-CHK</td>
<td>Check if the key token to be translated or reformatted can have the PCI-HSM 2016 compliance tag.</td>
</tr>
<tr>
<td>COMP-TAG</td>
<td>Convert the input key token into a PCI-HSM 2016 compliant-tagged token. This requires that the domain at first is in full PCI-HSM 2016 compliance mode and from there enters into the migration mode, which is a temporary reduced mode of an active PCI-HSM 2016 mode (see also “Migration mode” on page 1090).</td>
</tr>
<tr>
<td><strong>REFORMAT</strong></td>
<td>Reformat the <code>input_key_token</code>.</td>
</tr>
<tr>
<td></td>
<td>- When the <code>input_key_token</code> is a DES key token, reformat with the Key Wrapping Method specified.</td>
</tr>
<tr>
<td></td>
<td>- When the <code>input_key_token</code> is an operational AES key token, either reformat an AES DATA key (version X'04') to an AES CIPHER key (version X'05') or the reverse (version X'05' to version X'04').</td>
</tr>
<tr>
<td><strong>TRANSLAT</strong></td>
<td>Translate the <code>input_key_token</code> from encipherment under the <code>input_KEYK_IDENTIFIER</code> to encipherment under the <code>output_KEYK_IDENTIFIER</code>. This is the default.</td>
</tr>
<tr>
<td>V1PYLD</td>
<td>Reencipher an input variable-length AES key token (version X'05') to a payload version1 (fixed-length) key token. This keyword is only valid for the CIPHER, EXPORTER and IMPORTER key types.</td>
</tr>
<tr>
<td>V0PYLD</td>
<td>Reencipher an input variable-length AES key token (version X'05') to a payload version 0 (variable-length) key token. This keyword is only valid for the CIPHER, EXPORTER and IMPORTER key types.</td>
</tr>
<tr>
<td><strong>Key-wrapping method</strong></td>
<td><em>(optional, valid only if input_key_token is an external DES key token)</em></td>
</tr>
<tr>
<td>USECONFIG</td>
<td>This is the default. Specifies to wrap the key using the configuration setting for the default wrapping method. The default wrapping method configuration setting may be changed using the TKE. This keyword is ignored for AES keys.</td>
</tr>
<tr>
<td>WRAP-ENH</td>
<td>Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.</td>
</tr>
<tr>
<td>WRAP-ECB</td>
<td>Use original key wrapping method, which uses ECB wrapping for DES key tokens.</td>
</tr>
<tr>
<td><strong>Translation control</strong></td>
<td><em>(Optional, valid only with WRAP-ENH)</em></td>
</tr>
<tr>
<td>ENH-ONLY</td>
<td>Restrict the re-wrapping of the <code>output_key_token</code>. Once the token has been wrapped with the enhanced method, it cannot be re-wrapped using the original method.</td>
</tr>
<tr>
<td><strong>Algorithm</strong></td>
<td><em>(One required, if the V0PYLD or V1PYLD keyword is specified)</em></td>
</tr>
<tr>
<td>AES</td>
<td>Specifies that the input key is an AES key. Where used, the key-encrypting keys will be AES transport keys.</td>
</tr>
<tr>
<td>DES</td>
<td>Specifies that the input key is a DES key. Where used, the key-encrypting keys will be DES transport keys. This is the default.</td>
</tr>
<tr>
<td>HMAC</td>
<td>Specifies that the input key is an HMAC key. Where used, the key-encrypting keys will be AES transport keys.</td>
</tr>
</tbody>
</table>

`input_key_token_length`

Direction:  Input

324  Common Cryptographic Architecture Application Programmer's Guide
Key Translate2 (CSNBKTR2)

Type: Integer

The length of the input_key_token in bytes. The maximum value allowed is 725.

input_key_token

Direction: Input
Type: String

A variable length string variable containing the external key token. The external key token contains the key to be re-enciphered (or re-wrapped).

input_KEK_key_identifier_length

Direction: Input
Type: Integer

The length of the input_KEK_key_identifier in bytes. The maximum value allowed is 725.

input_KEK_key_identifier

Direction: Input/Output
Type: String

A variable length string variable containing the internal key token or the key label of an internal key token record in the key storage file. The internal key token contains the key-encrypting key used to decipher the key. The internal key token must contain a control vector that specifies an IMPORTER or IKEYXLAT key type. The control vector for an IMPORTER key must have the XLATE bit set to B'1'.

output_KEK_key_identifier_length

Direction: Input
Type: Integer

The length of the output_KEK_key_identifier in bytes. The maximum value is 725.

If the REFORMAT keyword is specified, this value must be 0.

output_KEK_key_identifier

Direction: Input/Output
Type: String

A variable length string variable containing the internal key token or the key label of an internal key token record in the key storage file. The internal key token contains the key-encrypting key used to encipher the key. The internal key token must contain a control vector that specifies an EXPORTER or OKEYXLAT key type. The control vector for an exporter key must have the XLATE bit set to B'1'.

If the REFORMAT keyword is specified, this parameter is ignored.

output_key_token_length

Direction: Input/Output
Type: Integer

On input, the length of the output area provided for the output_key_token. This must be at least 64 bytes. On output, the parameter is updated with the length of the token copied to the output_key_token.
Key Translate2 (CSNBKTR2)

output_key_token

Direction: Input/Output
Type: String

A variable length string variable containing an external key token. The external key token contains the re-enciphered key.

Restrictions

The restrictions for CSNBKTR2.

This verb does not support version X'10' external DES key tokens (RKX key tokens).

This verb was introduced with CCA 4.1.0.

Required commands

The required commands for CSNBKTR2.

This verb requires the **Key Translate2 - Allow use of REFORMAT** command (offset X'014B') to be enabled in the active role if the **REFORMAT** reencipherment keyword is used.

Otherwise, the verb requires the **Key Translate2** command (offset X'0149') to be enabled.

To use the translation control keyword WRAP-ECB or WRAP-ENH when the default key-wrapping method setting does not match the keyword, the **Key Translate2 - Allow wrapping override keywords** command (offset X'014A') must be enabled.

If the WRAP-ECB translation-control keyword is specified and the key in the input key token is wrapped by the enhanced wrapping method (WRAP-ENH), the verb requires the **CKDS Conversion2 - Convert from enhanced to original** command (offset X'0147') to be enabled. An active role with offset X'0149' enabled can also use the Key Token Change verb to translate a key from the enhanced key-wrapping method to the less-secure legacy method.

The **Key Translate2 - Disallow AES ver 5 to ver 4 conversion** command (offset X'032A') prevents CIPHER keys, which are in variable-length AES key tokens (newer version X'05') and wrapped under the AES master-key, from being reformatted into DATA keys, which are in fixed-length AES key tokens (older version X'04') and wrapped under the less-secure DES master-key. This command overrides the **Key Translate2 - Allow use of REFORMAT** command (offset X'014B').

Usage notes

The usage notes for CSNBKTR2.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBKTR2J.

See “Building Java applications using the CCA JNI” on page 28.
PKA Decrypt (CSNDPKD)

Use this verb to decrypt (unwrap) a formatted key value. This verb unwraps the key, parses it, and returns the parsed value to the application in the clear.

PKCS 1.2 and ZERO-PAD formatting are supported. For PKCS 1.2, the decrypted data is examined to ensure that it meets RSA DSI PKCS #1 block type 2 format specifications. ZERO-PAD is supported only for external or clear RSA private keys.

For the PKCOSOAEP recovery method keyword, the decrypted data is examined to ensure that it meets the RSAES-OAEP scheme of the RSA PKCS #1 v2.0 standard. See also “PKCS #1 hash formats” on page 1044.

This verb allows the use of clear or encrypted RSA private keys. If an external clear key token is used, the master keys are not required to be installed in any cryptographic coprocessor and PKA verbs do not have to be enabled.

Format

The format of CSNDPKD.

Parameters

The parameters for CSNDPKD.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.
PKA Decrypt (CSNDPKD)

rule_array_count

Direction:    Input
Type:        Integer

A pointer to an integer variable containing the number of elements in the
rule_array variable. This value must be 1 or 2.

rule_array

Direction:    Input
Type:        String array

The keyword that provides control information to the verb. The keyword is
left-aligned in an 8-byte field and padded on the right with blanks. The
rule_array keywords are described in [Table 94].

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recovery method</td>
<td>Specifies the method to use to recover the key value.</td>
</tr>
<tr>
<td>PKCS-1.2</td>
<td>Specifies that the key is formatted as defined in the RSA PKCS #1 v2.0 standard for the RSAES-PKCS1-v1_5 encryption/decryption scheme. Formerly known as the block-type 02 method. See &quot;PKCS #1 hash formats&quot; on page 1044.</td>
</tr>
<tr>
<td>PKCSOAEP</td>
<td>Specifies that the key is formatted as defined in the RSA PKCS #1 v2.0 standard for the RSAES-OAEP encryption/decryption scheme. See &quot;PKCS #1 hash formats&quot; on page 1044.</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>The input PKA_enciphered_keyvalue is decrypted using the RSA private key. The entire result (including leading zeros) is returned in the target_keyvalue field. The PKA_key_identifier must be an external RSA token or the label of an external token.</td>
</tr>
<tr>
<td>Hash method</td>
<td>(One required for PKCSOAEP, not allowed for any other recovery method).</td>
</tr>
<tr>
<td>SHA-1</td>
<td>Specifies to use the SHA-1 hash method to calculate the OAEP message hash.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Specifies to use the SHA-256 hash method to calculate the OAEP message hash.</td>
</tr>
</tbody>
</table>

PKA_enciphered_keyvalue_length

Direction:    Input
Type:        Integer

The length of the PKA_enciphered_keyvalue parameter in bytes. The maximum size that you can specify is 256 bytes. The length should be the same as the modulus length of the PKA_key_identifier.

PKA_enciphered_keyvalue

Direction:    Input
Type:        String

This field contains the key value protected under an RSA public key. This byte-length string is left-aligned within the PKA_enciphered_keyvalue parameter.

data_structure_length

Direction:    Input
Type:        Integer

This value must be 0.

data_structure

Direction:    Input
Type: String

This parameter is ignored.

**PKA_key_identifier_length**

**Direction:** Input  
**Type:** Integer

The length of the *PKA_key_identifier* parameter. When the *PKA_key_identifier* is a key label, this field specifies the length of the label. The maximum size that you can specify is 2500 bytes.

**PKA_key_identifier**

**Direction:** Input  
**Type:** String

An internal RSA private key token, the label of an internal RSA private key token, or an external RSA private key token containing a clear RSA private key in Modulus-Exponent or Chinese Remainder Theorem format. The corresponding public key was used to wrap the key value.

**target_keyvalue_length**

**Direction:** Input/Output  
**Type:** Integer

The length of the *target_keyvalue* parameter. The maximum size that you can specify is 256 bytes. On return, this field is updated with the actual length of *target_keyvalue*.

If **ZERO-PAD** is specified, this length will be the same as the *PKA_enciphered_keyvalue_length* which is equal to the RSA modulus byte length.

**target_keyvalue**

**Direction:** Output  
**Type:** String

This field will contain the decrypted, parsed key value. If **ZERO-PAD** is specified, the decrypted key value, including leading zeros, will be returned.

**Restrictions**

The restrictions for CSNDPKD.

- The exponent of the RSA public key must be odd.

- Rule array keywords PKCSOAEP, SHA-1, and SHA-256 are not supported in releases before Release 4.4.

- The **PKA Decipher - Key Data Disallow PKCS-1.2** command (offset X'020A'), the **PKA Decipher - Key Data Disallow ZERO-PAD** command (offset X'020B'), and the **PKA Decipher - Key Data Disallow PKCSOAEP** command (offset X'020C') are not defined in releases before Release 4.4.

**Required commands**

The required commands for CSNDPKD.

This verb requires the **PKA Decrypt** command (offset X'011F') to be enabled in the active role.
The PKA Decrypt verb also requires the following commands to be enabled in the active role to disallow the unwrapping of an encrypted key that has been formatted:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKCS-1.2</td>
<td>X'020A'</td>
<td>PKA Decipher - Key Data Disallow PKCS-1.2</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>X'020B'</td>
<td>PKA Decipher - Key Data Disallow ZERO-PAD</td>
</tr>
<tr>
<td>PKCSOAEP</td>
<td>X'020C'</td>
<td>PKA Decipher - Key Data Disallow PKCSOAEP</td>
</tr>
</tbody>
</table>

**Usage notes**

The usage notes for CSNDPKD.

The RSA private key must be enabled for key management functions.

The hardware configuration sets the limit on the modulus size of keys for key management; thus, this verb will fail if the RSA key modulus bit length exceeds this limit.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNDPKDJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
class CSNDPKDJ{
    public native void CSNDPKDJ(
        hikmNativeNumber return_code,
        hikmNativeNumber reason_code,
        hikmNativeNumber exit_data_length,
        byte[] exit_data,
        hikmNativeNumber rule_array_count,
        byte[] rule_array,
        hikmNativeNumber PKA_enciphered_keyvalue_length,
        byte[] PKA_enciphered_keyvalue,
        hikmNativeNumber data_structure_length,
        byte[] data_structure,
        hikmNativeNumber PKA_key_identifier_length,
        byte[] PKA_key_identifier,
        hikmNativeNumber target_keyvalue_length,
        byte[] target_keyvalue);
}
```

**PKA Encrypt (CSNDPKE)**

This verb encrypts a supplied clear key value under an RSA public key.

The supplied key can be formatted using the PKCS 1.2 or ZERO-PAD methods prior to encryption. Beginning with Release 4.4, the supplied key can also be formatted using the PKCSOAEP method. The rule_array keyword specifies the format of the key prior to encryption.
Format

The format of CSNDPKE.

```
CSNDPKE(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    keyvalue_length,
    keyvalue,
    data_structure_length,
    data_structure,
    PKA_key_identifier_length,
    PKA_key_identifier,
    PKA_enciphered_keyvalue_length,
    PKA_enciphered_keyvalue)
```

Parameters

The parameters for CSNDPKE.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1 or 2.

**rule_array**

- **Direction:** Input
- **Type:** String array

A keyword that provides control information to the verb. The keyword is left-aligned in an 8-byte field and padded on the right with blanks. The `rule_array` keywords are described in Table 95.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formatting Method (One, required). Specifies the method to use to format the key value prior to encryption.</td>
<td></td>
</tr>
<tr>
<td>MRP</td>
<td>The key value will be padded on the left with binary zeros to the length of the PKA key modulus. The RSA public key can have an even or odd exponent.</td>
</tr>
<tr>
<td>PKCS-1.2</td>
<td>RSA DSI PKCS #1 block type 02 format will be used to format the supplied key value. In the RSA PKCS #1 v2.0 standard, RSA terminology describes this as the RSAES-PKCS1-v1_5 format.</td>
</tr>
<tr>
<td>PKCSOAEP</td>
<td>Specifies that the key is formatted as defined in the RSA PKCS #1 v2.0 standard for the RSAES-OAEP encryption/decryption scheme. See “PKCS #1 hash formats” on page 1044.</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>Places the clear_source_data variable in the low-order bit positions of a bit string of the same length as the modulus. The data is padded on the left as needed with binary zeros to the length of the PKA key modulus. The RSA public key must have an odd exponent.</td>
</tr>
<tr>
<td>Hash method (One required for PKCSOAEP, not allowed for any other recovery method).</td>
<td></td>
</tr>
<tr>
<td>SHA-1</td>
<td>Specifies to use the SHA-1 hash method to calculate the OAEP message hash.</td>
</tr>
</tbody>
</table>
Table 95. Keywords for PKA Encrypt control information  (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-256</td>
<td>Specifies to use the SHA-256 hash method to calculate the OAEP message hash.</td>
</tr>
</tbody>
</table>

**keyvalue_length**

Direction: Input
Type: Integer

The length of the *keyvalue* parameter. The maximum field size is 256 bytes. The actual maximum size depends on the modulus length of *PKA_key_identifier* and the formatting method you specify in the *rule_array* parameter. See "Usage notes" on page 333.

**keyvalue**

Direction: Input
Type: String

This field contains the supplied clear key value to be encrypted under the *PKA_key_identifier*.

**data_structure_length**

Direction: Input
Type: Integer

This value must be 0.

**data_structure**

Direction: Input
Type: String

This field is currently ignored.

**PKA_key_identifier_length**

Direction: Input
Type: Integer

The length of the *PKA_key_identifier* parameter. When the *PKA_key_identifier* is a key label, this field specifies the length of the label. The maximum size that you can specify is 2500 bytes.

**PKA_key_identifier**

Direction: Input
Type: String

The RSA public or private key token or the label of the RSA public or private key to be used to encrypt the supplied key value.

**PKA_enciphered_keyvalue_length**

Direction: Input/Output
Type: Integer

The length of the *PKA_enciphered_keyvalue* parameter in bytes. The maximum size that you can specify is 256 bytes. On return, this field is updated with the actual length of *PKA_enciphered_keyvalue*.
This length should be the same as the modulus length of the
PKA_key_identifier.

PKA_enciphered_keyvalue

Direction: Output
Type: String

This field contains the key value protected under an RSA public key. This
byte-length string is left-aligned within the PKA_enciphered_keyvalue parameter.

Restrictions

The restrictions for CSNDPKE.

- A message can be encrypted provided that it is smaller than the public key
  modulus.
  
  The term 'smaller' refers to the exact bit count, not the byte count of the
  modulus. For example, counting bits, the hexadecimal number 'X'FF' is several
  bits longer than the number 'X'1F', even though both numbers are one byte long
  as represented in computer memory.

- The exponent of the RSA public key must be odd unless the MRP keyword is
  supplied.

- The RSA public key modulus size (key size) is limited by the Function Control
  Vector to accommodate governmental export and import regulations.

- Rule array keywords PKCSOAEP, SHA-1, and SHA-256 are not supported in
  releases before Release 4.4.

- The PKA Encipher - Clear Key Disallow MRP (offset X'0208'), PKA Encipher -
  Clear Key Disallow PKCS-1.2 (offset X'0206'), PKA Encipher - Clear Key
  Disallow PKCSOAEP (offset X'0209'), and PKA Encipher - Clear Key Disallow
  ZERO-PAD (offset X'0207') are not defined in releases before Release 4.4.

Required commands

The required commands for CSNDPKE.

This verb requires the PKA Encrypt command (offset X'011E') to be enabled in the
active role.

The PKA Encrypt verb also requires the following commands to be enabled in the
active role to disallow the encryption of clear source data:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRP</td>
<td>X'0208'</td>
<td>PKA Encipher - Clear Key Disallow MRP</td>
</tr>
<tr>
<td>PKCS-1.2</td>
<td>X'0206'</td>
<td>PKA Encipher - Clear Key Disallow PKCS-1.2</td>
</tr>
<tr>
<td>PKCSOAEP</td>
<td>X'0209'</td>
<td>PKA Encipher - Clear Key Disallow PKCSOAEP</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>X'0207'</td>
<td>PKA Encipher - Clear Key Disallow ZERO-PAD</td>
</tr>
</tbody>
</table>

Usage notes

The usage notes for CSNDPKE.

- For RSA DSI PKCS #1 formatting, the key value length must be a minimum of
  11 bytes less than the modulus length of the RSA key.

- The hardware configuration sets the limit on the modulus size of keys for key
  management; thus, this service will fail if the RSA key modulus bit length
  exceeds this limit.
PKA Encrypt (CSNDPKE)

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNDPKEJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNDPKEJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber key_value_length,
    byte[] key_value,
    hikmNativeNumber data_struct_length,
    byte[] data_struct,
    hikmNativeNumber PKA_key_identifier_length,
    byte[] PKA_key_identifier,
    hikmNativeNumber PKA_enciphered_keyvalue_length,
    byte[] PKA_enciphered_keyvalue);
```

Prohibit Export (CSNBPEX)

Use this verb to modify an operational key so that it cannot be exported.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.

Format

The format of CSNBPEX.

```plaintext
CSNBPEX(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier)
```

Parameters

The parameters for CSNBPEX.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

`key_identifier`

**Direction:** Input/Output  
**Type:** String

A 64-byte string variable containing the internal key token to be modified. The returned `key_identifier` will be encrypted under the current master key.

Restrictions

The restrictions for CSNBPEX.

None.
Required commands

The required commands for CSNBPEX.

This verb requires the Prohibit Export command (offset X'00CD') to be enabled in the active role.

The following access-control points are added beginning with Release 4.3:

- To disable the wrapping of a key with a weaker master key, the Prohibit weak wrapping - Master keys command (offset X'0333') must be enabled in the active role.
- To receive a warning when wrapping a key with a weaker master key, enable the Warn when weak wrap - Master keys command (offset X'0332') in the active role. The Prohibit weak wrapping - Master keys command overrides this command.

Usage notes

The usage notes for CSNBPEX.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBPEXJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBPEXJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] key_identifier);
```

Prohibit Export Extended (CSNBPEXX)

Use this verb to modify an exportable external CCA DES key-token so that its key can no longer be exported.

This verb performs the following functions:

- Multiply deciphers the source key under a key formed by the XOR of the source key’s control vector and the specified key-encrypting key (KEK).
- Turns from on to off the XPORT-OK bit in the source key’s control vector (bit 17).
- Multiply enciphers the key under a key formed by the XOR of the KEK key and the source key’s modified control vector. The encrypted key and the modified control vector are stored in the source-key key token, and the TVV is updated.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.
Prohibit Export Extended (CSNBPEXX)

Format
The format of CSNBPEXX.

```
CSNBPEXX(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    source_key_token,
    KEK_key_identifier)
```

Parameters
The parameters for CSNBPEXX.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

`source_key_token`
Direction: Input/Output
Type: String

A pointer to a string variable containing an external key-token.

`KEK_key_identifier`
Direction: Input
Type: String

A pointer to a string variable containing an internal key-encrypting token, or the key label of an internal key-encrypting token record.

Restrictions
The restrictions for CSNBPEXX.

This verb does not support version X'10' external DES key tokens (RKX key tokens).

Required commands
The required commands for CSNBPEXX.

This verb requires the Prohibit Export Extended command (offset X'0301') to be enabled in the active role.

Usage notes
The usage notes for CSNBPEXX.

None.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNBPEXXJ.

See “Building Java applications using the CCA JNI” on page 28.
Format

```java
public native void CSNBPXXJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] source_key_token,
    byte[] KEK_key_identifier);
```

Restrict Key Attribute (CSNBRKA)

Use the Restrict Key Attribute verb to modify an exportable internal or external variable-length symmetric key-token so that its key can no longer be exported.

**Note:** This verb supports PCI-HSM 2016 compliant-tagged key tokens.

Format

The format of CSNBRKA.

```
CSNBRKA (  
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array
    key_identifier_length
    key_identifier
    key_encrypting_key_identifier_length
    key_encrypting_key_identifier
    opt_parameter1_length
    opt_parameter1
    opt_parameter2_length
    opt_parameter2)
```

Parameters

The parameters for CSNBRKA.

For the definitions of the `return_code, reason_code, exit_data_length,` and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

`rule_array_count`

- **Direction:** Input
- **Type:** Integer

The number of keywords you supplied in the `rule_array` parameter. This value must be 1 - 10.

`rule_array`

- **Direction:** Input
- **Type:** String array

The `rule_array` contains keywords that provide control information to the verb. The keywords must be in contiguous storage with each of the keywords left-aligned in its own 8-byte location and padded on the right with blanks. The `rule_array` keywords are described in Table 96 on page 338.
# Restrict Key Attribute (CSNBRKA)

Table 96. Keywords for Restrict Key Attribute control information

<table>
<thead>
<tr>
<th>Token algorithm (one required)</th>
<th>AES</th>
<th>DES</th>
<th>HMAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies to further restrict one or more attributes of a variable-length AES key-token.</td>
<td>Specifies to further restrict one or more attributes of a fixed-length AES key-token.</td>
<td>Specifies to further restrict one or more attributes of a variable-length HMAC key-token.</td>
<td></td>
</tr>
</tbody>
</table>

### Keywords for AES or HMAC variable-length key tokens (key token for clear key not allowed)

**General attribute restriction** (one, optional). Ignored if attribute restriction keyword is specified.

- **NOEXPORT** Prohibits the key from being exported by any verb. This keyword is equivalent to providing all of the export control attribute restrictions keywords (NOEX-AES, NOEX-DES, NOEX-RSA, NOEX-RAW, NOEXSYM, NOEXAASY, and NOEXUASY). This is the default when no attribute restrictions keywords (export control and key usage) are specified.

**Export control attribute restrictions** (one or more, optional).

- **NOEX-AES** Prohibits the key from being exported using an AES key.
- **NOEX-DES** Prohibits the key from being exported using a DES key.
- **NOEX-RSA** Prohibits the key from being exported using an RSA key.
- **NOEX-RAW** Prohibits the key from being exported using a RAW key. Defined for future use. Currently ignored.
- **NOEX-SYM** Prohibits the key from being exported using a symmetric key.
- **NOEXAASY** Prohibits the key form being exported using an authenticated asymmetric key (for example, an RSA key in a trusted block).
- **NOEXUASY** Prohibits the key from being exported using an unauthenticated asymmetric key.

**Ciphertext translation restriction** (one, optional). Only valid for AES CIPHER keys.

- **C-XLATE** Restricts the key to being used by the Cipher_Text_Translate2 (CSNBCTT2) verb.

**KEK identifier rule** (one, optional). Not allowed if KEK_identifier_length variable is 0. Required if KEK identifier is a key label and the key in key storage is an RSA KEK. Release 4.2 or later.

- **IKEK-AES** The inbound key-encrypting key for the external key is an AES KEK. This is the default.
- **IKEK-PKA** The inbound key-encrypting key for the external key is an asymmetric key. Required if an RSA transport key is used (that is, the key being changed is wrapped using the PKOAEP2 method).

### Keywords for fixed-length key tokens

**General attribute restriction** (one, optional). Ignored if attribute restriction or key restriction keyword is specified.

- **NOEXPORT** Prohibits the key from being exported by any verb. Use this keyword to set XPORT-OK off (CV bit 17 = B'0') and NOT31XPT (CV bit 27 = B'1'). This is the default if no attribute restriction or key restriction keywords are specified.

**Attribute to restrict** (one, optional). Keywords **CCAXPORT** and **NOT31XPT** in this group are defaults if no keyword is provided. Only valid for DES. Make multiple calls to this verb as needed to restrict more than one of these attributes.

- **CCAXPORT** Prohibits the key from being exported by any verb. Use this keyword to set XPORT-OK off (CV bit 17 = B'0').
- **DOUBLE-O** For double-length DES key tokens that do not have equal key halves (ignoring parity bits), sets CV bit 40 = B'1' to guarantee that the two 8-byte key values of a DES double-length key are unique, that is, the key halves are not replicated. Key type must be EXPORTER, IKEYXLAT, IMPORTER, or OKEYXLAT.

**Note:** A double-length key with replicated key halves has an effective strength of a single-length key.
| NOT31XPT (Release 4.2 or later) | Sets export control CV to NOT31XPT (bit 57 = B'1') to prohibit TR-31 export of the key. |
| KEK identifier rule (one, optional). Not allowed if KEK_identifier_length variable is 0. Release 4.2 or later. |
| IKEK-DES | The inbound key-encrypting key for the external key is a DES KEK. This is the default. |

### Table 96. Keywords for Restrict Key Attribute control information (continued)

#### key_identifier_length
- **Direction:** Input
- **Type:** Integer

The length of the key_identifier parameter in bytes. The maximum value is 725.

#### key_identifier
- **Direction:** Input
- **Type:** String

The key for which the export control is to be updated. The parameter contains an internal token or the 64-byte label of the key in key storage. If a label is specified, the key token will be updated in key storage and not returned by this verb.

#### key_encrypting_key_identifier_length
- **Direction:** Input
- **Type:** Integer

The byte length of the key_encrypting_key_identifier parameter. This value must be 0.

#### key_encrypting_key_identifier
- **Direction:** Input
- **Type:** String

This parameter is ignored.

#### opt_parameter1_length
- **Direction:** Input
- **Type:** Integer

The byte length of the opt_parameter1 parameter. This value must be 0.

#### opt_parameter1
- **Direction:** Input
- **Type:** String

This parameter is ignored.

#### opt_parameter2_length
- **Direction:** Input
- **Type:** Integer

The byte length of the opt_parameter2 parameter. This value must be 0.
Restrict Key Attribute (CSNBRKA)

**opt_parameter2**

<table>
<thead>
<tr>
<th>Direction</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>

This parameter is ignored.

**Restrictions**

The restrictions for CSNBRKA.

This verb was introduced with CCA 4.1.0.

**Required commands**

The required commands for CSNBRKA.

The Restrict Key Attribute verb requires the following commands to be enabled in the active role:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES or HMAC</td>
<td>X'00E9'</td>
<td>Restrict Key Attribute - Export Control</td>
</tr>
<tr>
<td>DES</td>
<td>X'0154'</td>
<td>Restrict Key Attribute - Permit setting the TR-31 export bit</td>
</tr>
</tbody>
</table>

The following access-control points are added beginning with Release 4.3:

- To disable the wrapping of a key with a weaker master key, the **Prohibit weak wrapping - Master keys** command (offset X'0333') must be enabled in the active role.
- To receive a warning when wrapping a key with a weaker master key, enable the **Warn when weak wrap - Master keys** command (offset X'0332') in the active role. The **Prohibit weak wrapping - Master keys** command overrides this command.

**Usage notes**

The usage notes for CSNBPEXX.

This verb is available starting with CCA 4.1.0.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBRKAJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBRKAJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber key_identifier_length,
    byte[] key_identifier,
    hikmNativeNumber key_encrypting_key_identifier_length,
    byte[] key_encrypting_key_identifier,
```
Random Number Generate (CSNBRNG)

This verb uses the cryptographic feature to generate a cryptographic-quality random number.

Format

The format of CSNBRNG.

```
CSNBRNG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    form,
    random_number)
```

Parameters

The parameters for CSNBRNG.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

**form**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>String</td>
</tr>
</tbody>
</table>

The 8-byte keyword that defines the characteristics of the random number should be left-aligned and padded on the right with blanks. The keywords are listed in Table 97.

**Table 97. Keywords for Random Number Generate form parameter**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVEN</td>
<td>Generate a 64-bit random number with even parity in each byte.</td>
</tr>
<tr>
<td>ODD</td>
<td>Generate a 64-bit random number with odd parity in each byte.</td>
</tr>
<tr>
<td>RANDOM</td>
<td>Generate a 64-bit random number.</td>
</tr>
</tbody>
</table>

Parity is calculated on the seven high-order bits in each byte and is presented in the low-order bit in the byte.

**random_number**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>String</td>
</tr>
</tbody>
</table>

The generated number returned by the verb in an 8-byte variable.

Restrictions

The restrictions for CSNBRNG.

None.
Random Number Generate (CSNBRNG)

**Required commands**
The required commands for CSNBRNG.

This verb requires the **Key Generate - OP** command (offset X'008E') to be enabled in the active role.

**Usage notes**
The usage notes for CSNBRNG.

None.

**JNI version**
This verb has a Java Native Interface (JNI) version, which is named CSNBRNGJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBRNGJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] form,
    byte[] random_number);
```

Random Number Generate Long (CSNBRNGL)

This verb uses the cryptographic feature to generate a cryptographic-quality random number from 1 - 8192 bytes in length.

Choose the parity of each generated random byte as even, odd, or random. This verb returns the random number in a string variable.

Because this verb uses cryptographic processes, the quality of the output is better than that which higher-level language compilers typically supply.

**Format**
The format of CSNBRNGL.

```java
public native void CSNBRNGL(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    seed_length,
    seed,
    random_number_length,
    random_number);
```

**Parameters**
The parameters for CSNBRNGL.
Random Number Generate Long (CSNBRNGL)

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 1.

**rule_array**

- **Direction:** Input
- **Type:** String array

A pointer to a string variable containing an array of keywords. The keywords are eight bytes in length, and must be left-aligned and padded on the right with space characters. The rule_array keywords are described in Table 98.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parity adjust (One required)</td>
<td></td>
</tr>
<tr>
<td>EVEN</td>
<td>Specifies that each generated random byte is adjusted for even parity.</td>
</tr>
<tr>
<td>ODD</td>
<td>Specifies that each generated random byte is adjusted for odd parity.</td>
</tr>
<tr>
<td>RANDOM</td>
<td>Specifies that each generated random byte is not adjusted for parity.</td>
</tr>
</tbody>
</table>

**seed_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes in the seed variable. This value must be 0.

**seed**

- **Direction:** Input
- **Type:** String

This parameter is ignored.

**random_number_length**

- **Direction:** Input/Output
- **Type:** Integer

A pointer to an integer variable containing the number of bytes in the random_number variable. On input, the minimum value is 1 and the maximum value is 8192.

Use this variable to specify the number of random bytes that the verb is to return. On output, this variable contains the number of bytes returned by the verb in the random_number variable.

**random_number**

- **Direction:** Output
- **Type:** String

A pointer to a string variable containing the random number generated.
Random Number Generate Long (CSNBRNGL)

**Restrictions**

The restrictions for CSNBRNGL.

None.

**Required commands**

The required commands for CSNBRNGL.

None.

**Usage notes**

The usage notes for CSNBRNGL.

None.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBRNGLJ.

See [“Building Java applications using the CCA JNI” on page 28](#).

**Format**

```java
public native void CSNBRNGLJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber seed_length,
    byte[] seed,
    hikmNativeNumber random_number_length,
    byte[] random_number);
```

Symmetric Key Export (CSNDSYX)

Use the Symmetric Key Export verb to transfer an application-supplied AES DATA (version X'04'), DES DATA, or variable-length symmetric key token key from encryption under the AES or DES master key to encryption under an application-supplied RSA public key or AES EXPORTER key.

The application-supplied key must be an AES, DES or HMAC internal key token or the label of an AES or DES key token in the AES or DES key storage file.

The verb can also export an HMAC key that is contained in an internal variable-length symmetric key-token. The exported key is returned in an external variable-length symmetric key-token.

Use the Symmetric Key Import verb to import a key exported using the AES or DES algorithm, and the Symmetric Key Import2 verb to import a key exported using the HMAC algorithm.

Different methods are supported for formatting the output key. Not all of these methods are available for each supported source key-token. The AESKW key-formatting method uses an AES EXPORTER key-encrypting key to wrap the output key before returning it in an external variable-length symmetric key-token.
The other key formatting methods each use a different scheme to format the key before it is enciphered using an asymmetric RSA public-key. The formatted and enciphered key is returned as an opaque data buffer, and is not in a key token.

**Format**

The format of CSNDSYX.

```plaintext
CSNDSYX(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    source_key_identifier_length,
    source_key_identifier,
    transporter_key_identifier_length,
    transporter_key_identifier,
    enciphered_key_length,
    enciphered_key)
```

**Parameters**

The parameters for CSNDSYX.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see [“Parameters common to all verbs” on page 22](#).

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1, 2, or 3.

**rule_array**

- **Direction:** Input
- **Type:** String array

Keywords that provide control information to the verb. Each keyword is left-aligned in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage. The `rule_array` keywords are described in Table 99.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithm</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>Export an AES key. If <code>source_key_identifier</code> is a variable-length symmetric key token or label, only the <strong>PKOAEP2</strong> and <strong>AESKW</strong> key formatting methods are supported.</td>
</tr>
<tr>
<td>DES</td>
<td>Export a DES key. This is the default.</td>
</tr>
<tr>
<td>HMAC</td>
<td>Export an HMAC key. Only the <strong>PKOAEP2</strong> and <strong>AESKW</strong> key formatting methods are supported.</td>
</tr>
<tr>
<td>Recovery method (One, required)</td>
<td></td>
</tr>
<tr>
<td>AESKW</td>
<td>Specifies that the key is to be formatted using <strong>AESKW</strong> and placed in an external variable length CCA token. The <code>transport_key_identifier</code> must be an AES EXPORTER. This rule is not valid with the <strong>DES</strong> algorithm keyword or with AES DATA (version X’04’) keys.</td>
</tr>
</tbody>
</table>
## Table 99. Keywords for Symmetric Key Export control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AESKWCV</td>
<td>Specifies to return the key formatted using the ANSI X9.102 AESKW method creating a special variable-length symmetric key-token whose key type is DESUSECV. The AESKW payload that contains the DES key is encrypted by the AES EXPORTER key-encrypting key provided as the transport key, and returned in an external variable-length symmetric key-token with a token algorithm of DES. The DES control vector (with its key form bits masked to binary zeros to mask the key length) along with other significant key-token information is included in the associated data section of the variable-length symmetric key-token. Valid only with the DES algorithm, and only for a DES key in a fixed-length symmetric key-token.</td>
</tr>
<tr>
<td>PKCSOAEP</td>
<td>Specifies using the method found in RSA DSI PKCS #1 V2.0 OAEP. See &quot;PKCS #1 hash formats&quot; on page 1044. The default hash method is SHA-1. Use the SHA-256 keyword for the SHA-256 hash method. Use the SHA-384 keyword for the SHA-384 hash method. Use the SHA-512 keyword for the SHA-512 hash method.</td>
</tr>
<tr>
<td>PKCS-12</td>
<td>Specifies using the method found in RSA DSI PKCS #1 block type 02 to recover the symmetric key. In the RSA PKCS #1 v2.0 standard, RSA terminology describes this as the RSAES-PKCS1-v1_5 format. See &quot;PKCS #1 hash formats&quot; on page 1044.</td>
</tr>
<tr>
<td>PKOAEP2</td>
<td>Specifies that the key is formatted as defined in the RSA PKCS #1 v2.1 standard for the RSAES-OAEP encryption mechanism. Valid only with algorithm HMAC. This keyword was introduced with CCA 4.1.0. See &quot;PKCS #1 hash formats&quot; on page 1044.</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>The clear key is right-aligned in the field provided, and the field is padded to the left with zeros up to the size of the RSA encryption block (which is the modulus length).</td>
</tr>
</tbody>
</table>

### Hash method (One, optional for PKCSOAEP, required for PKOAEP2. Not valid with any other key formatting method)

<table>
<thead>
<tr>
<th>Hash Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>Specifies to use the SHA-1 hash method to calculate the OAEP message hash. Valid only with key-formating methods PKCSOAEP or PKOAEP2. This is the default for PKCSOAEP.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Specifies to use the SHA-256 hash method to calculate the OAEP message hash. Valid only with key-formating methods PKCSOAEP or PKOAEP2.</td>
</tr>
<tr>
<td>SHA-384</td>
<td>Specifies to use the SHA-384 hash method to calculate the OAEP message hash. Valid only with key-formating method PKOAEP2.</td>
</tr>
<tr>
<td>SHA-512</td>
<td>Specifies to use the SHA-512 hash method to calculate the OAEP message hash. Valid only with key-formating method PKOAEP2.</td>
</tr>
</tbody>
</table>

### Parameters

**source_key_identifier_length**

- **Direction:** Input
- **Type:** Integer

The length of the `source_key_identifier` parameter. The minimum size is 64 bytes. The maximum size is 725 bytes.

**source_key_identifier**

- **Direction:** Input
- **Type:** String

The label or internal token of a secure AES DATA (version X’04’), DES DATA, or variable-length symmetric key token to encrypt under the supplied RSA public key or AES EXPORTER key. The key in the key identifier must match the algorithm in the `rule_array`. DES is the default algorithm.

**transporter_key_identifier_length**

- **Direction:** Input
- **Type:** Integer
The length of the `transporter_key_identifier` parameter. The maximum size is 3500 bytes for an RSA key token or 725 for an AES EXPORTER key token. The length must be 64 if `transporter_key_identifier` is a label.

**transporter_key_identifier**

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing an RSA public key token, AES EXPORTER token, or label of the key to protect the exported symmetric key.

When the AESKW key formatting method is specified, this parameter must be an AES EXPORTER key token or label with the EXPORT bit on in the key-usage field. Otherwise, this parameter must be an RSA public key token or label.

**enciphered_key_length**

- **Direction:** Input/Output
- **Type:** Integer

The length of the `enciphered_key` parameter. This variable is updated with the actual length of the `enciphered_key` generated. The maximum size you can specify in this parameter is 900 bytes.

**enciphered_key**

- **Direction:** Output
- **Type:** String

A pointer to a string variable containing the key after it has been formatted and enciphered by the transport key. The enciphered key is returned either as an opaque data buffer or in an external variable-length symmetric key-token. For key-formatting method PKOAEP2, the key token has no key verification pattern.

**Restrictions**

The restrictions for CSNDSYX.

- The RSA public-key modulus size (key length) is limited by the function control vector (FCV) to accommodate potential government export and import regulations.
- Retained keys are not supported.
- The maximum public exponent is 17 bits for any key that has a modulus greater than 2048 bits.

**Required commands**

The required commands for CSNDSYX.

This verb requires the following commands to be enabled in the active role based on the key-formatting method and the algorithm:

<table>
<thead>
<tr>
<th>Key-formatting method</th>
<th>Algorithm</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>AESKW</td>
<td>AES</td>
<td>X'0327'</td>
<td>Symmetric Key Export - AESKW</td>
</tr>
<tr>
<td>AESKWCV (Release 4.4 or later)</td>
<td>DES</td>
<td>X'02B3'</td>
<td>Symmetric Key Export - AESKWCV</td>
</tr>
</tbody>
</table>
Symmetric Key Export (CSNDSYX)

<table>
<thead>
<tr>
<th>Key-formatting method</th>
<th>Algorithm</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKOAEP2</td>
<td>AES</td>
<td>X'00FC'</td>
<td>Symmetric Key Export - AES, PKOAEP2</td>
</tr>
<tr>
<td>PKOAEP2</td>
<td>HMAC</td>
<td>X'00F5'</td>
<td>Symmetric Key Export - HMAC, PKOAEP2</td>
</tr>
<tr>
<td>PKCSOAEP or PKCS-1.2</td>
<td>AES</td>
<td>X'0130'</td>
<td>Symmetric Key Export - AES, PKCSOAEP, PKCS-1.2</td>
</tr>
<tr>
<td>PKCSOAEP or PKCS-1.2</td>
<td>DES</td>
<td>X'0105'</td>
<td>Symmetric Key Export - DES, PKCS-1.2</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>AES</td>
<td>X'0131'</td>
<td>Symmetric Key Export - AES, ZERO-PAD</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>DES</td>
<td>X'023E'</td>
<td>Symmetric Key Export - DES, ZERO-PAD</td>
</tr>
</tbody>
</table>

To disallow the wrapping of a key with a weaker key-encrypting key, enable the `Prohibit weak wrapping - Transport keys` command (offset X'0328') in the active role. This command affects multiple verbs. See Chapter 24, “Access control points and verbs,” on page 1047.

To receive a warning when wrapping a key with a weaker key-encrypting key, enable the `Warn when weak wrap - Transport keys` command (offset X'032C') in the active role. The `Prohibit weak wrapping - Transport keys` command (offset X'0328') overrides this command.

**Usage notes**

The usage notes for CSNDSYX.

The hardware configuration sets the limit on the modulus size of keys for key management; thus, this verb will fail if the RSA key modulus bit length exceeds this limit.

The strength of the exporter key expected by Symmetric Key Export depends on the attributes of the key being exported. The resulting return code and reason code when using an exporter KEK that is weaker depends on the `Disallow Weak Key Wrap` command (offset X'0328') and the `Warn when Wrapping Weak Keys` command (offset X'032C'):

- If the `Disallow Weak Key Wrap` command (offset X'0328') is disabled (the default), the key strength requirement is not enforced. Using a weaker key results in return code 0 with a nonzero reason code if the `Warn when Wrapping Weak Keys` command (offset X'032C') is enabled. Otherwise, a reason code of zero is returned.
- If the `Disallow Weak Key Wrap` (offset X'0328') access control point is enabled (using TKE), the key strength requirement will be enforced, and attempting to use a weaker key results in return code 8.

For AES DATA and AES CIPHER keys, the AES EXPORTER key must be at least as long as the key being exported to be considered sufficient strength.

Note that wrapping an AES 192-bit key or an AES 256-bit key with any RSA key will always be considered a weak wrap.

For HMAC keys, the AES EXPORTER must be sufficient strength as described in Table 100 on page 349.
Table 100. AES EXPORTER strength required for exporting an HMAC key under an AES EXPORTER

<table>
<thead>
<tr>
<th>Key-usage field 2 in the HMAC key contains</th>
<th>Minimum strength of AES EXPORTER to adequately protect the HMAC key</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-256, SHA-384, SHA-512</td>
<td>256 bits</td>
</tr>
<tr>
<td>SHA-224</td>
<td>192 bits</td>
</tr>
<tr>
<td>SHA-1</td>
<td>128 bits</td>
</tr>
</tbody>
</table>

If an RSA public key is specified as the `transporter_key_identifier`, the RSA key used must have a modulus size greater than or equal to the total PKOAEP2 message bit length (key size plus total overhead), as described in Table 101.

Table 101. Minimum RSA modulus strength required to contain a PKOAEP2 block when exporting an AES key

<table>
<thead>
<tr>
<th>AES key size</th>
<th>Total message sizes (and therefore minimum RSA key size) when the hash method is:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SHA-1</td>
</tr>
<tr>
<td>128 bits</td>
<td>736 bits</td>
</tr>
<tr>
<td>192 bits</td>
<td>800 bits</td>
</tr>
<tr>
<td>256 bits</td>
<td>800 bits</td>
</tr>
</tbody>
</table>

For AES keys, the AES EXPORTER must be sufficient strength as described in Table 102.

Table 102. Minimum RSA modulus length to adequately protect an AES key

<table>
<thead>
<tr>
<th>AES key to be exported</th>
<th>Minimum strength of RSA wrapping key to adequately protect the AES key</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES 128</td>
<td>3072 bits</td>
</tr>
<tr>
<td>AES 192</td>
<td>7860 bits</td>
</tr>
<tr>
<td>AES 256</td>
<td>15360 bits</td>
</tr>
</tbody>
</table>

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNDSYXJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNDSYXJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber source_key_identifier_length,
    byte[] source_key_identifier,
    hikmNativeNumber transporter_key_identifier_length,
    byte[] transporter_key_identifier,
    hikmNativeNumber enciphered_key_length,
    byte[] enciphered_key);
```

Chapter 8. AES, DES, and HMAC cryptographic keys 349
Symmetric Key Export with Data (CSNDSXD)

Use the Symmetric Key Export with Data verb to export a symmetric key, along with some application supplied data, encrypted using an RSA key.

The clear key data is copied into the provided data field at the offset specified with the data_offset. Then it is encrypted using the PKCS-1.5 block type 2 formatting algorithm.

Format

The format of CSNDSXD.

```c
CSNDSXD(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    source_key_identifier_length,
    source_key_identifier,
    data_length,
    data_offset,
    data,
    RSA_public_key_identifier_length,
    RSA_public_key_identifier,
    RSA_enciphered_key_length,
    RSA_enciphered_key
)
```

Parameters

The parameters for CSNDSXD.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. The value must be 2.

**rule_array**

Direction: Input
Type: String array

A pointer to a string variable containing an array of keywords. The keywords are eight bytes in length and must be left-aligned and padded on the right with space characters. The rule_array keywords are described in Table 103.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>(One required)</td>
</tr>
<tr>
<td>AES</td>
<td>The key specified in source_key_identifier is an AES key.</td>
</tr>
</tbody>
</table>

Table 103. Keywords for Symmetric Key Export with Data control information
Table 103. Keywords for Symmetric Key Export with Data control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES</td>
<td>The key specified in source_key_identifier is a DES key.</td>
</tr>
</tbody>
</table>

**Key Formatting method** (One required)

| PKCS-EXT   | Copy the clear key data (length determined by the key length in the source key token) into the provided data field at the offset specified in the data_offset parameter. Then encrypt the key using the PKCS-1.5 block type 2 formatting algorithm. |

**source_key_identifier_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes in the source_key_identifier variable. This value is 64 when a label is supplied. When the key identifier is a key token, the value is the length of the token. For DES keys, the value must be 64. For AES keys, the maximum value is 725.

**source_key_identifier**

- **Direction:** Input
- **Type:** String

An internal key token, or the label of an operational symmetric key-token record in AES or DES key storage containing an operational AES or DES key token that is to be exported. If the key is a DES key, bit 17 of the control vector must be equal to ‘1'b (XPORT-OK). The key must have a control vector of DATAC or DKYGENDKY with subtype DKYL0, unless the Allow Symmetric Key Export with Data Special access control point is enabled.

If the AES key is in a fixed length key token, no control vector checking is needed. If the AES key is in a variable length token, the key type must be CIPHER. If the key type is not CIPHER, an access control point Allow Symmetric Key Export with Data Special must be enabled. If the key is an AES key, the key management field in the key must allow export by RSA keys and by unauthenticated asymmetric keys.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**data_length**

- **Direction:** Input
- **Type:** Integer

The length of the data parameter in bytes. The maximum value is the length of the modulus (in bytes) of the RSA_public_key_identifier minus 11. The overall maximum value is 501.

**data_offset**

- **Direction:** Input
- **Type:** Integer

The offset in bytes from the start of the clear data (data) where the clear DES or AES key is to be copied. The maximum value is data_length minus the key length of the clear source key.

**data**
Symmetric Key Export with Data (CSNDSXD)

Direction: Input
Type: String

The clear data. The deciphered key from parameter `source_key_identifier` is copied into this data at the specified offset, and then encrypted with the key from parameter `RSA_public_key_identifier`.

**RSA_public_key_identifier_length**

Direction: Input
Type: Integer

The length of the `RSA_public_key_identifier` field in bytes. This value is 64 when a label is supplied. When the key identifier is a key token, the value is the length of the token. The maximum value is 3500.

**RSA_public_key_identifier**

Direction: Input
Type: String

A PKA96 RSA internal or external key-token with the RSA public key of the remote node that imports the exported key.

**RSA_enciphered_key_length**

Direction: Input
Type: Integer

The length of the `RSA_enciphered_key` field in bytes. On output, the variable is updated with the actual length of the `RSA_enciphered_key` parameter. The maximum length is 512.

**RSA_enciphered_key**

Direction: Output
Type: String

The exported RSA-enciphered key.

**Restrictions**

The restrictions for CSNDSXD.

None.

**Required commands**

The required commands for CSNDSXD.

The Symmetric Key Export with Data verb requires the `Symmetric Key Export with Data` command (offset X'02B5') to be enabled in the active role.

In addition, the verb requires the following commands to be enabled in the active role based on the key-formatting method and the token algorithm:

**Table 104. Required commands for the Symmetric Key Export with Data verb.**

<table>
<thead>
<tr>
<th>Key-formatting method</th>
<th>Algorithm</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKCS-EXT</td>
<td>AES</td>
<td>X'0130'</td>
<td>Symmetric Key Export - AES, PKCSOAEP, PKCS-1.2</td>
</tr>
</tbody>
</table>
Table 104. Required commands for the Symmetric Key Export with Data verb (continued).

<table>
<thead>
<tr>
<th>Key-formatting method</th>
<th>Algorithm</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKCS-EXT</td>
<td>DES</td>
<td>X'0105'</td>
<td>Symmetric Key Export - DES, PKCS-1.2</td>
</tr>
</tbody>
</table>

The **Symmetric Key Export with Data - Special** command (offset X'02B6') affects which key types are allowed for the source key token. When offset X'02B6' is enabled in the active role, any key type can be used. When it is not enabled in the active role, the following rules apply:
- **Token algorithm AES:**
  - If the source AES key is in a fixed-length symmetric key-token, the key is always allowed. If the source AES key is in a variable-length symmetric key-token, the key type must be CIPHER.
- **Token algorithm DES:**
  - The source DES key must be in a fixed-length symmetric key-token and have one of the following:
    - A control vector with bit 61 = B'1' (NOT-CCA)
    - A key type of DATAC
    - A key type of DKYGENKY with subtype DKYL0

**Usage notes**

The usage notes for CSNDSXD.

None.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNDSXDJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNDSXDJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber source_key_identifier_length,
    byte[] source_key_identifier,
    hikmNativeNumber data_length,
    byte[] data,
    hikmNativeNumber RSA_public_key_identifier_length,
    byte[] RSA_public_key_identifier,
    hikmNativeNumber RSA_enciphered_key_length,
    byte[] RSA_enciphered_key);
```

**Symmetric Key Generate (CSNDSYG)**

Use the Symmetric Key Generate verb to generate an AES or DES DATA key and return the key in two forms: enciphered under the master key and encrypted under an RSA public key.
Symmetric Key Generate (CSNDSYG)

You can import the RSA public key encrypted form by using the Symmetric Key Import or Symmetric Key Import2 verbs at the receiving node.

Also use the Symmetric Key Generate verb to generate any DES importer or exporter key-encrypting key encrypted under a RSA public key according to the PKA92 formatting structure. See “PKA92 key format and encryption process” on page 1042 for more details about PKA92 formatting.

Format

The format of CSNDSYG.

```c
CSNDSYG(  return_code,  reason_code,  exit_data_length,  exit_data,  rule_array_count,  rule_array,  key_encrypting_key_identifier,  RSA_public_key_identifier_length,  RSA_public_key_identifier,  local_enciphered_key_identifier_length,  local_enciphered_key_identifier,  RSA_enciphered_key_length,  RSA_enciphered_key)
```

Parameters

The parameters for CSNDSYG.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

`rule_array_count`

**Direction:** Input  
**Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1 - 7.

`rule_array`

**Direction:** Input  
**Type:** String array

Keywords that provide control information to the verb. The recovery method is the method to use to recover the symmetric key. Each keyword is left-aligned in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage. The `rule_array` keywords are described in Table 105.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm (One, optional)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>Specifies to generate an AES key.</td>
</tr>
<tr>
<td>DES</td>
<td>Specifies to generate a DES key. This is the default.</td>
</tr>
</tbody>
</table>

**Key-formatting method (One required)**
### Table 105. Keywords for Symmetric Key Generate control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKA92</td>
<td>Specifies the key-encrypting key is to be encrypted under a PKA96 RSA public key according to the PKA92 formatting structure.</td>
</tr>
<tr>
<td>PKCSOAEP</td>
<td>Specifies to use the method found in RSA DSI PKCS #1V2 OAEP. Supported by the DES and AES algorithms. The default hash method is SHA-1. Use the SHA-256 keyword for the SHA-256 hash method.</td>
</tr>
<tr>
<td>PKCS-1.2</td>
<td>Specifies the method found in RSA DSI PKCS #1 block type 02. In the RSA PKCS #1 v2.0 standard, RSA terminology describes this as the RSAES-PKCS1-v1_5 format.</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>The clear key is right-aligned in the field provided, and the field is padded to the left with zeros up to the size of the RSA encryption block (which is the modulus length).</td>
</tr>
<tr>
<td>Key length</td>
<td>(One, optional use with PKA92)</td>
</tr>
<tr>
<td>SINGLE-R</td>
<td>Generates a key-encrypting key that has equal left and right halves allowing it to perform as a single-length key. Valid only for the recovery method of PKA92.</td>
</tr>
<tr>
<td>Key length</td>
<td>(One, optional use with PKCSOAEP, PKCS-1.2, or ZERO-PAD)</td>
</tr>
<tr>
<td>SINGLE, KEYLN8</td>
<td>Generates a single-length DES key. This is the default for DES keys.</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>Generates a double-length DES key. Valid only for DES keys.</td>
</tr>
<tr>
<td>KEYLN16</td>
<td>Generates a double-length DES DATA key. This is the default for AES keys.</td>
</tr>
<tr>
<td>KEYLN24</td>
<td>Generates a triple-length DES DATA key. Valid only for AES keys</td>
</tr>
<tr>
<td>KEYLN32</td>
<td>Generates a 32-byte AES key. Valid only for AES keys</td>
</tr>
<tr>
<td>Encipherment method</td>
<td>for the local enciphered copy of the key (One, optional for use with PKCSOAEP, PKCS-1.2, and ZERO-PAD)</td>
</tr>
<tr>
<td>EX</td>
<td>The DES enciphered key is enciphered by an EXPORTER key that is provided through the key_encrypting_key_identifier parameter.</td>
</tr>
<tr>
<td>IM</td>
<td>The DES enciphered key is enciphered by an IMPORTER key that is provided through the key_encrypting_key_identifier parameter.</td>
</tr>
<tr>
<td>OP</td>
<td>The DES enciphered key is enciphered by the master key. The key_encrypting_key_identifier parameter is ignored. This is the default.</td>
</tr>
<tr>
<td>Key-wrapping method</td>
<td>(One, optional)</td>
</tr>
<tr>
<td>USECONFIG</td>
<td>This is the default. Specifies to wrap the key using the configuration setting for the default wrapping method. The default wrapping method configuration setting may be changed using the TKE. This keyword is ignored for AES keys.</td>
</tr>
<tr>
<td>WRAP-ENH</td>
<td>Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.</td>
</tr>
<tr>
<td>WRAP-ECB</td>
<td>Use original key wrapping method, which uses ECB wrapping for DES key tokens and CBC wrapping for AES key tokens.</td>
</tr>
<tr>
<td>Translation control</td>
<td>(Optional) This is valid only with key-wrapping method WRAP-ENH or with USECONFIG when the default wrapping method is WRAP-ENH. This option cannot be used on a key with a control vector valued to binary zeros.</td>
</tr>
<tr>
<td>ENH-ONLY</td>
<td>Specifies to restrict the key from being wrapped with the legacy wrapping method after it has been wrapped with the enhanced wrapping method. Sets bit 56 (ENH-ONLY) of the control vector to B’1’. This keyword was introduced with CCA 4.1.0.</td>
</tr>
<tr>
<td>Hash method</td>
<td>(Optional). Valid only with keyword PKCSOAEP.</td>
</tr>
<tr>
<td>SHA-1</td>
<td>Specifies to use the SHA-1 hash method to calculate the OAEP message hash. This is the default.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Specifies to use the SHA-256 hash method to calculate the OAEP message hash.</td>
</tr>
</tbody>
</table>

**key_encrypting_key_identifier**
**Symmetric Key Generate (CSNDSYG)**

**Direction:** Input/Output  
**Type:** String

The label or internal token of a key-encrypting key. If the `rule_array` specifies IM, this DES key must be an IMPORTER. If the `rule_array` specifies EX, this DES key must be an EXPORTER.

**RSA_public_key_identifier_length**

**Direction:** Input  
**Type:** Integer

The length of the `RSA_public_key_identifier` parameter. If the `RSA_public_key_identifier` parameter is a label, this parameter specifies the length of the label. The maximum size is 3500 bytes.

**RSA_public_key_identifier**

**Direction:** Input  
**Type:** String

The token, or label, of the RSA public key to be used for protecting the generated symmetric key.

**local_enciphered_key_identifier_length**

**Direction:** Input/Output  
**Type:** Integer

The length of the `local_enciphered_key_identifier`. This field is updated with the actual length of the `local_enciphered_key_identifier` that is generated. The maximum length is 3500 bytes. However, this value should be 64 as in current CCA practice a DES key-token or a key label is always a 64-byte structure.

**local_enciphered_key_identifier**

**Direction:** Input/Output  
**Type:** String

A pointer to a string variable containing either a key name or a key token. The control vector for the local key is taken from the identified key token. On output, the generated key is inserted into the identified key token.

On input, you must specify a token type consistent with your choice of local-key encryption. If you specify IM or EX, you must specify an external key-token. Otherwise, specify an internal key-token or a null key-token.

When PKCSOAEP, PKCS-1.2, or ZERO-PAD is specified, a null key-token can be specified. In this case, an AES DATA or DES DATA key is returned. For an internal key (OP), a default AES DATA or DATA control-vector is returned in the key token. For an external key (IM or EX), the control vector is set to null.

**RSA_enciphered_key_length**

**Direction:** Input/Output  
**Type:** Integer

The length of the `RSA_enciphered_key` parameter. This verb updates this with the actual length of the `RSA_enciphered_key` it generates. The maximum size is 3500 bytes.

**RSA_enciphered_key**
Direction: Input/Output
Type: String

A pointer to a string variable containing the generated RSA-enciphered key returned by the verb. If you specify PKCSOAEP, PKCS-1.2, or ZERO-PAD, on input specify a null key token. If you specify PKA92, on input specify an internal (operational) CCA DES key-token.

Restrictions
The restrictions for CSNDSYG.

None.

Required commands
The required commands for CSNDSYG.

This verb requires the following commands to be enabled in the active role based on the key-formatting method and the algorithm:

<table>
<thead>
<tr>
<th>Key-formatting method</th>
<th>Algorithm</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKCSOAEP or PKCS-1.2</td>
<td>AES</td>
<td>X'012C'</td>
<td>Symmetric Key Generate - AES_ PKCSOAEP_ PKCS-1.2</td>
</tr>
<tr>
<td>PKCSOAEP or PKCS-1.2</td>
<td>DES</td>
<td>X'023F'</td>
<td>Symmetric Key Generate - DES_ PKCS-1.2</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>AES</td>
<td>X'012D'</td>
<td>Symmetric Key Generate - AES_ ZERO-PAD</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>DES</td>
<td>X'023C'</td>
<td>Symmetric Key Generate - DES_ ZERO-PAD</td>
</tr>
<tr>
<td>PKA92</td>
<td>DES</td>
<td>X'010D'</td>
<td>Symmetric Key Generate - DES_ PKA92</td>
</tr>
</tbody>
</table>

The use of the WRAP-ECB or WRAP-ENH key-wrapping method keywords requires the Symmetric Key Generate - Allow wrapping override keywords command (offset X'013E') to be enabled.

The following access-control points are added beginning with Release 4.3:

- To disable the wrapping of a key with a weaker master key, the Prohibit weak wrapping - Master keys command (offset X'0333') must be enabled in the active role.
- To receive a warning when wrapping a key with a weaker master key, enable the Warn when weak wrap - Master keys command (offset X'0332') in the active role. The Prohibit weak wrapping - Master keys command overrides this command.

Usage notes
The usage notes for CSNDSYG.

The hardware configuration sets the limit on the modulus size of keys for key management; thus, this verb will fail if the RSA key modulus bit length exceeds this limit.

Specification of PKA92 with an input NOCV key-encrypting key token is not supported.
Symmetric Key Generate (CSNDSYG)

Use the PKA92 key-formating method to generate a key-encrypting key. The verb enciphers one key copy using the key encipherment technique employed in the IBM Transaction Security System (TSS) 4753, 4755, and AS/400 cryptographic product PKA92 implementations (see “PKA92 key format and encryption process” on page 1042). The control vector for the RSA-enciphered copy of the key is taken from an internal (operational) DES key token that must be present on input in the RSA_enciphered_key variable.

Only key-encrypting keys that conform to the rules for an OPEX case under the Key Generate verb are permitted. The control vector for the local key is taken from a DES key token that must be present on input in the local_enciphered_key_identifier variable. The control vector for one key copy must be from the EXPORTER class, while the control vector for the other key copy must be from the IMPORTER class.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNDSYGJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNDSYGJ(
  hikmNativeNumber return_code,
  hikmNativeNumber reason_code,
  hikmNativeNumber exit_data_length,
  byte[] exit_data,
  hikmNativeNumber rule_array_count,
  byte[] rule_array,
  byte[] key_encrypting_key_identifier,
  hikmNativeNumber RSA_public_key_identifier_length,
  byte[] RSA_public_key_identifier,
  hikmNativeNumber local_enciphered_key_identifier_length,
  byte[] local_enciphered_key_identifier,
  hikmNativeNumber RSA_enciphered_key_length,
  byte[] RSA_enciphered_key);
```

Symmetric Key Import (CSNDSYI)

Use the Symmetric Key Import verb to import a symmetric AES DATA or DES DATA key enciphered under an RSA public key. The verb returns the key in operational form, enciphered under the master key.

This verb also supports import of a PKA92-formatted DES key-encrypting key under a PKA96 RSA public key.
Format

The format of CSNDSYI.

```c
CSNDSYI(    return_code,    reason_code,    exit_data_length,    exit_data,    rule_array_count,    rule_array,    RSA_enciphered_key_length,    RSA_enciphered_key,    RSA_private_key_identifier_length,    RSA_private_key_identifier,    target_key_identifier_length,    target_key_identifier)
```

Parameters

The parameters for CSNDSYI.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

`rule_array_count`

Direction: Input  
Type: Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1 - 5.

`rule_array`

Direction: Input  
Type: String array

The keyword that provides control information to the verb. The recovery method is the method to use to recover the symmetric key. The keyword is left-aligned in an 8-byte field and padded on the right with blanks. The `rule_array` keywords are described in Table 106.

**Table 106. Keywords for Symmetric Key Import control information**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithm</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>Export an AES key.</td>
</tr>
<tr>
<td>DES</td>
<td>Export a DES key. This is the default.</td>
</tr>
<tr>
<td><strong>Recovery method</strong> (One required)</td>
<td></td>
</tr>
<tr>
<td>PKA92</td>
<td>Specifies the key-encrypting key is encrypted under a PKA96 RSA public key according to the PKA92 formatting structure.</td>
</tr>
<tr>
<td>PKCSOAEP</td>
<td>Specifies to use the method found in RSA DSI PKCS #1V2 OAEP. Supported by the DES and AES algorithms. The default hash method is SHA-1. Use the SHA-256 keyword for the SHA-256 hash method.</td>
</tr>
<tr>
<td>PKCS-1.2</td>
<td>Specifies the method found in RSA DSI PKCS #1 block type 02. In the RSA PKCS #1 v2.0 standard, RSA terminology describes this as the RSAES-PKCS1-v1_5 format.</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>The clear key is right-aligned in the field provided, and the field is padded to the left with zeros up to the size of the RSA encryption block (which is the modulus length).</td>
</tr>
</tbody>
</table>
### Table 106. Keywords for Symmetric Key Import control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key-wrapping method</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>USECONFIG</td>
<td>This is the default. Specifies to wrap the key using the configuration setting for the default wrapping method. The default wrapping method configuration setting may be changed using the TKE. This keyword is ignored for AES keys.</td>
</tr>
<tr>
<td>WRAP-ENH</td>
<td>Specifies to wrap the key using the legacy wrapping method. This keyword is ignored for AES keys. This keyword was introduced with CCA 4.1.0.</td>
</tr>
<tr>
<td>WRAP-ECB</td>
<td>Specifies to wrap the key using the enhanced wrapping method. Valid only for DES keys. This keyword was introduced with CCA 4.1.0.</td>
</tr>
<tr>
<td><strong>Translation control</strong> (Optional) This is valid only with key-wrapping method WRAP-ENH or with USECONFIG when the default wrapping method is WRAP-ENH. This option cannot be used on a key with a control vector valued to binary zeros.</td>
<td></td>
</tr>
<tr>
<td>ENH-ONLY</td>
<td>Specifies to restrict the key from being wrapped with the legacy wrapping method after it has been wrapped with the enhanced wrapping method. Sets bit 56 (ENH-ONLY) of the control vector to B'1'. This keyword was introduced with CCA 4.1.0.</td>
</tr>
<tr>
<td><strong>Hash method</strong> (Optional). Valid only with keyword PKCSOAEP.</td>
<td></td>
</tr>
<tr>
<td>SHA-1</td>
<td>Specifies to use the SHA-1 hash method to calculate the OAEP message hash. This is the default.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Specifies to use the SHA-256 hash method to calculate the OAEP message hash.</td>
</tr>
</tbody>
</table>

**RSA_enciphered_key_length**

- **Direction:** Input
- **Type:** Integer

The length of the `RSA_enciphered_key` parameter. The maximum size is 3500 bytes.

**RSA_enciphered_key**

- **Direction:** Input
- **Type:** String

The key to import, protected under an RSA public key. The encrypted key is in the low-order bits (right-aligned) of a string whose length is the minimum number of bytes that can contain the encrypted key. This string is left-aligned within the `RSA_enciphered_key` parameter.

**RSA_private_key_identifier_length**

- **Direction:** Input
- **Type:** Integer

The length of the `RSA_private_key_identifier` parameter. When the `RSA_private_key_identifier` parameter is a key label, this field specifies the length of the label. The maximum size is 3500 bytes.

**RSA_private_key_identifier**

- **Direction:** Input
- **Type:** String

An internal RSA private key token or label whose corresponding public key protects the symmetric key.

**target_key_identifier_length**
Symmetric Key Import (CSNDSYI)

Direction: Input/Output
Type: Integer

The length of the target_key_identifier parameter. This field is updated with the actual length of the target_key_identifier that is generated. The maximum length is 3500 bytes.

target_key_identifier

Direction: Input/Output
Type: String

This field contains the internal token of the imported symmetric key.

Except for PKA92 processing, this verb produces a DATA key token with a key of the same length as that contained in the imported token.

Restrictions

The restrictions for CSNDSYI.

None.

Required commands

The required commands for CSNDSYI.

This verb requires the following commands to be enabled in the active role based on the key-formatting method and the algorithm:

<table>
<thead>
<tr>
<th>Key-formatting method</th>
<th>Algorithm</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKA92 and DATA, MAC, MACVER, KEYGENKY, EXPORTER, or OKEYXLAT key</td>
<td>DES</td>
<td>X'0235'</td>
<td>Symmetric Key Import - DES, PKA92 KEK</td>
</tr>
<tr>
<td>PKCSOAEP or PKCS-1.2</td>
<td>AES</td>
<td>X'012F'</td>
<td>Symmetric Key Import - AES, PKCSOAEP, PKCS-1.2</td>
</tr>
<tr>
<td></td>
<td>DES</td>
<td>X'0106'</td>
<td>Symmetric Key Import - AES, PKCS-1.2</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>AES</td>
<td>X'012F'</td>
<td>Symmetric Key Import - AES, ZERO-PAD</td>
</tr>
<tr>
<td></td>
<td>DES</td>
<td>X'023D'</td>
<td>Symmetric Key Import - AES, ZERO-PAD</td>
</tr>
<tr>
<td>WRAP-ECB or WRAP-ENH, when the default key-wrapping method setting does not match the keyword</td>
<td>DES</td>
<td>X'0144'</td>
<td>Symmetric Key Import - Allow wrapping override keywords</td>
</tr>
</tbody>
</table>

The following access control points control the use of weak transport keys:

- To disallow the import of a key wrapped with a weaker transport key, the `Symmetric Key Import - allow weak import` command (offset X'032B') must be enabled in the active role.

- To disable the wrapping of a key with a weaker transport key, the `Prohibit weak wrapping - Transport keys` command (offset X'0328') must be enabled in the active role.

- To receive an informational message when wrapping a key with a weaker key-encrypting key, enable the `Warn when weak wrap - Transport keys` command (offset X'032C') in the active role. The `Prohibit weak wrapping - Transport keys` command overrides this command.
Symmetric Key Import (CSNDSYI)

The following access control points control the use of weak master keys:

- To disable the wrapping of a key with a weaker master key, the Prohibit weak wrapping - Master keys command (offset X'0333') must be enabled in the active role.
- To receive a warning when wrapping a key with a weaker master key, enable the Warn when weak wrap - Master keys command (offset X'0332') in the active role. The Prohibit weak wrapping - Master keys command overrides this command.

Usage notes

The usage notes for CSNDSYI.

The hardware configuration sets the limit on the modulus size of keys for key management; thus, this verb will fail if the RSA key modulus bit length exceeds this limit.

Use of PKA92 with an input NOCV key-encrypting key token is not supported.

During initialization of a CEX*C, an Environment Identifier (EID) of zero will be set in the coprocessor. This will be interpreted by the Symmetric Key Import verb to mean that environment identification checking is to be bypassed. Thus it is possible on a Linux on Z system for a key-encrypting key RSA-enciphered at a node (EID) to be imported at the same node.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNDSYIJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNDSYIJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber RSA_enciphered_key_length,
    byte[] RSA_enciphered_key,
    hikmNativeNumber RSA_private_key_identifier_length,
    byte[] RSA_private_key_identifier,
    hikmNativeNumber target_key_identifier_length,
    byte[] target_key_identifier);
```

Symmetric Key Import2 (CSNDSYI2)

Use the Symmetric Key Import2 verb to import a symmetric key that is contained in an external variable-length symmetric key-token, and that has been exported by the Symmetric Key Export verb or the Key Generate2 verb, into an internal variable-length symmetric key-token, or into an internal fixed-length DES key-token.

The enciphered input key to be imported can be one of the following, depending on what release is used:
In all releases, the input key can't be an HMAC key that has been previously formatted using key-formatting method PKOAEP2.

AES keys are supported along with support for the AES token algorithm. With this support, the input key can also be an HMAC key that has been previously formatted using key-formatting method AESKW, provided that the operational AES key-encrypting key used to encipher the key is provided. Likewise, an AES key can either be in an external AES variable-length symmetric key-token enciphered under an AES key-encrypting key (AESKW), or an RSA public-key (PKOAEP2).

DES keys are supported, along with support for the DES token algorithm. With this support, the input key must have a key type of DESUSECV. A DESUSECV key contains the control vector and other information necessary to recreate the original internal fixed-length DES key-token.

When importing a DES key, the verb must decide whether to use the legacy ECB mode or the enhanced CBC mode when wrapping the key in the target key-token. New optional key-wrapping method keywords are added to select which key-wrapping method to use.

Also when importing a DES key, a new optional translation control keyword allows the target key to be restricted to being wrapped only with the enhanced CBC method once it has been wrapped with the enhanced method.

Before importing a DES key (Release 4.4 or later), the verb must determine whether to wrap the target key in legacy ECB mode or in enhanced CBC mode. These factors influence the key-wrapping method used for the imported target key-token:

1. The first is the default internal key-token key-wrapping preference of the receiving system where the target key token will be created. The receiving system can be set to a preference to wrap internal key tokens in either ECB or Enhanced modes.
2. The second is the key-wrapping method used to wrap the original key which was exported from the originating system.
3. The third is the key-wrapping method used by the verb which is specified by an optional key-wrapping method keyword or by default.

Table 107 and Table 108 on page 364 show how the key-wrapping method will be determined for the target key, based on the previously explained factors.

Table 107. Symmetric Key Import2 key-wrapping method of target key when system default is ECB (Legacy)

<table>
<thead>
<tr>
<th>Wrap method of original key that was exported</th>
<th>Key-wrapping method keyword</th>
<th>Key-wrapping method used for the imported target key</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECB (Legacy)</td>
<td>USECONF</td>
<td>ECB (Legacy).</td>
</tr>
<tr>
<td>WRAP-ECB</td>
<td></td>
<td>Under control of command Symmetric Key Import2 - Allow wrapping override keywords (offset X'02B9'):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If X'02B9' is enabled, ECB (Legacy).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If X'02B9' is not enabled, not authorized error is returned.</td>
</tr>
<tr>
<td>WRAP-ENH</td>
<td></td>
<td>Under control of offset X'02B9':</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If X'02B9' is enabled, Enhanced. In addition, if ENH-ONLY keyword, CV bit 56 is set to B'1' (ENH-ONLY).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If X'02B9' is not enabled, not authorized error is returned.</td>
</tr>
</tbody>
</table>
### Table 107. Symmetric Key Import2 key-wrapping method of target key when system default is ECB (Legacy) (continued)

<table>
<thead>
<tr>
<th>Enhanced with CV bit 56 = B’0’ (not ENH-ONLY)</th>
<th>USECONF</th>
<th>ECB (Legacy).</th>
</tr>
</thead>
</table>
| | WRAP-ECB | Under control of command offset X’02B9’:
| | | • If X’02B9’ is enabled, ECB (Legacy).
| | | • If X’02B9’ is not enabled, not authorized error is returned. |
| | WRAP-ENH | Under control of offset X’02B9’:
| | | • If X’02B9’ is enabled, Enhanced. In addition, if ENH-ONLY keyword, CV bit 56 is set to B’1’ (ENH-ONLY).
| | | • If X’02B9’ is not enabled, not authorized error is returned. |
| Enhanced with CV bit 56 = B’1’ (ENH-ONLY) | USECONF | Control information in the key token conflicts with that in the rule array error is returned. |
| | WRAP-ECB | Under control of offset X’02B9’:
| | | • If X’02B9’ is enabled, control information in the key token conflicts with that in the rule array error is returned.
| | | • If X’02B9’ is not enabled, not authorized error is returned. |
| | WRAP-ENH | Under control of offset X’02B9’:
| | | • If X’02B9’ is enabled, Enhanced and CV bit 56 is set to B’1’ (ENH-ONLY).
| | | • If X’02B9’ is not enabled, error not authorized is returned. |

**Note:** Conversion of an original key-token wrapped in Enhanced mode to an imported target key-token wrapped in ECB mode reduces security for that key. In this case, a command to override the system key-wrapping default is required.

### Table 108. Symmetric Key Import2 key-wrapping method of target key when system default is CBC (Enhanced)

<table>
<thead>
<tr>
<th>Wrap method of original key that was exported</th>
<th>Key-wrapping method keyword</th>
<th>Key-wrapping method used for the imported target key</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECB (Legacy)</td>
<td>USECONF</td>
<td>Enhanced. In addition, if ENH-ONLY keyword, CV bit 56 is set to B’1’ (ENH-ONLY).</td>
</tr>
</tbody>
</table>
| | WRAP-ECB | Under control of offset X’02B9’:
| | | • If X’02B9’ is enabled, ECB (Legacy).
| | | • If X’02B9’ is not enabled, not authorized error is returned. |
| | WRAP-ENH | Enhanced. In addition, if ENH-ONLY keyword, CV bit 56 is set to B’1’ (ENH-ONLY). |
| Enhanced with CV bit 56 = B’0’ (not ENH-ONLY) | USECONF | Enhanced. If ENH-ONLY keyword, CV bit 56 is set to B’1’ (ENH-ONLY). |
| | WRAP-ECB | Under control of offset X’02B9’:
| | | • If X’02B9’ is enabled, ECB (Legacy).
| | | • If X’02B9’ is not enabled, not authorized error is returned. |
| | WRAP-ENH | Enhanced. In addition, if ENH-ONLY keyword, CV bit 56 is set to B’1’ (ENH-ONLY). |
Symmetric Key Import2 (CSNDSYI2)

Table 108. Symmetric Key Import2 key-wrapping method of target key when system default is CBC (Enhanced) (continued)

<table>
<thead>
<tr>
<th>Enhanced with CV bit 56 = B’1’ (ENH-ONLY)</th>
<th>USECONFIG</th>
<th>WRAP-ECB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Enhanced and CV bit 56 is set to B’1’ (ENH-ONLY).</td>
<td>Under control of offset X’02B9’:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If X’02B9’ is enabled, control information in the key token conflicts with that in the rule array error is returned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If X’02B9’ is not enabled, not authorized error is returned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WRAP-ENH</td>
</tr>
<tr>
<td></td>
<td>Under control of offset X’02B9’:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• If X’02B9’ is enabled, enhanced and CV bit 56 is set to B’1’ (ENH-ONLY).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• If X’02B9’ is not enabled, not authorized error is returned.</td>
<td></td>
</tr>
</tbody>
</table>

Note: Conversion of an original key-token wrapped in ECB (Legacy) mode to an imported target key-token wrapped in Enhanced (CBC) mode improves security for that key.

Format
The format of CSNDSYI2.

```
CSNDSYI2(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    enciphered_key_length,
    enciphered_key,
    transport_key_identifier_length,
    transport_key_identifier,
    key_name_length,
    key_name,
    target_key_identifier_length,
    target_key_identifier)
```

Parameters
The parameters for CSNDSYI2.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

- **rule_array_count**
  - Direction: Input
  - Type: Integer

  The number of keywords you supplied in the `rule_array` parameter. This value must be 2.

- **rule_array**
  - Direction: Input
  - Type: String array

  The keywords that provide control information to the verb. The following table provides a list. The recovery method is the method to use to recover the symmetric key. The keywords must be 8 bytes of contiguous storage with the keyword left-aligned in its 8-byte location and padded on the right with
Symmetric Key Import2 (CSNDSYI2)

blanks. The rule_array keywords are described in Table 109.

Table 109. Keywords for Symmetric Key Import2 control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>(One, required)</td>
</tr>
<tr>
<td>AES</td>
<td>The key being imported is an AES key.</td>
</tr>
<tr>
<td>DES</td>
<td>The key being imported is a DES key.</td>
</tr>
<tr>
<td>HMAC</td>
<td>The key being imported is an HMAC key. Only the PKOAEP2 recovery method is supported.</td>
</tr>
<tr>
<td>Recovery</td>
<td>(One, required)</td>
</tr>
<tr>
<td>AESKW</td>
<td>Specifies the enciphered key has been wrapped with the AESKW formatting method.</td>
</tr>
<tr>
<td>AESKWCV</td>
<td>Specifies that the key is to be formatted using AESKW and placed in a symmetric variable length CCA token of type DESUSECV. The transport_key_identifier must be an AES EXPORTER key. The DES control vector and other significant token information is in the associated data section of the variable length key token. Only valid with the DES token algorithm.</td>
</tr>
<tr>
<td>PKOAEP2</td>
<td>Specifies to format the key according to the method found in RSA DSI PKCS #1 v2.1 RSAES-OAEP documentation.</td>
</tr>
<tr>
<td>Key wrapping method</td>
<td>(Optional, valid only for DES algorithm. The access control point Symmetric Key Import2 – Allow wrapping override keywords must be enabled to specify these keywords)</td>
</tr>
<tr>
<td>USECONFIG</td>
<td>This is the default. Specifies to wrap the key using the configuration setting for the default wrapping method. The default wrapping method configuration setting may be changed using the TKE. This keyword is ignored for AES keys.</td>
</tr>
<tr>
<td>WRAP-ENH</td>
<td>Specifies that the new enhanced wrapping method is to be used to wrap the key.</td>
</tr>
<tr>
<td>WRAP-ECB</td>
<td>Specifies that the original wrapping method is to be used.</td>
</tr>
<tr>
<td>Translation Control</td>
<td>(Optional, valid only for enhanced wrapping)</td>
</tr>
<tr>
<td>ENH-ONLY</td>
<td>Specify this keyword to indicate that the key once wrapped with the enhanced method cannot be wrapped with the original method. This restricts translation to the original method.</td>
</tr>
</tbody>
</table>

Note: There is no need for a hash method keyword, because the hash method is encoded in the external key-token carrying the encoded and encrypted payload.

enciphered_key_length

Direction: Input
Type: Integer

The length of the enciphered_key parameter. The maximum size is 900 bytes.

enciphered_key

Direction: Input
Type: String

The key to import, protected under either an RSA public key or an AES KEK. If the recovery method is PKOAEP2, the encrypted key is in the low-order bits (right-aligned) of a string whose length is the minimum number of bytes that can contain the encrypted key. If the recovery method is AESKW, the encrypted key is an AES key or HMAC key in the external variable length key token.

transport_key_identifier_length

Direction: Input
Type: Integer
The length of the transport_key_identifier parameter. When the transport_key_identifier parameter is a key label, this field must be 64. The maximum size is 3500 bytes for an RSA private key, or 725 bytes for an AES IMPORTER KEK.

**transport_key_identifier**

**Direction:** Input  
**Type:** String

An internal RSA private key token, internal AES IMPORTER KEK, or the 64-byte label of a key token whose corresponding key protects the symmetric key.

When the AESKW key formatting method is specified, this parameter must be an AES IMPORTER key with the IMPORT bit on in the key-usage field. Otherwise, this parameter must be an RSA private key.

**key_name_length**

**Direction:** Input  
**Type:** Integer

The length of the key_name parameter for target_key_identifier. Valid values are 0 and 64.

**key_name**

**Direction:** Input  
**Type:** String

A 64-byte key store label to be stored in the associated data structure of target_key_identifier.

**target_key_identifier_length**

**Direction:** Input/Output  
**Type:** Integer

On input, the length in bytes of the buffer for the target_key_identifier parameter. The buffer must be large enough to receive the target key token. The maximum value is 725 bytes.

On output, the parameter will hold the actual length of the target key token.

**target_key_identifier**

**Direction:** Output  
**Type:** String

This parameter contains the internal token of the imported symmetric key.

**Restrictions**

The restrictions for CSNDSYI2.

The exponent of the RSA public key must be odd.

**Required commands**

The required commands for CSNDSYI2.

The Symmetric Key Import2 verb requires the following commands to be enabled in the active role:
Symmetric Key Import2 (CSNDSYI2)

<table>
<thead>
<tr>
<th>Key-formatting method keyword</th>
<th>Token algorithm keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>AESKW (Rel. 4.2 or later)</td>
<td>AES or HMAC</td>
<td>X'0329'</td>
<td>Symmetric Key Import2 - AESKW</td>
</tr>
<tr>
<td>AESKWCV (Release 4.4 or later)</td>
<td>DES</td>
<td>X'02B4'</td>
<td>Symmetric Key Import2 - AESKWCV</td>
</tr>
<tr>
<td>WRAP-ECB or WRAP-ENH, when the default key-wrapping method setting does not match the keyword</td>
<td>DES</td>
<td>X'02B9'</td>
<td>Symmetric Key Import2 - Allow wrapping override keywords</td>
</tr>
<tr>
<td>PKOAEP2</td>
<td>AES (Rel. 4.2 or later)</td>
<td>X'00FD'</td>
<td>Symmetric Key Import2 - AES,PKOAEP2</td>
</tr>
<tr>
<td>PKOAEP2</td>
<td>HMAC</td>
<td>X'00F4'</td>
<td>Symmetric Key Import2 - HMAC,PKOAEP2</td>
</tr>
</tbody>
</table>

To disallow the import of a key wrapped with a weaker transport key, the Symmetric Key Import2 - disallow weak import command (offset X'032B') must be enabled in the active role. This command affects multiple verbs. See Chapter 24, “Access control points and verbs,” on page 1047.

To receive a warning against the wrapping of a stronger key with a weaker key, the Warn when weak wrap - Transport keys command (offset X'032C') must be enabled in the active role. The Symmetric Key Import2 - disallow weak import command overrides this command.

To disable the wrapping of a key with a weaker master key, the Prohibit weak wrapping - Master keys command (offset X'0333') must be enabled in the active role.

To receive a warning when wrapping a key with a weaker master key, enable the Warn when weak wrap - Master keys command (offset X'0332') in the active role. The Prohibit weak wrapping - Master keys command overrides this command.

Usage notes
The usage notes for CSNDSYI2.

This is the message layout used to encode the key material exported with the PKOAEP2 formatting method.

Table 110. PKCS#1 OAEP encoded message layout (PKOAEP2)

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash field</td>
<td>32 bytes</td>
<td>SHA-256 hash of associated data section in the source key identifier</td>
</tr>
<tr>
<td>Key bit length</td>
<td>2 bytes</td>
<td>Variable</td>
</tr>
<tr>
<td>Key material</td>
<td>length in bytes of the key material (rounded up to the nearest byte)</td>
<td>Variable</td>
</tr>
</tbody>
</table>

Hash field
The associated data for the HMAC variable length token is hashed using SHA-256.

Key bit length
A 2-byte key bit length field.

Key material
The key material is padded to the nearest byte with '0' bits.
The hardware configuration sets the limit on the modulus size of keys for key management; thus, this verb will fail if the RSA key modulus bit length exceeds this limit.

Specification of PKA92 with an input NOCV key-encrypting key token is not supported.

During initialization of a CEX*C, an Environment Identifier (EID) of zero is set in the coprocessor. This is interpreted by the Symmetric Key Import2 verb to mean that environment identification checking is to be bypassed. Thus it is possible on a Linux on Z platform for a key-encrypting key RSA-enciphered at a node (EID) to be imported at the same node.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNDSYI2J.

See “Building Java applications using the CCA JNI” on page 28.

Format
```
public native void CSNDSYI2J(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber enciphered_key_length,
    byte[] enciphered_key,
    hikmNativeNumber transport_key_identifier_length,
    byte[] transport_key_identifier,
    hikmNativeNumber key_name_length,
    byte[] key_name,
    hikmNativeNumber target_key_identifier_length,
    byte[] target_key_identifier);
```

Unique Key Derive (CSNBUKD)

The Unique Key Derive verb performs the key derivation process as defined in ANSI X9.24 Part 1.

The process derives keys from two values: the base derivation key and the derivation data:

- The base derivation key is the key from which the others are derived. This must be a KEYGENKY with the UKPT bit (bit 18) set to 1 in the Control Vector.
- The derivation data is used to make the derived key specific to a particular device and to a specific transaction from that device. The derivation data, called the Current Key Serial Number (CKSN), is the 80-bit concatenation of the device's 59-bit Initial Key Serial Number value and the 21-bit value of the current encryption counter which the device increments for each new transaction.

The Initial Pin Encryption Key (IPEK) is derived from the base derivation key and the initial derivation data. Specify the K3IPEK rule array keyword to return the IPEK.

Rule array keywords determine the types and number of keys derived on a particular call. See the Rule Array parameter description for more information.
Unique Key Derive (CSNBUKD)

Output keys are wrapped using the mode configured as the default wrapping mode, either enhanced wrapping mode (WRAP-ENH) or original ECB wrapping mode (WRAP-ECB).

**Note:** This verb supports PCI-HSM 2016 compliant-tagged key tokens.

**Format**

The format of CSNBUKD.

```c
CSNBUKD(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    base_derivation_key_identifier_length,
    base_derivation_key_identifier,
    derivation_data_length,
    derivation_data,
    generated_key_identifier1_length,
    generated_key_identifier1,
    generated_key_identifier2_length,
    generated_key_identifier2,
    generated_key_identifier3_length,
    generated_key_identifier3,
    transport_key_identifier_length,
    transport_key_identifier,
    reserved1_length,
    reserved1,
    reserved2_length,
    reserved2,
    reserved3_length,
    reserved3,
    reserved4_length,
    reserved4,
    reserved5_length,
    reserved5)
```

**Parameters**

The parameter definitions for CSNBUKD.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. Values are in the range 1 - 5.

**rule_array**

- **Direction:** Input
- **Type:** String array

An array of 8-byte keywords providing the processing control information to the verb. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with
blanks. The **rule_array** keywords are described in Table 111.

### Table 111. Keywords for Unique Key Derive control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithm</strong></td>
<td>(One, optional. The default is DES.) Specifies that the keys to be generated are DES (Triple DES) keys. All input skeleton tokens must be DES tokens and all generated output tokens are DES tokens.</td>
</tr>
<tr>
<td>DES</td>
<td>Specifies that the keys to be generated are DES (Triple DES) keys. All input skeleton tokens must be DES tokens and all generated output tokens are DES tokens.</td>
</tr>
<tr>
<td><strong>Token output type</strong></td>
<td>(One, required for K3IPEK.) Specifies that the output IPEK should be wrapped by the TDES transport key and returned in an external TDES token.</td>
</tr>
<tr>
<td>TDES-TOK</td>
<td>Specifies that the output IPEK should be wrapped by the TDES transport key and returned in an external TDES token.</td>
</tr>
<tr>
<td>TR31-TOK</td>
<td>Specifies that the output IPEK should be wrapped by the TDES transport key and returned in a TR-31 key block.</td>
</tr>
<tr>
<td><strong>Key wrapping method</strong></td>
<td>(One, optional. The default is USECONFG.) The access control point Unique Key Derive – Override Default Wrapping Method must be enabled to specify these keywords.</td>
</tr>
<tr>
<td>USECONFG</td>
<td>Specifies to wrap the key using the configuration setting for the default wrapping method. The default wrapping method configuration setting may be changed using the TKE. This keyword is ignored for AES keys.</td>
</tr>
<tr>
<td>WRAP-ECB</td>
<td>Specifies to wrap the key using the original wrapping method.</td>
</tr>
<tr>
<td>WRAP-ENH</td>
<td>Specifies to wrap the key using the enhanced wrapping method.</td>
</tr>
<tr>
<td><strong>Output key selection keywords</strong></td>
<td>(One required, up to 3 can be specified.) Neither the PIN-DATA nor the K3IPEK keyword can be specified with any other Output Key Selection keywords. Any combination of the other keywords (K1DATA, K2MAC, and K3PIN) can be specified, enabling a program to produce up to 3 different output keys with one call.</td>
</tr>
<tr>
<td>K1DATA</td>
<td>The returned key type for this keyword is a DATA ENCRYPTION key. This is the output key selection keyword for the generated_key_identifier1_length and generated_key_identifier1 parameters. The output value generated_key_identifier1 is created and is a data encryption key. The skeleton token provided in that parameter on input must be one of the permitted data encryption key types for this callable service. For valid values see Table 112 on page 375.</td>
</tr>
<tr>
<td>K2MAC</td>
<td>The returned key type for this keyword is a MAC key. This is the output key selection keyword for the generated_key_identifier2_length and generated_key_identifier2 parameters. The output value generated_key_identifier2 is created and is a MAC key. The skeleton token provided in that parameter on input must be one of the permitted MAC key types for this callable service. For valid values, see Table 112 on page 375.</td>
</tr>
<tr>
<td>K3PIN</td>
<td>The returned key type for this keyword is a PIN key. This is an output key selection keyword for the generated_key_identifier3_length and generated_key_identifier3 parameters. The output value generated_key_identifier3 is created and is a PIN key. The skeleton token provided in that parameter on input must be one of the permitted PIN key types for this callable service. For valid values see Table 112 on page 375.</td>
</tr>
<tr>
<td>K3IPEK</td>
<td>The returned key for this keyword is the IPEK. This is an output key selection keyword for the generated_key_identifier3_length and generated_key_identifier3 parameters. The output value generated_key_identifier3 will be created and will be the initial PIN encryption key wrapped by the TDES transport key and returned in an external symmetric token or TR-31 key block as indicated by the token output type keyword. The skeleton token provided in that parameter on input must be one of the permitted PIN key types for this callable service. For valid values see Table 112 on page 375. This keyword may not be combined with any other output key selection keyword.</td>
</tr>
</tbody>
</table>
**Unique Key Derive (CSNBUKD)**

Table 111. Keywords for Unique Key Derive control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
</table>
| PIN-DATA    | The returned key type for this keyword is a PIN key, which is returned in a DATA key token. This is an output key selection keyword for the `generated_key_identifier3_length` and `generated_key_identifier3` parameters. The output value `generated_key_identifier3` is created and will be a DATA key. The skeleton token provided in that parameter on input must be one of the permitted "PIN key with rule keyword PIN-DATA" key types for this callable service. For valid values, see Table 112 on page 375. To use this option:  
  • Control Vector bit 61 (Not-CCA) is set to a B'1'.  
  • Access control point Unique Key Derive – Allow PIN-DATA processing must be enabled. |

**base_derivation_key_identifier_length**

Direction: Input  
Type: Integer  
Length of the `base_derivation_key_identifier` parameter in bytes. This value must be 64.

**base_derivation_key_identifier**

Direction: Input/Output  
Type: String  
The base derivation key is the key from which the operational keys are derived using the DUKPT algorithms defined in ANSI X9.24 Part 1. The base derivation key must be an internal key token or the label of an internal key token containing a double-length KEYGENKY key with the UKPT bit (bit 18) set to B'1' in the control vector.

**derivation_data_length**

Direction: Input  
Type: Integer  
Length of the `derivation_data` parameter in bytes. This value must be 10.

**derivation_data**

Direction: Input  
Type: String  
The derivation data is an 80-bit (10-byte) string that contains the Current Key Serial Number (CKSN) of the device concatenated with the 21-bit value of the current Encryption Counter which the device increments for each new transaction.

**generated_key_identifier1_length**

Direction: Input/Output  
Type: Integer  
Length of the `generated_key_identifier1` parameter in bytes. Values are 0 and 64.

**generated_key_identifier1**
Direction: Input/Output  Type: String

On input, this parameter must be a DES Data encryption key token or a skeleton token of a DES Data encryption key, with one of the Data encryption control vectors as shown in Table 112 on page 375.

On output, generated_key_identifier1 contains the data encryption token with the derived data encryption key.

generated_key_identifier2_length

Direction: Input/Output  Type: Integer

Length of the generated_key_identifier2 parameter in bytes. Values are 0 and 64.

generated_key_identifier2

Direction: Input/Output  Type: String

On input, this must be a DES MAC key token or a skeleton token of a DES MAC key, with one of the MAC control vectors as shown in Table 112 on page 375.

On output, generated_key_identifier2 contains the MAC token with the derived MAC key.

generated_key_identifier3_length

Direction: Input/Output  Type: Integer

Length of the generated_key_identifier3 parameter in bytes. When the rule array keyword is K3IPEK, the length must be at least 64 bytes. Otherwise, values are 0 and 64.

generated_key_identifier3

Direction: Input/Output  Type: String

The input and output values for this parameter depends on the keyword specified in the rule_array parameter. The rule_array keyword for the generation_key_identifier3 parameter can be either PIN-DATA or K3PIN.

- When rule array keyword is PIN-DATA, input must be a data key token or skeleton token of a data key with one of the PIN key with rule keyword PIN-DATA-control-vectors as shown in Table 112 on page 375. On output, this parameter contains the data token with the derived PIN key.
- When rule array keyword is K3PIN, input must be a DES PIN key token or a skeleton token of a DES PIN key, with one of the PIN control vectors as shown in Table 112 on page 375. On output, this parameter contains the PIN token with the derived PIN key.
- When rule array keyword is K3IPEK, input must be a DES PIN key token or a skeleton token of a DES PIN key left-justified in the field, with one of the PIN control vectors as shown in Table 112 on page 375. On output, this parameter contained the TDES wrapped IPEK in an external symmetric key token or TR-31 key block.
Unique Key Derive (CSNBUKD)

transport_key_identifier_length

Direction: Input
Type: Integer

Length of the transport_key_identifier parameter in bytes. If the transport key identifier is not used, the length must be 0. Otherwise, the length must be 64.

transport_key_identifier

Direction: Input/Output
Type: String

If the K3IPEK keyword is specified, the transport_key_identifier contains the label or key token for the key encrypting key to be used to wrap the IPEK. The transport key must be a DES EXPORTER KEK. Otherwise this field is ignored.

reserved1_length

Direction: Input
Type: Integer

This parameter must be zero.

reserved1

Direction: Ignored
Type: String

This parameter is ignored.

reserved2_length

Direction: Input
Type: Integer

This parameter must be zero.

reserved2

Direction: Ignored
Type: String

This parameter is ignored.

reserved3_length

Direction: Input
Type: Integer

This parameter must be zero.

reserved3

Direction: Ignored
Type: String

This parameter is ignored.

reserved4_length

Direction: Input
Type: Integer
This parameter must be zero.

**reserved4**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>String</td>
</tr>
</tbody>
</table>

This parameter is ignored.

**reserved5_length**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Integer</td>
</tr>
</tbody>
</table>

This parameter must be zero.

**reserved5**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>String</td>
</tr>
</tbody>
</table>

This parameter is ignored.

### Restrictions

The restrictions for CSNBUKD.

Table 112 shows the valid skeleton tokens depending on the key type to be derived.

**Table 112. Valid Control Vectors for Derived Keys.**

<table>
<thead>
<tr>
<th>Key to be derived</th>
<th>Supported key types in the skeleton token</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data encryption key</td>
<td>CIPHER 00 03 71 00 03 41 00 00 00 03 71 00 03 21 00 00</td>
</tr>
<tr>
<td></td>
<td>ENCIPHER 00 03 60 00 03 41 00 00 00 03 60 00 03 21 00 00</td>
</tr>
<tr>
<td></td>
<td>DECIPHER 00 03 50 00 03 41 00 00 00 03 50 00 03 21 00 00</td>
</tr>
<tr>
<td>Message authentication code(MAC)</td>
<td>MAC 00 05 4D 00 03 41 00 00 00 05 4D 00 03 21 00 00</td>
</tr>
<tr>
<td></td>
<td>MACVER 00 05 44 00 03 41 00 00 05 44 00 03 21 00 00</td>
</tr>
<tr>
<td>PIN key</td>
<td>IPINENC 00 21 5F 00 03 41 00 00 21 5F 00 03 21 00 00</td>
</tr>
<tr>
<td></td>
<td>OPINENC 00 24 77 00 03 41 00 00 24 77 00 03 21 00 00</td>
</tr>
<tr>
<td>PIN key with rule keyword PIN-DATA</td>
<td>DATA PIN 00 00 7D 00 03 41 00 00 00 7D 00 03 21 00 00</td>
</tr>
</tbody>
</table>

Note that the following bits of the control vector are not checked and may have a value of either 0 or 1:

- Bit 17 - Export control
- Bit 56 – Enhanced wrapping control
- Bit 57 – TR-31 export control
- Bits 4 and 5 – UDX

Additional control vector bit that is not checked for PIN key with rule keyword PIN-DATA:

- Bit 61 - Not-CCA
**Unique Key Derive (CSNBUKD)**

**Required commands**

The required commands for CSNBUKD.

The Unique Key Derive verb requires the **Unique Key Derive** command (offset X'01C8') to be enabled in the active role.

In addition, these commands are required to be enabled in the active role, depending on the rule-array keyword or keywords:

*Table 113. Required commands for the Symmetric Key Export with Data verb*

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>K3IPEK</td>
<td>X'035</td>
<td>Unique Key Derive - K3IPEK</td>
</tr>
<tr>
<td>PIN-DATA</td>
<td>X'01C9'</td>
<td>Unique Key Derive - Allow PIN-DATA processing</td>
</tr>
<tr>
<td>WRAP-ECB or WRAP-ENH and default key-wrapping method setting does not match keyword</td>
<td>X'01CA'</td>
<td>Unique Key Derive - Override default wrapping</td>
</tr>
</tbody>
</table>

The following access control points control the use of weak transport keys:

- To disallow the import of a key wrapped with a weaker transport key, the **Symmetric Key Import2 - disallow weak import** command (offset X'032B') must be enabled in the active role.
- To receive a warning against the wrapping of a key with a weaker key, the **Warn when weak wrap - Transport keys** command (offset X'032C') must be enabled in the active role. The **Symmetric Key Import2 - disallow weak import** command overrides this command.

The following access control points control the use of weak master keys:

- To disable the wrapping of a key with a weaker master key, the **Prohibit weak wrapping - Master keys** command (offset X'0333') must be enabled in the active role.
- To receive a warning when wrapping a key with a weaker master key, enable the **Warn when weak wrap - Master Keys** command (offset X'0332') in the active role. The **Prohibit weak wrapping - Master keys** command overrides this command.

**Usage notes**

The usage notes for CSNBUKD.

*Table 114 indicates the variants used for each output key type to be derived.*

*Table 114. Derivation variants*

<table>
<thead>
<tr>
<th>Key type</th>
<th>DUKPT derivation variant</th>
<th>DUKPT key usage description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPINENC or OPINENC PIN key (using PIN-DATA rule array keyword)</td>
<td>00000000000000FF 00000000000000FF</td>
<td>PIN Encryption</td>
</tr>
<tr>
<td>MAC</td>
<td>00000000000000FF00 00000000000000FF00</td>
<td>MAC, request or both ways</td>
</tr>
<tr>
<td>MACVER</td>
<td>00000000FF000000 00000000FF000000</td>
<td>MAC, response only</td>
</tr>
</tbody>
</table>
Table 114. Derivation variants (continued)

<table>
<thead>
<tr>
<th>Key type</th>
<th>DUKPT derivation variant</th>
<th>DUKPT key usage description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIPHER ENCIPHER</td>
<td>0000000000FF00000000FF0000</td>
<td>Data Encryption, request or both ways</td>
</tr>
<tr>
<td>DECIPHER</td>
<td>000000FF000000000FF000000</td>
<td>Data Encryption, response only</td>
</tr>
</tbody>
</table>

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBUKDJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBUKDJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber base_derivation_key_identifier_length,
    byte[] base_derivation_key_identifier,
    hikmNativeNumber derivation_data_length,
    byte[] derivation_data,
    hikmNativeNumber generated_key_identifier1_length,
    byte[] generated_key_identifier1,
    hikmNativeNumber generated_key_identifier2_length,
    byte[] generated_key_identifier2,
    hikmNativeNumber generated_key_identifier3_length,
    byte[] generated_key_identifier3,
    hikmNativeNumber transport_key_identifier_length,
    byte[] transport_key_identifier,
    hikmNativeNumber reserved1_length,
    byte[] reserved1,
    hikmNativeNumber reserved2_length,
    byte[] reserved2,
    hikmNativeNumber reserved3_length,
    byte[] reserved3,
    hikmNativeNumber reserved4_length,
    byte[] reserved4,
    hikmNativeNumber reserved5_length,
    byte[] reserved5);```

Unique Key Derive (CSNBUKD)
Unique Key Derive (CSNUKD)
Chapter 9. Protecting data

Use CCA to protect sensitive data stored on your system, sent between systems, or stored off your system on magnetic tape.

To protect data, encipher it under a key. When you want to read the data, decipher it from ciphertext to plaintext form.

CCA provides Encipher and Decipher verbs to perform these functions. If you use a key to encipher data, you must use the same key to decipher the data. The Encipher and Decipher verbs use encrypted keys as input. You can also use clear keys, indirectly, by first using the Clear Key Import verb and then using the Encipher and Decipher verbs.

This topic describes the following verbs used for protecting data using DES or AES:

- "Decipher (CSNBDEC)" on page 381
- "Encipher (CSNBENC)" on page 385
- "Symmetric Algorithm Decipher (CSNBSAD)" on page 390
- "Symmetric Algorithm Encipher (CSNBSAE)" on page 397
- "Cipher Text Translate2 (CSNBCTT2)" on page 404

Modes of operation

Different algorithms are used to encipher or decipher DES data or keys and AES data or keys.

To encipher or decipher DES data or keys, CCA uses the U.S. National Institute of Standards and Technology (NIST) Data Encryption Standard (DES) algorithm, with single-length, double-length, or triple-length keys.

To encipher or decipher AES data or keys, CCA uses the U.S. National Institute of Standards and Technology (NIST) Advanced Encryption Standard (AES) algorithm, with 16-byte, 24-byte or 32-byte keys.

The Encipher and Decipher verbs operate in DES CBC (Cipher Block Chaining) mode.

Cipher Block Chaining (CBC) mode

The CBC mode uses an initial chaining vector (ICV) in its processing.

The CBC mode processes blocks of data only in exact multiples of the blocksize. The ICV is exclusive ORed with the first block of plaintext prior to the encryption step. The block of ciphertext just produced is exclusive-ORed with the next block of plaintext, and so on. You must use the same ICV to decipher the data. This disguises any pattern that may exist in the plaintext. CBC mode is the default for encrypting and decrypting data using the Encipher and Decipher verbs. "Ciphering methods" on page 1026 describes the cipher processing rules in detail.
Electronic Code Book (ECB) mode

In ECB mode, each block of plaintext is separately enciphered and each block of the ciphertext is separately deciphered.

In other words, the encipherment or decipherment of a block is totally independent of other blocks.

Processing rules

You can use different types of processing rules for block chaining.

"Ciphering methods” on page 1026 describes the cipher processing rules in detail.

CCA handles chaining for each block of data, from the first block until the last complete block of data in each Encipher or Symmetric Algorithm Encipher call.

There are different types of processing rules you can choose for block chaining:

ANSI X9.23
- Data is not necessarily in exact multiples of the block size. This processing rule pads the plaintext so the ciphertext produced is in exact multiples of the block size.

Cipher block chaining (CBC)
- Data must be an exact multiple of the block size, and output will have the same length.

Cryptographic Unit Support Program (CUSP)
- CBC mode (cipher block chaining) that is compatible with IBM’s CUSP and PCF products. The data need not be in exact multiples of the block size. The ciphertext is the same length as the plaintext.

Electronic Code Book (ECB)
- The data length must be a multiple of the block size. See “Electronic Code Book (ECB) mode.”

Information Protection System (IPS)
- CBC mode that is compatible with IBM’s IPS product. The data need not be in exact multiples of the block size. The ciphertext is the same length as the plaintext.

PKCS-PAD
- The data is padded on the right with between one and 16 bytes of pad characters, making ciphertext a multiple of the block size.

The resulting chaining value (except for ECB mode), after an Encipher or Symmetric Algorithm Encipher call, is known as an output chaining vector (OCV). When there are multiple cipher requests, the application can pass the OCV from the previous Encipher or Symmetric Algorithm Encipher call, as the input chaining vector (ICV) in the next Encipher or Symmetric Algorithm Encipher call. This produces chaining between successive calls, which is known as record chaining.

CCA provides the ICV selection keyword CONTINUE in the rule_array parameter used to select record chaining with the CBC processing rule.

Triple DES encryption

Triple DES encryption uses a triple-length DATA key comprised of three 8-byte DES keys to encipher eight bytes of data.

To encipher the data it uses the following method:
• Encipher the data using the first key
• Decipher the result using the second key
• Encipher the second result using the third key

The procedure is reversed to decipher data that has been triple-DES enciphered:
• Decipher the data using the third key
• Encipher the result using the second key
• Decipher the second result using the first key

A variation of the triple-DES algorithm supports the use of a double-length DATA key comprised of two 8-byte DATA keys. In this method, the first 8-byte key is reused in the last encipherment step.

Due to export regulations, triple-DES encryption might not be available on your processor.

Decipher (CSNBDEC)

Use the Decipher verb to decipher data using the DES cipher block chaining mode.

CCA supports the following processing rules to decipher data. You choose the type of processing rule that the Decipher verb should use for block chaining.

ANSI X9.23
For cipher block chaining. The ciphertext must be an exact multiple of eight bytes, but the plaintext will be between 1 and 8 bytes shorter than the ciphertext. The text_length will also be reduced to show the original length of the plaintext.

Cipher Block Chaining (CBC)
The ciphertext must be an exact multiple of eight bytes and the plaintext will have the same length.

Cryptographic Unit Support Program (CUSP)
CBC mode (cipher block chaining) that is compatible with IBM’s CUSP and PCF products. The data need not be in exact multiples of eight bytes. The ciphertext is the same length as the plaintext.

Information Protection System (IPS)
CBC mode (cipher block chaining) that is compatible with IBM’s IPS product. The data need not be in exact multiples of eight bytes. The ciphertext is the same length as the plaintext.

The cipher block chaining (CBC) mode uses an initial chaining value (ICV) in its processing. The first eight bytes of ciphertext is deciphered and then the ICV is XORed with the resulting eight bytes of data to form the first 8-byte block of plaintext. Thereafter, the 8-byte block of ciphertext is deciphered and XORed with the previous 8-byte block of ciphertext until all the ciphertext is deciphered.

The selection between single-DES decryption mode and triple-DES decryption mode is controlled by the length of the key supplied in the key_identifier parameter. If a single-length key is supplied, single-DES decryption is performed. If a double-length or triple-length key is supplied, triple-DES decryption is performed.

A different ICV could be passed on each call to the Decipher verb. However, the same ICV that was used in the corresponding Encipher verb must be passed.
Decipher (CSNBDEC)

Short blocks are text lengths of between one and seven bytes. A short block can be the only block. Trailing short blocks are blocks of between one and seven bytes that follow an exact multiple of eight bytes. For example, if the text length is 21, there are two 8-byte blocks and a trailing short block of five bytes. Because the DES processes text only in exact multiples of eight bytes, some special processing is required to decipher such short blocks.

These methods of treating short blocks and trailing short blocks do not increase the length of the ciphertext compared to the length of the plaintext. If the plaintext was padded during encipherment, the length of the ciphertext will always be an exact multiple of eight bytes.

CCA supports the ANSI X9.23 padding method.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.

Host CPU acceleration: CPACF

Only keys with a key type of DATA can be used successfully with the CPACF exploitation layer through this verb.

Specifically, a DATA key has a CV (Control Vector) of all X'00' bytes for all active bytes of the CV (eight bytes for 8-byte DES keys, 16 bytes for 16-byte DES keys, and 16 bytes for 24-byte DES keys).

For details about CPACF, see "CPACF support" on page 14.

Format

The format of CSNBDEC.

```
CSNBDEC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier,
    text_length,
    cipher_text,
    initialization_vector,
    rule_array_count,
    rule_array,
    chaining_vector,
    clear_text)
```

Parameters

The parameters for CSNBDEC.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see "Parameters common to all verbs" on page 22.

key_identifier

Direction: Input/Output
Type: String

A 64-byte string that is the internal key token containing the data-encrypting key or the label of a DES key storage record containing a data-encrypting key to be used for deciphering the data. If the key token or key label contains a
single-length key, single-DES decryption is performed. If the key token or key label contains a double-length or triple-length key, triple-DES decryption is performed.

Double length CIPHER and DECIPHER keys are also supported.

text_length

Direction: Input/Output
Type: Integer

On entry, you supply the length of the ciphertext. The maximum length of text is 214,783,647 bytes. A zero value for the text_length parameter is not valid. If the returned deciphered text (clear_text parameter) is a different length because of the removal of padding bytes, the value is updated to the length of the plaintext.

The application program passes the length of the ciphertext to the verb. The verb returns the length of the plaintext to your application program.

cipher_text

Direction: Input
Type: String

The text to be deciphered.

initialization_vector

Direction: Input
Type: String

The 8-byte supplied string for the cipher block chaining. The first block of the ciphertext is deciphered and XORed with the initial chaining vector (ICV) to get the first block of cleartext. The input block is the next ICV. To decipher the data, you must use the same ICV used when you enciphered the data.

rule_array_count

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 1, 2, or 3.

rule_array

Direction: Input
Type: String array

An array of 8-byte keywords providing the processing control information. The array is positional. The first keyword in the array is the processing rule. You choose the processing rule you want the verb to use for deciphering the data. The second keyword is the ICV selection keyword. The third keyword (or the second if the ICV selection keyword is allowed to default) is the encryption algorithm to use. The rule_array keywords are described in Table 115.

Table 115. Keywords for Decipher control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Rule</td>
<td>(One, required)</td>
</tr>
</tbody>
</table>
Decipher (CSNBDEC)

Table 115. Keywords for Decipher control information  (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBC</td>
<td>Performs ANSI X3.102 cipher block chaining. The data must be a multiple of eight bytes. An OCV is produced and placed in the chaining_vector parameter. If the ICV selection keyword CONTINUE is specified, the CBC OCV from the previous call is used as the ICV for this call.</td>
</tr>
<tr>
<td>CUSP</td>
<td>Performs Cryptographic Unit Support Program (CUSP) cipher block chaining.</td>
</tr>
<tr>
<td>IPS</td>
<td>Performs Information Protection System (IPS) cipher block chaining.</td>
</tr>
<tr>
<td>X9.23</td>
<td>Deciphers with cipher block chaining and text length reduced to the original value. This is compatible with the requirements in ANSI standard X9.23. The ciphertext length must be an exact multiple of eight bytes. Padding is removed from the plaintext.</td>
</tr>
</tbody>
</table>

ICV Selection (One, optional)

| CONTINUE | This specifies taking the initialization vector from the output chaining vector (OCV) contained in the work area to which the chaining_vector parameter points. CONTINUE is valid only for the CBC processing rule. |
| INITIAL | This specifies taking the initialization vector from the initialization_vector parameter. INITIAL is the default value. |

Encryption algorithm (Optional)

| DES | This specifies using the data encryption standard and ignoring the token marking. |

“Ciphering methods” on page 1026 describes the cipher processing rules in detail.

chaining_vector

Direction: Input/Output
Type: String

An 18-byte field CCA uses as a system work area. Your application program must not change the data in this string. The chaining vector holds the output chaining vector (OCV) from the caller. The OCV is the first eight bytes in the 18-byte string.

The direction is Output if the ICV selection keyword of the rule_array parameter is INITIAL. The direction is Input/Output if the ICV selection keyword of the rule_array parameter is CONTINUE.

clear_text

Direction: Output
Type: String

The field where the verb returns the deciphered text.

Restrictions

The restrictions for CSNBDEC.

This verb fails if the key token contains double or triple-length keys and triple-DES is not enabled.

Required commands

The required commands for CSNBDEC.

This verb requires the Decipher - DES command (offset X'000F') to be enabled in the active role.
Usage notes

The usage notes for CSNBDEC.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBDECJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBDECJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] key_identifier,
    hikmNativeNumber text_length,
    byte[] cipher_text,
    byte[] initialization_vector,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] chaining_vector,
    byte[] clear_text);
```

Encipher (CSNBENC)

Use the Encipher verb to encipher data using the DES cipher block chaining mode.

CCA supports the following processing rules to encipher data. You choose the type of processing rule that the Encipher verb should use for the block chaining.

Cipher block chaining (CBC)

In exact multiples of eight bytes.

Cryptographic Unit Support Program (CUSP)

CBC mode (cipher block chaining) that is compatible with IBM’s CUSP and PCF products. The data need not be in exact multiples of eight bytes. The ciphertext is the same length as the plaintext.

Information Protection System (IPS)

CBC mode (cipher block chaining) that is compatible with IBM’s IPS product. The data need not be in exact multiples of eight bytes. The ciphertext is the same length as the plaintext.

ANSI X9.23

For block chaining not necessarily in exact multiples of eight bytes. This process rule pads the plaintext so that ciphertext produced is an exact multiple of eight bytes.

For more information about the processing rules, see Table 116 on page 385 and “Ciphering methods” on page 1026.

The cipher block chaining (CBC) mode of operation uses an initial chaining vector (ICV) in its processing. The ICV is XORed with the first eight bytes of plaintext before the encryption step and thereafter, the 8-byte block of ciphertext just produced is XORed with the next 8-byte block of plaintext and so on. This disguises any pattern that might exist in the plaintext.
The selection between single-DES encryption mode and triple-DES encryption mode is controlled by the length of the key supplied in the key_identifier parameter. If a single-length key is supplied, single-DES encryption is performed. If a double-length or triple-length key is supplied, triple-DES encryption is performed.

To nullify the CBC effect on the first 8-byte block, supply eight bytes of zero. However, the ICV might require zeros.

Cipher block chaining also produces a resulting chaining value called the output chaining vector (OCV). The application can pass the OCV as the ICV in the next encipher call. This results in record chaining.

Note that the OCV that results is the same, whether an Encipher or a Decipher verb was invoked, assuming the same text, ICV, and key were used.

Short blocks are text lengths of between one and seven bytes. A short block can be the only block. Trailing short blocks are blocks of between one and seven bytes that follow an exact multiple of eight bytes. For example, if the text length is 21, there are two 8-byte blocks, and a trailing short block of five bytes.

An alternative method is to pad the plaintext and produce a ciphertext that is longer than the plaintext. The plaintext can be padded with up to eight bytes using one of several padding methods. This padding produces a ciphertext that is an exact multiple of eight bytes in length.

If the cleartext is already a multiple of eight, the ciphertext can be created using any processing rule.

Because of padding, the returned ciphertext length is longer than the provided plaintext. Therefore, the text_length parameter is modified. The returned ciphertext field should be eight bytes longer than the length of the plaintext to accommodate the maximum amount of padding.

Attention: If you lose the data-encrypting key under which the data (plaintext) is enciphered, the data enciphered under that key (ciphertext) cannot be recovered.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.

Host CPU acceleration: CPACF

Only keys with a key type of DATA can be used successfully with the CPACF exploitation layer through this verb.

Specifically, a DATA key has a CV (Control Vector) of all X'00' bytes for all active bytes of the CV (eight bytes for 8-byte DES keys, 16 bytes for 16-byte DES keys, and 16 bytes for 24-byte DES keys).

For details about CPACF, see "CPACF support" on page 14.
Format

The format of CSNBENC.

```
CSNBENC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier,
    text_length,
    clear_text,
    initialization_vector,
    rule_array_count,
    rule_array,
    pad_character,
    chaining_vector,
    cipher_text)
```

Parameters

The parameters for CSNBENC.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**key_identifier**

- **Direction:** Input/Output
- **Type:** String

A 64-byte string that is the internal key token containing the data-encrypting key or the label of a DES key storage record containing the data-encrypting key, to be used for encrypting the data. If the key token or key label contains a single-length key, single-DES encryption is performed. If the key token or key label contains a double-length or triple-length key, triple-DES encryption is performed.

Single and double-length CIPHER and ENCIPHER keys are also supported.

**text_length**

- **Direction:** Input/Output
- **Type:** Integer

On entry, the length of the plaintext (`clear_text` parameter) you supply. The maximum length of text is 214,783,647 bytes. A zero value for the `text_length` parameter is not valid. If the returned enciphered text (`cipher_text` parameter) is a different length because of the addition of padding bytes, the value is updated to the length of the ciphertext.

The application program passes the length of the plaintext to the verb. This verb returns the length of the ciphertext to the application program.

**clear_text**

- **Direction:** Input
- **Type:** String

The text that is to be enciphered.

**initialization_vector**

- **Direction:** Input
Type: String

The 8-byte supplied string for the cipher block chaining. The first eight bytes (or less) block of the data is XORed with the ICV and then enciphered. The input block is enciphered and the next ICV is created. You must use the same ICV to decipher the data.

**rule_array_count**

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 1, 2, or 3.

**rule_array**

Direction: Input
Type: String array

An array of 8-byte keywords providing the processing control information. The array is positional. The first keyword in the array is the processing rule. You choose the processing rule you want the verb to use for enciphering the data. The second keyword is the ICV selection keyword. The third keyword (or the second if the ICV selection keyword is allowed to default to INITIAL) is the encryption algorithm to use. The rule_array keywords are described in Table 116.

**Table 116. Keywords for Encipher control information**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processing Rule (1, required)</strong></td>
<td></td>
</tr>
<tr>
<td>CBC</td>
<td>Performs ANSI X3.102 cipher block chaining. The data must be a multiple of 8 bytes. An OCV is produced and placed in the chaining_vector parameter. If the ICV selection keyword CONTINUE is specified, the CBC OCV from the previous call is used as the ICV for this call.</td>
</tr>
<tr>
<td>CUSP</td>
<td>Performs Cryptographic Unit Support Program (CUSP) cipher block chaining.</td>
</tr>
<tr>
<td>IPS</td>
<td>Performs Information Protection System (IPS) cipher block chaining.</td>
</tr>
<tr>
<td>X9.23</td>
<td>Performs cipher block chaining with 1 - 8 bytes of padding. This is compatible with the requirements in ANSI X9.23. If the data is not in exact multiples of eight bytes, X9.23 pads the plaintext so the ciphertext produced is an exact multiple of 8 bytes. The plaintext is padded to the next multiple eight bytes, even if 8 bytes are added. An OCV is produced.</td>
</tr>
<tr>
<td><strong>ICV Selection (1, optional)</strong></td>
<td></td>
</tr>
<tr>
<td>CONTINUE</td>
<td>Specifies to take the initialization vector from the output chaining vector (OCV) contained in the work area to which the chaining_vector parameter points. CONTINUE is valid only for the CBC processing rule.</td>
</tr>
<tr>
<td>INITIAL</td>
<td>Specifies to take the initialization vector from the initialization_vector parameter. INITIAL is the default value.</td>
</tr>
<tr>
<td><strong>Encryption Algorithm (Optional)</strong></td>
<td></td>
</tr>
<tr>
<td>DES</td>
<td>Specifies to use the data encryption standard and to ignore the token marking.</td>
</tr>
</tbody>
</table>

“Ciphering methods” on page 1026 describes the cipher processing rules in detail.

**pad_character**

Direction: Input
Type: Integer
An integer between 0 and 255 that is used as a padding character for the X9.23 process rule (rule_array parameter).

**chaining_vector**

**Direction:** Input/Output  
**Type:** String

An 18-byte field CCA uses as a system work area. Your application program must not change the data in this string. The chaining vector holds the output chaining vector (OCV) from the caller. The OCV is the first eight bytes in the 18-byte string.

The direction is Output if the ICV selection keyword of the rule_array parameter is **INITIAL**.

The direction is Input/Output if the ICV selection keyword of the rule_array parameter is **CONTINUE**.

**cipher_text**

**Direction:** Output  
**Type:** String

The enciphered text the verb returns. The length of the ciphertext is returned in the text_length parameter. The cipher_text could be eight bytes longer than the length of the clear_text field because of the padding that is required for some processing rules.

**Restrictions**

The restrictions for CSNBENC.

This verb will fail if the key token contains double-length or triple-length keys and triple-DES is not enabled.

**Required commands**

The required commands for CSNBENC.

This verb requires the *Encipher - DES* command (offset X'000E') to be enabled in the active role.

**Usage notes**

The usage notes for CSNBENC.

None.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBENCJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBENCJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] key_identifier,
```

Encipher (CSNBENC)
Symmetric Algorithm Decipher (CSNBSAD)

Use the Symmetric Algorithm Decipher verb to decipher data with an Advanced Encryption Standard (AES) algorithm.

Data can be deciphered in either Cipher Block Chaining (CBC) mode with or without padding, or in Electronic Code Book (ECB) mode. Beginning with Release 5.2, data can be deciphered in Galois/Counter Mode (GCM). Also see “Symmetric Algorithm Encipher (CSNBSAE)” on page 397.

CCA supports the following processing rules to decipher data. You choose the type of processing rule that the verb should use for block chaining.

Cipher Block Chaining (CBC)
The plaintext must be an exact multiple of eight bytes, and the ciphertext will have the same length.

Electronic Code Book (ECB)
The plaintext length must be a multiple of the block size.

Galois/Counter Mode (GCM)
The plaintext can be deciphered in Galois/Counter Mode (GCM). For more information on GCM, read NIST SP 800-38D Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, November 2007. Available at http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf.

Public Key Cryptography Standards Pad (PKCS-PAD)
The plaintext was padded on the right with 1 - 16 bytes of pad characters, making the padded text a multiple of the block size.

The AES key used to decipher the data can either be 16, 24, or 32 bytes (128, 192, or 256 bits) in length. The key can be supplied to the verb in any of three forms:
1. A cleartext key consisting of only the key bytes, not contained in a key token.
2. A cleartext key contained in an internal fixed or variable length AES key-token.
3. An encrypted key contained in an internal fixed or variable length AES key-token, where the key is wrapped (encrypted) with the AES master key.

To use this verb, specify:
- The rule_array parameter:
  1. The algorithm identifier keyword AES, which is the only symmetric algorithm currently supported.
  2. An optional processing rule using keyword CBC (the default), ECB, GCM, or PKCS-PAD, which selects the decryption mode.
  3. An optional key rule using the keyword KEY-CLR (the default) or KEYIDENT, which selects whether the key_identifier parameter points to a
16-byte, 24-byte, or 32-byte clear key, or a key contained in a 64-byte AES key-token, either in application storage or a key label of such a key in key storage.

4. For processing rule CBC or ECB, specify an optional ICV (initial chaining value) selection using the keyword INITIAL (the default). Or use keyword CONTINUE, which indicates whether it is the first or a subsequent request, and which parameter points to the initialization vector.

For processing rule GCM, specify an optional ICV keyword ONLY (default).

- For a key rule of KEY-CLR, a key identifier containing a 16-byte, 24-byte, or 32-byte clear key. For a key rule of KEYIDENT, a fixed-length or variable-length internal AES key-token or the key label of such a key in AES key-storage. The key token can contain either a clear or enciphered key.

A variable-length AES key-token must have a key type of CIPHER that can be used for decryption (key-usage field 1 high-order byte = B'x1xx xxxx') and cannot be used for data translation (KUF1 high-order byte = B'xx0x xxxx'). Also, for processing rule CBC or PKCS-PAD, key usage must allow the key to be used for Cipher Block Chaining (KUF2 high-order byte = X'00' or X'FF').

For processing rule ECB, key usage must allow the key to be used for Electronic Code Book (KUF2 high-order byte = X'01' or X'FF').

For processing rule GCM, the key usage must allow the key to be used for Galois/Counter mode (KUF2 high-order byte = X'04' or X'FF').

- A block size of 16 for the cryptographic algorithm.

- For cipher block chaining, specify one of these:
  1. For an ICV selection of INITIAL, a 16-byte initialization vector of your choosing and a 32-byte chain data buffer.
  2. For an ICV selection of CONTINUE, no initialization vector and the 32-byte chain data buffer from the output of the previous chained call. The electronic code book algorithm does not use an initialization vector or a chain data buffer.
  3. For an ICV selection of ONLY, an initialization vector greater than 0 and a maximum of $2^{32} - 1$, and a 104-byte chain data buffer.

- The ciphertext to be deciphered.

- A cleartext buffer large enough to receive the deciphered output.

This verb does the following when it deciphers the data:

1. Verifies the AES key-token for keyword KEYIDENT.
2. Verifies that the ciphertext length is a multiple of the block size.
3. Deciphers the input AES key if the key is encrypted (MKVP was present in token).
4. Deciphers the ciphertext with the AES clear key according to the encryption mode specified.
5. Removes from 1 - 16 pad characters from the right of the clear data for keyword PKCS-PAD.
6. Returns the cleartext data and its length.
7. Returns the chain data and its length if keyword ECB is not specified.

CPACF exploitation for CSNBSAD requires fixed-length AES key-tokens with a key type of DATA or variable-length AES key-tokens of type CIPHER with DECRYPT key-usage enabled. A fixed-length AES DATA key has a control vector (CV) of all X'00' bytes for all active bytes of the CV. For details about CPACF, see "CPACF support" on page 14.
Symmetric Algorithm Decipher (CSNBSAD)

Format
The format of CSNBSAD.

```
CSNBSAD(  return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    key_identifier_length,  
    key_identifier,  
    key_parms_length,  
    key_parms,  
    block_size,  
    initialization_vector_length,  
    initialization_vector,  
    chain_data_length,  
    chain_data,  
    cipher_text_length,  
    cipher_text,  
    clear_text_length,  
    clear_text,  
    optional_data_length,  
    optional_data)
```

Parameters
The parameters for CSNBSAD.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

rule_array_count
Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1, 2, 3, or 4.

rule_array
Direction: Input
Type: String array

An array of 8-byte keywords providing the processing control information. The keywords must be left-aligned and padded on the right with space characters. The `rule_array` keywords are described in Table 117.

Table 117. Keywords for Symmetric Algorithm Decipher control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Decryption algorithm</strong> (One required)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>Specifies use of the Advanced Encryption Standard (AES) as the deciphering algorithm. The block size for AES is 16 bytes, and the key length is 16, 24, or 32 bytes. AES is the only algorithm currently supported by this verb.</td>
</tr>
<tr>
<td><strong>Processing rule</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>CBC</td>
<td>Performs ANSI X3.102 cipher block chaining. The data must be a multiple of eight bytes. An OCV is produced and placed in the <code>initialization_vector</code> parameter. If the ICV selection keyword <code>CONTINUE</code> is specified, the CBC OCV from the previous call is used as the ICV for this call.</td>
</tr>
</tbody>
</table>
Table 117. Keywords for Symmetric Algorithm Decipher control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECB</td>
<td>Specifies deciphering in Electronic Code Book mode. The ciphertext length must be a multiple of the block size.</td>
</tr>
<tr>
<td>GCM</td>
<td>Specifies decryption in Galois/Counter Mode. The ciphertext length must be a multiple of the algorithm block size and less than or equal to (2^{32} - 1).</td>
</tr>
<tr>
<td>PKCS-PAD</td>
<td>Specifies that the cleartext was padded on the right with 1 - 16 bytes of pad characters, making the padded text a multiple of the block size, before the data was enciphered. Each pad character is valued to the number of pad characters added. The cleartext length must be greater than 0. The output cleartext is stripped of any pad characters and the cleartext length is 1 - 16 bytes less than the ciphertext length.</td>
</tr>
</tbody>
</table>

*Key rule* (One, optional)

| KEY-CLR | Specifies that the `key_identifier` parameter points to a cleartext AES key. Only the key value is allowed; the key is not contained in a key token. This is the default value. |
| KEYIDENT| Specifies that the `key_identifier` parameter points to an internal AES key-token or the label of an internal key-token in AES key-storage. |

*Initial chaining value (ICV) selection* (One, optional)

| CONTINUE | This specifies taking the initialization vector from the output chaining vector (OCV) contained in the work area to which the `chaining_vector` parameter points. This keyword is not valid with the ECB or GCM processing rule keyword. |
| INITIAL  | This specifies taking the initialization vector from the `initialization_vector` parameter. Not valid with the GCM processing rule keyword. Otherwise this is the default. |
| ONLY     | Specifies that this is the only request. The initialization vector is used as input to decipher the block of data, and must be the same value used to encipher the ciphertext. Only valid with GCM processing rule keyword. This is the default for GCM. |

“Ciphering methods” on page 1026 describes the cipher processing rules in detail.

**key_identifier_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the `key_identifier` variable. This value must be 16, 24, 32, or \(\geq 64\).

**key_identifier**

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing either a cleartext AES key or the internal key-token or a label for an internal key-token record in AES key-storage. This is the key used to decipher the data pointed to by the `ciphertext` parameter.

For `rule_array` keyword `KEY-CLR`, a 16-byte, 24-byte, or 32-byte clear AES key is required. For `rule_array` keyword `KEYIDENT`, a fixed-length or a variable-length internal AES key-token or key label for such a key in AES key-storage is required.
A variable-length AES key-token must have a key type of CIPHER and must allow the key to be used for decryption (key-usage field 1 high-order byte = B’x1xx xxxx’). In addition, the key token must have the following key usage based on processing rule keyword:

- **CBC** must allow the key to be used for Cipher Block Chaining (KUF2 high-order byte = X’00’ or X’FF’).
- **ECB** must allow the key to be used for Electronic Code Book (KUF2 high-order byte = X’01’ or X’FF’).
- **GCM** must allow the key to be used for Galois/Counter mode (KUF2 high-order byte = X’04’ or X’FF’).

**key_parms_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the **key_parms** parameter. For processing rule GCM, this value can be 4 or 8 (which is strongly discouraged), or 12 - 16. Otherwise, this value must be 0.

**key_parms**

- **Direction:** Input
- **Type:** String

A pointer to a string variable for key-related parameters. For processing rule GCM, this variable contains the verified authentication tag for the data identified by the ciphertext parameter and any additional authenticated data identified by the optional_data parameter. No other usage is currently defined.

**block_size**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the block size used by the cryptographic algorithm. This value must be 16.

**initialization_vector_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the **initialization_vector** variable. For cipher block chaining (CBC or PKCS-PAD) with an INITIAL ICV selection, this value must be 16. For processing rule GCM, NIST recommends a length of 12, but any length from 1 to a maximum of \(2^{32} - 1\) can be used. Otherwise, set this value to 0.

**initialization_vector**

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing the initialization vector for the INITIAL call to CBC mode decryption, or if the ICV selection is ONLY. It is not used if the processing rule is ECB or the ICV selection is CONTINUE. The same initialization vector must have been used to encipher the data.

**chain_data_length**

- **Direction:** Input/Output
Symmetric Algorithm Decipher (CSNBSAD)

Type: Integer

A pointer to an integer variable containing the number of bytes of data in the chain_data variable. On input, set this variable to a value of at least 32 for CBC mode decryption, 0 for ECB mode encryption, or 104 for GCM mode decryption.

On output, the variable is updated with the length of the data returned in the chain_data variable. The chain_data_length parameter must not be changed by the calling application until chained operations are complete.

chain_data

Direction: Input/Output
Type: String

A pointer to a string variable used as a work area for CBC encipher requests. This work area is not used for ECB mode decryption or 104 for GCM mode decryption.

When the verb performs a CBC decipher operation and the ICV selection is INITIAL, the chain_data variable is an output-only buffer that receives data used as input for deciphering the next part of the input data, if any.

When the ICV selection is CONTINUE, the chain_data variable is both an input and output buffer.

When the ICV selection is ONLY, the chain_data variable is an output-only buffer that receives data in the event that the amount of ciphertext is greater than the host code can send to the coprocessor in a single call.

The application must not change any intermediate data in this string.

ciphertext_length

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the ciphertext variable. The ciphertext_length value must be a multiple of the algorithm block size. For processing rule GCM, the value can be a minimum of 0 up to a maximum of $2^{32} - 1$, otherwise the value must not be 0. If PKCS-PAD is specified, set the output cleartext_length variable from 1 - 16 bytes less than the ciphertext_length value.

Note: Do not make the ciphertext_length and cleartext_length parameters point to the same variable.

ciphertext

Direction: Input
Type: String

A pointer to a string variable containing the data to be deciphered, including any pad bytes.

cleartext_length

Direction: Input/Output
Type: Integer

On input, this parameter is a pointer to an integer variable containing the number of bytes of data in the cleartext variable. On output, this variable is updated to contain the actual length of text output in the cleartext variable. If
Symmetric Algorithm Decipher (CSNBSAD)

PKCS-PAD is specified, the cleartext value is updated with 1 - 16 bytes of data less than the ciphertext_length value.

cleartext

Direction: Input/Output
Type: String

A pointer to a string variable used to contain the data to be deciphered, excluding any pad bytes.

optional_data_length

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the optional_data variable. For processing rule GCM, set this value to a minimum of 0 up to a maximum of $2^{32} - 1$, otherwise set this value to 0.

optional_data

Direction: Input
Type: String

A pointer to a string variable containing optional data for the decryption. For processing rule GCM, this parameter identifies any additional authenticated data (AAD). No other usage is currently defined.

Restrictions

The restrictions for CSNBSAD.

None.

Required commands

The required commands for CSNBSAD.

The Symmetric Algorithm Decipher verb requires the command Symmetric Algorithm Decipher - secure AES keys (offset X'012B') to be enabled in the active role. The verb also requires the command shown in the following table to be enabled in the active role based on rule-array keyword:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCM</td>
<td>X'01CE'</td>
<td>Symmetric Algorithm Decipher - AES GCM</td>
</tr>
</tbody>
</table>

Usage notes

The usage notes for CSNBSAD.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBSADJ.

See "Building Java applications using the CCA JNI" on page 28.
Symmetric Algorithm Decipher (CSNBSAD)

Format

```java
public native void CSNBSADJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber key_identifier_length,
    byte[] key_identifier,
    hikmNativeNumber key_parms_length,
    byte[] key_parms,
    hikmNativeNumber block_size,
    hikmNativeNumber initialization_vector_length,
    byte[] initialization_vector,
    hikmNativeNumber chain_data_length,
    byte[] chain_data,
    hikmNativeNumber cipher_text_length,
    byte[] cipher_text,
    hikmNativeNumber clear_text_length,
    byte[] clear_text,
    hikmNativeNumber optional_data_length,
    byte[] optional_data);
```

Symmetric Algorithm Encipher (CSNBSAE)

Use the Symmetric Algorithm Encipher verb to encipher data using the AES algorithm.

CCA supports the following processing rules to encipher data. You choose the type of processing rule that the verb should use for block chaining.

**Cipher Block Chaining (CBC)**

The plaintext must be an exact multiple of eight bytes, and the ciphertext will have the same length.

**Electronic Code Book (ECB)**

The plaintext length must be a multiple of the block size.

**Galois/Counter Mode (GCM)**

The plaintext can be enciphered in Galois/Counter Mode (GCM). See also “Symmetric Algorithm Decipher (CSNBSAD)” on page 390. For more information on GCM, read NIST SP 800-38D Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, November 2007. Available at [http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf](http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf)

**Public Key Cryptography Standards Pad (PKCS-PAD)**

The plaintext was padded on the right with 1 - 16 bytes of pad characters, making the padded text a multiple of the block size.

The AES key used to encipher the data can either be 16, 24, or 32 bytes (128, 192, or 256 bits) in length. The key can be supplied to the verb in any of three forms:

1. A cleartext key consisting of only the key bytes, not contained in a key token.
2. A cleartext key contained in an internal fixed or variable length AES key-token.
3. An encrypted key contained in an internal fixed or variable length AES key-token, where the key is wrapped (encrypted) with the AES master key.

To use this verb, specify:

- The **rule_array**:
Symmetric Algorithm Encipher (CSNBSAE)

1. The algorithm identifier keyword **AES**, which is the only symmetric algorithm currently supported.
2. An optional processing rule using keyword **CBC** (the default), **ECB**, **GCM**, or **PKCS-PAD**, which selects the encryption mode.
3. An optional key rule using the keyword **KEY-CLR** (the default) or **KEYIDENT**, which selects whether the *key_identifier* parameter points to a 16-byte, 24-byte, or 32-byte clear key, or a key contained in a 64-byte AES key-token in application storage or a key label of such a key in AES key-storage
4. For processing rule **CBC** or **ECB**, an optional ICV (initial chaining value) selection using the keyword **INITIAL** (the default) or **CONTINUE**, which indicates whether it is the first or a subsequent request, and which parameter points to the initialization vector. For processing rule **GCM**, specify an optional ICV keyword **ONLY** (default).
   - For a key rule of **KEY-CLR**, a key identifier containing a 16-byte, 24-byte, or 32-byte clear key. For a key rule of **KEYIDENT**, a fixed-length or variable-length internal AES key-token or the key label of such a key in AES key-storage. The key token can contain either a clear or enciphered key.
     A variable-length AES key-token must have a key that can be used for encryption (key-usage field 1 high-order byte is B’x1xx xxxx’) and cannot be used for data translation (KUF1 high-order byte = B’xx0x xxxx’). Also, for processing rule **CBC** or **PKCS-PAD**, key usage must allow the key to be used for Cipher Block Chaining (KUF2 high-order byte is X’00’ or X’FF’). For processing rule **ECB**, key usage must allow the key to be used for Electronic Code Book (KUF2 high-order byte is X’01’ or X’FF’).
     For processing rule **GCM**, the key usage must allow the key to be used for Galois/Counter mode (KUF2 high-order byte = X’04’ or X’FF’).
   - A block size of 16 for the cryptographic algorithm.
   - For cipher block chaining, either one of these:
     1. For an ICV selection of **INITIAL**, a 16-byte initialization vector of your choosing and a 32-byte chain data buffer.
     2. For an ICV selection of **CONTINUE**, no initialization vector and the 32-byte chain data buffer from the output of the previous chained call. The electronic code book algorithm does not use an initialization vector or a chain data buffer.
     3. For an ICV selection of **ONLY**, an initialization vector greater than 0 and a maximum of $2^{32} - 1$, and a 104-byte chain data buffer.
     - The cleartext to be enciphered.
     - A ciphertext buffer large enough to receive the enciphered output.

This verb does the following when it enciphers the data:
1. Verifies the AES key-token for keyword **KEYIDENT**.
2. Deciphers the input AES key if the key is encrypted (MKVP was present in token).
3. Pads the cleartext data with 1 - 16 bytes on the right for keyword **PKCS-PAD**, otherwise verifies that the cleartext length is a multiple of the block size.
4. Enciphers the cleartext, including any pad characters, with the AES clear key according to the encryption mode specified.
5. Returns the ciphertext data and its length.
6. Returns the chain data and its length if keyword **ECB** is not specified.
7. Constructs an authentication tag in the key_parms variable if the GCM processing rule is specified in the rule array.

CPACF exploitation for CSNBSAE requires fixed-length AES key-tokens with a key type of DATA or variable-length AES key-tokens of type CIPHER with ENCRYPT key-usage enabled. A fixed-length AES DATA key has a control vector (CV) of all X'00' bytes for all active bytes of the CV. For details about CPACF, see “CPACF support” on page 14.

Format

The format of CSNBSAE.

```
CSNBSAE(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  key_identifier_length,
  key_identifier,
  key_parms_length,
  key_parms,
  block_size,
  initialization_vector_length,
  initialization_vector,
  chain_data_length,
  chain_data,
  clear_text_length,
  clear_text,
  cipher_text_length,
  cipher_text,
  optional_data_length,
  optional_data)
```

Parameters

The parameters for CSNBSAE.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 1, 2, 3, or 4.

**rule_array**

Direction: Input
Type: String array

A pointer to a string variable containing an array of keywords. The keywords are eight bytes in length, and must be left-aligned and padded on the right with space characters. The rule_array keywords are described in Table 118 on page 400.
## Table 118. Keywords for Symmetric Algorithm Encipher control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Encryption algorithm</strong> (Required)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>Specifies use of the Advanced Encryption Standard (AES) as the encryption algorithm. The block size for AES is 16 bytes, and the key length is 16, 24, or 32 bytes. AES is the only algorithm currently supported by this verb.</td>
</tr>
<tr>
<td><strong>Processing rule</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>CBC</td>
<td>Performs ANSI X3.102 cipher block chaining. The data must be a multiple of eight bytes. An OCV is produced and placed in the <code>chaining_vector</code> parameter. If the ICV selection keyword CONTINUE is specified, the CBC OCV from the previous call is used as the ICV for this call.</td>
</tr>
<tr>
<td>ECB</td>
<td>Specifies enciphering in Electronic Code Book mode. The cleartext length must be a multiple of the block size.</td>
</tr>
<tr>
<td>GCM</td>
<td>Specifies encryption in Galois/Counter Mode. The cleartext length must be less than or equal to $2^{32} - 1$.</td>
</tr>
<tr>
<td>PKCS-PAD</td>
<td>Specifies padding of the cleartext on the right with 1 - 16 bytes of pad characters, making the padded text a multiple of the block size. Each pad character is valued to the number of pad characters added. The ciphertext length must be large enough to include the added pad characters. The ciphertext length must be large enough to include the added pad characters. The padded cleartext is enciphered in Cipher Block Chaining mode.</td>
</tr>
<tr>
<td><strong>Key rule</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>KEY-CLR</td>
<td>Specifies that the <code>key_identifier</code> parameter points to a cleartext AES key. Only the key value is allowed; the key is not contained in a key token. This is the default value.</td>
</tr>
<tr>
<td>KEYIDENT</td>
<td>Specifies that the <code>key_identifier</code> parameter points to an internal AES key-token or the label of an internal key-token in AES key-storage.</td>
</tr>
<tr>
<td><strong>ICV selection</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>CONTINUE</td>
<td>Specifies that the request is part of a sequence of chained requests and is not the first (initial) request in the sequence. The chain data is used as input to encipher the next block of data. The chain data buffer is updated with output that is used as input to subsequent chained requests. This keyword is not valid with the ECB or GCM processing rule keyword.</td>
</tr>
<tr>
<td>INITIAL</td>
<td>Specifies that this is either the first request of a sequence of chained requests, or the only request. The initialization vector is used as input to encipher the first block of data. The chain data buffer is updated with output that is used as input to subsequent chained requests. Not valid with the GCM processing rule keyword, otherwise this is the default.</td>
</tr>
<tr>
<td>ONLY</td>
<td>Specifies that this is the only request. The initialization vector is used as input to encipher the block of data. Only valid with the GCM processing rule keyword. This is the default for GCM.</td>
</tr>
</tbody>
</table>

"Ciphering methods" on page 1026 describes the cipher processing rules in detail.

**key_identifier_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the `key_identifier` variable. This value must be 16, 24, 32, or ≥ 64.

**key_identifier**

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing either a cleartext AES key or the
internal key-token or a label for an internal key-token record in AES key-storage. This is the key used to encipher the data pointed to by the cleartext parameter. For rule_array keyword KEY-CLR, a 16-byte, 24-byte, or 32-byte clear AES key is required. For rule_array keyword KEYIDENT, a 64-byte fixed or variable-length internal AES key-token or key label for such a key in AES key-storage is required.

A variable-length AES key-token must have a key type of CIPHER and must allow the key to be used for encryption (key-usage field 1 high-order byte = B'1xxx xxxx'). In addition, the key token must have the following key usage based on processing rule keyword:

- **CBC**: must allow the key to be used for Cipher Block Chaining (key usage field (KUF) 2 high-order byte = X'00' or X'FF').
- **ECB**: must allow the key to be used for Electronic Code Book (KUF2 high-order byte = X'01' or X'FF').
- **GCM**: must allow the key to be used for Galois/Counter mode (KUF2 high-order byte = X'04' or X'FF').

**key_parms_length**

Direction: Input  
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the key_parms parameter. For processing rule GCM, this value can be 4 or 8 (which is strongly discouraged by NIST), or 12 - 16. Otherwise, this value must be 0.

**key_parms**

Direction: Input  
Type: String

A pointer to a string variable for key-related parameters. For processing rule GCM, this variable receives the generated authentication tag for the data identified by the cleartext parameter and any additional authenticated data identified by the optional_data parameter. No other usage is currently defined.

**block_size**

Direction: Input  
Type: Integer

A pointer to an integer variable containing the block size used by the cryptographic algorithm. This value must be 16.

**initialization_vector_length**

Direction: Input  
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the initialization_vector variable. For cipher block chaining (CBC or PKCS-PAD) with an INITIAL ICV selection, this value must be 16. For processing rule ECB or ICV selection CONTINUE, this value should be 0. For processing rule GCM, NIST recommends a length of 12, but any length from 1 to a maximum of 2^{32} - 1 can be used.

**initialization_vector**

Direction: Input
Symmetric Algorithm Encipher (CSNBSAE)

Type: String

A pointer to a string variable containing the initialization vector for the INITIAL call to CBC mode encryption, or if the ICV selection is ONLY. It is not used if the processing rule is ECB or the ICV selection is CONTINUE. The same initialization vector must be used when deciphering the data.

chain_data_length

Direction: Input/Output
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the chain_data variable. On input, depending on the processing rule keyword, set this variable to a value of at least 32 for CBC, 0 for ECB, or 104 for GCM mode encryption.

On output, the variable is updated with the length of the data returned in the chain_data variable. The chain_data_length parameter must not be changed by the calling application until chained operations are complete.

chain_data

Direction: Input/Output
Type: String

A pointer to a string variable used as a work area for CBC encipher requests. This work area is not used for ECB mode encryption, while GCM mode encryption uses 104 bytes for its work area.

When the verb performs a CBC encipher operation and the ICV selection is INITIAL, the chain_data variable is an output-only buffer that receives data used as input for enciphering the next part of the input data, if any. When the ICV selection is CONTINUE, the chain_data variable is both an input and output buffer. When the ICV selection is ONLY, the chain_data variable is an output-only buffer that receives data in the event that the amount of cleartext is greater than the host code can send to the coprocessor in a single call. The application must not change any intermediate data in this string.

cleartext_length

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the cleartext variable. The cleartext_length value must be a multiple of the algorithm block size unless processing rule PKCS-PAD is specified. For processing rule GCM, the value can be a minimum of 0 up to a maximum of \(2^{32} - 1\), otherwise the value must not be 0.

If PKCS-PAD is specified, set the output cleartext_length variable from 1 - 16 bytes more than the cleartext_length value to a multiple of the algorithm block size.

Note: Do not make the ciphertext_length and cleartext_length parameters point to the same variable.

cleartext

Direction: Input
Type: String
A pointer to a string variable used to contain the data to be enciphered, excluding any pad bytes.

ciphertext_length

Direction: Input/Output
Type: Integer

On input, the ciphertext_length parameter is a pointer to an integer variable containing the number of bytes of data in the ciphertext variable. On output, the ciphertext_length variable is updated to contain the actual length of text output in the ciphertext variable. If PKCS-PAD is specified, the ciphertext_length value must be greater than or equal to the next higher multiple of 16 as the cleartext_length value (from 1 - 16 bytes longer). Otherwise, the ciphertext_length value must be greater than or equal to the cleartext_length variable.

ciphertext

Direction: Input/Output
Type: String

A pointer to a string variable used as an output buffer where the verb returns the enciphered data. If PKCS-PAD is specified, on output the ciphertext buffer contains 1 - 16 bytes of data more than the cleartext input buffer contains.

optional_data_length

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the optional_data variable. For processing rule GCM, set this value to a minimum of 0 up to a maximum of $2^{32} - 1$, otherwise set this value to 0.

optional_data

Direction: Input
Type: String

A pointer to a string variable containing optional data for the encryption. For processing rule GCM, this parameter identifies any additional authenticated data (AAD). No other usage is currently defined.

Restrictions

The restrictions for CSNBSAE.

None.

Required commands

The required commands for CSNBSAE.

The Symmetric Algorithm Encipher verb requires the **Symmetric Algorithm Encipher - secure AES keys** command (offset X'012A') to be enabled in the active role. The verb also requires the command shown in the following table below to be enabled in the active role based on rule-array keyword:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCM</td>
<td>X'01CD'</td>
<td><strong>Symmetric Algorithm Encipher - AES GCM</strong></td>
</tr>
</tbody>
</table>
Symmetric Algorithm Encipher (CSNBSAE)

Usage notes

The usage notes for CSNBSAE.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBSAEJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBSAEJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber key_identifier_length,
    byte[] key_identifier,
    hikmNativeNumber key_parms_length,
    byte[] key_parms,
    hikmNativeNumber block_size,
    hikmNativeNumber initialization_vector_length,
    byte[] initialization_vector,
    hikmNativeNumber chain_data_length,
    byte[] chain_data,
    hikmNativeNumber clear_text_length,
    byte[] clear_text,
    hikmNativeNumber cipher_text_length,
    byte[] cipher_text,
    hikmNativeNumber optional_data_length,
    byte[] optional_data);
```

Cipher Text Translate2 (CSNBCTT2)

This callable service deciphers encrypted data (ciphertext) under one ciphertext translation key and re-enciphers it under another ciphertext translation key without having the data appear in the clear outside the cryptographic coprocessor. Such a function is useful in a multiple node network, where sensitive data is passed through multiple nodes prior to it reaching its final destination.

Use the Cipher Text Translate2 verb to decipher text under an input key and then to encipher the text under an output key. Both AES and DES algorithms are supported. Translation between AES and DES is allowed with restrictions controlled by access control points.

The encryption modes supported are:

- DES – CBC, CUSP and IPS
- AES – CBC and ECB

The padding methods supported are:

- DES – X9.23
- AES – PKCSPAD
Scenario for using the CSNBCTT2 verb

This scenario uses the Encipher (CSNBENC), Cipher Text Translate2 (CSNBCTT2), and Decipher (CSNBDEC) callable services with four network nodes: A, B, C, and D. You want to send data from your network node A to a destination node D. You cannot communicate directly with node D, because nodes B and C are situated between A and D. You do not want nodes B and C to decipher your data.

At node A, you use the CSNBENC service. Node D uses the CSNBDEC service. Node B and C will use the CSNBCTT2 service.

Consider the keys that are needed to support this process:
1. At your node, generate one key in two forms: OPEX CIPHER CIPHERXI.
2. Send the exportable CIPHERXI key to node B.
3. Node B and C need to share a key, so generate a different key in two forms: EXEX CIPHERX0 CIPHERXI.
4. Send the exportable CIPHERX0 key to node B.
5. Send the exportable CIPHERXI key to node C.
6. Node C and node D need to share a CIPHERX0 key and a CIPHER key. Node D can generate one key in two forms: OPEX CIPHERX0 CIPHERXI.
7. Node D sends the exportable CIPHERX0 key to node C.

The communication process is shown as:

Node: A B C D
Service: CSNBENC CSNBCTT2 CSNBCTT2 CSNBDEC
Keys: CIPHER CIPHERXI CIPHERX0 CIPHERXI CIPHERXI CIPHERXI CIPHER
Key pairs: |___ = ___| |___ = ___| |___ = ___|

Therefore, you need three keys, each in two different forms. You can generate two of the keys at node A, and node D can generate the third key. Note that the key used in the decipher callable service at node D is not the same key used in the encipher callable service at node A.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.
Cipher Text Translate 2 (CSNBCTT2)

Format

The format of CSNBCTT2.

```
CSNBCTT2(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_in_length,
    key_identifier_in,
    init_vector_in_length,
    init_vector_in,
    cipher_text_in_length,
    cipher_text_in,
    chaining_vector_length,
    chaining_vector,
    key_identifier_out_length,
    key_identifier_out,
    init_vector_out_length,
    init_vector_out,
    cipher_text_out_length,
    cipher_text_out,
    reserved1_length,
    reserved1,
    reserved2_length,
    reserved2
)
```

Parameters

The parameter definitions for CSNBCTT2.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Integer</td>
</tr>
</tbody>
</table>

A pointer to an integer variable containing the number of elements in the `rule_array` variable. The value must be 4 or 5.

**rule_array**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>String array</td>
</tr>
</tbody>
</table>

The keyword that provides control information to the verb. The processing method is the algorithm used to create the generated key. The keyword is left-aligned and padded on the right with blanks. The `rule_array` keywords are described in Table 119.

Table 119. Keywords for Cipher Text Translate2 control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inbound Processing Rule</strong> (One required)</td>
<td></td>
</tr>
<tr>
<td>I-CBC</td>
<td>Specifies encryption using CBC mode for the inbound ciphertext. The text length must be a multiple of the block size. The DES block size is 8 bytes. The AES block size is 16 bytes.</td>
</tr>
</tbody>
</table>
### Table 119. Keywords for Cipher Text Translate2 control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-CUSP</td>
<td>Specifies that CBC with CUSP processing for the inbound ciphertext. The ciphertext may be any length. The ciphertext is the same length as the plaintext. This keyword is only valid with DES.</td>
</tr>
<tr>
<td>I-ECB</td>
<td>Specifies encryption using ECB mode for the inbound ciphertext. The text must be a multiple of the block size. This keyword is only valid for AES encryption.</td>
</tr>
<tr>
<td>I-IPS</td>
<td>Specifies that CBC with IPS processing has been used for the inbound ciphertext. The ciphertext may be any length. The ciphertext is the same length as the plaintext. This keyword is only valid with DES.</td>
</tr>
<tr>
<td>IPKCSPAD</td>
<td>Specifies that CBC with PKCS padding was used for the inbound ciphertext. The text was padded on the right with 1 - 16 bytes of pad characters, making the padded text a multiple of the AES block size, before the data was enciphered. Each pad character is valued to the number of pad characters added. This keyword is only valid for AES encryption.</td>
</tr>
<tr>
<td>I-X923</td>
<td>Specifies that CBC with X9.24 padding was used for the inbound ciphertext. This is compatible with the requirements in ANSI Standard X9.23. This keyword is only valid for DES encryption.</td>
</tr>
</tbody>
</table>

#### Outbound processing rule (One required)

| O-CBC     | Specifies that encryption in CBC mode is used for the outbound ciphertext. The text length must be a multiple of the block size. The DES block size is 8 bytes. The AES block size is 16 bytes. |
| O-CUSP    | Specifies that CBC with CUSP processing is used for the outbound text. The outbound ciphertext has the same length as the plaintext. This keyword is only valid with DES. |
| O-ECB     | Specifies that encryption using ECB mode is used for the outbound ciphertext. The text must be a multiple of the block size. This keyword is only valid for AES encryption. |
| O-IPS     | Specifies that CBC with IPS processing is used for the outbound text. The outbound ciphertext has the same length as the plaintext. This keyword is only valid with DES. |
| OPKCSPAD  | Specifies that CBC with PKCS padding is used for the outbound text. The outbound text is padded on the right with 1 - 16 bytes of pad characters, making the padded text a multiple of the AES block size, before the data was enciphered. Each pad character is valued to the number of pad characters added. This keyword is only valid for AES encryption. |
| O-X923    | Specifies that CBC with X9.24 padding is used for the outbound text. This is compatible with the requirements in ANSI Standard X9.23. This keyword option is only valid for DES encryption. |

#### Segmenting control (One optional)

| CONTINUE  | Specifies the initialization vectors are taken from the chaining vector. The chaining vector is updated and must not be modified between calls. This keyword is ignored for I-ECB and O-ECB processing rules. The CONTINUE keyword is not valid with the I-X923 or O-X923 keywords. |
| INITIAL   | Specifies that the initialization vectors are taken from the `init_vector_in` and `init_vector_out` parameters. This is the default. This keyword is ignored for I-ECB and O-ECB processing rules. |

#### Inbound key identifier (One required)

| IKEY-DES  | Specifies that the inbound key identifier is a DES key. |
| IKEY-AES  | Specifies that the inbound key identifier is an AES key. |

#### Outbound key identifier (One required)

| OKEY-DES  | Specifies that the outbound key identifier is a DES key. |
| OKEY-AES  | Specifies that the outbound key identifier is an AES key. |

#### `key_identifier_in_length`

- **Direction:** Input
- **Type:** Integer

Length of the `key_identifier_in` in bytes. The value is 64 when a label is
supplied. When the key identifier is a key token, the value is the length of the token. The maximum value is 725.

**key_identifier_in**

Direction: Input/Output  
Type: String

An internal key token or the label of an AES or DES key storage record containing the cipher translation key for the inbound ciphertext.

Acceptable DES key types are DATA, CIPHER, CIPHERXI, CIPHERXL, and DECIPHER. The keys must have bit 19 for DECIPHER set on in the control vector. The key may be a single-, double-, or triple-length key. If the **Cipher Text translate2 - Allow only cipher text translate types** access control point is enabled, only CIPHERXI and CIPHERXL are allowed.

Acceptable AES key types include the 64-byte AES DATA key and the variable length token CIPHER key with the DECRYPT bit on in the key usage field. The C-XLATE bit can optionally be on. If the **Cipher Text translate2 - Allow only cipher text translate types** access control point is enabled, the C-XLATE bit must be turned on in the key usage field.

**init_vector_in_length**

Direction: Input  
Type: Integer

Length of the **init_vector_in** field in bytes. For AES keys, the length is 16. For DES keys, the length is 8. When the initialization vector is not required (segmenting rule CONTINUE, processing rule I-ECB), the value must be 0.

**init_vector_in**

Direction: Input  
Type: String

The initialization vector that is used to decipher the input data. This parameter is the initialization vector used at the previous cryptographic node. This parameter is required for segmenting rule INITIAL.

**cipher_text_in_length**

Direction: Input  
Type: Integer

The length of the **cipher_text_in** to be processed. See the table of ciphertext length restrictions in "Usage notes" on page 411.

**cipher_text_in**

Direction: Input  
Type: String

The text that is to be translated. The text is enciphered under the cipher key specified in the **key_identifier_in** parameter.

**chaining_vector_length**

Direction: Input  
Type: Integer

The length of the **chaining_vector** parameter in bytes. The **chaining_vector** field must be 128 bytes long.
chaining_vector

Direction: Input/Output
Type: String

The chaining_vector parameter is a work area used by the service to carry segmented data between procedure calls. This area must not be modified between calls to the service.

key_identifier_out_length

Direction: Input
Type: Integer

The length of the key_identifier_out parameter in bytes. This value is 64 when a label is supplied. When the key identifier is a key token, the value is the length of the token. The maximum value is 725.

key_identifier_out

Direction: Input/Output
Type: String

An internal key token or the label of an AES or DES key storage record containing the cipher translation key for the outbound ciphertext.

Acceptable DES key types are DATA, CIPHER, CIPHERXL, CIPHERXO, and ENCPHER. The key may be a double- or triple-length key. If the Cipher Text translate2 – Allow only cipher text translate types access control point is enabled, only CIPHERXO and CIPHERXL are allowed. Acceptable DES key types are DATA, CIPHER, CIPHERXL, CIPHERXO, and ENCPHER. The keys must have bit 18 for ENCPHER set on in the control vector. The key may be a double- or triple-length key. If the Cipher Text translate2 – Allow only cipher text translate types access control point is enabled, only CIPHERXO and CIPHERXL are allowed.

Acceptable AES key types include the 64-byte AES DATA key and the variable length token CIPHER key with the ENCRYPT bit on in the key usage field.

The C-XLATE bit can optionally be on. If the Cipher Text translate2 – Allow only cipher text translate types access control point is enabled, the C-XLATE bit must be turned on in the key usage field.

init_vector_out_length

Direction: Input
Type: Integer

The length of the init_vector_out parameter in bytes. For AES keys, the length is 16. For DES keys, the length is 8. When the initialization vector is not required (segmenting rule CONTINUE, processing rule O-ECB), the value must be 0.

init_vector_out

Direction: Input
Type: String

The initialization vector that is used to encipher the input data. This is the new initialization vector used when the callable service enciphers the plaintext. This parameter is required for segmenting rule INITIAL.

cipher_text_out_length
Cipher Text Translate 2 (CSNBCTT2)

Direction: Input/Output
Type: Integer

The length of the cipher_text_out parameter in bytes. This parameter is updated with the actual length of the data in the cipher_text_out parameter. Note that padding may require this value to be larger than the cipher_text_in_length parameter (see Table 120 on page 411).

cipher_text_out
Direction: Output
Type: String

The field where the callable service returns the translated text.

reserved1_length
Direction: Input
Type: Integer

The length of the reserved1 parameter in bytes. The value must be zero.

reserved1
Direction: Input
Type: String

This parameter is ignored.

reserved2_length
Direction: Input
Type: Integer

The length of the reserved2 parameter in bytes. The value must be zero.

reserved2
Direction: Input
Type: String

This parameter is ignored.

Restrictions

The restrictions for CSNBCTT2.

None.

Required commands

The CSNBCTT2 required commands.

This verb requires the following commands to be enabled in the active role:

<table>
<thead>
<tr>
<th>Command</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cipher Text Translate2</td>
<td>X'01C0'</td>
<td>Enable the Ciphertext Translate2 service</td>
</tr>
<tr>
<td>Cipher Text Translate2 – Allow translate from AES to TDES</td>
<td>X'01C1'</td>
<td>Allow translation from an AES key to 2 or 3 key triple DES key.</td>
</tr>
</tbody>
</table>
Cipher Text Translate 2 (CSNBCTT2)

<table>
<thead>
<tr>
<th>Command</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cipher Text Translate2 – Allow</td>
<td>X’01C2’</td>
<td>Allow translation from a stronger to weaker AES key. (For example, IN key AES256 and OUT key AES128.)</td>
</tr>
<tr>
<td>translate to weaker AES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cipher Text Translate2 – Allow</td>
<td>X’01C3’</td>
<td>Allow translation from a triple-length DES key to a weaker double-length DES key.</td>
</tr>
<tr>
<td>translate to weaker DES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cipher Text Translate2 – Allow</td>
<td>X’01C4’</td>
<td>When enabled, the verb only accepts these key-translation-only key types, which cannot be used in normal cipher or MAC data operations:</td>
</tr>
<tr>
<td>only cipher text translate types</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• AES key tokens with key type CIPHER and key usage set to allow data translate (C-XLATE) only</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DES key tokens with a key type CIPHERXI, CIPHERXO, or CIPHERXL</td>
</tr>
</tbody>
</table>

In other words, with offset X’01C4’ enabled in the active role, AES key type AESDATA and DES key types CIPHER, DATA, DATAC, and ENCIPHER, are not allowed. This restricts the verb from using keys that can be used in data operations other than translation.

Usage notes

Usage notes for CSNBCTT2.

Restriction: The Cipher Text Translate2 callable service is only available on the IBMzEC12 and later servers.

The initialization vectors must have already been established between the communicating applications or must be passed with the data.

Table 120 outlines the restrictions for the cipher_text_in_length and cipher_text_out_length parameters. The DES blocks referred to in this table are 8 bytes. The AES blocks referred to in this table are 16 bytes.

<table>
<thead>
<tr>
<th>Input cipher method</th>
<th>Output cipher method</th>
<th>Input ciphertext length restriction(s)</th>
<th>Output ciphertext length restriction(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES CBC</td>
<td>DES CBC</td>
<td>Input ciphertext must be a multiple of a DES block.</td>
<td>Output ciphertext length must be greater than or equal to the sum of the length of the input ciphertext and a DES block.</td>
</tr>
<tr>
<td></td>
<td>X9.23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DES CBC</td>
<td>AES CBC</td>
<td>Input ciphertext must be a multiple of a DES block.</td>
<td>If the input ciphertext is not a multiple of an AES block, then the output ciphertext length must be greater than or equal to the sum of the input ciphertext length and a DES block. If the input ciphertext is a multiple of an AES block, then the output ciphertext length must be greater than or equal to the sum of the input ciphertext length and an AES block.</td>
</tr>
<tr>
<td></td>
<td>PKCSPAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DES CBC</td>
<td>DES CUSP or IPS</td>
<td>Input ciphertext must be a multiple of a DES block.</td>
<td>Output ciphertext length must be greater than or equal to the input ciphertext length.</td>
</tr>
<tr>
<td>Input cipher method</td>
<td>Output cipher method</td>
<td>Input ciphertext length restriction(s)</td>
<td>Output ciphertext length restriction(s)</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------</td>
<td>---------------------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>DES CBC</td>
<td>DES CBC</td>
<td>Input ciphertext must be a multiple of a DES block.</td>
<td>Output ciphertext length must be greater than or equal to the input ciphertext length.</td>
</tr>
<tr>
<td>DES CBC</td>
<td>AES CBC</td>
<td>Input ciphertext must be a multiple of an AES block.</td>
<td>Output ciphertext length must be greater than or equal to the input ciphertext length.</td>
</tr>
<tr>
<td>DES CBC</td>
<td>AES CBC</td>
<td>Input ciphertext must be a multiple of an AES block.</td>
<td>Output ciphertext length must be greater than or equal to the input ciphertext length.</td>
</tr>
<tr>
<td>DES CBC CUSP or IPS</td>
<td>DES CBC CUSP or IPS</td>
<td>No restrictions</td>
<td>Output ciphertext length must be greater than or equal to the input ciphertext length.</td>
</tr>
<tr>
<td>DES CBC CUSP or IPS</td>
<td>AES CBC</td>
<td>Input ciphertext must be a multiple of a DES block.</td>
<td>Output ciphertext length must be greater than or equal to the input ciphertext length.</td>
</tr>
<tr>
<td>DES CBC CUSP or IPS</td>
<td>AES CBC or ECB</td>
<td>Input ciphertext must be a multiple of an AES block.</td>
<td>Output ciphertext length must be greater than or equal to the input ciphertext length.</td>
</tr>
<tr>
<td>DES CBC CUSP or IPS</td>
<td>DES CBC X9.23</td>
<td>No restrictions</td>
<td>Output ciphertext length must be greater than or equal to the sum of the input ciphertext length and a DES block.</td>
</tr>
<tr>
<td>DES CBC CUSP or IPS</td>
<td>AES CBC PKCSPAD</td>
<td>No restrictions</td>
<td>Output ciphertext length must be greater than or equal to the sum of the input ciphertext length and an AES block.</td>
</tr>
<tr>
<td>DES CBC X9.23</td>
<td>DES CBC X9.23</td>
<td>Input ciphertext must be a multiple of a DES block.</td>
<td>Output ciphertext length must be greater than or equal to the input ciphertext length.</td>
</tr>
<tr>
<td>DES CBC X9.23</td>
<td>AES CBC PKCSPAD</td>
<td>Input ciphertext must be a multiple of a DES block.</td>
<td>Output ciphertext length must be greater than or equal to the sum of the input ciphertext length and a DES block.</td>
</tr>
<tr>
<td>DES CBC X9.23</td>
<td>DES CBC CUSP or IPS</td>
<td>Input ciphertext must be a multiple of a DES block.</td>
<td>Output ciphertext length must be greater than or equal to the input ciphertext length.</td>
</tr>
<tr>
<td>DES CBC X9.23</td>
<td>DES CBC</td>
<td>Input ciphertext must be a multiple of a DES block.</td>
<td>Output ciphertext length must be greater than or equal to the input ciphertext length. Note: This operation is not possible if the padding is determined by the adapter to be from 1-7 bytes.</td>
</tr>
<tr>
<td>DES CBC X9.23</td>
<td>AES CBC</td>
<td>Input ciphertext must be a multiple of a DES block but must not be a multiple of an AES block.</td>
<td>Output ciphertext length must be greater than or equal to the input ciphertext length. Note: This operation is not possible if the padding is determined by the adapter to be from 1-7 bytes.</td>
</tr>
<tr>
<td>DES CBC X9.23</td>
<td>AES ECB</td>
<td>Input ciphertext must be a multiple of a DES block but must not be a multiple of an AES block.</td>
<td>Output ciphertext length must be greater than or equal to the input ciphertext length. Note: This operation is not possible if the padding is determined by the adapter to be from 1-7 bytes.</td>
</tr>
<tr>
<td>AES CBC or ECB</td>
<td>DES CBC X9.23</td>
<td>Input ciphertext must be a multiple of an AES block.</td>
<td>Output ciphertext length must be greater than or equal to the sum of the input ciphertext length and a DES block.</td>
</tr>
<tr>
<td>AES CBC or ECB</td>
<td>AES CBC PKCSPAD</td>
<td>Input ciphertext must be a multiple of an AES block.</td>
<td>Output ciphertext length must be greater than or equal to the sum of the input ciphertext length and an AES block.</td>
</tr>
<tr>
<td>AES CBC or ECB</td>
<td>DES CBC CUSP or IPS</td>
<td>Input ciphertext must be a multiple of an AES block.</td>
<td>Output ciphertext length must be greater than or equal to the input ciphertext length.</td>
</tr>
<tr>
<td>AES CBC or ECB</td>
<td>DES CBC</td>
<td>Input ciphertext must be a multiple of an AES block.</td>
<td>Output ciphertext length must be greater than or equal to the input ciphertext length.</td>
</tr>
</tbody>
</table>

Note: This operation is not possible if the padding is determined by the adapter to be from 1-7 bytes.
There are requirements for the keys for the `key_identifier_in` and `key_identifier_out` parameters. The `key_identifier_in` key must be able to decipher text. The `key_identifier_out` key must be able to encipher text.

Table 121 on page 414 shows the valid key types which are allowed for the `key_identifier_in` and `key_identifier_out` parameters. In the table, a variable length key token cipher key is denoted by vCIPHER. vCIPHER is the default which has the ENCRYPT and DECRYPT bits on in the usage field. vCIPHERe has only the ENCRYPT bit on in the usage field. vCIPHERd has only the DECRYPT bit on in the usage field. Adding x to either of the preceding names means the TRANSLATE bit is on in the usage field for that key. For example, vCIPHERex means a variable length token with the ENCRYPT and TRANSLAT bits turned on.

AESDATA is the 64-byte AES DATA key type.
Cipher Text Translate 2 (CSNBCTT2)

Table 121. Cipher Text Translate2 key usage.

<table>
<thead>
<tr>
<th>key_identifier_in (DEC bit except DATA and AESDATA)</th>
<th>key_identifier_out (ENC bit except DATA and AESDATA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA, CIPHER, DECRYPTER, CIPHERXI, CIPHERXL</td>
<td>DATA, CIPHER, DECRYPTER, CIPHERXI, CIPHERXL</td>
</tr>
<tr>
<td>AESDATA, vCIPHER, vCIPHERd, vCIPHERdx, vCIPHERedx</td>
<td>DATA (must be at least double-length key with ACP)</td>
</tr>
<tr>
<td></td>
<td>CIPHER (requires ACP to be enabled)</td>
</tr>
<tr>
<td></td>
<td>ENCIPHER (requires ACP to be enabled)</td>
</tr>
<tr>
<td></td>
<td>CIPHERXO (requires ACP to be enabled)</td>
</tr>
<tr>
<td></td>
<td>CIPHERXL (requires ACP to be enabled)</td>
</tr>
<tr>
<td></td>
<td>AESDATA, vCIPHER, vCIPHERd, vCIPHERdx, vCIPHERedx</td>
</tr>
</tbody>
</table>

Note:
1. Translation from stronger encryption to single-key DES is not allowed.
2. Translation from a triple-length DES key to a double-length DES key requires the Cipher Text Translate2 - Allow translate to weaker DES access control point (offset X'01C3') to be enabled.
3. When the Cipher Text Translate2 - Allow only cipher text translate types access control point (offset X'01C4') is enabled, only CIPHERXI, CIPHERXL, and CIPHERXO DES key types are allowed and AES key tokens with key type CIPHER must be set to allow data translate (C-XLATE).

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBCTT2.

See “Building Java applications using the CCA JNI” on page 28.

Format

public native void CSNBCTT2J(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber key_identifier_in_len,
    byte[] key_identifier_in,
    hikmNativeNumber init_vector_in_length,
    byte[] init_vector_in,
    hikmNativeNumber cipher_text_in_length,
    byte[] cipher_text_in,
);
hikmNativeNumber chaining_vector_length, 
byte[] chaining_vector,  
hikmNativeNumber key_identifier_out_length, 
byte[] key_identifier_out, 
hikmNativeNumber init_vector_out_length, 
byte[] init_vector_out, 
hikmNativeNumber cipher_text_out_length, 
byte[] cipher_text_out, 
hikmNativeNumber reserved1_length, 
byte[] reserved1, 
hikmNativeNumber reserved2_length, 
byte[] reserved2);
Cipher Text Translate 2 (CSNBCTT2)
Chapter 10. Verifying data integrity and authenticating messages

CCA provides methods to verify the integrity of transmitted messages and stored data.

The methods provided are:

- Message authentication code (MAC)
- Hash functions, including Modification Detection Code (MDC) processing and one-way hash generation

**Note:** You can also use digital signatures (see Chapter 14, “Using digital signatures,” on page 675) to authenticate messages.

The choice of verb depends on the security requirements of the environment in which you are operating. If you need to ensure the authenticity of the sender as well as the integrity of the data and both the sender and receiver can share a secret key, consider Message Authentication Code processing. If you need to ensure the integrity of transmitted data in an environment where it is not possible for the sender and the receiver to share a secret cryptographic key, consider hashing functions.

The verbs described in this topic include:

- “HMAC Generate (CSNBHMG)” on page 419
- “HMAC Verify (CSNBHMV)” on page 423
- “MAC Generate (CSNBMGN)” on page 426
- “MAC Generate2 (CSNBMGN2)” on page 430
- “MAC Verify (CSNBMVR)” on page 433
- “MAC Verify2 (CSNBMVR2)” on page 438
- “MDC Generate (CSNBMDG)” on page 441
- “One-Way Hash (CSNBOWH)” on page 445

How MACs are used

When a message is sent, an application program can generate an authentication code for it using the MAC Generate verb.

CCA supports the ANSI X9.9-1 basic procedure and both the ANSI X9.19 basic procedure and optional double key MAC procedure. The MAC Generate verb computes the text of the Message Authentication Code using the algorithm and a key. The ANSI X9.9-1 or ANSI X9.19 basic procedures accept either a single-length MAC generation (MAC) key or a data-encrypting (DATA) key, and the message text. The ANSI X9.19 optional double key MAC procedure accepts a double-length MAC key and the message text. The originator of the message sends the MAC with the message text.

When the receiver gets the message, an application program calls the MAC Verify verb. The MAC Generate verb generates a MAC using the same algorithm as the sender and either the single-length or double-length MAC verification key, the single-length or double-length MAC generation key, or DATA key, and the message text.
text. The MAC Verify verb compares the MAC it generates with the one sent with the message and issues a return code that indicates whether the MACs match. If the return code indicates that the MACs match, the receiver can accept the message as genuine and unaltered. If the return code indicates that the MACs do not match, the receiver can assume the message is either fraudulent or has been altered. The newly computed MAC is not revealed outside the cryptographic coprocessor.

In a similar manner, MACs can be used to ensure the integrity of data stored on the system or on removable media, such as tape.

Secure use of the MAC Generate and MAC Verify verbs requires the use of MAC and MACVER keys in these verbs, respectively. To accomplish this, the originator of the message generates a MAC/MACVER key pair, uses the MAC key in the MAC Generate verb, and exports the MACVER key to the receiver. The originator of the message enforces key separation on the link by encrypting the MACVER key under a transport key that is not an NOCV key before exporting the key to the receiver. With this type of key separation enforced, the receiver can receive only a MACVER key and can use only this key in the MAC Verify verb. This ensures that the receiver cannot alter the message and produce a valid MAC with the altered message. These security features are not present if DATA keys are used in the MAC Generate verb or if DATA or MAC keys are used in the MAC Verify verb.

By using MACs you get the following benefits:

- **For data transmitted over a network**, you can validate the authenticity of the message as well as ensure the data has not been altered during transmission. For example, an active eavesdropper can tap into a transmission line and interject fraudulent messages or alter sensitive data being transmitted. If the data is accompanied by a MAC, the recipient can use a verb to detect whether the data has been altered. Because both the sender and receiver share a secret key, the receiver can use a verb that calculates a MAC on the received message and compares it to the MAC transmitted with the message. If the comparison is equal, the message could be accepted as unaltered. Furthermore, because the shared key is secret, when a MAC is verified it can be assumed that the sender was, in fact, the other person who knew the secret key.

- **For data stored on tape or DASD**, you can ensure that the data read back onto the system was the same as the data written onto the tape or DASD. For example, someone might be able to bypass access controls. Such an access might escape the notice of auditors. However, if a MAC is stored with the data, and verified when the data is read, you can detect alterations to the data.

**How hashing functions and MDCs are used**

Hashing functions include the MDC and one-way hash.

You need to hash text before submitting it to the Digital Signature Generate and Digital Signature Verify verbs (see Chapter 14, “Using digital signatures,” on page 675). CCA supports the SHA-1, MD5, and RIPEMD-160 hashing functions.

When a message is sent, an application program can generate a hash or a Modification Detection Code (MDC) for it using the One-Way Hash verb. This verb computes the hash or MDC, a short, fixed-length value, using a one-way cryptographic function and the message text. The originator of the message
ensures the hash or MDC is transmitted with integrity to the intended receiver of the message. For example, the value could be published in a reliable source of public information.

When the receiver gets the message, an application program calls the One-Way Hash verb to generate a new hash or MDC using the same function and message text that were used by the sender. The application program can compare the new value with the one generated by the originator of the message. If the two values match, the receiver knows the message was not altered.

In a similar manner, hashes and MDCs can be used to ensure the integrity of data stored on the system or on removable media, such as tape.

By using hashes and MDCs, you get the following benefits:

- **For data transmitted over a network between locations that do not share a secret key**, you can ensure the data has not been altered during transmission. It is easy to compute a hash or MDC for specific data, yet hard to find data that will result in a given hash or MDC. In effect, the problem of ensuring the integrity of a large file is reduced to ensuring the integrity of a short, fixed-length value.

- **For data stored on tape or DASD**, you can ensure that the data read back onto the system was the same as the data written onto the tape or DASD. After a hash has been established for a file, the One-Way Hash verb can be run at any later time on the file. The resulting value can be compared with the stored value to detect deliberate or inadvertent modification.

For more information, see “Modification Detection Code calculation” on page 1024.

**HMAC Generate (CSNBHMG)**

Use the HMAC Generate verb to generate a keyed hash Message Authentication Code (HMAC) for the message string provided as input.

An HMAC key that can be used for generate is required to calculate the HMAC.

**Format**

The format of CSNBHMG.

```plaintext
CSNBHMG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_length,
    key_identifier,
    text_length,
    text,
    chaining_vector_length,
    chaining_vector,
    mac_length,
    mac)
```

**Parameters**

The parameters for CSNBHMG.
HMAC Generate (CSNBHMG)

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

Direction: Input  
Type: Integer  

The number of keywords you supplied in the rule_array parameter. This value must be 2 or 3.

**rule_array**

Direction: Input  
Type: String array  

Keywords that provide control information to the verb. The following table lists the keywords. Each keyword is left-aligned in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage. The rule_array keywords are described in Table 122.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Token algorithm</strong> (One required)</td>
<td></td>
</tr>
<tr>
<td>HMAC</td>
<td>Specifies the HMAC algorithm to be used to generate the MAC.</td>
</tr>
<tr>
<td><strong>Hash method</strong> (One required).</td>
<td></td>
</tr>
<tr>
<td>SHA-1</td>
<td>Specifies the FIPS-198 HMAC procedure using the SHA-1 hash method, a symmetric key and text to produce a 20-byte (160-bit) MAC.</td>
</tr>
<tr>
<td>SHA-224</td>
<td>Specifies the FIPS-198 HMAC procedure using the SHA-224 hash method, a symmetric key and text to produce a 28-byte (224-bit) MAC.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Specifies the FIPS-198 HMAC procedure using the SHA-256 hash method, a symmetric key and text to produce a 32-byte (256-bit) MAC.</td>
</tr>
<tr>
<td>SHA-384</td>
<td>Specifies the FIPS-198 HMAC procedure using the SHA-384 hash method, a symmetric key and text to produce a 48-byte (384-bit) MAC.</td>
</tr>
<tr>
<td>SHA-512</td>
<td>Specifies the FIPS-198 HMAC procedure using the SHA-512 hash method, a symmetric key and text to produce a 64-byte (512-bit) MAC.</td>
</tr>
<tr>
<td><strong>Segmenting control</strong> (One optional)</td>
<td></td>
</tr>
<tr>
<td>FIRST</td>
<td>First call, this is the first segment of data from the application program.</td>
</tr>
<tr>
<td>LAST</td>
<td>Last call; this is the last data segment.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>Middle call; this is an intermediate data segment.</td>
</tr>
<tr>
<td>ONLY</td>
<td>Only call; segmenting is not employed by the application program. This is the default value.</td>
</tr>
</tbody>
</table>

**key_identifier_length**

Direction: Input  
Type: Integer  

The length of the key_identifier parameter. The maximum value is 725.

**key_identifier**

Direction: Input  
Type: String  

The 64-byte label or internal token of an encrypted HMAC key.
Table 123. Hash methods and HMAC key sizes

<table>
<thead>
<tr>
<th>Specified hash method</th>
<th>Minimum HMAC key size (in bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>80</td>
</tr>
<tr>
<td>SHA-224</td>
<td>112</td>
</tr>
<tr>
<td>SHA-256</td>
<td>128</td>
</tr>
<tr>
<td>SHA-384</td>
<td>192</td>
</tr>
<tr>
<td>SHA-512</td>
<td>256</td>
</tr>
</tbody>
</table>

text_length

| Direction: | Input          |
| Type:      | Integer        |

The length of the text you supply in the text parameter. The maximum length of text is 214783647 bytes. For FIRST and MIDDLE calls, the text_length must be a multiple of 64 for SHA-1, SHA-224 and SHA-256 hash methods, and a multiple of 128 for SHA-384 and SHA-512 hash methods.

text

| Direction: | Input          |
| Type:      | String         |

The application-supplied text for which the MAC is generated.

chaining_vector_length

| Direction: | Input/Output   |
| Type:      | Integer        |

The length of the chaining_vector in bytes. This value must be 128.

chaining_vector

| Direction: | Input/Output   |
| Type:      | String         |

An 128-byte string used as a system work area. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.

On the first call, initialize this parameter as binary zeros.

mac_length

| Direction: | Input/Output   |
| Type:      | Integer        |

The length of the mac parameter in bytes. This parameter is updated to the actual length of the mac parameter on output. The minimum value is 4, and the maximum value is 64.

mac

| Direction: | Output         |
| Type:      | String         |

The field in which the verb returns the MAC value if the segmenting rule is ONLY or LAST.
Restrictions
The restrictions for CSNBHMG.

This verb was introduced with CCA 4.1.0.

Required commands
The required commands for CSNBHMG.

This verb requires the commands shown in the following table to be enabled in the active role:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>X’00E4’</td>
<td>HMAC Generate - SHA-1</td>
</tr>
<tr>
<td>SHA-224</td>
<td>X’00E5’</td>
<td>HMAC Generate - SHA-224</td>
</tr>
<tr>
<td>SHA-256</td>
<td>X’00E6’</td>
<td>HMAC Generate - SHA-256</td>
</tr>
<tr>
<td>SHA-384</td>
<td>X’00E7’</td>
<td>HMAC Generate - SHA-384</td>
</tr>
<tr>
<td>SHA-512</td>
<td>X’00E8’</td>
<td>HMAC Generate - SHA-512</td>
</tr>
</tbody>
</table>

Usage notes
The usage notes for CSNBHMG.

None.

Related information
More information about CSNBHMG is provided in other topics.

The HMAC Verify verb is described in “HMAC Verify (CSNBHMOV)” on page 423.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNBHMGJ.

See “Building Java applications using the CCA JNI” on page 28.

Format
```java
public native void CSNBHMGJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber key_identifier_length
    byte[] key_identifier,
    hikmNativeNumber text_length,
    byte[] text,
    hikmNativeNumber chaining_vector_length,
    byte[] chaining_vector,
    hikmNativeNumber mac_length,
    byte[] mac);
```
HMAC Verify (CSNBHMV)

Use the HMAC Verify verb to verify a keyed hash Message Authentication Code (HMAC) for the message string provided as input.

A MAC key contained in an internal variable-length symmetric key-token is required to verify the HMAC. The key must have the same value as the key used to generate the HMAC.

Format

The format of CSNBHMV.

```
CSNBHMV(
   return_code,
   reason_code,
   exit_data_length,
   exit_data,
   rule_array_count,
   rule_array,
   key_identifier_length,
   key_identifier,
   text_length,
   text,
   chaining_vector_length,
   chaining_vector,
   mac_length,
   mac)
```

Parameters

The parameters for CSNBHMV.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

The number of keywords you supplied in the `rule_array` parameter. This value must be 2 or 3.

**rule_array**

- **Direction:** Input
- **Type:** String array

Keywords that provide control information to the verb. The following table lists the keywords. Each keyword is left-aligned in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage. The `rule_array` keywords are described in Table 124.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Token algorithm</strong> (One required)</td>
<td>Specifies that the HMAC algorithm is to be used to verify the MAC.</td>
</tr>
<tr>
<td><strong>Hash method</strong> (One required)</td>
<td></td>
</tr>
</tbody>
</table>

Table 124. Keywords for HMAC Verify control information
### Keywords for HMAC Verify control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>Specifies the FIPS-198 HMAC procedure using the SHA-1 hash method, a symmetric key and text to produce a 20-byte (160-bit) MAC.</td>
</tr>
<tr>
<td>SHA-224</td>
<td>Specifies the FIPS-198 HMAC procedure using the SHA-224 hash method, a symmetric key and text to produce a 28-byte (224-bit) MAC.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Specifies the FIPS-198 HMAC procedure using the SHA-256 hash method, a symmetric key and text to produce a 32-byte (256-bit) MAC.</td>
</tr>
<tr>
<td>SHA-384</td>
<td>Specifies the FIPS-198 HMAC procedure using the SHA-384 hash method, a symmetric key and text to produce a 48-byte (384-bit) MAC.</td>
</tr>
<tr>
<td>SHA-512</td>
<td>Specifies the FIPS-198 HMAC procedure using the SHA-512 hash method, a symmetric key and text to produce a 64-byte (512-bit) MAC.</td>
</tr>
</tbody>
</table>

**Segmenting control (Optional)**

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIRST</td>
<td>First call, this is the first segment of data from the application program.</td>
</tr>
<tr>
<td>LAST</td>
<td>Last call; this is the last data segment.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>Middle call; this is an intermediate data segment.</td>
</tr>
<tr>
<td>ONLY</td>
<td>Only call; segmenting is not employed by the application program. This is the default value.</td>
</tr>
</tbody>
</table>

**key_identifier_length**

- **Direction:** Input
- **Type:** Integer

The length of the `key_identifier` parameter. The maximum value is 725.

**key_identifier**

- **Direction:** Input/Output
- **Type:** String

The 64-byte label or internal token of an encrypted HMAC or HMACVER key.

**text_length**

- **Direction:** Input
- **Type:** Integer

The length of the text you supply in the `text` parameter. The maximum length of `text` is 214783647 bytes. For `FIRST` and `MIDDLE` calls, the `text_length` must be a multiple of 64 for SHA-1, SHA-224, and SHA-256 hash methods, and a multiple of 128 for SHA-384 and SHA-512 hash methods.

**text**

- **Direction:** Input
- **Type:** String

The application-supplied text for which the HMAC is to be verified.

**chaining_vector_length**

- **Direction:** Input/Output
- **Type:** Integer

The length of the `chaining_vector` in bytes. This value must be 128.
HMAC Verify (CSNBHMV)

Direction: Input/Output
Type: String

An 128-byte string used as a system work area. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.

On the first call, initialize this parameter as binary zeros.

**mac_length**

Direction: Input
Type: Integer

The length of the mac parameter in bytes. The maximum value is 64.

**mac**

Direction: Input
Type: String

The field that contains the MAC value you want to verify.

**Restrictions**

The restrictions for CSNBHMV.

This verb was introduced with CCA 4.1.0.

**Required commands**

The required commands for CSNBHMV.

This verb requires the commands shown in the following table to be enabled in the active role:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>X'00F7'</td>
<td>HMAC Verify - SHA-1</td>
</tr>
<tr>
<td>SHA-224</td>
<td>X'00F8'</td>
<td>HMAC Verify - SHA-224</td>
</tr>
<tr>
<td>SHA-256</td>
<td>X'00F9'</td>
<td>HMAC Verify - SHA-256</td>
</tr>
<tr>
<td>SHA-384</td>
<td>X'00FA'</td>
<td>HMAC Verify - SHA-384</td>
</tr>
<tr>
<td>SHA-512</td>
<td>X'00FB'</td>
<td>HMAC Verify - SHA-512</td>
</tr>
</tbody>
</table>

**Usage notes**

The usage notes for CSNBHMV.

None.

**Related information**

Additional information about CSNBHMV,

The HMAC Generate verb is described in "HMAC Generate (CSNBHMG)" on page 419.
JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBHMVJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

public native void CSNBHMVJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber key_identifier_length,
    byte[] key_identifier,
    hikmNativeNumber text_length,
    byte[] text,
    hikmNativeNumber chaining_vector_length,
    byte[] chaining_vector,
    hikmNativeNumber mac_length,
    byte[] mac);

MAC Generate (CSNBMGN)

When a message is sent, an application program can generate an authentication code for it using the MAC Generate verb.

This verb generates a 4-byte, 6-byte, or 8-byte Message Authentication Code (MAC) for an application-supplied text string.

This verb computes the Message Authentication Code using one of the following methods:

• Using the ANSI X9.9-1 single key algorithm, a single-length MAC generation key or data-encrypting key, and the message text.
• Using the ANSI X9.19 optional double key algorithm, a double-length MAC generation key and the message text.
• Using the Europay, MasterCard and Visa (EMV) padding rules.

The MAC can be the leftmost 32 or 48 bits of the last block of the ciphertext or the entire last block (64 bits) of the ciphertext. The originator of the message sends the Message Authentication Code with the message text.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.

Host CPU acceleration: CPACF

Only keys with a key type of DATA can be used successfully with the CPACF exploitation layer through this verb.

Specifically, a DATA key has a CV (Control Vector) of all X’00’ bytes for all active bytes of the CV (eight bytes for 8-byte DES keys, 16 bytes for 16-byte DES keys, and 16 bytes for 24-byte DES keys).

For details about CPACF, see “CPACF support” on page 14.
MAC Generate (CSNBMGN)

Format

The format of CSNBMGN.

```
CSNBMGN(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier,
    text_length,
    text,
    rule_array_count,
    rule_array,
    chaining_vector,
    mac)
```

Parameters

The parameters for CSNBMGN.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

**key_identifier**

- **Direction:** Input/Output
- **Type:** String

The 64-byte key label or internal key token that identifies a single-length or double-length MAC generate key or a single-length DATA or DATAM key. The type of key depends on the MAC process rule in the `rule_array` parameter.

**text_length**

- **Direction:** Input
- **Type:** Integer

The length of the text you supply in the `text` parameter. If the `text_length` is not a multiple of eight bytes and if the ONLY or LAST keyword of the `rule_array` parameter is called, the text is padded in accordance with the processing rule specified.

**text**

- **Direction:** Input
- **Type:** String

The application-supplied text for which the MAC is generated.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0, 1, 2, or 3.

**rule_array**

- **Direction:** Input
- **Type:** String array

Zero to three keywords that provide control information to the verb. The
MAC Generate (CSNBMGN)

Keywords are described in Table 125. The keywords must be in 24 bytes of contiguous storage with each of the keywords left-aligned in its own 8-byte location and padded on the right with blanks. For example,

'X9.9-1 MIDDLE MACLEN4'

The order of the rule_array keywords is not fixed.

You can specify one of the MAC processing rules and then choose one of the segmenting control keywords and one of the MAC length keywords. The rule_array keywords are described in Table 125.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC process rules (One, optional)</td>
<td></td>
</tr>
<tr>
<td>EMVMAC</td>
<td>EMV padding rule with a single-length MAC key. The key_identifier parameter must identify a single-length MAC or a single-length DATA key. The text is always padded with 1 - 8 bytes so the resulting text length is a multiple of eight bytes. The first pad character is X'80'. The remaining pad characters are X'00'.</td>
</tr>
<tr>
<td>EMVMACD</td>
<td>EMV padding rule with a double-length MAC key. The key_identifier parameter must identify a double-length MAC key. The padding rules are the same as for keyword EMVMAC.</td>
</tr>
<tr>
<td>TDES-MAC</td>
<td>ANSI X9.9-1 procedure using ISO 16609 CBC mode triple-DES (TDES) encryption of the data. Uses a double-length key.</td>
</tr>
<tr>
<td>X9.19OPT</td>
<td>ANSI X9.19 optional double key MAC procedure. The key_identifier parameter must identify a double-length MAC key. The padding rules are the same as for keyword X9.9-1.</td>
</tr>
<tr>
<td>X9.9-1</td>
<td>ANSI X9.9-1 and X9.19 basic procedure. The key_identifier parameter must identify a single-length MAC or a single-length DATA key. X9.9-1 causes the MAC to be computed from all of the data. The text is padded only if the text length is not a multiple of eight bytes. If padding is required, the pad character X'00' is used. This is the default value.</td>
</tr>
<tr>
<td>Segmenting control (One, optional)</td>
<td></td>
</tr>
<tr>
<td>FIRST</td>
<td>First call; this is the first segment of data from the application program.</td>
</tr>
<tr>
<td>LAST</td>
<td>Last call; this is the last data segment.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>Middle call; this is an intermediate data segment.</td>
</tr>
<tr>
<td>ONLY</td>
<td>Only call; segmenting is not employed by the application program. This is the default value.</td>
</tr>
<tr>
<td>MAC length and presentation (One, optional)</td>
<td></td>
</tr>
<tr>
<td>HEX-8</td>
<td>Generates a 4-byte MAC value and presents it as 8 hexadecimal characters.</td>
</tr>
<tr>
<td>HEX-9</td>
<td>Generates a 4-byte MAC value and presents it as two groups of 4 hexadecimal characters with a space between the groups.</td>
</tr>
<tr>
<td>MACLEN4</td>
<td>Generates a 4-byte MAC value. This is the default value.</td>
</tr>
<tr>
<td>MACLEN6</td>
<td>Generates a 6-byte MAC value.</td>
</tr>
<tr>
<td>MACLEN8</td>
<td>Generates an 8-byte MAC value.</td>
</tr>
</tbody>
</table>

Chaining vector

Direction: Input/Output
Type: String

An 18-byte string that CCA uses as a system work area. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.

On the first call, initialize this parameter as binary zeros.

mac

428 Common Cryptographic Architecture Application Programmer's Guide
The 8-byte or 9-byte field in which the verb returns the MAC value if the segmenting rule is ONLY or LAST. Allocate an 8-byte field for MAC values of four bytes, six bytes, eight bytes, or HEX-8. Allocate a 9-byte MAC field if you specify HEX-9 in the rule_array parameter.

**Restrictions**

The restrictions for CSNBMGN.

It might seem intuitive that a DATAM key should also be usable for the MAC Generate verb, and a DATAMV key for the MAC Verify verb, with the CPACF exploitation layer. However, this would violate the security restrictions imposed by the user when the user creates a key of type DATAM or DATAMV. A DES key that has been translated for use with the CPACF (see “CPACF support” on page [14](#)) can be used with CPACF DES encrypt and decrypt operations, an operation that is by definition not allowed for a DATAM or DATAMV key type. Also note that by definition both through z/OS CCA-ICSF and in this S390 Linux CCA access layer, a DATA key of 16 bytes or 24 bytes in length is restricted from use with the X9.19OPT and EMVMACD rule_array keyword specified MAC algorithms. The only available MAC algorithm for a 16-byte or 24-byte DATA key is the TDES-MAC algorithm.

Also note that the CPACF exploitation layer is activated only for MAC Generate or MAC Verify calls that specify the ONLY rule_array keyword for segmenting control (this is the default segmenting control if no segmenting control rule_array keyword is specified). The reason for this is that the intermediate MAC context for normal CEX*C calls to MAC Generate and MAC Verify is protected by the adapter Master Key. Because the same security cannot be provided for intermediate results from the host-based CPACF exploitation layer (they are returned in the clear by the CPACF) the FIRST, MIDDLE, and LAST segmenting control keywords will direct operations to the CEX*C.

**Required commands**

The required commands for CSNBMGN.

This verb requires the MAC Generate command (offset X’0010’) to be enabled in the active role.

**Usage notes**

The usage notes for CSNBMGN.

None.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBMGNJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBMGNJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
```
MAC Generate (CSNBMGN)

byte[] exit_data,
byte[] key_identifier,
hikmNativeNumber text_length,
byte[] text,
hikmNativeNumber rule_array_count,
byte[] rule_array,
byte[] chaining_vector,
byte[] mac);

MAC Generate2 (CSNBMGN2)

Use the MAC Generate2 service to generate a keyed hash message authentication code (HMAC) or a ciphered message authentication code (CMAC) for the message string provided as input. A MAC key with key usage that can be used for generate is required to calculate the MAC.

The MAC generate key must be in a variable-length HMAC key token for HMAC and an AES MAC token for CMAC.

Format

The format of CSNBMGN2.

| CSNBMGN2( |
| return_code, |
| reason_code, |
| exit_data_length, |
| exit_data, |
| rule_array_count, |
| rule_array, |
| key_identifier_length, |
| key_identifier, |
| message_text_length, |
| message_text, |
| chaining_vector_length, |
| chaining_vector, |
| mac_length, |
| mac) |

Parameters

The parameters for CSNBMGN2.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

rule_array_count

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 0, 1, 2, or 3.

rule_array

Direction: Input
Type: String array

The rule_array contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with
Table 126. Keywords for MAC Generate2 control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token algorithm (One, required)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>Specifies the use of the AES CMAC algorithm to generate a MAC.</td>
</tr>
<tr>
<td>HMAC</td>
<td>Specifies the use of the HMAC algorithm to generate a MAC.</td>
</tr>
<tr>
<td>Hash method (One, required for HMAC only)</td>
<td></td>
</tr>
<tr>
<td>SHA-1</td>
<td>Specifies the use of the SHA-1 hash method.</td>
</tr>
<tr>
<td>SHA-224</td>
<td>Specifies the use of the SHA-224 hash method.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Specifies the use of the SHA-256 hash method.</td>
</tr>
<tr>
<td>SHA-384</td>
<td>Specifies the use of the SHA-384 hash method.</td>
</tr>
<tr>
<td>SHA-512</td>
<td>Specifies the use of the SHA-512 hash method.</td>
</tr>
<tr>
<td>Segmenting Control (One, optional)</td>
<td></td>
</tr>
<tr>
<td>ONLY</td>
<td>Only call. Segmenting is not employed by the application program. This is the default value.</td>
</tr>
<tr>
<td>FIRST</td>
<td>First call. This is the first segment of data from the application program.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>Middle call. This is an intermediate data segment.</td>
</tr>
<tr>
<td>LAST</td>
<td>Last call. This is the last data segment.</td>
</tr>
</tbody>
</table>

**key_identifier_length**

- **Direction:** Input/Output
- **Type:** String

The length in bytes of the key_identifier parameter. If the key_identifier parameter contains a label, the value must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**key_identifier**

- **Direction:** Input/Output
- **Type:** String

The identifier of the key to generate the MAC. The key identifier is an operational token or the key label of an operational token in key storage.

For the HMAC algorithm, the key algorithm must be HMAC and the key usage fields must indicate GENONLY or GENERATE and the hash method selected. For the AES algorithm, the key algorithm must be AES, the key type must be MAC, and the key usage fields must indicate GENONLY or GENERATE and must indicate CMAC.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**message_text_length**

- **Direction:** Input
- **Type:** Integer

The length of the text you supply in the message_text parameter. The maximum length of text is 214783647 bytes. For FIRST and MIDDLE calls, the message_text_length must be:
  - a multiple of 64 for the SHA-1, SHA-224, and SHA-256 hash methods
MAC Generate2 (CSNBMGN2)

- a multiple of 128 for the SHA-384 and SHA-512 hash methods
- a multiple of 16 for the AES CMAC method.

**message_text**

- **Direction:** Input
- **Type:** String

The application-supplied text for which the MAC is generated.

**chaining_vector_length**

- **Direction:** Input/Output
- **Type:** Integer

A pointer to an integer variable specifying the length in bytes of the **chaining_vector** parameter. The value must be 128.

**chaining_vector**

- **Direction:** Input/Output
- **Type:** String

A pointer to a string variable containing a work area that the security server uses to carry segmented data between procedure calls. When the segmenting control is **FIRST** or **ONLY**, this value is ignored, but must be declared.

**mac_length**

- **Direction:** Input/Output
- **Type:** Integer

The length of the **mac** parameter in bytes. This parameter is updated to the actual length of the **MAC** parameter on output. For HMAC, the minimum value is 4 and the maximum value is 64. For AES, the value must be 16.

**mac**

- **Direction:** Output
- **Type:** String

The field in which the callable service returns the MAC value if the segmenting rule is **ONLY** or **LAST**.

**Restrictions**

The restrictions for CSNBMGN2.

None.

**Required commands**

The required commands for CSNBMGN2.

The MAC Generate2 verb requires the commands listed in [Table 127](#) to be enabled in the active role based on the keyword specified for the process rule:

*Table 127. CSNBMGN2 access control points.*

<table>
<thead>
<tr>
<th>Hash method / Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES CMAC</td>
<td>X'0336'</td>
<td>MAC Generate2 - AES CMAC</td>
</tr>
</tbody>
</table>
Table 127. CSNBMGN2 access control points (continued).

<table>
<thead>
<tr>
<th>Hash method / Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>X'00E4'</td>
<td>HMAC Generate - SHA-1</td>
</tr>
<tr>
<td>SHA-224</td>
<td>X'00E5'</td>
<td>HMAC Generate - SHA-224</td>
</tr>
<tr>
<td>SHA-256</td>
<td>X'00E6'</td>
<td>HMAC Generate - SHA-256</td>
</tr>
<tr>
<td>SHA-384</td>
<td>X'00E7'</td>
<td>HMAC Generate - SHA-384</td>
</tr>
<tr>
<td>SHA-512</td>
<td>X'00E8'</td>
<td>HMAC Generate - SHA-512</td>
</tr>
</tbody>
</table>

1) A role with this offset enabled can also use the HMAC Generate verb.

Usage notes
The usage notes for CSNBMGN2.

None.

Related information
Additional information about CSNBMGN2.

The MAC Verify2 verb is described in “MAC Verify2 (CSNBMVR2)” on page 438.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNBMGN2J.

See “Building Java applications using the CCA JNI” on page 28.

Format
public native void CSNBMGN2J(
  hikmNativeNumber return_code,
  hikmNativeNumber reason_code,
  hikmNativeNumber exit_data_length,
  byte[] exit_data,
  hikmNativeNumber rule_array_count,
  byte[] rule_array,
  hikmNativeNumber key_identifier_length,
  byte[] key_identifier,
  hikmNativeNumber message_text_length,
  byte[] message_text,
  hikmNativeNumber chaining_vector_length,
  byte[] chaining_vector,
  hikmNativeNumber mac_length,
  byte[] mac);

MAC Verify (CSNBMVR)
When the receiver gets a message, an application program calls the MAC Verify verb.

This verb verifies a 4-byte, 6-byte, or 8-byte Message Authentication Code (MAC) for an application-supplied text string. This verb verifies a MAC by generating another MAC and comparing it with the MAC received with the message. This process takes place entirely within the secure module on the coprocessor. If the two codes are the same, the message sent was the same one received. A return code
indicates whether the MACs are the same. The generated MAC never appears in
storage and is not revealed outside the cryptographic feature.

The MAC Verify verb can use any of the following methods to generate the MAC
for authentication:

- The ANSI X9.9-1 single key algorithm, a single-length MAC verification or MAC
generation key (or a data-encrypting key), and the message text.
- The ANSI X9.19 optional double key algorithm, a double-length MAC
verification or MAC generation key and the message text.
- Using the Europay, MasterCard and Visa (EMV) padding rules.

The method used to verify the MAC should correspond with the method used to
generate the MAC.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.

Host CPU acceleration: CPACF

Only keys with a key type of DATA can be used successfully with the CPACF
exploitation layer through this verb.

Specifically, a DATA key has a CV (Control Vector) of all X'00' bytes for all active
bytes of the CV (eight bytes for 8-byte DES keys, 16 bytes for 16-byte DES keys,
and 16 bytes for 24-byte DES keys).

For details about CPACF, see "CPACF support" on page 14.

Format

The format of CSNBMVR.

<table>
<thead>
<tr>
<th>CSNBMVR(</th>
</tr>
</thead>
<tbody>
<tr>
<td>return_code,</td>
</tr>
<tr>
<td>reason_code,</td>
</tr>
<tr>
<td>exit_data_length,</td>
</tr>
<tr>
<td>exit_data,</td>
</tr>
<tr>
<td>key_identifier,</td>
</tr>
<tr>
<td>text_length,</td>
</tr>
<tr>
<td>text,</td>
</tr>
<tr>
<td>rule_array_count,</td>
</tr>
<tr>
<td>rule_array,</td>
</tr>
<tr>
<td>chaining_vector,</td>
</tr>
<tr>
<td>mac)</td>
</tr>
</tbody>
</table>

Parameters

The parameters for CSNBMVR.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data
parameters, see "Parameters common to all verbs" on page 22.

key_identifier

Direction: Input/Output
Type: String

The 64-byte key label or internal key token that identifies a single-length or
double-length MAC verify key, a single-length or double-length MAC
generation key, or a single-length DATA key. The type of key depends on the
MAC process rule in the rule_array parameter.

**text_length**

Direction: Input  
Type: Integer

You supply the length of the cleartext in the text parameter. If the text_length parameter is not a multiple of eight bytes and if the ONLY or LAST keyword of the rule_array parameter is called, the text is padded in accordance with the processing rule specified.

**text**

Direction: Input  
Type: String

The application-supplied text for which the MAC is verified.

**rule_array_count**

Direction: Input  
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 0, 1, 2, or 3.

**rule_array**

Direction: Input  
Type: String array

Zero to three keywords that provide control information to the verb. The keywords are described in Table 128. The keywords must be in 24 bytes of contiguous storage with each of the keywords left-aligned in its own 8-byte location and padded on the right with blanks. For example, 'X9.9-1 MIDDLE MACLEN4 '

The order of the rule_array keywords is not fixed.

You can specify one of the MAC processing rules, and then choose one of the segmenting control keywords and one of the MAC length keywords. The rule_array keywords are described in Table 128.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAC process rules</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td><strong>EMVMAC</strong></td>
<td>EMV padding rule with a single-length MAC key. The key_identifier parameter must identify a single-length MAC, MACVER, or DATA key. The text is always padded with 1 - 8 bytes, so that the resulting text length is a multiple of eight bytes. The first pad character is X'80'. The remaining pad characters are X'00'.</td>
</tr>
<tr>
<td><strong>EMVMACD</strong></td>
<td>EMV padding rule with a double-length MAC key. The key_identifier parameter must identify a double-length MAC or MACVER key. The padding rules are the same as for EMVMAC.</td>
</tr>
<tr>
<td><strong>TDES-MAC</strong></td>
<td>ANSI X9.9-1 procedure using ISO 16609 CBC mode triple-DES (TDES) encryption of the data. Uses a double-length key.</td>
</tr>
<tr>
<td><strong>X9.9-1</strong></td>
<td>ANSI X9.9-1 and X9.19 basic procedure. The key_identifier parameter must identify a single-length MAC, MACVER, or DATA key. X9.9-1 causes the MAC to be computed from all the data. The text is padded only if the text length is not a multiple of eight bytes. If padding is required, the pad character X'00' is used. This is the default value.</td>
</tr>
</tbody>
</table>
Table 128. Keywords for MAC Verify control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X9.19OPT</td>
<td>ANSI X9.19 optional double-length MAC procedure. The key_identifier parameter must identify a double-length MAC or MACVER key. The padding rules are the same as for X9.9-1.</td>
</tr>
</tbody>
</table>

**Segmenting control (Optional)**

| FIRST      | First call. This is the first segment of data from the application program. |
| LAST       | Last call. This is the last data segment.                                  |
| MIDDLE     | Middle call. This is an intermediate data segment.                         |
| ONLY       | Only call. The application program does not employ segmenting. This is the default value. |

**MAC length and presentation (Optional)**

| HEX-8      | Verifies a 4-byte MAC value represented as 8 hexadecimal characters.       |
| HEX-9      | Verifies a 4-byte MAC value represented as two groups of 4 hexadecimal characters with a space character between the groups. |
| MACLEN4    | Verifies a 4-byte MAC value. This is the default value.                    |
| MACLEN6    | Verifies a 6-byte MAC value.                                              |
| MACLEN8    | Verifies an 8-byte MAC value.                                             |

**chaining_vector**

- **Direction:** Input/Output
- **Type:** String

An 18-byte string CCA uses as a system work area. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.

On the first call, initialize this parameter to binary zeros.

**mac**

- **Direction:** Input
- **Type:** String

The 8-byte or 9-byte field that contains the MAC value you want to verify. The value in the field must be left-aligned and padded with zeros. If you specified the HEX-9 keyword in the rule_array parameter, the input MAC is nine bytes in length.

**Restrictions**

The restrictions for CSNBMR.

It might seem intuitive that a DATAM key should also be usable for the MAC Generate verb, and a DATAMV key for the MAC Verify verb, with the CPACF exploitation layer. However, this would violate the security restrictions imposed by the user when the user creates a key of type DATAM or DATAMV. A DES key that has been translated for use with the CPACF (see "CPACF support" on page 14) can be used with CPACF DES encrypt and decrypt operations, an operation that is by definition not allowed for a DATAM or DATAMV key type. Also note that by definition both through z/OS CCA-ICSF and in this S390 Linux CCA access layer, a DATA key of 16 bytes or 24 bytes in length is restricted from use with the X9.19OPT and EMVMACD rule_array keyword specified MAC algorithms. The only available MAC algorithm for a 16-byte or 24-byte DATA key is the TDES-MAC algorithm.
Also note that the CPACF exploitation layer is activated only for MAC Generate or MAC Verify calls that specify the **ONLY** `rule_array` keyword for segmenting control (this is the default segmenting control if no segmenting control `rule_array` keyword is specified). The reason for this is that the intermediate MAC context for normal CEX*C calls to MAC Generate and MAC Verify is protected by the adapter Master Key. Because the same security cannot be provided for intermediate results from the host-based CPACF exploitation layer (they are returned in the clear by the CPACF) the **FIRST, MIDDLE, and LAST** segmenting control keywords will direct operations to the CEX*C.

**Required commands**

The required commands for CSNBMVR.

This verb requires the **MAC Verify** command (offset X'0011') to be enabled in the active role.

**Usage notes**

The usage notes for CSNBMVR.

To verify a MAC in one call, specify the **ONLY** keyword on the segmenting rule keyword for the `rule_array` parameter. For two or more calls, specify the **FIRST** keyword for the first input block, **MIDDLE** for intermediate blocks (if any), and **LAST** for the last block.

For a given text string, the MAC resulting from the verification process is the same regardless of how the text is segmented or how it was segmented when the original MAC was generated.

**Related information**

Additional information for CSNBMVR.

The MAC Generate verb is described in “MAC Generate (CSNBMGN)” on page 426.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBMVRJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```
public native void CSNBMVRJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] key_identifier,
    hikmNativeNumber text_length,
    byte[] text,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] chaining_vector,
    byte[] mac);
```
MAC Verify2 (CSNBMVR2)

Use the MAC Verify2 verb to verify a keyed hash message authentication code (HMAC) or a ciphered message authentication code (CMAC) for the message text provided as input. A MAC key with key usage that can be used for verify is required to verify the MAC.

The MAC verify key must be in a variable-length HMAC key token for HMAC and an AES MAC token for CMAC.

Format

The format of CSNBMVR2.

```c
CSNBMVR2(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_length,
    key_identifier,
    message_text_length,
    message_text,
    chaining_vector_length,
    chaining_vector,
    mac_length,
    mac
)
```

Parameters

The parameters for CSNBMVR2.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1, 2, or 3.

**rule_array**

- **Direction:** Input
- **Type:** String array

The `rule_array` contains keywords that provide control information to the MAC Verify2 callable service. The keywords are described in Table 129.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token algorithm</td>
<td>(One, required)</td>
</tr>
<tr>
<td>AES</td>
<td>Specifies the use of the AES CMAC algorithm to generate a MAC.</td>
</tr>
<tr>
<td>HMAC</td>
<td>Specifies the use of the HMAC algorithm to generate a MAC.</td>
</tr>
<tr>
<td>Hash method</td>
<td>(One required for HMAC only)</td>
</tr>
</tbody>
</table>
Table 129. Keywords for MAC Verify2 control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>Specifies the use of the SHA-1 hash method.</td>
</tr>
<tr>
<td>SHA-224</td>
<td>Specifies the use of the SHA-224 hash method.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Specifies the use of the SHA-256 hash method.</td>
</tr>
<tr>
<td>SHA-384</td>
<td>Specifies the use of the SHA-384 hash method.</td>
</tr>
<tr>
<td>SHA-512</td>
<td>Specifies the use of the SHA-512 hash method.</td>
</tr>
</tbody>
</table>

Segmenting Control (One optional)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIRST</td>
<td>First call. This is the first segment of data from the application program.</td>
</tr>
<tr>
<td>LAST</td>
<td>Last call. This is the last data segment.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>Middle call. This is an intermediate data segment.</td>
</tr>
<tr>
<td>ONLY</td>
<td>Only call. Segmenting is not employed by the application program. This is the default value.</td>
</tr>
</tbody>
</table>

**key_identifier_length**

Direction: Input  
Type: Integer

Length of the *key_identifier* parameter in bytes. If the *key_identifier* parameter contains a label, the value must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**key_identifier**

Direction: Input/Output  
Type: String

The identifier of the key to verify the MAC. The key identifier is an operational token or the key label of an operational token in key storage.

For the HMAC algorithm, the key algorithm must be HMAC and the key usage fields must indicate GENERATE or VERIFY and the hash method selected. For the AES algorithm, the key algorithm must be AES, the key type must be MAC, and the key usage fields must indicate GENERATE or VERIFY and must indicate CMAC.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**message_text_length**

Direction: Input  
Type: Integer

The length of the cleartext you supply in the *message_text* parameter. The maximum length of text is 214783647 bytes. For **FIRST** and **MIDDLE** calls, the *message_text_length* must be:

- a multiple of 64 for the SHA-1, SHA-224, and SHA-256 hash methods,
- a multiple of 128 for the SHA-384 and SHA-512 hash methods,
- a multiple of 16 for the AES CMAC method.

**message_text**

Direction: Input  
Type: String
**MAC Verify2 (CSNBMVR2)**

The application-supplied text for which the MAC is generated.

**chaining_vector_length**

**Direction:** Input  
**Type:** Integer

Specifies the length in bytes of the *chaining_vector* parameter. The value must be 128.

**chaining_vector**

**Direction:** Input/Output  
**Type:** String

A pointer to a string variable containing a work area that the security server uses to carry segmented data between procedure calls. When the segmenting control is **FIRST** or **ONLY**, this value is ignored but must be declared. Important: Application programs must not alter the contents of this variable between related **FIRST**, **MIDDLE**, and **LAST** calls.

**mac_length**

**Direction:** Input  
**Type:** Integer

Specifies the length in bytes of the *mac* parameter. The value must be equal to the number of MAC bytes to be verified, up to a maximum of 64.

**mac**

**Direction:** Input  
**Type:** String

The field that contains the MAC value you want to verify.

**Restrictions**

The restrictions for CSNBMVR2.

None.

**Required commands**

The required commands for CSNBMVR2.

The CSNBMVR2 verb requires the commands listed in Table 130 to be enabled in the active role.

*Table 130. Required commands for CSNBMVR2.*

<table>
<thead>
<tr>
<th>Hash method/Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>X'0337'</td>
<td>MAC Verify2 - AES CMAC</td>
</tr>
<tr>
<td>SHA-1</td>
<td>X'00F7'</td>
<td>HMAC Verify - SHA-1</td>
</tr>
<tr>
<td>SHA-224</td>
<td>X'00F8'</td>
<td>HMAC Verify - SHA-224</td>
</tr>
<tr>
<td>SHA-256</td>
<td>X'00F9'</td>
<td>HMAC Verify - SHA-256</td>
</tr>
<tr>
<td>SHA-384</td>
<td>X'00FA'</td>
<td>HMAC Verify - SHA-384</td>
</tr>
<tr>
<td>SHA-512</td>
<td>X'00FB'</td>
<td>HMAC Verify - SHA-512</td>
</tr>
</tbody>
</table>

1) A role with this offset enabled can also use the HMAC Verify verb.
Usage notes

The usage notes for CSNBMVR2.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBMVR2J.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBMVR2J(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber key_identifier_length,
    byte[] key_identifier,
    hikmNativeNumber message_text_length,
    byte[] message_text,
    hikmNativeNumber chaining_vector_length,
    byte[] chaining_vector,
    hikmNativeNumber mac_length,
    byte[] mac);
```

MDC Generate (CSNBMDG)

Use this verb to create a 128-bit hash value (Modification Detection Code) on a data string whose integrity you intend to confirm.

After using this verb to generate an MDC, you can compare the MDC to a known value or communicate the value to another entity so that they can compare the MDC hash value to one that they calculate. This verb enables you to perform the following tasks:

- Specify the two-encipherment or four-encipherment version of the algorithm.
- Segment your text into a series of verb calls.
- Use the default or a keyed-hash algorithm.

The user must enable the MDC Generate command (offset X'008A') with a Trusted Key Entry (TKE) workstation before using this verb.

For a description of the MDC calculations, see “Modification Detection Code calculation” on page 1024.

Specifying two or four encipherments: Four encipherments per algorithm round improve security; two encipherments per algorithm round improve performance. To specify the number of encipherments, use the MDC-2, MDC-4, PADMDC-2, or PADMDC-4 keyword with the rule_array parameter. Two encipherments create results that differ from four encipherments; ensure that the same number of encipherments are used to verify the MDC.

Segmenting text: This verb lets you segment text into a series of verb calls. If you can present all of the data to be hashed in a single invocation of the verb (32 MB) of data, use the rule_array keyword ONLY. Alternatively, you can segment your
MDC Generate (CSNBMDG)

text and present the segments with a series of verb calls. Use the rule_array keywords and LAST for the first and last segments. If more than two segments are used, specify the rule_array keyword MIDDLE for the additional segments.

Between verb calls, unprocessed text data and intermediate information from the partial MDC calculation is stored in the chaining_vector variable and the MDC key in the MDC variable. During segmented processing, the application program must not change the data in either of these variables.

**Keyed hash:** This verb can be used with a default key, or as a keyed-hash algorithm. A default key is used whenever the ONLY or FIRST segmenting and key control keywords are used. To use the verb as a keyed-hash algorithm, do the following:

1. On the first call to the verb, place the non-null key into the MDC variable.
2. Ensure that the chaining_vector variable is set to null (18 bytes of X'00').
3. Decide if the text will be processed in a single segment or multiple segments.
   - For a single segment of text, use the LAST keyword.
   - For multiple segments of text, begin with the MIDDLE keyword and continue using the MIDDLE keyword up to the final segment of text. For the final segment, use the LAST keyword.

As with the default key, you must not alter the value of the MDC or chaining_vector variables between calls.

**Format**
The format of CSNBMDG.

```c
CSNBMDG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    text_length,
    text,
    rule_array_count,
    rule_array,
    chaining_vector,
    mdc)
```

**Parameters**
The parameters for CSNBMDG.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see "Parameters common to all verbs" on page 22.

**text_length**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Integer</td>
</tr>
</tbody>
</table>

A pointer to an integer variable containing the number of bytes of data in the text variable. See "Restrictions" on page 444.

**text**

| Direction: | Input |
Type: String

A pointer to a string variable containing the text for which the verb calculates the MDC value.

rule_array_count

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value can be 0, 1, or 2.

rule_array

Direction: Input
Type: String array

Keywords that provide control information to the verb. A keyword specifies the method for calculating the RSA digital signature. Each keyword is left-aligned in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage. The rule_array keywords are described in Table 131.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segmenting and key control (One, optional)</td>
<td></td>
</tr>
<tr>
<td>ONLY</td>
<td>Specifies that segmenting is not used and the default key is used. This is the default.</td>
</tr>
<tr>
<td>FIRST</td>
<td>Specifies the first segment of text, and use of the default key.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>Specifies an intermediate segment of text, or the first segment of text and use of a user-supplied key.</td>
</tr>
<tr>
<td>LAST</td>
<td>Specifies the last segment of text, or that segmenting is not used, and use of a user-supplied key.</td>
</tr>
<tr>
<td>Algorithm mode (One, optional)</td>
<td></td>
</tr>
<tr>
<td>MDC-2</td>
<td>Specifies two encipherments for each 8-byte block using MDC procedures. This is the default.</td>
</tr>
<tr>
<td>MDC-4</td>
<td>Specifies four encipherments for each 8-byte block using MDC procedures.</td>
</tr>
<tr>
<td>PADMDC-2</td>
<td>Specifies two encipherments for each 8-byte block using PADMDC procedures.</td>
</tr>
<tr>
<td>PADMDC-4</td>
<td>Specifies four encipherments for each 8-byte block using PADMDC procedures.</td>
</tr>
</tbody>
</table>

chaining_vector

Direction: Input/Output
Type: String

A pointer to an 18-byte string variable the security server uses as a work area to hold segmented data between verb invocations.

Important: When segmenting text, the application program must not change the data in this string between verb calls to the MDC Generate verb.

mdc

Direction: Input/Output
Type: String

A pointer to a user-supplied MDC key or to a 16-byte string variable containing the MDC value. This value can be the key that the application...
MDC Generate (CSNBMDG)

This variable is also used to hold the intermediate MDC result when segmenting text.

**IMPORTANT:** When segmenting text, the application program must not change the data in this string between verb calls to the MDC Generate verb.

### Restrictions

The restrictions for CSNBMDG.

- When padding is requested (by specifying an algorithm mode keyword of `PADMDC-2` or `PADMDC-4`), a text length of zero is valid for any segment-control keyword specified in the `rule_array` variable `FIRST`, `MIDDLE`, `LAST`, or `ONLY`). When `LAST` or `ONLY` is specified, the supplied text is padded with X'FF' bytes and a padding count in the last byte to bring the total text length to the next multiple of 8 that is greater than or equal to 16.

- When no padding is requested (by specifying an algorithm mode keyword of `MDC-2` or `MDC-4`), the total length of text provided (over a single or segmented calls) must be a minimum of 16 bytes and a multiple of eight bytes. For segmented calls (that is, segmenting and key control keyword is not `ONLY`), a text length of zero is valid on any of the calls.

### Required commands

The required commands for CSNBMDG.

In releases prior to CCA 4.1.0 and for installations without CPACF support this verb requires the **MDC Generate** command (offset X'008A') to be enabled in the active role. This command is no longer required in CCA 4.1.0 when CPACF clear-key function is available (KM, function 1), and enabled for CCA use (environment variable CSU_HCPUACLR is set to '1', the default value).

The user must enable the **MDC Generate** command with a Trusted Key Entry (TKE) workstation before using this verb.

### Usage notes

The usage notes for CSNBMDG.

None.

### JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBMDGJ.

See [“Building Java applications using the CCA JNI” on page 28](#).

### Format

```java
public native void CSNBMDGJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber text_length,
    byte[] text_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] chaining_vector,
    byte[] mdc);
```
One-Way Hash (CSNBOWH)

Use the One-Way Hash verb to generate a one-way hash on specified text.

These SHA based hashing functions are supported with the CPACF exploitation layer: SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512. For details about CPACF, see “CPACF support” on page 14.

Format

The format of CSNBOWH.

```
CSNBOWH(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    text_length,
    text,
    chaining_vector_length,
    chaining_vector,
    hash_length,
    hash)
```

Parameters

The parameters for CSNBOWH.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1 or 2.

**rule_array**

- **Direction:** Input
- **Type:** String array

These keywords provide control information to the verb. The optional chaining flag keyword indicates whether calls to this verb are chained together logically to overcome buffer size limitations. Each keyword is left-aligned in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage. The `rule_array` keywords are described in Table 132.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash method (One_required). The SHA-based hashing functions use CPACF by default. For details about CPACF, see “CPACF support” on page 14.</td>
<td></td>
</tr>
<tr>
<td>MD5</td>
<td>Hash algorithm is MD5 algorithm. Use this hash method for PKCS-1.0 and PKCS-1.1. Length of hash generated is 16 bytes.</td>
</tr>
<tr>
<td>RPMD-160</td>
<td>Hash algorithm is RIPEMD-160. Length of hash generated is 20 bytes.</td>
</tr>
</tbody>
</table>
Table 132. Keywords for One-Way Hash control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>Hash algorithm is SHA-1 algorithm. Length of hash generated is 20 bytes.</td>
</tr>
<tr>
<td>SHA-224</td>
<td>Hash algorithm is SHA-224 algorithm. Length of hash generated is 20 bytes.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Hash algorithm is SHA-256 algorithm. Length of hash generated is 20 bytes.</td>
</tr>
<tr>
<td>SHA-384</td>
<td>Hash algorithm is SHA-384 algorithm. Length of hash generated is 20 bytes.</td>
</tr>
<tr>
<td>SHA-512</td>
<td>Hash algorithm is SHA-512 algorithm. Length of hash generated is 20 bytes.</td>
</tr>
</tbody>
</table>

**Chaining flag (One, optional)**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIRST</td>
<td>Specifies this is the first call in a series of chained calls. Intermediate results are stored in the hash field.</td>
</tr>
<tr>
<td>LAST</td>
<td>Specifies this is the last call in a series of chained calls.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>Specifies this is a middle call in a series of chained calls. Intermediate results are stored in the hash field.</td>
</tr>
<tr>
<td>ONLY</td>
<td>Specifies this is the only call and the call is not chained. This is the default.</td>
</tr>
</tbody>
</table>

**text_length**

- **Direction:** Input
- **Type:** Integer

The length of the text parameter in bytes.

**Note:** If you specify the FIRST or MIDDLE keyword, the text length must be a multiple of the block size of the hash method. For MD5, RPMD-160, and SHA-1, this is a multiple of 64 bytes.

For ONLY and LAST, this verb performs the required padding according to the algorithm specified.

**text**

- **Direction:** Input
- **Type:** String

The application-supplied text on which this verb performs the hash.

**chaining_vector_length**

- **Direction:** Input
- **Type:** Integer

The byte length of the chaining_vector parameter. This must be 128 bytes.

**chaining_vector**

- **Direction:** Input/Output
- **Type:** String

This field is a 128-byte work area. Your application must not change the data in this string. The chaining vector permits chaining data from one call to another.

**hash_length**

- **Direction:** Input
- **Type:** Integer
The length of the supplied hash field in bytes.

Note: For SHA-1 and RPMD-160 this must be a minimum of 20 bytes. For MD5 this must be a minimum of 16 bytes.

hash

Direction: Input/Output
Type: String

This field contains the hash, left-aligned. The processing of the rest of the field depends on the implementation. If you specify the FIRST or MIDDLE keyword, this field contains the intermediate hash value. Your application must not change the data in this field between the sequence of FIRST, MIDDLE, and LAST calls for a specific message.

Restrictions

The restrictions for CSNBOWH.

None.

Required commands

The required commands for CSNBOWH.

None.

Usage notes

The usage notes for CSNBOWH.

Although the algorithms accept zero bit length text, it is not supported for any hashing method.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBOWHJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBOWHJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber text_length,
    byte[] text,
    hikmNativeNumber chaining_vector_length,
    byte[] chaining_vector,
    hikmNativeNumber hash_length,
    byte[] hash);
```
One-Way Hash (CSNBOWH)
Chapter 11. Key storage mechanisms

You can use key storage mechanisms and the associated key record verbs to perform operations on key tokens and key records located in AES, DES, and PKA key storage.

A key-token record consists of a key-token name (key label) and a key token of format null, internal, or external. The operations to be performed are: creating, writing, reading, listing, and deleting key tokens or key records.

The verbs described in this chapter include:

- “AES Key Record Create (CSNBAKRC)” on page 455
- “AES Key Record Delete (CSNBAKRD)” on page 457
- “AES Key Record List (CSNBAKRL)” on page 459
- “AES Key Record Read (CSNBAKRR)” on page 462
- “AES Key Record Write (CSNBAKRW)” on page 464
- “DES Key Record Create (CSNBKRC)” on page 466
- “DES Key Record Delete (CSNBKRD)” on page 467
- “DES Key Record List (CSNBKRL)” on page 469
- “DES Key Record Read (CSNBKRR)” on page 471
- “DES Key Record Write (CSNBKRW)” on page 473
- “PKA Key Record Create (CSNDKRC)” on page 474
- “PKA Key Record Delete (CSNDKRD)” on page 476
- “PKA Key Record List (CSNDKRL)” on page 478
- “PKA Key Record Read (CSNDKRR)” on page 480
- “PKA Key Record Write (CSNDKRW)” on page 482
- “Retained Key Delete (CSNDRKD)” on page 484
- “Retained Key List (CSNDRKL)” on page 487

Key labels and key-storage management

Use these verbs to manage AES, DES, and PKA key storage.

The CCA software manages key storage as an indexed repository of key records. Access key storage using a key label with verbs that have a key-label or key-identifier parameter.

An independent key-storage system can be used to manage records for AES key records, DES key records, and PKA key records:

**AES key storage**

Holds external, internal, and null AES or HMAC key tokens

**DES key storage**

Holds external, internal, and null DES key tokens

**PKA key storage**

Holds external, internal, and null PKA (ECC and RSA) key tokens, and both internal and external public and private PKA key tokens
Private RSA keys are generated and optionally retained within the coprocessor using the PKA Key Generate verb. Depending on the other uses for coprocessor storage, between 75 and 150 keys can normally be retained within the coprocessor.

Key storage must be initialized before any records are created. Before a key token can be stored in key storage, a key-storage record must be created using the AES Key Record Create, DES Key Record Create, or PKA Key Record Create verb.

Use the AES Key Record Delete, DES Key Record Delete, or PKA Key Record Delete verb to delete a key token from a key record, or to entirely delete the key record from key storage.

Use the AES Key Record List, DES Key Record List, or PKA Key Record List verb to determine the existence of key records in key storage. These list verbs create a key-record-list file with information about select key records. The wildcard character, represented by an asterisk (*), is used to obtain information about multiple key records. The file can be read using conventional workstation-data-management services.

Individual key tokens can be read using the AES Key Record Read, DES Key Record Read, and PKA Key Record Read verbs or written using the AES Key Record Write, DES Key Record Write, and PKA Key Record Write verbs.

**Environment variables for the key storage file**

These environment variables contain the name of the key storage file.

There is one for each type: AES, DES, and PKA.

**CSUAESDS**

AES key storage file.

**CSUDESDS**

DES key storage file.

**CSUPKADS**

PKA key storage file.

**Key-label content**

Use a key label to identify a record in key storage managed by a CCA implementation.

The key label must be left-aligned in the 64-byte string variable used as input to the verb. Some verbs use a key label while others use a key identifier. Calls that use a key identifier accept either a key token or a key label.

A key-label character string has the following properties:

- It contains 64 bytes of data.
- The first character is within the range X'20' - X'FE'. If the first character is within this range, the input is treated as a key label, even if it is otherwise not valid. Inputs beginning with a byte valued in the range X'00' - X'1F' are considered to be some form of key token. A first byte valued to X'FF' is not valid.
- The first character of the key label cannot be numeric (0 - 9).
- The label is ended by a space character on the right (in ASCII it is X'20', and in EBCDIC it is X'40'). The remainder of the 64-byte field is padded with space characters.
• Construct a label with 1 - 7 name tokens, each separated by a period (.). The key label must not end with a period.

• A name token consists of 1 - 8 characters in the character set A - Z, 0 - 9, and three additional characters relating to different character symbols in the various national language character sets as listed in Table 133.

Table 133. Valid symbols for the name token

<table>
<thead>
<tr>
<th>ASCII systems</th>
<th>EBCDIC systems</th>
<th>USA graphic (for reference)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'23'</td>
<td>X'7B'</td>
<td>#</td>
</tr>
<tr>
<td>X'24'</td>
<td>X'5B'</td>
<td>$</td>
</tr>
<tr>
<td>X'40'</td>
<td>X'7C'</td>
<td>@</td>
</tr>
</tbody>
</table>

The alphabetic and numeric characters and the period should be encoded in the normal character set for the computing platform that is in use, either ASCII or EBCDIC.

Note:
1. Some CCA implementations accept the characters a - z and fold these to their uppercase equivalents, A - Z. For compatibility reasons, only use the uppercase alphabetic characters.
2. Some implementations internally transform the EBCDIC encoding of a key label to an ASCII string. Also, the label might be put in tokenized form by dropping the periods and formatting each name token into 8-byte groups, padded on the right with space characters.

Some verbs accept a key label containing a wild card represented by an asterisk (*). (X'2A' in ASCII; X'5C' in EBCDIC). When a verb permits the use of a wild card, the wild card can appear as the first character, as the last character, or as the only character in a name token. Any of the name tokens can contain a wild card.

Examples of valid key labels include the following:

A
ABCD.2.3.4.5555
ABCD.EFGH
BANKSYS.XXXXX.43*PDQ

Examples of key labels that are not valid are listed in Table 134.

Table 134. Key labels that are not valid

<table>
<thead>
<tr>
<th>Key label not valid</th>
<th>Problem with key label</th>
</tr>
</thead>
<tbody>
<tr>
<td>A/./B</td>
<td>A slash is an unacceptable character</td>
</tr>
<tr>
<td>ABCDEFGH9</td>
<td>Name token is greater than 8 characters</td>
</tr>
<tr>
<td>1111111.2.3.4.55555</td>
<td>First character cannot be numeric</td>
</tr>
<tr>
<td>A1111111.2.3.4.55555.6.7.8</td>
<td>Number of name tokens exceeds 7</td>
</tr>
<tr>
<td>BANKSYS.XXXXX.<em>43</em>.PDQ</td>
<td>Number of wild cards exceeds 1</td>
</tr>
<tr>
<td>A.B.</td>
<td>Last character cannot be a period</td>
</tr>
</tbody>
</table>

Key storage with Linux on Z, in contrast to z/OS

Key storage for IBM z/OS and for Linux on the IBM platforms other than IBM Z, diverged in design at their very inception.
Background information about master key management

There are four types (or sets) of master keys (Symmetric DES, AES, Asymmetric RSA (PKA), and APKA).

There are three master key registers for each of the four types of master key. In other words, there are a total of twelve master key registers.

The APKA master-key register set, introduced to CCA beginning with Release 4.1.0, is used to encrypt and decrypt the Object Protection Key (OPK) that is itself used to wrap the key material of an Elliptic Curve Cryptography (ECC) key. ECC keys are asymmetric.

For each of the four types, there is a master key register in one of these three categories:

New master-key (NMK) register
This register holds a master key that is not yet usable for decrypting key tokens for normal cryptographic operations.

The NMK register can be in one of these states:

EMPTY
No key parts have been loaded yet.

PARTIALLY FULL
Some key parts have been loaded, but not the LAST key part. See “Master Key Process (CSNBMKP)” on page 157.

FULL
The LAST key part has been loaded, but the SET command has not yet been called. See “Master Key Process (CSNBMKP)” on page 157.

Current master-key (CMK) register
This register holds a master key that can be used to decrypt internal key tokens for keys in use with normal cryptographic operations. Internal key tokens are protected under the master key; the keys are actually stored outside the adapter.

The CMK register can be in one of these states:

EMPTY
No valid key has yet been established with the SET command in the life of this adapter, or the adapter has been re-initialized to clear the master key registers.

VALID
A master key has been loaded with the SET command.

Old master-key (OMK) register
This is the master key that previously has been the CMK, before the master key that is now in the CMK register. The OMK register can also be used to decrypt internal key tokens, but for these keys a warning with return code 0 and reason code 2 is returned, along with the results from the requested cryptographic operation.

The OMK register can be in one of these states:

EMPTY
No valid key is in this register.

VALID
A master key that previously was in the CMK register has been
shifted to the OMK register by the SET command. The same invocation of the SET command also shifted the contents of the NMK register into the CMK register.

**SET command**
The SET command transfers the current master-key to the old master-key register, and the new master-key to the current master-key register. It then clears the new master-key register.

The SET command is invoked with "Master Key Process (CSNBMKP)" on page 157, and performs these commands:
1. The master key from the CMK register is copied to the OMK register.
2. The master key from the FULL NMK register is copied to the CMK register.
3. The NMK register status is changed to EMPTY.

**Key Storage on z/OS (RTNMK-focused)**

*Design point* - Keys should be re-enciphered to a master key in the NMK register.

This forces the following process to be followed when changing the master key:

- Load all the master key parts for a NMK, such that the LAST key part has been loaded, but the SET command has not been issued. Now the NMK register is in the FULL state.
- Re-encipher all of (for example: CKDS) an existing key storage to a copy of that key storage that is not online, using the RTNMK rule_array keyword of "Key Token Change (CSNBKTC)" on page 301 (for AES or DES) or "PKA Key Token Change (CSNDKTC)" on page 710 (for PKA), creating CKDS-pending. Keys in this copy are enciphered under the NMK register, and so are not usable for normal cryptographic operations.
- Invoke the SET command for the NMK. See "SET command." Now the master keys in the current CKDS are enciphered under the OMK (because of the shift), and are usable. Also, the master keys in the CKDS-pending are also usable because the NMK has now become the CMK.
- Delete the old CKDS and change CKDS-pending to be the normal CKDS, completing the process.

**Key Storage for traditional IBM systems other than IBM Z (RTCMK-focused: Linux, AIX, Windows)**

*Design point* - Keys should be re-enciphered to a master key in the CMK register.

This forces the following process to be followed when changing the master key:

- Load all the master key parts for a NMK, such that the LAST key part has been loaded, then issue the SET command. Now the previous OMK is gone, the previous CMK is now the OMK, and the CMK contains the newly-loaded value. See "SET command."
- Re-encipher all of an existing CCA host key storage data file's key tokens, which are enciphered under the OMK, to be enciphered under the CMK. This is done using the RTCMK rule_array keyword of "Key Token Change (CSNBKTC)" on page 301 or "PKA Key Token Change (CSNDKTC)" on page 710.
  - This immediately replaces operational keys with the re-enciphered version.
  - The CCA key storage file has a data structure with the verification pattern of the most recently SET master key. The key storage implementation also allows
writing external tokens into the key storage. This means that external key
tokens, and the internal key tokens encrypted under current master key, will
be allowed into the key storage.

It is impossible with current implementation to use RTN MK together with
CCA key storage.

• During the re-encipherment:
  – Some of the keys in the CCA key storage files are enciphered under the OMK
    (because of the shift) and are usable
  – Some of the keys in the CCA key storage files are enciphered under the CMK,
    either because they are new or because they have been re-enciphered.
  – No new key tokens can be created with the key wrapped using the OMK.
    Both types are usable for cryptographic operations.

**Changing the master key for two or more adapters that have
the same master key, with shared CCA key storage**

Because the verification pattern of the current master key (CMK) is stored in a
header in key storage, changing the master key for a configuration of multiple
adapters required extra care in prior releases.

Starting with the CCA 6.0 host library, the master key verification pattern in the
key storage header is not checked with the purpose of allowing or disallowing the
use of an adapter. Therefore, the old workaround (change the master key for one
card, move key storage files to a temporary location, change the master key for the
other cards, move the key storage files back), is no longer needed. There is no
change to the format of the key storage files themselves, the change is only to the
key storage handling in the CCA host library.

**Key storage file ownership**

The last user to access the key storage file owns it, due to the internals of the key
storage functions.

The file is recreated after being compressed, and due to the file creation the owner
is changed.

Having the set-group-id bit (g+s) on in the directory permission causes the file to
be created with the group owner the same as the directory group owner. The
group read/write permissions on the file then allow the other members of the
group continued access to the file.

**The Linux on Z approach**

Because the CCA key storage design point for the Linux platform host release has
always been CMK-focused, this design point was taken forward for the Linux on Z
approach.

At this time, CCA host key storage does not support nor ship with an additional
utility to manage the ‘store-in-pending’ approach to re-enciphering key tokens. This
additional utility is necessary to work with use of the RTN MK keyword for
"Key
Token Change (CSNBKTC)" on page 301 and "PKA Key Token Change
(CSNDKTC)" on page 710. Therefore, it is suggested that users wanting to make
use of CCA host key storage management follow the ‘RTN MK-focused’ approach
described in "Key Storage for traditional IBM systems other than IBM Z
(RTN MK-focused: Linux, AIX, Windows)" on page 453.
However it is also desirable to provide as much host-support equivalence with the z/OS approach as possible, given that the underlying system is running on an IBM Z platform and likely to collaborate with z/OS software. Therefore, the RTNMK keyword is provided for “Key Token Change (CSNBKTC)” on page 301 and “PKA Key Token Change (CSNDKTC)” on page 710 to allow users who have their own utility or key storage management facility to manage key tokens using the method most familiar from z/OS:

- The key tokens to be enciphered should be passed directly (not by label) to “Key Token Change (CSNBKTC)” on page 301 and “PKA Key Token Change (CSNDKTC)” on page 710 for re-encipherment, and stored outside CCA host key storage.
- When re-encipherment is complete and the “Master Key Process (CSNBMKP)” on page 157 SET’ command has been issued, the re-enciphered key tokens can be reintroduced to CCA host key storage if desired, using the standard mechanisms.

**AES Key Record Create (CSNBAKRC)**

Use the AES Key Record Create verb to create an AES or HMAC key-token record in AES key-storage.

The new key record can be a null AES key-token or a valid internal fixed-length AES key-token or an external or internal variable-length AES or HMAC key-token. It is identified by the key label specified with the `key_label` parameter.

After creating an AES key-record, use any of the following verbs to add or update a key token in the record:
- AES Key Record Delete
- AES Key Record Write
- Key Generate
- Key Token Change
- Key Token Change2
- Symmetric Key Generate
- Symmetric Key Import
- Symmetric Key Import2

**Note:**
1. To delete a key record from AES key-storage, use the AES Key Record Delete verb.
2. AES key records are stored in the external key-storage file defined by the CSUAESDS environment variable.
AES Key Record Create (CSNBAKRC)

Format
The format of CSNBAKRC.

```
CSNBAKRC (  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array_count,  
    key_label,  
    key_token_length,  
    key_token)
```

Parameters
The parameters for CSNBAKRC.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

`rule_array_count`
- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0.

`rule_array`
- **Direction:** Input
- **Type:** String array

This parameter is ignored.

`key_label`
- **Direction:** Input
- **Type:** String

A pointer to a string variable containing the key label of the AES key-record to be created.

`key_token_length`
- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the `key_token` variable. If the value of the `key_token_length` variable is zero, a record with a null AES key-token is created.

`key_token`
- **Direction:** Input
- **Type:** String

A pointer to a string variable containing the key token being written to AES key-storage.
Restrictions
The restrictions for CSNBAKRC.

The record must have a unique label. Therefore, there cannot be another record in the AES key storage file with the same label and a different key type.

Required commands
The required commands for CSNBAKRC.

None.

Usage notes
The usage notes for CSNBAKRC.

None.

Related information
Additional information about CSNBAKRC.

See “Key storage with Linux on Z, in contrast to z/OS” on page 451.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNBAKRCJ.

See “Building Java applications using the CCA JNI” on page 28.

Format
public native void CSNBAKRCJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] key_label,
    hikmNativeNumber key_token_length,
    byte[] key_token);

AES Key Record Delete (CSNBAKRD)

Use the AES Key Record Delete verb to perform one of the tasks listed in the AES key storage file.

- Overwrite (delete) a key token or key tokens in AES key-storage, replacing the key token of each selected record with a null AES key-token.
- Delete an entire key record or key records, including the key label and the key token of each selected record, from AES key-storage.

Identify a task with the rule_array keyword, and the key record or records with the key_label parameter. To identify multiple records, use a wild card (*) in the key label.

Note: AES key records are stored in the external key-storage file defined by the CSUAESDS environment variable.
**AES Key Record Delete (CSNBAKRD)**

**Format**

The format of CSNBAKRD.

```plaintext
CSNBAKRD (  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    key_label)
```

**Parameters**

The parameters for CSNBAKRD.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

- **rule_array_count**
  - Direction: Input
  - Type: Integer
  
  A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0 or 1.

- **rule_array**
  - Direction: Input
  - Type: String array
  
  A pointer to a string variable containing an array of keywords. The keywords are eight bytes in length and must be left-aligned and padded on the right with space characters. The `rule_array` keywords are described in Table 135.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task (One, optional)</td>
<td>Deletes a key token from a key record in AES key storage. This is the default.</td>
</tr>
<tr>
<td>TOKEN-DL</td>
<td>Deletes an entire key record, including the key label, from AES key storage.</td>
</tr>
<tr>
<td>LABEL-DL</td>
<td>Deletes an entire key record, including the key label, from AES key storage.</td>
</tr>
</tbody>
</table>

- **key_label**
  - Direction: Input
  - Type: String
  
  A pointer to a string variable containing the key label of a key-token record or records in AES key-storage. Use a wild card (*) in the `key_label` variable to identify multiple records in key storage.

**Restrictions**

The restrictions for CSNBAKRD.

The record defined by the `key_label` must be unique.
Required commands

The required commands for CSNBAKRD.

None.

Usage notes

The usage notes for CSNBAKRD.

None.

Related information

Additional information about CSNBAKRD.

See “Key storage with Linux on Z, in contrast to z/OS” on page 451.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBAKRDJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBAKRDJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] key_label);
```

AES Key Record List (CSNBAKRL)

The AES Key Record List verb creates a key-record-list file containing information about specified key records in key storage.

Information listed includes whether record validation is correct, the type of key, and the date and time the record was created and last updated.

Specify the key records to be listed using the key-label variable. To identify multiple key records, use the wild card (*) in the key label.

Note:

1. To list all the labels in key storage, specify the key_label parameter with *, **, ***.*, and so forth, up to a maximum of seven name tokens (**.*.*.*.*.*.*).
2. AES key records are stored in the external key-storage file defined by the CSUAESDS environment variable.

This verb creates the key-record-list file and returns the name of the file and the length of the file name to the calling application. This file has a header record, followed by 0 - n detail records, where n is the number of key records with matching key-labels.

The file is kept in the /opt/IBM/CCA/keys/deslist directory (assuming the directory name was not changed during installation). These list files are created
AES Key Record List (CSNBAKRL)

under the ownership of the environment of the user that requests the list service. Make sure the files created kept the same group ID as your installation requires. This can also be achieved by setting the 'set-group-id-on-execution' bit on in this directory. See the g+s flags in the chmod command for full details. Not doing this might cause errors to be returned on key-record-list verbs.

Format

The format of CSNBAKRL.

```
CSNBAKRL(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_label,
    dataset_name_length,
    dataset_name,
    security_server_name)
```

Parameters

The parameters for CSNBAKRL.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

`rule_array_count`

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0.

`rule_array`

Direction: Input
Type: String array

A pointer to a string variable containing an array of keywords. This verb currently does not use keywords.

`key_label`

Direction: Input
Type: String

A pointer to a string variable containing the key label of a key-token record in key storage. In a key label, you can use a wildcard (*) to identify multiple records in key storage.

`dataset_name_length`

Direction: Output
Type: Integer

A pointer to an integer variable containing the number of bytes of data returned by the verb in the `dataset_name` variable. The maximum returned length is 64 bytes.
**dataset_name**

**Direction:** Output  
**Type:** String

A pointer to a string variable containing the name of the file returned by the verb. The file contains the AES key-record information. When the verb stores a key-record-list file, it overlays any older file with the same name.

The file name returned by this verb is defined by the CSUAESLD environment variable.

This verb returns the file name as a fully qualified file specification (for example, /opt/IBM/CCA/keys/KYRLLnnn.LST), where *nnn* is the numeric portion of the name. This value increases by one every time that you use this verb. When this value reaches 999, it resets to 001.

**security_server_name**

**Direction:** Output  
**Type:** String

A pointer to a string variable. The information in this variable is not currently used, but the variable must be declared.

**Restrictions**

The restrictions for CSNBAKRL.

None.

**Required commands**

The required commands for CSNBAKRL.

None.

**Usage notes**

The usage notes for CSNBAKRL.

None.

**Related information**

Additional information about CSNBAKRL.

See “Key storage with Linux on Z, in contrast to z/OS” on page 451.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBAKRLJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBAKRLJ(  
    hikmNativeNumber return_code,  
    hikmNativeNumber reason_code,  
    hikmNativeNumber exit_data_length,  
    byte[] exit_data,  
    hikmNativeNumber rule_array_count,
```
AES Key Record List (CSNBAKRL)

byte[] rule_array,
byte[] key_label,
hikmNativeNumber data_set_name_length,
byte[] data_set_name,
byte[] security_server_name);

AES Key Record Read (CSNBAKRR)

Use the AES Key Record Read verb to read a key-token record from AES key-storage and return a copy of the key token to application storage.

The returned key token can be null. In this event, the key_length variable contains a value of 64 and the key-token variable contains 64 bytes of X'00' beginning at offset 0.

Note: AES key records are stored in the external key-storage file defined by the CSUAESDS environment variable.

Format

The format of CSNBAKRR.

```c
CSNBAKRR (
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_label,
    key_token_length,
    key_token)
```

Parameters

The parameters for CSNBAKRR.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see "Parameters common to all verbs" on page 22.

rule_array_count

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 0.

rule_array

Direction: Input
Type: String array

This parameter is ignored.

key_label

Direction: Input
Type: String

A pointer to a string variable containing the key label of the record to be read from AES key-storage.
**key_token_length**

**Direction:** Input/Output  
**Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the  
*key_token* variable. The maximum length is 64.

**key_token**

**Direction:** Output  
**Type:** String

A pointer to a string variable containing the key token read from AES  
key-storage. This variable must be large enough to hold the AES key token  
being read. On completion, the *key_token_length* variable contains the actual  
length of the token being returned.

**Restrictions**

The restrictions for CSNBAKRR.

None.

**Required commands**

The required commands for CSNBAKRR.

None.

**Usage notes**

The usage notes for CSNBAKRR.

None.

**Related information**

Additional information about CSNBAKRR.

See “Key storage with Linux on Z, in contrast to z/OS” on page 451.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBAKRRJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBAKRRJ(  
    hikmNativeNumber return_code,  
    hikmNativeNumber reason_code,  
    hikmNativeNumber exit_data_length,  
    byte[] exit_data,  
    hikmNativeNumber rule_array_count,  
    byte[] rule_array,  
    byte[] key_label,  
    hikmNativeNumber key_token_length,  
    byte[] key_token);
```
AES Key Record Write (CSNBAKRW)

Use this verb to write a copy of an AES or HMAC key-token from application storage into AES key-storage.

This verb can perform the following processing options:

• Write the new key-token only if the old token was null.
• Write the new key-token regardless of content of the old token.

AES key records are stored in the external key-storage file defined by the CSUAESDS environment variable.

**Note:** Before using this verb, use the verb “AES Key Record Create (CSNBAKRC)” on page 455 to create a key record in the key storage file.

**Format**

The format of CSNBAKRW.

```c
CSNBAKRW (  
  return_code,  
  reason_code,  
  exit_data_length,  
  exit_data,  
  rule_array_count,  
  rule_array,  
  key_label,  
  key_token_length,  
  key_token)
```

**Parameters**

The parameters for CSNBAKRW.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0 or 1.

**rule_array**

- **Direction:** Input
- **Type:** String array

A pointer to a string variable containing an array of keywords. The keywords are eight bytes in length and must be left-aligned and padded on the right with space characters.

The `rule_array` keywords are described in Table 136.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing option</td>
<td>(One, optional)</td>
</tr>
</tbody>
</table>

Table 136. Keywords for AES Key Record Write control information
Table 136. Keywords for AES Key Record Write control information  (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECK</td>
<td>Specifies that the record is written only if a record of the same label in AES key-storage contains a null key-token. This is the default.</td>
</tr>
<tr>
<td>OVERLAY</td>
<td>Specifies that the record is overwritten regardless of the current content of the record in AES key-storage.</td>
</tr>
</tbody>
</table>

**key_label**

**Direction:** Input  
**Type:** String  
A pointer to a string variable containing the key label that identifies the record in AES key-storage where the key token is to be written.

**key_token_length**

**Direction:** Input  
**Type:** Integer  
A pointer to an integer variable containing the number of bytes of data in the key_token variable. This value must be 64.

**key_token**

**Direction:** Input  
**Type:** String  
A pointer to a string variable containing the AES key-token to be written into AES key-storage.

**Restrictions**

The restrictions for CSNBAKRW.

The record defined by the key_label parameter must be unique and must already exist in the key storage file.

**Required commands**

The required commands for CSNBAKRW.

None.

**Usage notes**

The usage notes for CSNBAKRW.

None.

**Related information**

Additional information about CSNBAKRW.

You can use this verb with the key record create verb to write an initial record to key storage. Use it following the Key Import and Key Generate verb to write an operational key imported or generated by these verbs directly to the key storage file.
AES Key Record Write (CSNBAKRW)

See “Key storage with Linux on Z, in contrast to z/OS” on page 451.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBAKRWJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBAKRWJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] key_label,
    hikmNativeNumber key_token_length,
    byte[] key_token);
```

DES Key Record Create (CSNGBKRC)

Use the DES Key Record Create verb to add a key record to the DES key storage file.

The record contains a key token set to binary zeros and is identified by the label passed in the `key_label` parameter. The key label must be unique.

DES key records are stored in the external key-storage file defined by the CSUDESDS environment variable.

Format

The format of CSNGBKRC.

```
CSNGBKRC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_label)
```

Parameters

The parameters for CSNGBKRC.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

`key_label`

- **Direction:** Input
- **Type:** String

The 64-byte label of a record in the DES key storage file that is the target of this verb. The created record contains a key token set to binary zeros and has a key type of NULL.

Restrictions

The restrictions for CSNGBKRC.
The record must have a unique label. Therefore, there cannot be another record in the DES key storage file with the same label and a different key type.

**Required commands**
The required commands for CSNBKRC.

None.

**Usage notes**
The usage notes for CSNBKRC.

None.

**Related information**
Additional information for CSNBKRC.

See “Key storage with Linux on Z, in contrast to z/OS” on page 451.

**JNI version**
This verb has a Java Native Interface (JNI) version, which is named CSNBKRCJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBKRCJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] key_label);
```

---

**DES Key Record Delete (CSNBKRD)**

Use the DES Key Record Delete verb to perform one of the following tasks in the DES key storage file.

- Replace the token in a key record with a null key token
- Delete an entire key record, including the key label, from the key storage file

DES key records are stored in the external key-storage file defined by the CSUDESDS environment variable.

**Format**
The format of CSNBKRD.

```java
CSNBKRD(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_label)
```
Parameters

The parameters for CSNBKRD.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 1.

**rule_array**

- **Direction:** Input
- **Type:** String array

The 8-byte keyword that defines the action to be performed. The rule_array keywords are described in Table 137.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN-DL</td>
<td>Deletes a key token from a key record in DES key storage.</td>
</tr>
<tr>
<td>LABEL-DL</td>
<td>Deletes an entire key record, including the key label, from DES key storage.</td>
</tr>
</tbody>
</table>

**key_label**

- **Direction:** Input
- **Type:** String

The 64-byte label of a record in the key storage file that is the target of this verb.

Restrictions

The restrictions for CSNBKRD.

The record defined by the key_label must be unique.

Required commands

The required commands for CSNBKRD.

None.

Usage notes

The usage notes for CSNBKRD.

None.
Related information

Additional information about CSNBKRD.

See “Key storage with Linux on Z, in contrast to z/OS” on page 451.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBKRDJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBKRDJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] key_label);
```

---

DES Key Record List (CSNBKRL)

The DES Key Record List verb creates a key-record-list file containing information about specified key records in key storage.

Information listed includes whether record validation is correct, the type of key, and the date and time the record was created and last updated.

Specify the key records to be listed using the key-label variable. To identify multiple key records, use the wild card (*) in the key label.

**Note:** To list all the labels in key storage, specify the key_label parameter with *, ***, ***, ***, and so forth, up to a maximum of seven name tokens (**/*.**/*.**/*.**/*.**/*.**/).

This verb creates the key-record-list file and returns the name of the file and the length of the file name to the calling application. This file has a header record, followed by $0 - n$ detail records, where $n$ is the number of key records with matching key-labels. The file is kept in the `/opt/IBM/CCA/keys/deslist` directory (assuming the directory name was not changed during installation). These list files are created under the ownership of the environment of the user that requests the list service. Make sure the files created kept the same group ID as your installation requires. This can also be achieved by setting the “set-group-id-on-execution” bit on in this directory. See the g+s flags in the `chmod` command for full details. Not doing this might cause errors to be returned on key-record-list verbs.

DES key records are stored in the external key-storage file defined by the `CSUDESIDS` environment variable.
DES Key Record List (CSNKRL)

Format
The format of CSNKRL.

```
CSNBKRL(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_label,
    dataset_name_length,
    dataset_name,
    security_server_name)
```

Parameters
The parameters for CSNKRL.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**key_label**
- Direction: Input
- Type: String

The `key_label` parameter is a pointer to a string variable containing the key label of a key-token record in key storage. In a key label, you can use a wild card (*) to identify multiple records in key storage.

**dataset_name_length**
- Direction: Output
- Type: Integer

The `dataset_name_length` parameter is a pointer to an integer variable containing the number of bytes of data returned by the verb in the `dataset_name` variable. The maximum returned length is 64 bytes.

**dataset_name**
- Direction: Output
- Type: String

The `dataset_name` parameter is a pointer to a 64-byte string variable containing the name of the file returned by the verb. The file contains the key-record information.

The verb returns the file name as a fully qualified file specification.

**Note:** When the verb stores a key-record-list file, it overlays any older file with the same name.

**security_server_name**
- Direction: Output
- Type: String

The `security_server_name` parameter is a pointer to a string variable. The information in this variable is not currently used, but the variable must be declared.
Restrictions
The restrictions for CSNBKRL.
None.

Required commands
The required commands for CSNBKRL.
None.

Usage notes
The usage notes for CSNBKRL.
None.

Related information
Additional information about CSNBKRL.
See “Key storage with Linux on Z, in contrast to z/OS” on page 451.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNBKRLJ.
See “Building Java applications using the CCA JNI” on page 28.

Format
```java
public native void CSNBKRLJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] key_label,
    hikmNativeNumber data_set_name_length,
    byte[] data_set_name,
    byte[] security_server_name);
```

DES Key Record Read (CSNBKRR)
Use the DES Key Record Read verb to copy an internal key token from the DES key storage file to application storage.

Other cryptographic services can then use the copied key token directly. The key token can also be used as input to the token copying functions of Key Generate or Key Import verbs to create additional NOCV keys.

DES key records are stored in the external key-storage file defined by the CSUDESDS environment variable.
DES Key Record Read (CSNBKRR)

Format
The format of CSNBKRR.

```
CSNBKRR(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_Label,
    key_token)
```

Parameters
The parameters for CSNBKRR.

For the definitions of the `return_code, reason_code, exit_data_length, and exit_data` parameters, see “Parameters common to all verbs” on page 22.

`key_label`
  Direction: Input
  Type: String

The 64-byte label of a record in the DES key storage file. The internal key token in this record is returned to the caller.

`key_token`
  Direction: Output
  Type: String

The 64-byte internal key token retrieved from the DES key storage file.

Restrictions
The restrictions for CSNBKRR.

The record defined by the `key_label` parameter must be unique and must already exist in the key storage file.

Required commands
The required commands for CSNBKRR.

None.

Usage notes
The usage notes for CSNBKRR.

None.

Related information
Additional information about CSNBKRR.

See “Key storage with Linux on Z, in contrast to z/OS” on page 451.
JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBKRRJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBKRRJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] key_label,
    byte[] key_token);
```

DES Key Record Write (CSNBKRW)

Use the DES Key Record Write verb to copy an internal DES key token from application storage into the DES key storage file.

The key label must be unique and the record must already exist in the key storage file.

DES key records are stored in the external key-storage file defined by the CSUDESDES environment variable.

Note: Before you use this verb, use the DES Key Record Create verb (see “DES Key Record Create (CSNBKRC)” on page 466) to create a key record in the key storage file.

Format

The format of CSNBKRW.

```java
CSNBKRW(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_token,
    key_label)
```

Parameters

The parameters for CSNBKRW.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

`key_token`

- **Direction:** Input/Output
- **Type:** String

The 64-byte internal key token that is written to the DES key storage file.

`key_label`

- **Direction:** Input
- **Type:** String
DES Key Record Write (CSNBKRW)

The 64-byte label of a record in the DES key storage file that is the target of this verb. The record is updated with the internal key token supplied in the key_token parameter.

Restrictions
The restrictions for CSNBKRW.

The record defined by the key_label parameter must be unique and must already exist in the key storage file.

Required commands
The required commands for CSNBKRW.

None.

Usage notes
The usage notes for CSNBKRW.

None.

Related information
Additional information about CSNBKRW.

You can use this verb with the key record create verb to write an initial record to key storage. Use it following the Key Import and Key Generate verb to write an operational key imported or generated by these verbs directly to the key storage file.

See “Key storage with Linux on Z, in contrast to z/OS” on page 451.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNBKRWJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

public native void CSNBKRWJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] key_token,
    byte[] key_label);

PKA Key Record Create (CSNDKRC)

This verb writes a new ECC or RSA key-token record to the PKA key storage file.

The new key record can be a null PKA key-token or an external or internal ECC or RSA key-token.

PKA key records are stored in the external key-storage file defined by the CSUPKADS environment variable.
PKA Key Record Create (CSNDKRC)

Format

The format of CSNDKRC.

```
CSNDKRC(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    key_label,  
    key_token_length,  
    key_token)
```

Parameters

The parameters for CSNDKRC.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 0.

**rule_array**

- **Direction:** Input
- **Type:** String array

This parameter is ignored.

**key_label**

- **Direction:** Input
- **Type:** String

The label of the record to be created, 64-byte character string.

**key_token_length**

- **Direction:** Input
- **Type:** Integer

The length of the field containing the token to be written to the PKA key storage file. If zero is specified, a null token will be added to the file. The maximum value of key_token_length is the maximum length of a private RSA token.

**key_token**

- **Direction:** Input
- **Type:** String

Data to be written to the PKA key storage file if key_token_length is nonzero. An RSA private token in either external or internal format, or an RSA public token.
PKA Key Record Create (CSNDKRC)

Restrictions
The restrictions for CSNDKRC.

None.

Required commands
The required commands for CSNDKRC.

None.

Usage notes
The usage notes for CSNDKRC.

None.

Related information
Additional information about CSNDKRC.

See “Key storage with Linux on Z, in contrast to z/OS” on page 451.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNDKRCJ.

See “Building Java applications using the CCA JNI” on page 28.

Format
public native void CSNDKRCJ(
  hikmNativeNumber return_code,
  hikmNativeNumber reason_code,
  hikmNativeNumber exit_data_length,
  byte[] exit_data,
  hikmNativeNumber rule_array_count,
  byte[] rule_array,
  byte[] key_label,
  hikmNativeNumber key_token_length,
  byte[] key_token);

PKA Key Record Delete (CSNDKRD)

Use PKA Key Record Delete to delete a record from the PKA key storage file.

PKA key records are stored in the external key-storage file defined by the CSUPKADS environment variable.

Format
The format of CSNDKRD.

```
CSNDKRD(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  key_label)
```
Parameters

The parameters for CSNDKRD.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

rule_array_count

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 0 or 1.

rule_array

Direction: Input
Type: String array

Keywords that provide control information to the verb. Each keyword is left-aligned in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage. The rule_array keywords are described in Table 138.

Table 138. Keywords for PKA Key Record Delete control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deletion mode</td>
<td>(One, optional). Specifies whether the record is to be deleted entirely or whether only its contents are to be erased.</td>
</tr>
<tr>
<td>LABEL-DL</td>
<td>Specifies the record will be deleted from the PKA key storage file entirely. This is the default deletion mode.</td>
</tr>
<tr>
<td>TOKEN-DL</td>
<td>Specifies only the contents of the record are to be deleted. The record will still exist in the PKA key storage file, but will contain only binary zeros.</td>
</tr>
</tbody>
</table>

key_label

Direction: Input
Type: String

The label of the record to be deleted, a 64-byte character string.

Restrictions

The restrictions for CSNDKRD.

None.

Required commands

The required commands for CSNDKRD.

None.

Usage notes

The usage notes for CSNDKRD.

None.
PKA Key Record Delete (CSNDKRD)

Related information

Additional information about CSNDKRD.

See “Key storage with Linux on Z, in contrast to z/OS” on page 451.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNDKRDJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNDKRDJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] key_label);
```

PKA Key Record List (CSNDKRL)

The PKA Key Record List verb creates a key-record-list file containing information about specified key records in PKA key-storage.

Information includes whether record validation is correct, the type of key, and the dates and times when the record was created and last updated.

Specify the key records to be listed using the `key_label` parameter. To identify multiple key records, use the wild card (*) in a key label.

**Note:** To list all the labels in key storage, specify the `key_label` parameter with *, **, ***, and so forth, up to a maximum of seven name tokens (*****...***).

This verb creates the list file and returns the name of the file and the length of the file name to the calling application. This verb also returns the name of the security server where the file is stored. The PKA Key Record List file has a header record, followed by 0 - n detail records, where n is the number of key records with matching key labels. The file is kept in the `/opt/IBM/CCA/keys/pkalist` directory (assuming the directory name was not changed during installation). These list files are created under the ownership of the environment of the user that requests the list verb. Make sure the files created kept the same group ID as your installation requires. This can also be achieved by setting the “set-group-id-on-execution” bit on in this directory. See the g+s flags in the `chmod` command for full details. Not doing this might cause errors to be returned on key-record-list verbs.

PKA key records are stored in the external key-storage file defined by the CSUPKADS environment variable.

For information concerning the location of the key-record-list directory, refer to the *IBM 4764 PCI-X Cryptographic Coprocessor CCA Support Program Installation Manual*. 

478  Common Cryptographic Architecture Application Programmer's Guide
Format

The format of CSNDKRL.

```c
CSNDKRL(
    return_code,
    reason_code,
    exit_data_length,
    edit_data,
    rule_array_count,
    rule_array,
    key_label,
    dataset_name_length,
    dataset_name,
    security_server_name)
```

Parameters

The parameters for CSNDKRL.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

`rule_array_count`

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0.

`rule_array`

- **Direction:** Input
- **Type:** String array

This parameter is ignored.

`key_label`

- **Direction:** Output
- **Type:** String

The `key_label` parameter is a pointer to a string variable containing a key record in PKA key-storage. You can use a wild card (*) to identify multiple records in key storage.

`dataset_name_length`

- **Direction:** Input
- **Type:** Integer

The `dataset_name_length` parameter is a pointer to an integer variable containing the number of bytes of data returned in the `dataset_name` variable. The maximum returned length is 64 bytes.

`dataset_name`

- **Direction:** Output
- **Type:** String

The `dataset_name` parameter is a pointer to a 64-byte string variable containing the name of the file returned by the verb. The file contains the key-record information.
PKA Key Record List (CSNDKRL)

The verb returns the file name as a fully qualified file specification.

**Note:** When the verb stores a key-record-list file, it overlays any older file with the same name.

**security_server_name**
- **Direction:** Output
- **Type:** String

The `security_server_name` parameter is a pointer to a string variable. The information in this variable is not currently used, but the variable must be declared.

**Restrictions**
- The restrictions for CSNDKRL.
  - None.

**Required commands**
- The required commands for CSNDKRL.
  - None.

**Usage notes**
- The usage notes for CSNDKRL.
  - None.

**Related information**
- Additional information about CSNDKRL.
  - See “Key storage with Linux on Z, in contrast to z/OS” on page 451.

**JNI version**
- This verb has a Java Native Interface (JNI) version, which is named CSNDKRLJ.
  - See “Building Java applications using the CCA JNI” on page 28.

**Format**
- `public native void CSNDKRLJ(`
  - `hikmNativeNumber return_code,`
  - `hikmNativeNumber reason_code,`
  - `hikmNativeNumber exit_data_length,`
  - `byte[] exit_data,`
  - `hikmNativeNumber rule_array_count,`
  - `byte[] rule_array,`
  - `byte[] key_label,`
  - `hikmNativeNumber data_set_name_length,`
  - `byte[] data_set_name,`
  - `byte[] security_server_name);`

PKA Key Record Read (CSNDKRR)

Reads a record from the PKA key storage file and returns the content of the record. This is true even when the record contains a null PKA token.
PKA key records are stored in the external key-storage file defined by the CSUPKADS environment variable.

**Format**

The format of CSNDKRR.

```plaintext
CSNDKRR(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_label,
    key_token_length,
    key_token)
```

**Parameters**

The parameters for CSNDKRR.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0.

**rule_array**

- **Direction:** Input
- **Type:** String array

This parameter is ignored.

**key_label**

- **Direction:** Input
- **Type:** String

The label of the record to be read, a 64-byte character string.

**key_token_length**

- **Direction:** Input/Output
- **Type:** Integer

The length of the area to which the record is to be returned. On successful completion of this verb, `key_token_length` will contain the actual length of the record returned.

**key_token**

- **Direction:** Output
- **Type:** String

This is the area into which the returned record will be written. The area should be at least as long as the record.
PKA Key Record Read (CSNDKRR)

Restrictions
The restrictions for CSNDKRR.
None.

Required commands
The required commands for CSNDKRR.
None.

Usage notes
The usage notes for CSNDKRR.
None.

Related information
Additional information about CSNDKRR.
See “Key storage with Linux on Z, in contrast to z/OS” on page 451.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNDKRRJ.
See “Building Java applications using the CCA JNI” on page 28.

Format
public native void CSNDKRRJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] key_label,
    hikmNativeNumber key_token_length,
    byte[] key_token);

PKA Key Record Write (CSNDKRW)

Use this verb to write over an existing ECC or RSA key-token record from application storage in the PKA key storage file.

PKA key records are stored in the external key-storage file defined by the CSUPKADS environment variable.
PKA Key Record Write (CSNDKRW)

Format

The format of CSNDKRW.

```c
CSNDKRW(  return_code,
         reason_code,
         exit_data_length,
         exit_data,
rule_array_count,
rule_array,
key_label,
key_token_length,
key_token)
```

Parameters

The parameters for CSNDKRW.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

**rule_array_count**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Integer</td>
</tr>
</tbody>
</table>

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0 or 1.

**rule_array**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>String array</td>
</tr>
</tbody>
</table>

Keywords that provide control information to the verb. Each keyword is left-aligned in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage. The `rule_array` keywords are described in Table 139.

Table 139. Keywords for PKA Key Record Write control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write mode</td>
<td>(One, optional). Specifies the circumstances under which the record is to be written.</td>
</tr>
<tr>
<td>CHECK</td>
<td>Specifies the record will be written only if a record of type NULL with the same label exists in the PKA key storage file. If such a record exists, it is overwritten. This is the default condition.</td>
</tr>
<tr>
<td>OVERLAY</td>
<td>Specifies the record will be overwritten regardless of the current content of the record. If a record with the same label exists in the PKA key storage file, is overwritten.</td>
</tr>
</tbody>
</table>

**key_label**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>String</td>
</tr>
</tbody>
</table>

The label of the record to be overwritten, a 64-byte character string.

**key_token_length**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Integer</td>
</tr>
</tbody>
</table>

PKA Key Record Write (CSNDKRW)

The length of the field containing the token to be written to the PKA key storage file.

**key_token**

- **Direction:** Input
- **Type:** String

A string variable containing an RSA or ECC private token in either external or internal format, or an RSA or ECC public token that is to be written to the PKA key storage file.

**Restrictions**

The restrictions for CSNDKRW.

None.

**Required commands**

The required commands for CSNDKRW.

None.

**Usage notes**

The usage notes for CSNDKRW.

None.

**Related information**

Additional information about CSNDKRW.

See “Key storage with Linux on Z, in contrast to z/OS” on page 451.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNDKRWJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNDKRWJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] key_label,
    hikmNativeNumber key_token_length,
    byte[] key_token);
```

Retained Key Delete (CSNDRKD)

Use this verb to delete a PKA key-record currently retained within the cryptographic engine.
Both public and private keys can be retained within the cryptographic engine using verbs such as PKA Key Generate. A list of retained keys can be obtained using the Retained Key List verb.

**Important:** Before using this verb, see the information about retained keys in "Using retained keys."

### Format

The format of CSNDRKD.

```plaintext
CSNDRKD(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_label)
```

### Parameters

The parameters for CSNDRKD.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

**rule_array_count**

Direction: Input  
Type: Integer  
A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0.

**rule_array**

Direction: Input  
Type: String array  
This parameter is ignored.

**key_label**

Direction: Input  
Type: String  
A pointer to a string variable containing the key label of an RSA key-record that has been retained within the cryptographic engine. The use of a wild card in the `key_label` variable is not permitted.

### Using retained keys

Retained key use is discouraged on the IBM Z platform because a retained key can exist only in one CEX/C Cryptographic adapter, by definition.

- This has potential problems:
  - The key cannot be exported, so it cannot be backed up.
  - The key cannot be exported to another card in the same group, so operations concerning the retained key cannot participate in load-balancing.
There is an exception to the above points, in that keys generated in a deterministic fashion using externally saved regeneration data (it is possible to save so-called 'regen data' securely) can be recreated from that data or created in multiple cards across a card group.

However, this is a very sophisticated topic, and is beyond the scope of this document. Also, the complexity required to implement this properly, as well as the sophistication involved in its data management, present formidable obstacles.

Retained key support is offered in this release, however. The following verbs work with retained keys:

- "PKA Key Generate (CSNDPKG)" on page 689 generates an RSA retained key. The same restrictions that Integrated Cryptographic Service Facility (ICSF) has for retained key creation are implemented here. These are:
  - Notice that PKA Key Token Build will let you create 'key-mgmt' skeleton key tokens, and this is as designed. You can still pass these to PKA Key Generate and have a key pair created. What is not allowed is specifying that this 'key-mgmt' token is to be generated in PKA Key Generate as a RETAIN key token: a retained key. Such an attempt will fail with error 12 reason code 3046.
  - The maximum modulus size is 2048 bits.
  - The usage flags are restricted to signature generation.
    Specifically, key management usage for retained keys is not allowed because of the dangers of losing your key encrypting key (kek) for important keys, when that kek exists only inside a single adapter.
- "Retained Key List (CSNDRKL)" on page 487 lists the retained keys inside an adapter.
- "Retained Key Delete (CSNDRKD)" on page 484 deletes a retained key from adapter internal storage.

Restrictions

The restrictions for CSNDRKD.

None.

Required commands

The required commands for CSNDRKD.

This verb requires the Retained Key Delete command (offset X'0203') to be enabled in the active role.

Usage notes

The usage notes for CSNDRKD.

This verb is not impacted by the AUTOSELECT option. See "Verbs that ignore AUTOSELECT" on page 12 for more information.

Related information

Additional information about CSNDRKD.

See “Key storage with Linux on Z, in contrast to z/OS” on page 451.
JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNDRKDJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

public native void CSNDRKDJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] key_label);

Retained Key List (CSNRKL)

Use this verb to list the key labels of selected PKA key records that have been retained within the cryptographic engine.

Specify the keys to be listed using the key_label_mask variable. To identify multiple keys, use a wildcard (*) in the mask. Only labels with matching characters to those in the mask up to the first “*” is returned. To list all retained key labels, specify a mask of an *, followed by 63 space characters. For example, if the cryptographic engine has retained key labels a.a, a.a1, a.b.c.d, and z.a, and you specify the mask a.*, the verb returns a.a and a.a1. If you specify a mask of a.a*, the verb returns a.a and a.a1.

To retain PKA keys within the coprocessor, use the PKA Key Generate verb. To delete retained keys from the coprocessor, use the Retained Key Delete verb.

Important: Before using this verb, see the information about retained keys in “Using retained keys” on page 485.

Format

The format of CSNRKL.

[CSNRKL(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_label_mask,
    retained_keys_count,
    key_labels_count,
    key_labels)
]

Parameters

The parameters for CSNRKL.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

rule_array_count
Retained Key List (CSNDRKL)

- **Direction:** Input
  **Type:** Integer

  A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0.

  **rule_array**

  - **Direction:** Input
  - **Type:** String array

    This parameter is ignored.

- **key_label_mask**

  - **Direction:** Input
  - **Type:** String

    A pointer to a string variable containing a key-label mask that is used to filter the list of key names returned by the verb. Use a wild card (*) to identify multiple key records retained within the coprocessor.

- **retained_keys_count**

  - **Direction:** Input/Output
  - **Type:** Integer

    A pointer to an integer variable to receive the total number of retained-key records stored within the coprocessor.

- **key_labels_count**

  - **Direction:** Input/Output
  - **Type:** Integer

    A pointer to an integer variable which on input defines the maximum number of key labels to be returned, and which on output defines the number of key labels returned by the coprocessor.

- **key_labels**

  - **Direction:** Output
  - **Type:** String array

    A pointer to a string array variable containing the returned key labels. The coprocessor returns zero or more 64-byte array elements, each of which contains the key label of a PKA key-record retained within the coprocessor.

**Restrictions**

The restrictions for CSNDRKL.

None.

**Required commands**

The required commands for CSNDRKL.

This verb requires the **Retained Key List** command (offset X'0230') to be enabled in the active role.
Usage notes

The usage notes for CSNDRKL.

This verb is not impacted by the AUTOSELECT option. See “Verbs that ignore AUTOSELECT” on page 12 for more information.

Related information

Related information about CSNDRKL.

See “Key storage with Linux on Z, in contrast to z/OS” on page 451.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNDRKLJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNDRKLJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] key_label_mask,
    hikmNativeNumber retained_keys_count,
    hikmNativeNumber key_labels_count,
    byte[] key_labels);
```
Retained Key List (CSNDRKL)
Chapter 12. Financial services

The process of validating personal identities in a financial transaction system is called personal authentication.

The personal identification number (PIN) is the basis for verifying the identity of a customer across financial industry networks. CCA provides verbs to translate, verify, and generate PINs. You can use the verbs to prevent unauthorized disclosures when organizations handle PINs.

The following verbs are described in this topic:

- “Authentication Parameter Generate (CSNBAPG)” on page 508
- “Clear PIN Encrypt (CSNBCPE)” on page 511
- “Clear PIN Generate (CSNPBPGN)” on page 515
- “Clear PIN Generate Alternate (CSNBPCA)” on page 518
- “CVV Generate (CSNBCSG)” on page 523
- “CVV Key Combine (CSNBCKC)” on page 526
- “CVV Verify (CSNBCSV)” on page 531
- “Encrypted PIN Generate (CSNBEPG)” on page 535
- “Encrypted PIN Translate (CSNBTPTR)” on page 539
- “Encrypted PIN Translate Enhanced (CSNBPTRE)” on page 545
- “Encrypted PIN Verify (CSNBPVR)” on page 555
- “FPE Decipher (CSNBFPED)” on page 560
- “FPE Encipher (CSNBFPEE)” on page 568
- “FPE Translate (CSNBFPET)” on page 575
- “PIN Change/Unblock (CSNBPCU)” on page 583
- “Recover PIN from Offset (CSNBFO)” on page 591
- “Secure Messaging for Keys (CSNBSKY)” on page 595
- “Secure Messaging for PINs (CSNBSPN)” on page 599
- “Transaction Validation (CSNBTRV)” on page 603

How personal identification numbers (PINs) are used

CCA allows your applications to generate PINs, to verify supplied PINs, and to translate PINs from one format or encryption key to another.

Many people are familiar with PINs, which are used to access an automated teller machine (ATM). From the system point of view, PINs are used primarily in financial networks to authenticate users. Typically, a user is assigned a PIN and enters the PIN at automated teller machines (ATMs) to gain access to his or her accounts. It is extremely important that the PIN be kept private so no one other than the account owner can use it.
How Visa card verification values are used

The Visa International Service Association (VISA) and MasterCard International, Incorporated have specified a cryptographic method to calculate a value that relates to the personal account number (PAN), the card expiration date, and the service code.

The Visa card-verification value (CVV) and the MasterCard card-verification code (CVC) can be encoded on either track 1 or track 2 of a magnetic striped card or chip card and are used to detect forged cards. Because most online transactions use track-2, the CCA verbs generate and verify the CVV\(^1\) by the track-2 method.

The Visa CVV Generate verb calculates a 1-byte to 5-byte value through the DES-encryption of the PAN, the card expiration date, and the service code using two data-encrypting keys or two MAC keys. The Visa CVV Verify verb calculates the CVV by the same method, compares it to the CVV supplied by the application (which reads the credit card’s magnetic stripe or chip) in the CVV\_value, and issues a return code that indicates whether the card is authentic.

The CVV Key Combine verb combines two operational DES keys into one operational TDES key. The verb accepts as input two single-length keys that are suitable for use with the CVV (card-verification value) algorithm. The resulting double-length key meets a more recent industry standard of using TDES to support PIN-based transactions. In addition, the double-length key is in a format that can be wrapped using the Key Export to TR31 verb.

Translating data and PINs in networks

More and more data is being transmitted across networks where, for various reasons, the keys used on one network cannot be used on another network.

Encrypted data and PINs that are transmitted across these boundaries must be translated securely from encryption under one key to encryption under another key. For example, a traveler visiting a foreign city might want to use an ATM to access an account at home. The PIN entered at the ATM might need to be encrypted at the ATM and sent over one or more financial networks to the traveler’s home bank. At the home bank, the PIN must be verified before access is allowed. On intermediate systems (between networks), applications can use the Encrypted PIN Translate verb to re-encrypt a PIN block from one key to another. Running on CCA, such applications can ensure that PINs never appear in the clear and that the PIN-encrypting keys are isolated on their own networks.

Working with Europay-Mastercard-Visa Smart cards

There are several verbs you can use in secure communications with Europay-Mastercard-Visa (EMV) smart cards.

The processing capabilities are consistent with the specifications provided in these documents:

- EMV 2000 Integrated Circuit Card Specification for Payment Systems Version 4.0 (EMV4.0) Book 2

\(^1\) The Visa CVV and the MasterCard CVC refer to the same value. CVV is used here to mean both CVV and CVC.

EMV smart cards include the following processing capabilities:

- The Diversified Key Generate verb with rule-array options TDES-XOR, TDESEMV2, and TDESEMV4 enables you to derive a key used to cipher and authenticate messages, and more particularly message parts, for exchange with an EMV smart card. You use the derived key with verbs such as: Encipher, Decipher, MAC Generate, MAC Verify, Secure Messaging for Keys, and Secure Messaging for PINs. These message parts can be combined with message parts created using the Secure Messaging for Keys and Secure Messaging for PINs verbs.

- The Secure Messaging for Keys verb enables secure incorporation of a key into a message part (generally the value portion of a TLV component of a secure message for a card). Similarly, the Secure Messaging for PINs verb enables secure incorporation of a PIN block into a message part.

- PIN Change/Unblock verb enables encryption of a new PIN to send to a new EMV card, or to update the PIN value on an initialized EMV card. This verb generates both the required session key (from the master encryption key) and the required authentication code (from the master authentication key).

- The ZERO-PAD option of the PKA Encrypt enables validation of a digital signature created according to ISO 9796-2 standard by encrypting information that you format, including a hash value of the message to be validated. You compare the resulting enciphered data to the digital signature accompanying the message to be validated.

- The MAC Generate and MAC Verify verbs post-pad a X’80’...X’00’ string to a message as required for authenticating messages exchanged with EMV smart cards.

**PIN verbs**

CCA supports PIN verbs, various PIN algorithms, and PIN block formats.

It also explains the use of PIN-encrypting keys.

You use the PIN verbs to generate, verify, and translate PINs.

**Generating a PIN**

To generate personal identification numbers, call the Clear PIN Generate or Encrypted PIN Generate verb.

Using a PIN generation algorithm, data used in the algorithm, and the PIN generation key, the Clear PIN Generate verb generates a clear PIN and a PIN verification value, or offset. Using a PIN generation algorithm, data used in the algorithm, the PIN generation key, and an outbound PIN encrypting key, the Encrypted PIN Generate verb generates and formats a PIN and encrypts the PIN block.

**Encrypting a PIN**

To format a PIN into a supported PIN block format and encrypt the PIN block, call the Clear PIN Encrypt verb.
Generating a PIN validation value from an encrypted PIN block

To generate a clear VISA PIN validation value (PVV) from an encrypted PIN block, call the Clear PIN Generate Alternate verb.

The PIN block can be encrypted under an input PIN-encrypting key (IPINENC) or an output PIN encrypting key (OPINENC).

Verifying a PIN

To verify a supplied PIN, call the Encrypted PIN Verify verb.

You supply the enciphered PIN, the PIN-encrypting key that enciphers the PIN, and other data. You must also specify the PIN verification key and PIN verification algorithm. The Encrypted PIN Verify verb generates a verification PIN. This verb compares the two personal identification numbers and if they are the same, it verifies the supplied PIN.

Translating a PIN

To translate a PIN block format from one PIN-encrypting key to another or from one PIN block format to another, call the Encrypted PIN Translate verb.

You must identify the input PIN-encrypting key that originally enciphered the PIN. You also need to specify the output PIN-encrypting key that you want the verb to use to encipher the PIN. If you want to change the PIN block format, specify a different output PIN block format from the input PIN block format.

Algorithms for generating and verifying a PIN

CCA supports the following algorithms for generating and verifying personal identification numbers.

- IBM 3624 institution-assigned PIN
- IBM 3624 customer-selected PIN (through a PIN offset)
- IBM German Bank Pool PIN (verify through an institution key)
- VISA PIN through a VISA PIN validation value
- Interbank PIN

The algorithms are described in detail in Chapter 22, “PIN formats and algorithms,” on page 1005.

Using PINs on different systems

CCA allows you to translate different PIN block formats, which lets you use personal identification numbers on different systems.

CCA supports the following formats:

- IBM 3624
- IBM 3621 (same as IBM 5906)
- IBM 4704 encrypting PINPAD format
- ISO 0 (same as ANSI 9.8, VISA 1, and ECI 1)
- ISO 1 (same as ECI 4)
- ISO 2
- VISA 2
- VISA 3
• VISA 4
• ECI 2
• ECI 3

The algorithms are described in detail in Chapter 22, “PIN formats and algorithms,” on page 1005.

**PIN-Encrypting keys**

A unique master key variant enciphers each type of key.

Note that the PIN block variant constant (PBVC) are not supported in this version of CCA.

**Derived unique key per transaction algorithms**

CCA supports ANSI X9.24 derived unique key per transaction algorithms to generate PIN-encrypting keys from user data.

CCA supports both single-length and double-length key generation. Keywords for single-length and double-length key generation cannot be mixed.

**Encrypted PIN Translate**

The UKPTIPIN, IPKTOPIN, and UKPTBOTH keywords will cause the verb to generate single-length keys. and DUKPT-IP, DKPT-OP, and DUKPT-BH are the respective keywords to generate double-length keys.

The input_PIN_profile and output_PIN_profile parameters must supply the current key serial number when these keywords are specified.

**Encrypted PIN Verify**

The UKPTIPIN keyword will cause the verb to verify single-length keys. DUKPT-IP is the keyword for double-length key generation.

The input_PIN_profile parameter must supply the current key serial number when these keywords are specified.

---

**ANSI X9.8 PIN restrictions**

These access control points implement the PIN-block processing restrictions of the ANSI X9.8 standard implemented in CCA 4.1.0.

These access control points are available on the IBM z196 starting with the CEX3C feature. These access control points are disabled in the default role. A TKE Workstation is required to enable them.

These are the access control points:

- **ANSI X9.8 PIN - Enforce PIN block restrictions** (offset X'0350'). See “ANSI X9.8 PIN - Enforce PIN block restrictions” on page 496.
- **ANSI X9.8 PIN - Allow modification of PAN** (offset X'0351') See “ANSI X9.8 PIN - Allow modification of PAN” on page 496.
- **ANSI X9.8 PIN - Allow only ANSI PIN blocks** (offset X'0352') See “ANSI X9.8 PIN - Allow only ANSI PIN blocks” on page 497.
• **ANSI X9.8 PIN - Use stored decimalization tables only** (offset X'0356') See “Use stored decimalization tables only” on page 497.

These verbs are affected by these access control points:
- Clear PIN Generate Alternate (CSNBCPA)
- Encrypted PIN Generate (CSNBEGP)
- Encrypted PIN Translate (CSNBPTR)
- Encrypted PIN Verify (CSNPVNR)
- Secure Messaging for PINs (CSNBSPN)

PIN decimalization tables can be stored in the Coprocessor, starting with CEX3C, for use by CCA verbs. Only tables that have been activated can be used. A TKE Workstation is required to manage the tables in the coprocessors.

**ANSI X9.8 PIN - Enforce PIN block restrictions**

When the ANSI X9.8 PIN - Enforce PIN block restrictions access control point is enabled, restrictions are enforced.

The following restrictions are enforced:
- The Encrypted PIN Translate and Secure Messaging for PINs verbs will not accept IBM 3624 PIN format in the output profile parameter when the input profile parameter is not IBM 3624.
- The Encrypted PIN Translate verb will not accept ISO-0 or ISO-3 formats in the input PIN profile unless ISO-0 or ISO-3 is in the output PIN profile.
- The Encrypted PIN Translate and Secure Messaging for PINs verbs will not accept ISO-1 or ISO-2 formats in the output profile parameter when the input profile parameter contains ISO-0, ISO-3, or VISA4.
- When the input profile parameter for the Encrypted PIN Translate and Secure Messaging for PINs verbs contains either ISO-0 or ISO-3 formats, the PAN within the decrypted PIN block will be extracted. This PAN must be the same as the PAN that was supplied as the input PAN parameter, and this PAN must be the same as the PAN supplied as the output PAN parameter.
- The input PAN and output PAN parameters for the Encrypted PIN Translate and Secure Messaging for PINs verbs must be equivalent.
- When the rule array for the Clear PIN Generate Alternate verb contains VISA-PVV, the input PIN profile must contain ISO-0 or ISO-3 formats.

**ANSI X9.8 PIN - Allow modification of PAN**

In order to enable the ANSI X9.8 PIN - Allow modification of PAN access control point, the ANSI X9.8 PIN - Enforce PIN block restrictions must also be enabled.

The ANSI X9.8 PIN - Allow modification of PAN access control point cannot be enabled by itself.

When the ANSI X9.8 PIN - Allow modification of PAN access control point is enabled, the input PAN and output PAN parameters will be tested in the Encrypted PIN Translate and Secure Messaging for PINs verbs. The input PAN will be compared to the portions of the PAN that are recoverable from the decrypted PIN block. If the PANs are the same, the account number will be changed in the output PIN block.
ANSI X9.8 PIN - Allow only ANSI PIN blocks

In order to enable the ANSI X9.8 PIN - Allow only ANSI PIN blocks access control point, the ANSI X9.8 PIN - Enforce PIN block restrictions must also be enabled.

The ANSI X9.8 PIN - Allow only ANSI PIN blocks access control point cannot be enabled by itself.

When this access control point is enabled, the Encrypted PIN Translate verb allows reformatting of the PIN block as shown in Table 140.

<table>
<thead>
<tr>
<th>Reformat to:</th>
<th>ISO Format 0</th>
<th>ISO Format 1</th>
<th>ISO Format 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO Format 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISO Format 1</td>
<td>Reformat permitted</td>
<td>Reformat permitted</td>
<td></td>
</tr>
</tbody>
</table>

Use stored decimalization tables only

The ANSI X9.8 PIN - Use stored decimalization tables only access control point can be enabled by itself.

When this access control point is enabled, the Secure Messaging for PINs, Clear PIN Generate Alternate, Encrypted PIN Generate, and Encrypted PIN Verify verbs must supply a decimalization table that matches the active decimalization tables stored in the coprocessors. The decimalization table in the data_array parameter will be compared against the active decimalization tables in the coprocessor, and if the supplied table matches a stored table, the request will be processed. If the supplied table doesn’t match any of the stored tables or there are no stored tables, the request fails.

PIN decimalization tables can be stored in the in the Coprocessor, starting with CEX3C, for use by CCA verbs. Only tables that have been activated can be used. A TKE Workstation is required to manage the tables in the coprocessors.

The PIN profile

The PIN profile components include a block format, format control, pad digit, and key serial number.

The PIN profile consists of the following:

- PIN block format (see “PIN block format” on page 498)
- Format control (see “Format control” on page 500)
- Pad digit (see “Pad digit” on page 500)
- Current Key Serial Number (for UKPT and DUKPT – see “Current key serial number” on page 501)

Table 141 on page 498 shows the format of a PIN profile.
Table 141. Format of a PIN profile

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 7</td>
<td>PIN block format</td>
</tr>
<tr>
<td>8 - 15</td>
<td>Format control</td>
</tr>
<tr>
<td>16 - 23</td>
<td>Pad digit</td>
</tr>
<tr>
<td>24 - 47</td>
<td>Current Key Serial Number (for UKPT and DUKPT)</td>
</tr>
</tbody>
</table>

**PIN block format**

This keyword specifies the format of the PIN block. The 8-byte value must be left-aligned and padded with blanks.

Refer to Table 142 for a list of valid values.

Table 142. Format values of PIN blocks

<table>
<thead>
<tr>
<th>Format value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECI-2</td>
<td>Eurocheque International format 2</td>
</tr>
<tr>
<td>ECI-3</td>
<td>Eurocheque International format 3</td>
</tr>
<tr>
<td>ISO-0</td>
<td>ISO format 0, ANSI X9.8, VISA 1, and ECI 1</td>
</tr>
<tr>
<td>ISO-1</td>
<td>ISO format 1 and ECI 4</td>
</tr>
<tr>
<td>ISO-2</td>
<td>ISO format 2</td>
</tr>
<tr>
<td>ISO-3</td>
<td>ISO format 3</td>
</tr>
<tr>
<td>VISA-2</td>
<td>VISA format 2</td>
</tr>
<tr>
<td>VISA-3</td>
<td>VISA format 3</td>
</tr>
<tr>
<td>VISA-4</td>
<td>VISA format 4</td>
</tr>
<tr>
<td>3621</td>
<td>IBM 3621 and 5906</td>
</tr>
<tr>
<td>3624</td>
<td>IBM 3624</td>
</tr>
<tr>
<td>4704-EPP</td>
<td>IBM 4704 encrypting PIN pad</td>
</tr>
</tbody>
</table>

**PIN block format and PIN extraction method keywords**

In the Clear PIN Generate Alternate, Encrypted PIN Translate, and Encrypted PIN Verify verbs, you can specify a PIN extraction keyword for a given PIN block format.

In the table below, the allowable PIN extraction methods are listed for each PIN block format. The first PIN extraction method keyword listed for a PIN block format is the default.

Table 143. PIN block format and PIN extraction method keywords

<table>
<thead>
<tr>
<th>PIN block format</th>
<th>PIN extraction method keywords</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECI-2</td>
<td>PINLEN04</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINLEN04 format.</td>
</tr>
<tr>
<td>ECI-3</td>
<td>PINBLOCk</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINBLOCk format.</td>
</tr>
<tr>
<td>ISO-0</td>
<td>PINBLOCk</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINBLOCk format.</td>
</tr>
</tbody>
</table>
Table 143. PIN block format and PIN extraction method keywords (continued)

<table>
<thead>
<tr>
<th>PIN block format</th>
<th>PIN extraction method keywords</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO-1</td>
<td>PINBLOCK</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.</td>
</tr>
<tr>
<td>ISO-2</td>
<td>PINBLOCK</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.</td>
</tr>
<tr>
<td>ISO-3</td>
<td>PINBLOCK</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.</td>
</tr>
<tr>
<td>VISA-2</td>
<td>PINBLOCK</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.</td>
</tr>
<tr>
<td>VISA-3</td>
<td>PINBLOCK</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.</td>
</tr>
<tr>
<td>VISA-4</td>
<td>PINBLOCK</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.</td>
</tr>
<tr>
<td>3621</td>
<td>PADDIGIT, HEXDIGIT, PINLEN04 - PINLEN12, PADEXIST</td>
<td>The PIN extraction method keywords specify a PIN extraction method for an IBM 3621 PIN block format. The first keyword, PADDIGIT, is the default PIN extraction method for the PIN block format.</td>
</tr>
<tr>
<td>3624</td>
<td>PADDIGIT, HEXDIGIT, PINLEN04 - PINLEN16, PADEXIST</td>
<td>The PIN extraction method keywords specify a PIN extraction method for an IBM 3624 PIN block format. The first keyword, PADDIGIT, is the default PIN extraction method for the PIN block format.</td>
</tr>
<tr>
<td>4704-EPP</td>
<td>PINBLOCK</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.</td>
</tr>
</tbody>
</table>

The PIN extraction methods operate as follows:

**PINBLOCK**

Specifies that the service verb use one of these:
- The PIN length, if the PIN block contains a PIN length field
- The PIN delimiter character, if the PIN block contains a PIN delimiter character.

**PADDIGIT**

Specifies that the verb use the pad value in the PIN profile to identify the end of the PIN.

**HEXDIGIT**

Specifies that the verb use the first occurrence of a digit in the range from X’A’ to X’F’ as the pad value to determine the PIN length

**PINLEN** *nn*

Specifies that the verb use the length specified in the keyword, where *nn* can range from 04 - 16, the number of digits used to identify the PIN.

**PADEXIST**

Specifies that the verb use the character in the 16th position of the PIN block as the value of the pad value.
Enhanced PIN security mode

An enhanced PIN security mode is available. This optional mode is selected by enabling the **Enhanced PIN Security** (offset X'0313') access control point in the CEX*C default role.

When active, this control point affects all PIN verbs that extract or format a PIN using a PIN-block format of IBM 3624 with a PIN-extraction method of **PADDIGIT**.

Table 144 summarizes the verbs affected by the enhanced PIN security mode, and describes the effect that the mode has when the access control point is enabled.

<table>
<thead>
<tr>
<th>PIN-block format and PIN-extraction method</th>
<th>Affected verbs</th>
<th>PIN processing changes when Enhanced PIN Security Mode enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECI-2, 3621, or 3624 formats AND PINLENnn</td>
<td>Clear PIN Generate Alternate Encrypted PIN Translate Encrypted PIN Verify</td>
<td>The PINLEnn keyword in the rule_array parameter for PIN extraction method is not allowed if the Enhanced PIN Security Mode is enabled. <strong>Note:</strong> The verb will fail with return code 8 and reason code X'7E0'.</td>
</tr>
<tr>
<td>IBM 3624 format and HEXDIGIT, PADDIGIT, or PADEXIST</td>
<td>Clear PIN Generate Alternate Encrypted PIN Translate Encrypted PIN Verify PIN Change/Unblock</td>
<td>PIN extraction determines the PIN length by scanning from right to left until a digit, not equal to the PAD digit, is found. The minimum PIN length is set at four digits, so scanning ceases one digit past the position of the fourth PIN digit in the block.</td>
</tr>
<tr>
<td>IBM 3624 format and HEXDIGIT, PADDIGIT, or PADEXIST</td>
<td>Clear PIN Encrypt Encrypted PIN Generate Encrypted PIN Translate</td>
<td>PIN formatting does not examine the PIN in the output PIN block to see if it contains the PAD digit.</td>
</tr>
<tr>
<td>IBM 3624 format and HEXDIGIT, PADDIGIT, or PADEXIST</td>
<td>Encrypted PIN Translate</td>
<td>Restricted to non-decimal digit for PADDIGIT.</td>
</tr>
</tbody>
</table>

Format control

This keyword specifies whether there is any control on the user-supplied PIN format.

The 8-byte value must be left-aligned and padded with blanks. The only permitted value is **NONE**, which indicates no format control will be used.

Pad digit

Some PIN formats require the pad digit parameter.

If the PIN format does not need a pad digit, the verb ignores this parameter.

Table 145 shows the format of a pad digit. The PIN profile pad digit must be specified in upper case.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 - 22</td>
<td>Seven space characters</td>
</tr>
</tbody>
</table>
Table 145. Format of a pad digit (continued)

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>Character representation of a hexadecimal pad digit or a space if a pad digit is not needed. Characters must be one of the following: digits 0 - 9, letters A - F, or a blank.</td>
</tr>
</tbody>
</table>

Each PIN format supports only a pad digit in a certain range. Table 146 lists the valid pad digits for each PIN block format.

Table 146. Pad digits for PIN block formats

<table>
<thead>
<tr>
<th>PIN Block Format</th>
<th>Output PIN Profile</th>
<th>Input PIN Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECI-2</td>
<td>Pad digit is not used</td>
<td>Pad digit is not used</td>
</tr>
<tr>
<td>ECI-3</td>
<td>Pad digit is not used</td>
<td>Pad digit is not used</td>
</tr>
<tr>
<td>ISO-0</td>
<td>F</td>
<td>Pad digit is not used</td>
</tr>
<tr>
<td>ISO-1</td>
<td>Pad digit is not used</td>
<td>Pad digit is not used</td>
</tr>
<tr>
<td>ISO-2</td>
<td>Pad digit is not used</td>
<td>Pad digit is not used</td>
</tr>
<tr>
<td>ISO-3</td>
<td>Pad digit is not used</td>
<td>Pad digit is not used</td>
</tr>
<tr>
<td>VISA-2</td>
<td>0 - 9</td>
<td>Pad digit is not used</td>
</tr>
<tr>
<td>VISA-3</td>
<td>0 - F</td>
<td>Pad digit is not used</td>
</tr>
<tr>
<td>VISA-4</td>
<td>F</td>
<td>Pad digit is not used</td>
</tr>
<tr>
<td>3621</td>
<td>0 - F</td>
<td>0 - F</td>
</tr>
<tr>
<td>3624</td>
<td>0 - F</td>
<td>0 - F</td>
</tr>
<tr>
<td>4704-EPP</td>
<td>F</td>
<td>Pad digit is not used</td>
</tr>
</tbody>
</table>

The verb returns an error indicating that the PAD digit is not valid if all of these conditions are met:

- The Enhanced PIN Security (offset X’0313’) access control point is enabled in the active role.
- The output PIN profile specifies 3624 as the PIN-block format.
- The output PIN profile specifies a decimal digit (0 - 9) as the PAD digit.

Recommendations for the pad digit

IBM recommends to use a non-decimal pad digit in the range of A - F when processing IBM 3624 PIN blocks.

If you use a decimal pad digit, the creator of the PIN block must ensure that the calculated PIN does not contain the pad digit, or unpredictable results might occur.

For example, you can exclude a specific decimal digit from being in any calculated PIN by using the IBM 3624 calculation procedure and by specifying a decimalization table that does not contain the desired decimal pad digit.

Current key serial number

The current key serial number is the concatenation of the initial key serial number (a 59-bit value) and the encryption counter (a 21-bit value).

The concatenation is an 80-bit (10-byte) value. Table 147 on page 502 shows the format of the current key serial number.
When **UKPT** or **DUKPT** is specified, the PIN profile parameter is extended to a 48-byte field and must contain the current key serial number.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 - 47</td>
<td>Character representation of the current key serial number used to derive the initial PIN encrypting key. It is left-aligned and padded with 4 blanks.</td>
</tr>
</tbody>
</table>

**Decimalization tables**

Decimalization tables can be loaded in the coprocessors to restrict attacks using modified tables.

The management of the tables requires a TKE workstation.

These verbs make use of the stored decimalization tables:

- Clear PIN Generate (CSNBPGN)
- Clear PIN Generate Alternate (CSNBCPA)
- Encrypted PIN Generate (CSNBEPG)
- Encrypted PIN Verify (CSNPVR)

The **ANSI X9.8 PIN - Use stored decimalization table only** (offset X'0356') access control point is used to restrict the use of the stored decimalization tables. When the access control point is enabled, the table supplied by the verb will be compared against the active tables stored in the coprocessor. If the supplied table does not match any of the active tables, the request will fail.

A TKE workstation (Version 7.1 or later) is required to manage the PIN decimalization tables. The tables must be loaded and then activated. Only active tables are checked when the access control point is enabled.

**Note:** CCA routes work to all active coprocessors based on workload. All coprocessors must have the same set of decimalization tables for the decimalization table access control point to be effective.

**Visa Format Preserving Encryption**

Format preserving encryption (FPE) is a method of encryption where the resulting ciphertext has the same form as the input cleartext. The form of the text can vary according to the usage and the application. The contained information provides background information which is helpful for using the Visa format preserving encryption (VFPE) services provided by CCA. The Visa Format Preserving Encryption (VFPE) has an algorithm that uses an alphabet parameter. An alphabet assigns a sequential number set for all potential characters for a given field type that is used in the conversion of payment card data prior to encryption.

One example for format preserving encryption is a 16 digit credit card number. After using FPE to encrypt a credit card number, the resulting ciphertext is another 16 digit number. In this example of the credit card number, the output ciphertext is limited to numeric digits only.

The FPE services require some knowledge of the input cleartext character set in order to create the appropriate output ciphertext. The CSNFPEE, CSNFPED,
CSNBFPET, and CSNBPTRE callable services use the tables in the following subsections to determine valid character sets for the cleartext input parameters.

VFPE applies to these verbs:
- FPE Decipher (CSNBFPED)
- FPE Encipher (CSNBFPEE)
- FPE Translate (CSNBFPET)
- Encrypted PIN Translate Enhanced (CSNBPTRE)

These CCA verbs use the tables in the following subsections to determine valid character sets for the cleartext input parameters. These services convert payment card data as required to or from VFPE alphabet numbers as determined by rule-array keyword. The alphabet tables below are meant to provide a reference for the valid set of characters for each of the four Visa payment card data formats (namely, PAN, Cardholder Name, Track 1 Discretionary Data, and Track 2 Discretionary Data).

VFPE payment card data can be in any one of these formats:
- For Track 2, a special modified 5-bit ASCII format, which allows parity checking of the digits, as specified in ISO 7811
- For Track 1, a special modified 7-bit ASCII format, which allows parity checking of the digits, as specified in ISO 7811-2 and ISO 7813
- 4-bit Binary Coded Decimal (BCD)
- 7-bit American Standard Code for Information Interchange (ASCII)
- 8-bit Extended Binary Coded Decimal Interchange Code (EBCDIC)

The conversion of payment card data to a VFPE alphabet prior to encryption serves to standardize the data. The converted encryption result is presented to the terminal application for constructing payment transaction data. When the converted encryption result is decrypted, the VFPE alphabet data can be converted back to any desired format.

When VFPE is applied to a transaction, it must always be applied to all occurrences of the following fields (when present), and in the following order:
1. Primary Account Number (PAN)
2. Cardholder Name
3. Track 1 Discretionary Data
4. Track 2 Discretionary Data

Any missing data fields will be skipped.

Each character in the set of characters for a given field type is assigned a unique VFPE alphabet number. VFPE requires translation of each payment card data character to its assigned VFPE alphabet number prior to encryption. Refer to Table 148 on page 504.
Table 148. VFPE alphabet by field type. Table describing the VFPE alphabet by field type in three columns.

<table>
<thead>
<tr>
<th>Field type</th>
<th>VFPE alphabet used</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Account Number (PAN)</td>
<td>• For releases starting with 5.0: BASE-10 alphabet. Refer to Table 149 on page 505</td>
<td>PAN data must be converted into the VFPE BASE-10 alphabet prior to encryption.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data is obtained from Track 1, Track 2, chip medium-scale integration (MSI), or chip account number.</td>
</tr>
<tr>
<td>Cardholder Name</td>
<td>• For releases starting with 5.0: Track 1 Cardholder Name alphabet. Refer to the third column of Table 150 on page 505</td>
<td>Cardholder Name data must be converted into the VFPE Track 1 Cardholder Name alphabet prior to encryption.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data is obtained from Track 1 or chip data.</td>
</tr>
<tr>
<td>Track 1 Discretionary Data</td>
<td>• For releases starting with 5.0: Track 1 Discretionary Data alphabet. Refer to the second column of Table 150 on page 505</td>
<td>Track 1 Discretionary Data must be converted into the VFPE Track 1 Discretionary alphabet prior to encryption.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This table has all of the same characters as the VFPE Track 1 Cardholder Name alphabet plus two reserved name field characters, namely a period &quot;.&quot; and a slash &quot;/&quot;.</td>
<td>Data is obtained from magnetic stripe or chip data.</td>
</tr>
<tr>
<td>Track 2 Discretionary Data</td>
<td>• For release 5.0: BASE-16 alphabet. Refer to Table 151 on page 507</td>
<td>Track 2 Discretionary Data must be converted into the VFPE BASE-10 alphabet prior to encryption.</td>
</tr>
<tr>
<td></td>
<td>• For release 5.2: BASE-10 alphabet. Refer to Table 149 on page 505</td>
<td>Data is obtained from the magnetic stripe or chip data.</td>
</tr>
</tbody>
</table>

Note:
1. Characters that are not found in the alphabet table should be skipped and not encrypted.
2. Reserved characters that are not in the table are intentionally missing. Missing reserved characters can be used for hardware control, start sentinel, field separate, or end sentinel.

VFPE BASE-10 alphabet

The VFPE BASE-10 alphabet is used for converting data when the character set only consists of numbers zero through nine (0 - 9). VFPE requires translation (conversion) of the following data to the VFPE alphabet number in Table 149 on page 505:

- PAN data obtained from payment card Track 1, Track 2, chip MSI, or chip account number
- Track 2 Discretionary Data obtained from the magnetic strip or chip data.

Any of the data types shown in the table are supported. After the formatted encrypted data is decrypted, it can be translated to the same or a different data type than the original coding.
Table 149. VFPE BASE-10 alphabet for PAN data and Track 2 Discretionary Data

<table>
<thead>
<tr>
<th>Character</th>
<th>VFPE BASE-10 alphabet number</th>
<th>ISO 7811 modified 5-bit ASCII</th>
<th>ISO 7811-2 and ISO 7813 Modified 7-bit ASCII</th>
<th>Normal data type encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>4-bit binary coded decimal (BCD)</td>
<td>7-bit ASCII</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>000000</td>
<td>00100000</td>
<td>01100000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>000001</td>
<td>10100011</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>000010</td>
<td>10100100</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>100111</td>
<td>00101111</td>
<td>0101</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>001001</td>
<td>10101001</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>101011</td>
<td>00101011</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>010111</td>
<td>10101111</td>
<td>0111</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>001111</td>
<td>10110111</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>010000</td>
<td>10110000</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>110011</td>
<td>00110101</td>
<td>1001</td>
</tr>
</tbody>
</table>

VFPE Track 1 Discretionary Data and Cardholder Name alphabets

There are two VFPE alphabets for Track 1 data. One is for VFPE Track 1 Discretionary Data, and the other for VFPE Track 1 Cardholder Name data:

- The VFPE Track 1 Discretionary Data alphabet is used for converting the Track 1 Discretionary Data obtained from payment card magnetic stripe or chip data. This alphabet includes two reserved name field characters, namely a period "." and a slash "/", that the VFPE Track 1 Cardholder Name alphabet does not have. If a period ("." or a slash (/") character is encountered in Track 1 Discretionary Data, it should be converted and encrypted.

- The VFPE Track 1 Cardholder Name alphabet is used for converting the cardholder name data obtained from payment card Track 1 or chip data. If a period ("." or a slash (/") character is encountered in Cardholder Name data, it should be skipped and not encrypted.

Refer to Table 150. Any of the data types shown in the table are supported. After the formatted and encrypted data is decrypted, it can be translated to the same or a different data type than the original coding.

Table 150. VFPE Track 1 Discretionary Data and Cardholder Name alphabets

<table>
<thead>
<tr>
<th>Character</th>
<th>VFPE Track 1 Discretionary Data alphabet number</th>
<th>VFPE Track 1 Cardholder Name alphabet number</th>
<th>ISO 7811-2 and ISO 7813 Modified 7-bit ASCII data type</th>
<th>7-bit ASCII normal data type encoding</th>
<th>8-bit EBCDIC normal data type encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>space</td>
<td>0</td>
<td>0</td>
<td>1000000</td>
<td>0100000</td>
<td>01000000</td>
</tr>
<tr>
<td>#</td>
<td>1</td>
<td>1</td>
<td>1000011</td>
<td>0100011</td>
<td>01110111</td>
</tr>
<tr>
<td>$</td>
<td>2</td>
<td>2</td>
<td>0000100</td>
<td>0100100</td>
<td>01011111</td>
</tr>
<tr>
<td>(</td>
<td>3</td>
<td>3</td>
<td>0001000</td>
<td>0101000</td>
<td>01001101</td>
</tr>
<tr>
<td>)</td>
<td>4</td>
<td>4</td>
<td>1001001</td>
<td>0101001</td>
<td>01011111</td>
</tr>
<tr>
<td>-</td>
<td>5</td>
<td>5</td>
<td>0001101</td>
<td>0101101</td>
<td>01100000</td>
</tr>
</tbody>
</table>
Table 150. VFPE Track 1 Discretionary Data and Cardholder Name alphabets (continued)

<table>
<thead>
<tr>
<th>Character</th>
<th>VFPE Track 1 Discretionary Data alphabet number</th>
<th>VFPE Track 1 Cardholder Name alphabet number</th>
<th>ISO 7811-2 and ISO 7813 Modified 7-bit ASCII data type</th>
<th>7-bit ASCII normal data type encoding</th>
<th>8-bit EBCDIC normal data type encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>6</td>
<td>Skip</td>
<td>0001110</td>
<td>0101110</td>
<td>01001011</td>
</tr>
<tr>
<td>/</td>
<td>7</td>
<td>Skip</td>
<td>1001111</td>
<td>0101111</td>
<td>01100001</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>6</td>
<td>0010000</td>
<td>0110000</td>
<td>11110000</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>7</td>
<td>1010001</td>
<td>0110001</td>
<td>11110001</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>8</td>
<td>1010010</td>
<td>0110010</td>
<td>11110010</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>9</td>
<td>0010011</td>
<td>0110101</td>
<td>11110101</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>10</td>
<td>1010100</td>
<td>0110100</td>
<td>11110010</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>11</td>
<td>0010101</td>
<td>0110101</td>
<td>11110101</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>12</td>
<td>0010110</td>
<td>0110110</td>
<td>11110101</td>
</tr>
<tr>
<td>7</td>
<td>15</td>
<td>13</td>
<td>1010111</td>
<td>0110111</td>
<td>11110111</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>14</td>
<td>1011000</td>
<td>0111000</td>
<td>11110000</td>
</tr>
<tr>
<td>9</td>
<td>17</td>
<td>15</td>
<td>0011001</td>
<td>0111001</td>
<td>11110101</td>
</tr>
<tr>
<td>A</td>
<td>18</td>
<td>16</td>
<td>1100001</td>
<td>1000001</td>
<td>11000001</td>
</tr>
<tr>
<td>B</td>
<td>19</td>
<td>17</td>
<td>1100010</td>
<td>1000100</td>
<td>11000100</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
<td>18</td>
<td>0100011</td>
<td>1000111</td>
<td>11000011</td>
</tr>
<tr>
<td>D</td>
<td>21</td>
<td>19</td>
<td>1100100</td>
<td>1000100</td>
<td>11000100</td>
</tr>
<tr>
<td>E</td>
<td>22</td>
<td>20</td>
<td>0100101</td>
<td>1000101</td>
<td>11000101</td>
</tr>
<tr>
<td>F</td>
<td>23</td>
<td>21</td>
<td>0100110</td>
<td>1000110</td>
<td>11000110</td>
</tr>
<tr>
<td>G</td>
<td>24</td>
<td>22</td>
<td>1100111</td>
<td>1000111</td>
<td>11000111</td>
</tr>
<tr>
<td>H</td>
<td>25</td>
<td>23</td>
<td>1101000</td>
<td>1001000</td>
<td>11001000</td>
</tr>
<tr>
<td>I</td>
<td>26</td>
<td>24</td>
<td>0101001</td>
<td>1001001</td>
<td>11001001</td>
</tr>
<tr>
<td>J</td>
<td>27</td>
<td>25</td>
<td>0101010</td>
<td>1001010</td>
<td>11001001</td>
</tr>
<tr>
<td>K</td>
<td>28</td>
<td>26</td>
<td>1101101</td>
<td>1001011</td>
<td>11001001</td>
</tr>
<tr>
<td>L</td>
<td>29</td>
<td>27</td>
<td>0101100</td>
<td>1001100</td>
<td>11001011</td>
</tr>
<tr>
<td>M</td>
<td>30</td>
<td>28</td>
<td>1101101</td>
<td>1001101</td>
<td>11001010</td>
</tr>
<tr>
<td>N</td>
<td>31</td>
<td>29</td>
<td>1101110</td>
<td>1001110</td>
<td>11001010</td>
</tr>
<tr>
<td>O</td>
<td>32</td>
<td>30</td>
<td>0101111</td>
<td>1001111</td>
<td>11010110</td>
</tr>
<tr>
<td>P</td>
<td>33</td>
<td>31</td>
<td>1100000</td>
<td>1010000</td>
<td>11010111</td>
</tr>
<tr>
<td>Q</td>
<td>34</td>
<td>32</td>
<td>0110001</td>
<td>1010001</td>
<td>11011000</td>
</tr>
<tr>
<td>R</td>
<td>35</td>
<td>33</td>
<td>0110010</td>
<td>1010010</td>
<td>11011001</td>
</tr>
<tr>
<td>S</td>
<td>36</td>
<td>34</td>
<td>1110011</td>
<td>1010011</td>
<td>11100010</td>
</tr>
<tr>
<td>T</td>
<td>37</td>
<td>35</td>
<td>0110100</td>
<td>1010100</td>
<td>11100011</td>
</tr>
<tr>
<td>U</td>
<td>38</td>
<td>36</td>
<td>1110101</td>
<td>1010101</td>
<td>11100100</td>
</tr>
<tr>
<td>V</td>
<td>39</td>
<td>37</td>
<td>1110110</td>
<td>1010110</td>
<td>11100101</td>
</tr>
<tr>
<td>W</td>
<td>40</td>
<td>38</td>
<td>0110111</td>
<td>1010111</td>
<td>11100110</td>
</tr>
<tr>
<td>X</td>
<td>41</td>
<td>39</td>
<td>0111000</td>
<td>1011000</td>
<td>11101111</td>
</tr>
<tr>
<td>Y</td>
<td>42</td>
<td>40</td>
<td>1111001</td>
<td>1011001</td>
<td>11101000</td>
</tr>
</tbody>
</table>
### Table 150. VFPE Track 1 Discretionary Data and Cardholder Name alphabets (continued)

<table>
<thead>
<tr>
<th>Character</th>
<th>VFPE Track 1 Discretionary Data alphabet number</th>
<th>VFPE Track 1 Cardholder Name alphabet number</th>
<th>ISO 7811-2 and ISO 7813 Modified 7-bit ASCII data type</th>
<th>7-bit ASCII normal data type encoding</th>
<th>8-bit EBCDIC normal data type encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>43</td>
<td>41</td>
<td>1111010</td>
<td>1011010</td>
<td>11101001</td>
</tr>
<tr>
<td>[</td>
<td>44</td>
<td>42</td>
<td>0111011</td>
<td>1011011</td>
<td>1111010</td>
</tr>
<tr>
<td>\</td>
<td>45</td>
<td>43</td>
<td>1111100</td>
<td>1011100</td>
<td>11100000</td>
</tr>
<tr>
<td>]</td>
<td>46</td>
<td>44</td>
<td>0111110</td>
<td>1011101</td>
<td>10111011</td>
</tr>
</tbody>
</table>

### Base-16 alphabet

Cards are encoded with the special ISO 7811 modified 5-bit ASCII encoding for track 2. This data type allows parity checking of the digits. Many systems require this encoding to be converted into standard data types for processing. Other data fields may use base-16 encoding and would use this same alphabet when performing VFPE. These data types support values in the ranges 0 - 9 and A - F.

VFPE requires translation of the characters of the VFPE alphabet number prior to encryption. Therefore, any of the data types shown in Table 190 are supported. Decryption may use the same or a different data type than the original encoding. This alphabet requires the following values to be used in the VFPE algorithm:

**Number of characters in alphabet (’n’): 16**

### Table 151. Base-16 alphabet

<table>
<thead>
<tr>
<th>VFPE alphabet number</th>
<th>ISO 7811 modified 5-bit ASCII encoding</th>
<th>Normal data type encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Character</td>
<td>Binary</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>10000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>00001</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>00010</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>10011</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>00100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>10101</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>10110</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>00111</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>01000</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>11001</td>
</tr>
<tr>
<td>A</td>
<td>;</td>
<td>11010</td>
</tr>
<tr>
<td>B</td>
<td>:</td>
<td>01011</td>
</tr>
<tr>
<td>C</td>
<td>&lt;</td>
<td>11100</td>
</tr>
<tr>
<td>D</td>
<td>&gt;</td>
<td>01110</td>
</tr>
<tr>
<td>E</td>
<td>?</td>
<td>11111</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Usage notes for FPE Encipher (CSNBFPEE) and FPE Decipher (CSNBFPED) services

The CSNBFPEE and CSNBFPED services support two options:
1. the standard encryption or decryption option which uses the DES CBC mode of operation
2. the Visa Format Preserving Encryption (VFPE) option.

If the standard encryption or decryption option was selected, the plaintext data was formatted into blocks and then encrypted or decrypted with triple-DES with a static TDES key or a DUKPT double length data encryption or decryption key. For the decryption operation, the data blocks must be decrypted and unblocked to produce the plaintext. If the data was encrypted or decrypted with the VFPE option, it was processed in place without changing the data type or length of the field. Also, DUKPT key management is used.

These services can be used to encrypt or decrypt one or all of the following fields:
• the primary account number (PAN),
• the cardholder name,
• the track 1 discretionary data, or
• the track 2 discretionary data.

There are three encryption or decryption options:
1. the standard option with CBC mode TDES and DUKPT keys
2. the VFPE option with DUKPT keys
3. the standard option with CBC mode TDES and double-length TDES keys.

To use these services, you must specify the following:
• the processing method, which is limited to Visa Data Secure Platform (VDSP)
• the key management method, either STATIC or DUKPT
• the algorithm, which is limited to TDES
• the mode, either CBC or Visa Format Preserving Encryption (VFPE)
• the plaintext to be encrypted or decrypted
• the character set of each field to be encrypted or decrypted using rule-array keywords
• the base derivation key and key serial number if DUKPT key management is used, or a double-length TDES key if STATIC key management is used
• a compliance or non-compliance indicator for the check digit of the PAN to be processed if VFPE is specified.

The services return the encrypted or decrypted fields and optionally, the DUKPT PIN key, if the DUKPT key management is selected and the PINKEY rule is specified.

Authentication Parameter Generate (CSNBAPG)

The Authentication Parameter Generate service generates an authentication parameter (AP) and returns it encrypted using the key supplied in the AP_encrypting_key_identifier parameter.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.
Format

The format of CSNBAPG.

```c
CSNBAPG(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    inbound_PIN_encrypting_key_identifier_length,  
    inbound_PIN_encrypting_key_identifier,  
    encrypted_PIN_block,  
    issuer_domestic_code,  
    card_secure_code,  
    PAN_data,  
    AP_encrypting_key_identifier_length,  
    AP_encrypting_key_identifier,  
    AP_value)
```

Parameters

The parameters for CSNBAPG.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

`rule_array_count`

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0, 1, or 2.

`rule_array`

- **Direction:** Input
- **Type:** String array

The keywords that provide control information to the verb. The `rule_array` keywords are described in Table 152.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AP Protection Method</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>ENCRYPT</td>
<td>Specifies that the AP value should be returned encrypted under the <code>AP_encrypting_key_identifier</code> parameter. This is the default.</td>
</tr>
<tr>
<td>CLEAR</td>
<td>Specifies that the AP value should be returned in the clear.</td>
</tr>
<tr>
<td><strong>AP Value Format</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>BCD</td>
<td>Specifies the output format of the AP as packed binary coded decimal. This is the default.</td>
</tr>
</tbody>
</table>

`inbound_PIN_encrypting_key_identifier_length`

- **Direction:** Input
- **Type:** Integer
Length of the `inbound_PIN_encrypting_key_identifier` parameter in bytes. This value must be 64.

**inbound_PIN_encrypting_key_identifier**

- **Direction:** Input
- **Type:** String

An operational key token or the label of the AES or DES key storage record containing a double length IPINENC key that decrypts the PIN block.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**encrypted_PIN_block**

- **Direction:** Input
- **Type:** String

The ISO-0 PIN block encrypted with the `inbound_PIN_encrypting_key_identifier`. The PIN within the PIN block must be a 5-digit value.

**issuer_domestic_code**

- **Direction:** Input
- **Type:** String

A 5 byte alphanumeric character string.

**card_secure_code**

- **Direction:** Input
- **Type:** String

An 8 byte string of digits grouped into two 4 byte sections. The 4 digits in a section cannot all be zero, for example, the value 0000 is invalid.

**PAN_data**

- **Direction:** Input
- **Type:** String

The personal account number (PAN). Must be 12 characters long.

**AP_encrypting_key_identifier_length**

- **Direction:** Input
- **Type:** Integer

The length of the `AP_encrypting_key_identifier` parameter in bytes. This value is 64 when a label is supplied. When the key identifier is a key token, the value is the length of the token. The maximum value is 725. The value may be 0 when the CLEAR `rule_array` keyword is specified.

**AP_encrypting_key_identifier**

- **Direction:** Input
- **Type:** String

An internal key token or the label of the AES or DES key storage record containing a double length DATA key used to encrypt the `AP_value`. If the AP Protection Method was specified as CLEAR keyword in the `rule_array` parameter, this parameter is ignored.
Authentication Parameter Generate (CSNBAPG)

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**AP_value**

- **Direction:** Output
- **Type:** String

An 8 byte character string containing the generated authentication parameter.

**Restrictions**

The restrictions for CSNBAPG.

None.

**Required commands**

The required commands for CSNBAPG.

The Authentication Parameter Generate verb requires the **Authentication Parameter Generate** command (offset X'02B1') to be enabled in the active role.

In addition, rule-array keyword CLEAR requires the **Authentication Parameter Generate - Clear** command (offset X'02B2') to be enabled in the active role.

**Usage notes**

The usage notes for CSNBAPG.

None.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBAPGJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBAPGJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber inbound_PIN_encrypting_key_identifier_length,
    byte[] inbound_PIN_encrypting_key_identifier,
    byte[] encrypted_PIN_block,
    byte[] issuer_domestic_code,
    byte[] card_secure_code,
    byte[] PAN_data,
    hikmNativeNumber AP_encrypting_key_identifier_length,
    byte[] AP_encrypting_key_identifier,
    byte[] AP_value);
```

Clear PIN Encrypt (CSNBCPE)

The Clear PIN Encrypt verb formats a PIN into one of the following PIN block formats and encrypts the results.
Clear PIN Encrypt (CSNBCPE)

You can use this verb to create an encrypted PIN block for transmission. With the RANDOM keyword, you can have the verb generate random PIN numbers.

Note: A clear PIN is a sensitive piece of information. Ensure your application program and system design provide adequate protection for any clear PIN value.

- IBM 3621 format
- IBM 3624 format
- ISO-0 format (same as the ANSI X9.8, VISA-1, and ECI formats)
- ISO-1 format (same as the ECI-4 format)
- ISO-2 format
- ISO-3 format
- IBM 4704 encrypting PINPAD (4704-EPP) format
- VISA 2 format
- VISA 3 format
- VISA 4 format
- ECI2 format
- ECI3 format

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.

Format

The format of CSNBCPE.

```c
CSNBCPE(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    PIN_encrypting_key_identifier,
    rule_array_count,
    rule_array,
    clear_PIN,
    PIN_profile,
    PAN_data,
    sequence_number,
    encrypted_PIN_block)
```

Parameters

The parameter definitions for CSNBCPE.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see "Parameters common to all verbs" on page 22.

**PIN_encrypting_key_identifier**

- **Direction:** Input/Output
- **Type:** String

The 64-byte string containing an internal key token or a key label of an internal key token. The internal key token contains the key that encrypts the PIN block. The control vector in the internal key token must specify an OPINENC key type and have the CPINENC usage bit set to B’1’.

**rule_array_count**
Clear PIN Encrypt (CSNBCPE)

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. Valid values are 0 and 1.

rule_array

Direction: Input
Type: String array

Keywords that provide control information to the verb. The keyword is left-aligned in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage. The rule_array keywords are described in Table 153.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCRYPT</td>
<td>This is the default. Use of this keyword is optional.</td>
</tr>
<tr>
<td>RANDOM</td>
<td>Causes the verb to generate a random PIN value. The length of the PIN is</td>
</tr>
<tr>
<td></td>
<td>based on the value in the clear_PIN variable. Set the value of the clear</td>
</tr>
<tr>
<td></td>
<td>PIN to zero and use as many digits as the desired random PIN. Pad the</td>
</tr>
<tr>
<td></td>
<td>remainder of the clear PIN variable with space characters.</td>
</tr>
</tbody>
</table>

clear_PIN

Direction: Input
Type: String

A 16-character string with the clear PIN. The value in this variable must be left-aligned and padded on the right with space characters.

PIN_profile

Direction: Input
Type: String array

A 24-byte string containing three 8-byte elements with a PIN block format keyword, the format control keyword, NONE, and a pad digit as required by certain formats. See "The PIN profile" on page 497 for additional information.

PAN_data

Direction: Input
Type: String

A 12-byte PAN in character format. The verb uses this parameter if the PIN profile specifies the ISO-0, ISO-3, or VISA-4 keyword for the PIN block format. Otherwise, ensure this parameter is a 12-byte variable in application storage. The information in this variable will be ignored, but the variable must be specified.

Note: When using the ISO-0 or ISO-3 keyword, use the 12 rightmost digits of the PAN data, excluding the check digit. When using the VISA-4 keyword, use the 12 leftmost digits of the PAN data, excluding the check digit.

sequence_number

Direction: Input
Type: Integer
Clear PIN Encrypt (CSNBCPE)

The 4-byte character integer. The verb currently ignores the value in this variable. For future compatibility, the suggested value is 99999.

encrypted_PIN_block

Direction: Output
Type: String

The field that receives the 8-byte encrypted PIN block.

Restrictions

The restrictions for CSNBCPE.

The format control specified in the PIN profile must be NONE.

Required commands

The required commands for CSNBCPE.

This verb requires the Clear PIN Encrypt command (offset X'00AF') to be enabled in the active role.

An enhanced PIN security mode is available for formatting an encrypted PIN-block into IBM 3624 format using the PADDIGIT PIN-extraction method. This mode limits checking of the PIN to decimal digits. No other PIN-block consistency checking will occur. To activate this mode, enable the Enhanced PIN Security command (offset X'0313') in the active role.

Usage notes

Usage notes for CSNBCPE.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBCPEJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

public native void CSNBCPEJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] PIN_encrypting_key_identifier,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] clear_PIN,
    byte[] PIN_profile,
    byte[] PAN_data,
    hikmNativeNumber sequence_number,
    byte[] encrypted_PIN_block);
Clear PIN Generate (CSNBPGN)

Use the Clear PIN Generate verb to generate a clear PIN, a PIN validation value (PVV), or an offset according to an algorithm.

You supply the algorithm or process rule using the rule_array parameter.

- IBM 3624 (IBM-PIN or IBM-PINO)
- VISA PIN validation value (VISA-PVV)
- Interbank PIN (INBK-PIN)

For guidance information about VISA, see their appropriate publications.

**Note:** This verb supports PCI-HSM 2016 compliant-tagged key tokens.

**Format**

The format of CSNBPGN.

```plaintext
CSNBPGN(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    PIN_generating_key_identifier,
    rule_array_count,
    rule_array,
    PIN_length,
    PIN_check_length,
    data_array,
    returned_result)
```

**Parameters**

The parameters for CSNBPGN.

For the definitions of the **return_code**, **reason_code**, **exit_data_length**, and **exit_data** parameters, see [Parameters common to all verbs](#) on page 22.

**PIN_generating_key_identifier**

- **Direction:** Input/Output
- **Type:** String

The 64-byte key label or internal key token that identifies the PIN generation (PINGEN) key.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the **rule_array** variable. This value must be 1.

**rule_array**

- **Direction:** Input
- **Type:** String array

The process rule provides control information to the verb. The keyword is left-aligned in an 8-byte field and padded on the right with blanks.
Clear PIN Generate (CSNBPGN)

**rule_array** keyword is described in **Table 154**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Process Rule (One, required)</strong></td>
<td></td>
</tr>
<tr>
<td>GBP-PIN</td>
<td>The IBM German Bank Pool PIN, which uses the institution PINGEN key to generate an institution PIN (IPIN).</td>
</tr>
<tr>
<td>IBM-PIN</td>
<td>The IBM 3624 PIN, which is an institution-assigned PIN. It does not calculate the PIN offset.</td>
</tr>
<tr>
<td>IBM-PINO</td>
<td>The IBM 3624 PIN offset, which is a customer-selected PIN and calculates the PIN offset (the output).</td>
</tr>
<tr>
<td>INBK-PIN</td>
<td>The Interbank PIN that is generated.</td>
</tr>
<tr>
<td>VISA-PVV</td>
<td>The VISA PIN validation value. Input is the customer PIN.</td>
</tr>
</tbody>
</table>

**PIN_length**

Direction: Input  
Type: Integer

The length of the PIN used for the IBM algorithms only, **IBM-PIN** or **IBM-PINO**. Otherwise, this parameter is ignored. Specify an integer in the range 4 - 16.

**PIN_check_length**

Direction: Input  
Type: Integer

The length of the PIN offset used for the **IBM-PINO** process rule only. Otherwise, this parameter is ignored. Specify an integer from 4 - 16.

**Note:** The PIN check length must be less than or equal to the integer specified in the **PIN_length** parameter.

**data_array**

Direction: Input  
Type: String

Three 16-byte data elements required by the corresponding **rule_array** parameter. The data array consists of three 16-byte fields or elements whose specification depends on the process rule. If a process rule only requires one or two 16-byte fields, the rest of the data array is ignored by the verb. **Table 155** describes the array elements.

<table>
<thead>
<tr>
<th>Array element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear_PIN</td>
<td>Clear user selected PIN of 4 - 12 digits of 0 - 9. Left-aligned and padded with spaces. For <strong>IBM-PINO</strong>, this is the clear customer PIN (CSPIN).</td>
</tr>
</tbody>
</table>
| Decimalization_table| Decimalization table for IBM and GBP only. Sixteen digits of 0 - 9.  
If the **ANSI X9.8 PIN - Use stored decimalization table only** access control point (X'0356') is enabled in the active role, this table must match one of the active decimalization tables in the coprocessors. |

**Table 155. Array elements for the Clear PIN Generate verb**
Table 155. Array elements for the Clear PIN Generate verb (continued)

<table>
<thead>
<tr>
<th>Array element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trans_sec_parm</td>
<td>For VISA only, the leftmost sixteen digits. Eleven digits of the personal account number (PAN). One digit key index. Four digits of customer selected PIN. For Interbank only, sixteen digits. Eleven rightmost digits of the personal account number (PAN). A constant of 6. One digit key selector index. Three digits of PIN validation data.</td>
</tr>
<tr>
<td>Validation_data</td>
<td>Validation data for IBM and IBM German Bank Pool padded to 16 bytes. One to sixteen characters of hexadecimal account data left-aligned and padded on the right with blanks.</td>
</tr>
</tbody>
</table>

Table 156 lists the data array elements required by the process rule (rule_array parameter). The numbers refer to the process rule's position within the array.

Table 156. Array elements for Clear PIN Generate

<table>
<thead>
<tr>
<th>Process Rule</th>
<th>IBM-PIN</th>
<th>IBM-PINO</th>
<th>GBP-PIN</th>
<th>GBP-PINO</th>
<th>VISA-PVV</th>
<th>INBK-PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimalization_table</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Validation_data</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clear_PIN</td>
<td></td>
<td>3</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trans_sec_parm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Note: Generate offset for GBP algorithm is equivalent to IBM offset generation with PIN_check_length of 4 and PIN_length of 6.

direction_result

Direction: Output
Type: String

The 16-byte generated output, left-aligned, and padded on the right with blanks.

Restrictions

The restrictions for CSNBPGN.

None.

Required commands

The required commands for CSNBPGN.

This verb requires the Clear PIN Generate - 3624 command (offset X'00A0') to be enabled in the active role.

Whenever the ANSI X9.8 PIN - Use stored decimalization table only command (offset X'0356') is enabled in the active role, the Decimalization_table element of the data_array value must match one of the PIN decimalization tables that are in the active state on the coprocessor. Use of this command provides improved security and control for PIN decimalization tables.

Usage notes

The usage notes for CSNBPGN.
Clear PIN Generate (CSNBPGN)

If you are using the IBM 3624 PIN and IBM German Bank Pool PIN algorithms, you can supply an unencrypted customer selected PIN to generate a PIN offset.

Related information

Related information about CSNBPGN.

The algorithms are described in detail in Chapter 22, “PIN formats and algorithms,” on page 1005.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBPGNJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBPGNJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] PIN_generating_key_identifier,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber PIN_length,
    hikmNativeNumber PIN_check_length,
    byte[] data_array,
    byte[] returned_result);
```

Clear PIN Generate Alternate (CSNBCPA)

Use the Clear PIN Generate Alternate verb to generate a clear VISA PVV (PIN validation value) from an input encrypted PIN block or to produce a 3624 offset from a customer-selected encrypted PIN.

The PIN block can be encrypted under either an input PIN-encrypting key (IPINENC) or an output PIN-encrypting key (OPINENC).

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.

Format

The format of CSNBCPA.

```java
CSNBCPA(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    PIN_encryption_key_identifier,
    PIN_generation_key_identifier,
    PIN_profile,
    PAN_data,
    encrypted_PIN_block,
    rule_array_count,
    rule_array,
    PIN_check_length,
    data_array,
    returned_PVV)
```
Parameters

The parameters for CSNBCPA.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

**PIN_encryption_key_identifier**

**Direction:** Input/Output  
**Type:** String

A 64-byte string consisting of an internal token that contains an IPINENC or OPINENC key or the label of an IPINENC or OPINENC key that is used to encrypt the PIN block. If you specify a label, it must resolve uniquely to either an IPINENC or OPINENC key.

**PIN_generation_key_identifier**

**Direction:** Input/Output  
**Type:** String

A 64-byte string that consists of an internal token that contains a PIN generation (PINGEN) key or the label of a PINGEN key.

**PIN_profile**

**Direction:** Input  
**Type:** String

The three 8-byte character elements that contain information necessary to extract a PIN from a formatted PIN block. The pad digit is needed to extract the PIN from a 3624 or 3621 PIN block in the Clear PIN Generate Alternate verb. See “The PIN profile” on page 497 for additional information.

**PAN_data**

**Direction:** Input  
**Type:** String

A 12-byte field that contains 12 characters of PAN data. The personal account number recovers the PIN from the PIN block if the PIN profile specifies ISO-0 or VISA-4 block formats. Otherwise it is ignored, but you must specify this parameter.

For ISO-0 or ISO-3, use the rightmost 12 digits of the PAN, excluding the check digit. For VISA-4, use the leftmost 12 digits of the PAN, excluding the check digit.

**encrypted_PIN_block**

**Direction:** Input  
**Type:** String

An 8-byte field that contains the encrypted PIN that is input to the VISA PVV generation algorithm. The verb uses the IPINENC or OPINENC key that is specified in the PIN_encryption_key_identifier parameter to encrypt the block.

**rule_array_count**

**Direction:** Input  
**Type:** Integer

A pointer to an integer variable containing the number of elements in the
Clear PIN Generate Alternate (CSNBCPA)

(rule_array) variable. This value must be 1 or 2. If the default extraction method for a PIN block format is desired, specify the rule_array_count value as 1.

rule_array

Direction: Input
Type: String array

The process rule for the PIN generation algorithm. Specify IBM-PINO or VISA-PVV (the VISA PIN verification value) in an 8-byte field, left-aligned, and padded with blanks. The rule_array points to an array of one or two 8-byte elements. The rule_array keywords are described in Table 157.

Table 157. Keywords for Clear PIN Generate Alternate control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIN calculation method</td>
<td>(One required)</td>
</tr>
<tr>
<td>IBM-PINO</td>
<td>This keyword specifies use of the IBM 3624 PIN Offset calculation method.</td>
</tr>
<tr>
<td>VISA-PVV</td>
<td>This keyword specifies use of the VISA PVV calculation method.</td>
</tr>
</tbody>
</table>

PIN extraction method (One optional) See the text following this table.

If the PIN extraction method is provided, one of the PIN extraction method keywords shown in Table 143 on page 498 can be specified for the given PIN block format. See “PIN block format and PIN extraction method keywords” on page 498 for additional information. If the default extraction method for a PIN block format is desired, specify the rule_array_count value as 1.

The PIN extraction methods operate as follows:

PINBLOCK

Specifies that the verb use one of the following:
- The PIN length, if the PIN block contains a PIN length field
- The PIN delimiter character, if the PIN block contains a PIN delimiter character.

PADDIGIT

Specifies that the verb use the pad value in the PIN profile to identify the end of the PIN.

HEXDIGIT

Specifies that the verb use the first occurrence of a digit in the range from X’A’ to X’F’ as the pad value to determine the PIN length.

PINLENnn

Specifies that the verb use the length specified in the keyword, where nn can range from 04 - 16, to identify the PIN.

The PINLENnn keywords are disabled for this verb by default. If these keywords are used, return code 8 with reason code 33 is returned. To enable them, the Enhanced PIN Security command (bit X’0313’) must be enabled using a TKE.

PADEXIST

Specifies that the verb use the character in the 16th position of the PIN block as the value of the pad value.

PIN_check_length
Clear PIN Generate Alternate (CSNBCCPA)

**Direction:** Input  
**Type:** Integer

The length of the PIN offset used only for the **IBM-PINO** process rule. Otherwise, this parameter is ignored. Specify an integer from 4 - 16.

**Note:** The PIN check length must be less than or equal to the integer specified in the **PIN_length** parameter.

**data_array**

**Direction:** Input  
**Type:** String

Three 16-byte elements, [Table 158](#) describes the format when **IBM-PINO** is specified, [Table 159](#) describes the format when **VISA-PVV** is specified.

**Table 158. Array elements for Clear PIN Generate Alternate, data_array (IBM-PINO)**

<table>
<thead>
<tr>
<th>Array element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimalization_table</td>
<td>This element contains the decimalization table of 16 characters (0 - 9) that are used to convert hexadecimal digits (X'0' - X'F') of the enciphered validation data to the decimal digits (X'0' - X'9'). If the ANSI X9.8 PIN - Use stored decimalization table only access control point (X'0356') is enabled in the active role, this table must match one of the active decimalization tables in the coprocessors.</td>
</tr>
<tr>
<td>validation_data</td>
<td>This element contains 1 - 16 characters of account data. The data must be left-aligned and padded on the right with space characters.</td>
</tr>
<tr>
<td>Reserved-3</td>
<td>This field is ignored, but you must specify it.</td>
</tr>
</tbody>
</table>

**Table 159. Array elements for Clear PIN Generate Alternate, data_array (VISA-PVV)**

<table>
<thead>
<tr>
<th>Array element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trans_sec_parm</td>
<td>For <strong>VISA-PVV</strong> only, the leftmost twelve digits. Eleven digits of the personal account number (PAN). One digit key index. The rest of the field is ignored.</td>
</tr>
<tr>
<td>Reserved-2</td>
<td>This field is ignored, but you must specify it.</td>
</tr>
<tr>
<td>Reserved-3</td>
<td>This field is ignored, but you must specify it.</td>
</tr>
</tbody>
</table>

**returned_PVV**

**Direction:** Output  
**Type:** String

A 16-byte area that contains the 4-byte PVV left-aligned and padded with blanks.

**Restrictions**

The restrictions for CSNBCCPA.

None.

**Required commands**

The required commands for CSNBCCPA.
Clear PIN Generate Alternate (CSNBCPA)

This verb requires the commands shown in the following table to be enabled in the active role based on the keyword specified for the PIN-calculation method:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM-PINO</td>
<td>X'00A4'</td>
<td><strong>Clear PIN Generate Alternate - 3624 Offset</strong></td>
</tr>
<tr>
<td>VISA-PVV</td>
<td>X'00BB'</td>
<td><strong>Clear PIN Generate Alternate - VISA PVV</strong></td>
</tr>
</tbody>
</table>

An enhanced PIN security mode, on the CEX*C is available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3624 PIN-block. To do this, you must enable the Enhanced PIN Security (offset X'0313') access control point in the default role. When activated, this mode limits checking of the PIN to decimal digits and a PIN length minimum of 4 is enforced. No other PIN-block consistency checking will occur.

An enhanced PIN security mode starting with CEX3C is available beginning with Release 4.1.0, to implement restrictions required by the ANSI X9.8 PIN standard. The restrictions are to accept only a PIN_profile variable that contains a PIN-block format of ISO-0 or ISO-3. To enforce these restrictions, you must enable the following access control points in the default role:

- **ANSI X9.8 PIN - Enforce PIN block restrictions** (X'0350')

For more information, see “ANSI X9.8 PIN restrictions” on page 495.

Note: A role with offset X'0350' enabled also affects access control of the Encrypted PIN Translate and the Secure Messaging for PINs verbs.

Whenever the **ANSI X9.8 PIN - Use stored decimalization tables only** command (offset X'0356') is enabled in the active role, the Decimalization_table element of the data_array value must match one of the PIN decimalization tables that are in the active state on the coprocessor. Use of this command provides improved security and control for PIN decimalization tables. The VISA-PVV PIN-calculation method does not have a Decimalization_table element and is therefore not affected by this command.

**Usage notes**

The usage notes for CSNBCPA.

None.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBCPAJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```
public native void CSNBCPAJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] inbound_PIN_encrypting_key_identifier,
    byte[] PIN_generating_key_identifier,
    byte[] input_PIN_profile,
    byte[] PAN_data,
);```

522 Common Cryptographic Architecture Application Programmer's Guide
CVV Generate (CSNBCSG)

Use the CVV Generate verb to generate a VISA Card Verification Value (CVV) or MasterCard Card Verification Code (CVC) as defined for track 2.

This verb generates a CVV that is based on the information that the PAN_data, the expiration_date, and the service_code parameters provide. This verb uses the Key-A and the Key-B keys to cryptographically process this information. Key-A and Key-B can be single-length DATA or MAC keys, or a combined Key-A, Key-B double length DATA or MAC key. If the requested CVV is shorter than 5 characters, the CVV is padded on the right by space characters. The CVV is returned in the 5-byte variable that the CVV_value parameter identifies. When you verify a CVV, compare the result to the value that the CVV_value supplies.

See CVV Key Combine (CSNBCKC) for information on combining two single-length MAC-capable keys into one double-length key.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.

Format

The format of CSNBCSG.

```
CSNBCSG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    PAN_data,
    expiration_date,
    service_code,
    CVV_key_A_Identifier,
    CVV_key_B_Identifier,
    CVV_value)
```

Parameters

The parameters for CSNBCSG.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

rule_array_count

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 0, 1, or 2.

rule_array
Keywords that provide control information to the verb. Each keyword is left-aligned in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage. The rule_array keywords are described in Table 160.

Table 160. Keywords for CVV Generate control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAN data length</td>
<td>Specifies that the length of the PAN data is 13 bytes. <strong>PAN-13 is the default value.</strong></td>
</tr>
<tr>
<td>PAN-13</td>
<td>Specifies that the length of the PAN data is 13 bytes. <strong>PAN-13 is the default value.</strong></td>
</tr>
<tr>
<td>PAN-14</td>
<td>Specifies that the length of the PAN data is 14 bytes.</td>
</tr>
<tr>
<td>PAN-15</td>
<td>Specifies that the length of the PAN data is 15 bytes.</td>
</tr>
<tr>
<td>PAN-16</td>
<td>Specifies that the length of the PAN data is 16 bytes.</td>
</tr>
<tr>
<td>PAN-17</td>
<td>Specifies that the length of the PAN data is 17 bytes.</td>
</tr>
<tr>
<td>PAN-18</td>
<td>Specifies that the length of the PAN data is 18 bytes.</td>
</tr>
<tr>
<td>PAN-19</td>
<td>Specifies that the length of the PAN data is 19 bytes.</td>
</tr>
<tr>
<td>CVV length</td>
<td>Specifies that the CVV is to be computed as one byte, followed by four blanks. <em>CVV-1 is the default value.</em></td>
</tr>
<tr>
<td>CVV-1</td>
<td>Specifies that the CVV is to be computed as one byte, followed by four blanks. <em>CVV-1 is the default value.</em></td>
</tr>
<tr>
<td>CVV-2</td>
<td>Specifies that the CVV is to be computed as two bytes, followed by three blanks.</td>
</tr>
<tr>
<td>CVV-3</td>
<td>Specifies that the CVV is to be computed as three bytes, followed by two blanks.</td>
</tr>
<tr>
<td>CVV-4</td>
<td>Specifies that the CVV is to be computed as four bytes, followed by one blank.</td>
</tr>
<tr>
<td>CVV-5</td>
<td>Specifies that the CVV is to be computed as five bytes.</td>
</tr>
</tbody>
</table>

**PAN data**

The **PAN_data** parameter specifies an address that points to the place in application data storage that contains personal account number (PAN) information in character form. The PAN is the account number as defined for the track-2 magnetic-stripe standards. If the **PAN-nn** keyword is specified in the **rule_array**, where **nn** is a value between 13 and 19, then **nn** number of characters are processed.

If you specify the **PAN-nn** keyword in the **rule_array** where **nn** is less than 16, the server might copy 16 bytes to a work area. Therefore, ensure that the verb can address 16 bytes of storage.

**expiration_date**

The **expiration_date** parameter specifies an address that points to the place in application data storage that contains the card expiration date in numeric character form in a 4-byte field. The application programmer must determine
whether the CVV will be calculated with the date form of YYMM or MMYY.

**service_code**

**Direction:** Input  
**Type:** String

The *service_code* parameter specifies an address that points to the place in application data storage that contains the service code in numeric character form in a 3-byte field. The service code is the number that the track-2 magnetic-stripe standards define. The service code of X'000' is supported.

**CVV_key_A_Identifier**

**Direction:** Input/Output  
**Type:** String

A 64-byte string that is the internal key token containing a single or double-length DATA or MAC key, or the label of a key storage record containing a single or double-length DATA or MAC key.

When this key is a double-length key, *CVV_key_B_Identifier* must be 64 byte of binary zero. When a double-length MAC key is used, the CV bits 0 - 3 must indicate a CVVKEY-A key (B'0010').

A single-length key contains the key-A key that encrypts information in the CVV process. The left half of a double-length key contains the key-A key that encrypts information in the CVV process and the right half contains the key-B key that decrypts information.

**CVV_key_B_Identifier**

**Direction:** Input/Output  
**Type:** String

A pointer to a 64-byte internal key token or a key label of a single-length DATA or MAC key that decrypts information in the CCV process. The internal key token contains the Key-B key that decrypts information in the CVV process.

When *CVV_key_A_Identifier* is a double-length key, this parameter must be 64 byte of binary zero.

**CVV_value**

**Direction:** Output  
**Type:** String

A pointer to the location in application data storage that will be used to store the computed 5-byte character output value.

**Restrictions**

The restrictions for CSNBCSG.

None.

**Required commands**

The required commands for CSNBCSG.

This verb requires the VISA CVV Generate command (offset X'00DF') to be enabled in the active role.
Usage notes

The usage notes for CSNBCSG.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBCSGJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBCSGJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] PAN_data,
    byte[] expiration_date,
    byte[] service_code,
    byte[] CVV_key_A_Identifier,
    byte[] CVV_key_B_Identifier,
    byte[] CVV_value);
```

CVV Key Combine (CSNBCKC)

Use the CVV Key Combine verb to combine two operational DES keys into one operational TDES key.

The verb accepts as input two single-length keys that are suitable for use with the CVV (card-verification value) algorithm. The resulting double-length key meets a more recent industry standard of using TDES to support PIN-based transactions. In addition, the double-length key is in a format that can be wrapped using the Key Export to TR31 verb.

The CVV Generate and CVV Verify verbs use the CVV algorithm to generate and verify card security codes required by Visa (CVV) and MasterCard (CVC). Previously, these verbs only accepted as input two single-length MAC-capable keys. These verbs will additionally accept as input a double-length MAC or MAC-capable DATA key that contains key-A as the left half of the key, and key-B as the right half of the key. The double-length key must be usable with either the CVV Generate verb, the CVV Verify verb, or both.

The CVV Key Combine verb allows combining most pairs of single-length DES keys that formerly functioned as a separate key-A and key-B into one double-length CVVKEY-A key. The CVVKEY-A attribute in the control vector is now changed to mean single-length CVV key containing key-A or double-length CVV key containing key-A and key-B.

To use this verb, specify the following:

- Up to two optional rule-array keywords:
  1. A key wrapping method keyword that specifies whether to use the new enhanced wrapping method, the original wrapping method, or the wrapping method defined as the default according to a configuration setting.
2. A translation control keyword that restricts the translation method to the enhanced method.

- A single-length operational DES key for key-A
  Identify a single-length operational DES key that has a key type of MAC or DATA. The key identifier length must be 64, which is the length of a DES key-token or a key label. This parameter identifies the key-A key used with the CVV algorithm. It is placed in the left half of the double-length output key. When a MAC key is identified, it must have as its subtype extension ANY-MAC (CV bits 0 - 3 = B'0000') or CVVKEY-A (CV bits 0 - 3 = B'0010'). If a DATA key is identified, it must have its MAC generate bit on (CV bit 20), its MAC verify bit on (CV bit 21), or both bits on.

- A single-length operational DES key for key-B
  Identify a single-length operational DES key that has a key type of MAC or DATA. The key identifier length must be 64, which is the length of a DES key-token or a key label. This parameter identifies the key-B key used with the CVV algorithm. It is placed in the right half of the double-length output key. When a MAC key is identified, it must have as its subtype extension ANY-MAC (CV bits 0 - 3 = B'0000') or CVVKEY-B (CV bits 0 - 3 = B'0011'). If a DATA key is identified, it must have its MAC generate bit on (CV bit 20), its MAC verify bit on (CV bit 21), or both bits on.

- An output key identifier
  Identify a null key-token in a 64-byte buffer, or the key label of a DES null key-token. If the input parameter identifies a key label, the output key is placed in DES key-storage. otherwise, the output is returned in the buffer provided.

The following table shows the various output combinations that are returned for the MAC generate and MAC verify attributes. These results are based on the three possible MAC generate and MAC verify control-vector-bit combinations (bits 20 - 21) that the pair of input keys can have.

<table>
<thead>
<tr>
<th>CV bits 20 - 21 of input key</th>
<th>CV bits 20 - 21 of input key key-B, single length</th>
</tr>
</thead>
<tbody>
<tr>
<td>key-A, single length</td>
<td>MAC generate and MAC verify</td>
</tr>
<tr>
<td>MAC generate and MAC verify</td>
<td>MAC generate only</td>
</tr>
<tr>
<td>MAC verify only</td>
<td>MAC verify only</td>
</tr>
<tr>
<td>MAC generate only</td>
<td>MAC generate only</td>
</tr>
<tr>
<td>MAC verify only</td>
<td>MAC verify only</td>
</tr>
</tbody>
</table>

| MAC generate and MAC verify   | MAC generate only double-length key-A         |
| MAC generate only             | MAC generate only double-length key-A         |
| MAC verify only                | Invalid combination, control vector conflict  |
| MAC verify only                | MAC verify only                              |

| MAC generate only             | MAC generate only double-length key-A         |
| MAC verify only                | Invalid combination, control vector conflict  |
| MAC verify only                | MAC verify only                              |
CVV Key Combine (CSNBCKC)

Format
The format of CSNBCKC.

```
CSNBCKC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_A_identifier_length,
    key_A_identifier,
    key_B_identifier_length,
    key_B_identifier,
    output_key_identifier_length
    output_key_identifier)
```

Parameters
The parameters for CSNBCKC.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**
- Direction: Input
- Type: Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. The value must be 0, 1, or 2.

**rule_array**
- Direction: Input
- Type: String array

A pointer to a string variable containing an array of keywords. The keywords are 8 bytes in length and must be left-aligned and padded on the right with space characters. The `rule_array` keywords are:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key wrapping method (one, optional)</td>
<td></td>
</tr>
<tr>
<td>USECONFIG</td>
<td>This is the default. Specifies to wrap the key using the configuration setting for the default wrapping method. The default wrapping method configuration setting may be changed using the TKE. This keyword is ignored for AES keys.</td>
</tr>
<tr>
<td>WRAP-ECB</td>
<td>Specifies to wrap the key using the legacy wrapping method.</td>
</tr>
<tr>
<td>WRAP-ENH</td>
<td>Specifies to wrap the key using the enhanced wrapping method.</td>
</tr>
<tr>
<td>Translation control (optional). This is valid only with wrapping method WRAP-ENH or with USECONFIG when the default wrapping method is WRAP-ENH. This option cannot be used on a key with a control vector valued to binary zeros.</td>
<td></td>
</tr>
<tr>
<td>ENH-ONLY</td>
<td>Specifies to restrict the key from being wrapped with the legacy wrapping method after it has been wrapped with the enhanced wrapping method. Sets bit 56 (ENH-ONLY) of the control vector to B1.</td>
</tr>
</tbody>
</table>

There are restrictions on the available wrapping methods for the output key derived from the wrapping methods employed and control vector restrictions of the input keys. These are detailed in Table 161 on page 529.
Table 161. Key-wrapping matrix for the CVV Key Combine verb

<table>
<thead>
<tr>
<th>key-A or key-B wrapped using WRAP-ENH method</th>
<th>key-A or key-B has ENH-ONLY bit on (CV bit 56 = B'1')</th>
<th>WRAP-ENH (by keyword or by default)</th>
<th>ENH-ONLY keyword</th>
<th>Resulting form of output key or error</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>No to both</td>
<td>No</td>
<td>ECB wrapped</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>No to both</td>
<td>No</td>
<td>Wrap type conflict, 8/2161 (X'871')</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Yes to either</td>
<td>No</td>
<td>WRAP-ENH, CV bit 56 = B'0' (NOT set)</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Yes to either</td>
<td>No</td>
<td>WRAP-ENH, CV bit 56 = B'1' (IS set)</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Yes to either</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Yes to either</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes to either</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes to either</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>No to both</td>
<td>Yes</td>
<td>CV bit 56 conflict, 8/2111 (X'83F')</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>No to both</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>No to both</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>No to both</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

**key_A_identifier_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes in the `key_A_identifier` variable. This value must be 64.

**key_A_identifier**

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing either the operational single-length DES key-token of the key-A key, or the label of such a key token. This key must be a MAC key or a DATA key that can perform MAC operations.

**key_B_identifier_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes in the `key_B_identifier` variable. This value must be 64.

**key_B_identifier**

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing either the operational single-length DES key-token of the key-B key, or the label of such a key token. This key must be a MAC key or a DATA key that can perform MAC operations.

**output_key_identifier_length**
CVV Key Combine (CSNBCKC)

A pointer to an integer variable containing the number of bytes in the `output_key_identifier` variable. This value must be 64.

`output_key_identifier`

A pointer to a string variable containing a NULL key token, or the key label of a null DES key-token.

**Restrictions**

The restrictions for CSNBCKC.

Input key-A and input key-B cannot have different export control bits (CV bit 17 and 57); these bits must match. Use the Prohibit Export verb to change XPORT-OK to NO-XPORT (CV bit 17), or the Restrict Key Attribute verb to change T31XPTOK to NOT31XPT (CV bit 57). These two bits are propagated to the output key-token.

**Required commands**

The required commands for CSNBCKC.

The CVV Key Combine verb requires the **CVV Key Combine** command (offset X'0155') to be enabled in the active role.

In addition, these commands are required to be enabled in the active role, depending on the key-wrapping method keyword:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRAP-ECB</td>
<td>X'0156'</td>
<td>CVV Key Combine - Allow wrapping override keywords</td>
</tr>
<tr>
<td>WRAP-ENH</td>
<td>X'0156'</td>
<td>CVV Key Combine - Allow wrapping override keywords</td>
</tr>
</tbody>
</table>

One additional restriction is related to combining the key-A and key-B pair of keys when there are mixed types. To permit the combination of mixed key types into a single set of types (ANY-MAC, CVVKEY-A, CVVKEY-B, and DATA), enable the **CVV Key Combine - Permit mixed key types** command (offset X'0157') in the active role. See Table 162 for when this command is required:

Table 162. Required commands for the CVV Key Combine verb, mixed key types

<table>
<thead>
<tr>
<th>Input key-A</th>
<th>Input key-B</th>
<th>ANY-MAC</th>
<th>CVVKEY-A</th>
<th>CVVKEY-B</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANY-MAC</td>
<td></td>
<td>Always returns double-length CVVKEY-A key</td>
<td>Invalid combination, control vector conflict</td>
<td>Only returns double-length CVVKEY-A key if X'0157' enabled</td>
<td>Only returns double-length CVVKEY-A key if X'0157' enabled</td>
</tr>
<tr>
<td>CVVKEY-A</td>
<td></td>
<td>Only returns double-length CVVKEY-A key if X'0157' enabled, else access error</td>
<td>Invalid combination, control vector conflict</td>
<td>Always returns double-length CVVKEY-A key</td>
<td>Only returns double-length CVVKEY-A key if X'0157' enabled</td>
</tr>
<tr>
<td>CVVKEY-B</td>
<td></td>
<td>Invalid combination</td>
<td>Invalid combination</td>
<td>Invalid combination</td>
<td>Invalid combination</td>
</tr>
</tbody>
</table>
Table 162. Required commands for the CVV Key Combine verb, mixed key types (continued)

<table>
<thead>
<tr>
<th>Input key-A</th>
<th>ANY-MAC</th>
<th>CVVKEY-A</th>
<th>CVVKEY-B</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>Only returns double-length CVVKEY-A key if X’0157’ enabled, else access error</td>
<td>Invalid combination, control vector conflict</td>
<td>Only returns double-length CVVKEY-A key if X’0157’ enabled, else access error</td>
<td>Always returns double-length CVVKEY-A key</td>
</tr>
</tbody>
</table>

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBCKCJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBCKCJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber key_A_identifier_length,
    byte[] key_A_identifier,
    hikmNativeNumber key_B_identifier_length,
    byte[] key_B_identifier,
    hikmNativeNumber output_key_identifier_length,
    byte[] output_key_identifier);
```

CVV Verify (CSNBCSV)

Use the CVV Verify verb to verify a VISA Card Verification Value (CVV) or MasterCard Card Verification Code (CVC) as defined for track 2.

This verb generates a CVV based on the information the PAN_data, the expiration_date, and the service_code parameters provide. This verb uses the Key-A and the Key-B keys to cryptographically process this information. If the requested CVV is shorter than 5 characters, the CVV is padded on the right by space characters. The generated CVV is then compared to the value that the CVV_value supplies for verification.

See CVV Key Combine (CSNBCKC) for information on combining two single-length MAC-capable keys into one double-length key.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.
CVV Verify (CSNBCSV)

Format

The format of CSNBCSV.

```c
CSNBCSV(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    PAN_data,
    expiration_date,
    service_code,
    CVV_key_A_Identifier,
    CVV_key_B_Identifier,
    CVV_value)
```

Parameters

The parameters for CSNBCSV.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0, 1, or 2.

**rule_array**

- **Direction:** Input
- **Type:** String array

Keywords that provide control information to the verb. Each keyword is left-aligned in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage. The `rule_array` keywords are described in Table 163.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PAN data length</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>PAN-13</td>
<td>Specifies that the length of the PAN data is 13 bytes. <strong>PAN-13 is the default value.</strong></td>
</tr>
<tr>
<td>PAN-14</td>
<td>Specifies that the length of the PAN data is 14 bytes.</td>
</tr>
<tr>
<td>PAN-15</td>
<td>Specifies that the length of the PAN data is 15 bytes.</td>
</tr>
<tr>
<td>PAN-16</td>
<td>Specifies that the length of the PAN data is 16 bytes.</td>
</tr>
<tr>
<td>PAN-17</td>
<td>Specifies that the length of the PAN data is 17 bytes.</td>
</tr>
<tr>
<td>PAN-18</td>
<td>Specifies that the length of the PAN data is 18 bytes.</td>
</tr>
<tr>
<td>PAN-19</td>
<td>Specifies that the length of the PAN data is 19 bytes.</td>
</tr>
<tr>
<td><strong>CVV length</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>CVV-1</td>
<td>Specifies that the CVV is to be computed as one byte, followed by four blanks. <strong>CVV-1 is the default value.</strong></td>
</tr>
</tbody>
</table>
Table 163. Keywords for CVV Verify control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVV-2</td>
<td>Specifies that the CVV is to be computed as two bytes, followed by three blanks.</td>
</tr>
<tr>
<td>CVV-3</td>
<td>Specifies that the CVV is to be computed as three bytes, followed by two blanks.</td>
</tr>
<tr>
<td>CVV-4</td>
<td>Specifies that the CVV is to be computed as four bytes, followed by one blank.</td>
</tr>
<tr>
<td>CVV-5</td>
<td>Specifies that the CVV is to be computed as five bytes.</td>
</tr>
</tbody>
</table>

**PAN_data**

- **Direction:** Input
- **Type:** String

The `PAN_data` parameter specifies an address that points to the place in application data storage that contains personal account number (PAN) information in character form. The PAN is the account number as defined for the track-2 magnetic-stripe standards. If the `PAN-nn` keyword is specified in the `rule_array`, where `nn` is a value between 13 and 19, then `nn` number of characters are processed.

If you specify the `PAN-nn` keyword in the `rule_array` where `nn` is less than 16, the server might copy 16 bytes to a work area. Therefore, ensure that the verb can address 16 bytes of storage.

**expiration_date**

- **Direction:** Input
- **Type:** String

The `expiration_date` parameter specifies an address that points to the place in application data storage that contains the card expiration date in numeric character form in a 4-byte field. The application programmer must determine whether the CVV will be calculated with the date form of YYMM or MMYY.

**service_code**

- **Direction:** Input
- **Type:** String

The `service_code` parameter specifies an address that points to the place in application data storage that contains the service code in numeric character form in a 3-byte field. The service code is the number that the track-2 magnetic-stripe standards define. The service code of `X’000’` is supported.

**CVV_key_A_Identifier**

- **Direction:** Input/Output
- **Type:** String

A 64-byte string that is the internal key token containing a single or double-length DATA or MAC key, or the label of a key storage record containing a single or double-length DATA or MAC key.

When this key is a double-length key, `CVV_key_B_Identifier` must be 64 byte of binary zero. When a double-length MAC key is used, the CV bits 0 - 3 must indicate a CVVKEY-A key (B’0010’).
**CVV Verify (CSNBCSV)**

A single-length key contains the key-A key that encrypts information in the CVV process. The left half of a double-length key contains the key-A key that encrypts information in the CVV process and the right half contains the key-B key that decrypts information.

**CVV_key_B_Identifier**

*Direction:* Input/Output  
*Type:* String

A pointer to a 64-byte internal key token or a key label of a single-length DATA or MAC key that decrypts information in the CCV process. The internal key token contains the Key-B key that decrypts information in the CVV process.

When `CVV_key_A_Identifier` is a double-length key, this parameter must be 64 byte of binary zero.

**CVV_value**

*Direction:* Input  
*Type:* String

The `CVV_value` parameter specifies an address that contains the CVV value which will be compared to the computed CVV value. This is a 5-byte field.

**Restrictions**

The restrictions for CSNBCSV.

None.

**Required commands**

The required commands for CSNBCSV.

This verb requires the VISA CVV Verify command (offset X'00E0') to be enabled in the active role.

**Usage notes**

The usage notes for CSNBCSV.

None.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBCSVJ.

See "Building Java applications using the CCA JNI" on page 28.

**Format**

```java
public native void CSNBCSVJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] PAN_data,
    byte[] expiration_date,
)```

534  Common Cryptographic Architecture Application Programmer’s Guide
Encrypted PIN Generate (CSNBEPG)

The Encrypted PIN Generate verb formats a PIN and encrypts the PIN block.

To generate the PIN, the verb uses one of the following PIN calculation methods:
- IBM 3624 PIN (IBM-PIN)
- IBM German Bank Pool Institution PIN (GBP-PIN)
- Interbank PIN (NL-PIN-1)

To format the PIN, the verb uses one of the following PIN block formats:
- IBM 3621 format
- IBM 3624 format
- ISO-0 format (same as the ANSI X9.8, VISA-1, and ECI-1 formats)
- ISO-1 format (same as the ECI-4 format)
- ISO-2 format
- ISO-3 format
- IBM 4704 encrypting PINPAD (4704-EPP) format
- VISA 2 format
- VISA 3 format
- VISA 4 format
- ECI-2 format
- ECI-3 format

**Note:** This verb supports PCI-HSM 2016 compliant-tagged key tokens.

**Format**

The format of CSNBEPG.

```c
CSNBEPG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    PIN_generating_key_identifier,
    outbound_PIN_encrypting_key_identifier
    rule_array_count,
    rule_array,
    PIN_length,
    data_array,
    PIN_profile,
    PAN_data,
    sequence_number
    encrypted_PIN_block)
```

**Parameters**

The parameters for CSNBEPG.
Encrypted PIN Generate (CSNBEPG)

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

**PIN_generating_key_identifier**

Direction:    Input/Output  
Type:        String  

The 64-byte internal key token or a key label of an internal key token in the DES key storage file. The internal key token contains the PIN-generating key. The control vector must specify the PINGEN key type and have the EPINGEN usage bit set to B’1’.

**outbound_PIN_encrypting_key_identifier**

Direction:    Input  
Type:        String  

A 64-byte internal key token or a key label of an internal key token in the DES key storage file. The internal key token contains the key to be used to encrypt the formatted PIN and must contain a control vector that specifies the OPINENC key type and has the EPINGEN usage bit set to B’1’.

**rule_array_count**

Direction:    Input  
Type:        Integer  

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 1.

**rule_array**

Direction:    Input  
Type:        String array  

Keywords that provide control information to the verb. Each keyword is left-aligned in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage. The rule_array keywords are described in Table 164.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GBP-PIN</td>
<td>This keyword specifies the IBM German Bank Pool Institution PIN calculation method is to be used to generate a PIN.</td>
</tr>
<tr>
<td>IBM-PIN</td>
<td>This keyword specifies the IBM 3624 PIN calculation method is to be used to generate a PIN.</td>
</tr>
<tr>
<td>INBK-PIN</td>
<td>This keyword specifies the Interbank PIN calculation method is to be used to generate a PIN.</td>
</tr>
</tbody>
</table>

**PIN_length**

Direction:    Input  
Type:        String  

An integer defining the PIN length for those PIN calculation methods with variable length PINs. Otherwise, the variable should be set to zero.

**data_array**

Direction:    Input  

Table 164. Keywords for Encrypted PIN Generate control information

---

536  Common Cryptographic Architecture Application Programmer's Guide
Type: Integer

Three 16-byte character strings, which are equivalent to a single 48-byte string. The values in the data array depend on the keyword for the PIN calculation method. Each element is not always used, but you must always declare a complete data array. The numeric characters in each 16-byte string must be from 1 - 16 bytes in length, uppercase, left-aligned, and padded on the right with space characters. Table 165 describes the array elements.

Table 165. Array elements for Encrypted PIN Generate data_array parameter

<table>
<thead>
<tr>
<th>Array element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimalization_table</td>
<td>Decimalization table for IBM and GBP only. Sixteen characters that are used to map the hexadecimal digits (X'0' - X'F') of the encrypted validation data to decimal digits (X'0' - X'9'). If the ANSI X9.8 PIN - Use stored decimalization tables only command (offset X'0356') access control point is enabled in the active role, this table must match one of the active decimalization tables in the coprocessors.</td>
</tr>
<tr>
<td>Trans_sec_parm</td>
<td>For Interbank only, sixteen digits. Eleven rightmost digits of the personal account number (PAN). A constant of 6. One digit key selector index. Three digits of PIN validation data.</td>
</tr>
<tr>
<td>Validation_data</td>
<td>Validation data for IBM and IBM German Bank Pool padded to 16 bytes. 1 - 16 characters of hexadecimal account data left-aligned and padded on the right with blanks.</td>
</tr>
</tbody>
</table>

Table 166 lists the data array elements required by the process rule (rule_array parameter). The numbers refer to the process rule's position within the array.

Table 166. Keywords for Encrypted PIN Generate control information

<table>
<thead>
<tr>
<th>Process rule</th>
<th>IBM-PIN</th>
<th>GBP-PIN</th>
<th>INBK-PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimalization_table</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Validation_data</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Trans_sec_parm</td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

PIN_profile

Direction: Input
Type: String array

A 24-byte string containing the PIN profile including the PIN block format. See “The PIN profile” on page 497 for additional information.

PAN_data

Direction: Input
Type: String

A 12-byte string that contains 12 digits of Personal Account Number (PAN) data. The verb uses this parameter if the PIN profile specifies the ISO-0, ISO-3, or VISA-4 keyword for the PIN block format. Otherwise, ensure that this parameter is a 4-byte variable in application storage. The information in this variable will be ignored, but the variable must be specified.

Note: When using the ISO-0 or ISO-3 keywords, use the 12 rightmost digits of the PAN data, excluding the check digit. When using the VISA-4 keyword, use the 12 leftmost digits of the PAN data, excluding the check digit.
Direction: Input
Type: Integer

The 4-byte string that contains the sequence number used by certain PIN block formats. The verb uses this parameter if the PIN profile specifies the 3621 or 4704-EPP keyword for the PIN block format. Otherwise, ensure this parameter is a 4-byte variable in application data storage. The information in the variable will be ignored, but the variable must be declared. To enter a sequence number, do the following:

- Enter 99999 to use a random sequence number that the service generates.
- For the 3621 PIN block format, enter a value in the range from 0 - 65,535.
- For the 4704-EPP PIN block format, enter a value in the range from 0 - 255.

encrypted_PIN_block

Direction: Output
Type: String

The field where the verb returns the 8-byte encrypted PIN.

Restrictions

The restrictions for CSNBEPG.

The format control specified in the PIN profile must be NONE.

Required commands

The required commands for CSNBEPG.

This verb requires the commands, as shown in the following table, to be enabled in the active role based on the keyword specified for the PIN-calculation methods.

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM-PIN</td>
<td>X'00B0'</td>
<td>Encrypted PIN Generate - 3624</td>
</tr>
<tr>
<td>GBP-PIN</td>
<td>X'00B1'</td>
<td>Encrypted PIN Generate - GBP</td>
</tr>
<tr>
<td>INBK-PIN</td>
<td>X'00B2'</td>
<td>Encrypted PIN Generate - Interbank</td>
</tr>
</tbody>
</table>

An enhanced PIN security mode is available for formatting an encrypted PIN block into IBM 3624 format using the PADDIGIT PIN-extraction method. This mode limits checking of the PIN to decimal digits, and a minimum PIN length of 4 is enforced; no other PIN-block consistency checking will occur. To activate this mode, enable the Enhanced PIN Security command (offset X'0313') in the active role.

Whenever the ANSI X9.8 PIN - Use stored decimalization tables only command (offset X'0356') command is enabled in the active role, the Decimalization_table element of the data_array value must match one of the PIN decimalization tables that are in the active state on the coprocessor. Use of this command provides improved security and control for PIN decimalization tables. The INBK-PIN PIN-calculation method does not have a Decimalization_table element and is therefore not affected by this command.
Usage notes

The usage notes for CSNBEPG.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBEPGJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBEPGJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] PIN_generating_key_identifier,
    byte[] outbound_PIN_encrypting_key_identifier,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber PIN_length,
    byte[] data_array,
    byte[] PIN_profile,
    byte[] PAN_data,
    hikmNativeNumber sequence_number,
    byte[] encrypted_PIN_block);
```

Encrypted PIN Translate (CSNBPTCR)

Use the Encrypted PIN Translate verb to re-encipher a PIN block from one PIN-encrypting key to another and, optionally, to change the PIN block format, such as the pad digit or sequence number.

The unique-key-per-transaction key derivation for single and double-length keys is available for the Encrypted PIN Translate verb. This support is available for the input_PIN_encrypting_key_identifier and the output_PIN_encrypting_key_identifier parameters for both REFORMAT and TRANSLAT process rules. The rule_array keyword determines which PIN keys are derived keys.

The Encrypted PIN Translate verb can be used for unique-key-per-transaction key derivation.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.
Encrypted PIN Translate (CSNBPTR)

Format

The format of CSNBPTR.

```
CSNBPTR(
   return_code,
   reason_code,
   exit_data_length,
   exit_data,
   input_PIN_encrypting_key_identifier,
   output_PIN_encrypting_key_identifier,
   input_PIN_profile,
   input_PAN_data,
   input_PIN_block,
   rule_array_count,
   rule_array,
   output_PIN_profile,
   output_PAN_data,
   sequence_number,
   PIN_block_out)
```

Parameters

The parameters for CSNBPTR.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

**input_PIN_encrypting_key_identifier**

- **Direction:** Input/Output
- **Type:** String

The input PIN-encrypting key (IPINENC) for the PIN_block_in parameter specified as a 64-byte internal key token or a key label. If keyword UKPTOPIN, UKPTBOTH, DUKPT-IP, or DUKPT-BH is specified in the rule_array parameter, the `input_PIN_encrypting_key_identifier` must specify a key token or key label of a KEYGENKY with the UKPT usage bit enabled.

**output_PIN_encrypting_key_identifier**

- **Direction:** Input/Output
- **Type:** String

The output PIN-encrypting key (OPINENC) for the output_PIN_block parameter specified as a 64-byte internal key token or a key label. If keyword UKPTOPIN, UKPTBOTH, DUKPT-IP, or DUKPT-BH is specified in the rule_array parameter, the `output_PIN_encrypting_key_identifier` must specify a key token or key label of a KEYGENKY with the UKPT usage bit enabled.

**input_PIN_profile**

- **Direction:** Input
- **Type:** String

The three 8-byte character elements that contain information necessary to either create a formatted PIN block or extract a PIN from a formatted PIN block. A particular PIN profile can be either an input PIN profile or an output PIN profile depending on whether the PIN block is being enciphered or deciphered by the verb. See "The PIN profile" on page 497 for additional information.

If you choose the TRANSLAT processing rule or the REFORMAT processing rule in the rule_array parameter, the input PIN profile and output PIN profile
can have different PIN block formats. If you specify UKPTIPIN with DUKPT-IP or UKPTBOTH with DUKPT-BH in the rule_array parameter, the input_PIN_profile is extended to a 48-byte field and must contain the current key serial number. See “The PIN profile” on page 497 for additional information.

The pad digit is needed to extract the PIN from a 3624 or 3621 PIN block in the Encrypted PIN Translate verb with a process rule (rule_array parameter) of REFORMAT. If the process rule is TRANSLAT, the pad digit is ignored.

The PINLENnn keywords are disabled for this verb by default. If these keywords are used, return code 8 with reason code 33 is returned. To enable them, the PTR Enhanced PIN Security access control point (bit X’0313’) must be enabled using a TKE.

**input_PAN_data**
- **Direction:** Input
- **Type:** String

The personal account number (PAN) if the process rule (rule_array parameter) is REFORMAT and the input PIN format is ISO-0, ISO-3, or VISA-4 only. Otherwise, this parameter is ignored. Specify 12 digits of account data in character format.

For ISO-0 or ISO-3, use the rightmost 12 digits of the PAN, excluding the check digit.

For VISA-4, use the leftmost 12 digits of the PAN, excluding the check digit.

**input_PIN_block**
- **Direction:** Input
- **Type:** String

The 8-byte enciphered PIN block that contains the PIN to be translated.

**rule_array_count**
- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 1, 2, or 3.

**rule_array**
- **Direction:** Input
- **Type:** String array

The process rule for the verb is described in Table 167

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process rule (One, required)</td>
<td></td>
</tr>
<tr>
<td>REFORMAT</td>
<td>Changes the PIN format, the contents of the PIN block, and the PIN-encrypting key.</td>
</tr>
<tr>
<td>TRANSLAT</td>
<td>Changes the PIN-encrypting key only. It does not change the PIN format and the contents of the PIN block.</td>
</tr>
</tbody>
</table>

Table 167. Keywords for Encrypted PIN Translate control information
Table 167. Keywords for Encrypted PIN Translate control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIN block format and PIN extraction method (Optional)</td>
<td>See “PIN block format and PIN extraction method keywords” on page 498 for additional information and a list of PIN block formats and PIN extraction method keywords. Note: If a PIN extraction method is not specified, the first one listed in Table 143 on page 498 for the PIN block format will be the default.</td>
</tr>
<tr>
<td><strong>DUKPT keywords - Single length key derivation (One, optional)</strong></td>
<td></td>
</tr>
<tr>
<td>UKTIPIN</td>
<td>The input_PIN_encrypting_key_identifier is derived as a single length key. The input_PIN_encrypting_key_identifier must be a KEYGENKY key with the UKPT usage bit enabled. The output_PIN_profile must be 48 bytes and contain the key serial number.</td>
</tr>
<tr>
<td>UKTOPIN</td>
<td>The output_PIN_encrypting_key_identifier is derived as a single length key. The output_PIN_encrypting_key_identifier must be a KEYGENKY key with the UKPT usage bit enabled. The output_PIN_profile must be 48 bytes and contain the key serial number.</td>
</tr>
<tr>
<td>UKPTBOTH</td>
<td>Both the input_PIN_encrypting_key_identifier and the output_PIN_encrypting_key_identifier are derived as a single length key. Both the input_PIN_encrypting_key_identifier and the output_PIN_encrypting_key_identifier must be KEYGENKY keys with the UKPT usage bit enabled. Both the input_PIN_profile and the output_PIN_profile must be 48 bytes and contain the respective key serial number.</td>
</tr>
<tr>
<td><strong>DUKPT keywords - double length key derivation (One, optional)</strong></td>
<td></td>
</tr>
<tr>
<td>DUKPT-IP</td>
<td>The input_PIN_encrypting_key_identifier is derived as a double length key. The input_PIN_encrypting_key_identifier must be a KEYGENKY key with the UKPT usage bit enabled. The input_PIN_profile must be 48 bytes and contain the key serial number.</td>
</tr>
<tr>
<td>DUKPT-OP</td>
<td>The output_PIN_encrypting_key_identifier is derived as a double length key. The output_PIN_encrypting_key_identifier must be a KEYGENKY key with the UKPT usage bit enabled. The output_PIN_profile must be 48 bytes and contain the key serial number.</td>
</tr>
<tr>
<td>DUKPT-BH</td>
<td>Both the input_PIN_encrypting_key_identifier and the output_PIN_encrypting_key_identifier are derived as a double length key. Both the input_PIN_encrypting_key_identifier and the output_PIN_encrypting_key_identifier must be KEYGENKY keys with the UKPT usage bit enabled. Both the input_PIN_profile and the output_PIN_profile must be 48 bytes and contain the respective key serial number.</td>
</tr>
</tbody>
</table>

output_PIN_profile

- **Direction:** Input
- **Type:** String

The three 8-byte character elements that contain information necessary to either create a formatted PIN block or extract a PIN from a formatted PIN block. A particular PIN profile can be either an input PIN profile or an output PIN profile, depending on whether the PIN block is being enciphered or deciphered by the verb.

- If you choose the TRANSLATE processing rule in the rule_array parameter, the input_PIN_profile and the output_PIN_profile must specify the same PIN block format.
- If you choose the REFORMAT processing rule in the rule_array parameter, the input PIN profile and output PIN profile can have different PIN block formats.
- If you specify UKTOPIN or UKPTBOTH in the rule_array parameter, the output_PIN_profile is extended to a 48-byte field and must contain the current key serial number. See “The PIN profile” on page 497 for additional information.
If you specify DUKPT-OP or DUKPT-BH in the rule_array parameter, the output_PIN_profile is extended to a 48-byte field and must contain the current key serial number. See “The PIN profile” on page 497 for additional information.

**output_PAN_data**

- **Direction:** Input
- **Type:** String

The personal account number (PAN) if the process rule (rule_array parameter) is RÉFORMAT and the output PIN format is ISO-0, ISO-3, or VISA-4 only. Otherwise, this parameter is ignored. Specify 12 digits of account data in character format.

- For ISO-0 or ISO-3, use the rightmost 12 digits of the PAN, excluding the check digit.
- For VISA-4, use the leftmost 12 digits of the PAN, excluding the check digit.

**sequence_number**

- **Direction:** Output
- **Type:** Integer

The sequence number if the process rule (rule_array parameter) is RÉFORMAT and the output PIN block format is 3621 or 4704-EPP only. Specify the integer value 99999. Otherwise, this parameter is ignored.

**output_PIN_block**

- **Direction:** Input
- **Type:** String

The 8-byte output PIN block that is re-enciphered.

**Restrictions**

The restrictions for CSNBPTR.

None.

**Required commands**

The required commands for CSNBPTR.

This verb requires the commands, as shown in the following table, to be enabled in the active role based on the keyword specified for the PIN-calculation methods.

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Input profile format control keyword</th>
<th>Output profile format control keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSLAT</td>
<td>NONE</td>
<td>NONE</td>
<td>X'00B3'</td>
<td>Encrypted PIN Translate - Translate</td>
</tr>
<tr>
<td>REFORMAT</td>
<td>NONE</td>
<td>NONE</td>
<td>X'00B7'</td>
<td>Encrypted PIN Translate - Reformat</td>
</tr>
</tbody>
</table>

This verb also requires the UKPT - PIN Verify, PIN Translate command (offset X'00E1') to be enabled if you employ UKPT processing.

**Note:** A role with offset X'00E1' enabled can also use the Encrypted PIN Verify verb with UKPT processing.
An enhanced PIN security mode is available for extracting PINs from an IBM 3624 encrypted PIN-block and formatting an encrypted PIN block into IBM 3624 format using the PADDIGIT PIN-extraction method. This mode limits checking of the PIN to decimal digits, and a minimum PIN length of 4 is enforced; no other PIN-block consistency checking will occur. To activate this mode, enable the Enhanced PIN Security command (offset X'0313') in the active role.

The verb returns an error indicating that the PAD digit is not valid if all of these conditions are met:

1. The Enhanced PIN Security command is enabled in the active role.
2. The output PIN profile specifies 3624 as the PIN-block format.
3. The output PIN profile specifies a decimal digit (0 - 9) as the PAD digit.

Beginning with Release 4.1.0, three new commands are added (offsets X'0350', X'0351', and X'0352'). The list hereafter describes how these three commands affect the PIN processing.

1. Enable the ANSI X9.8 PIN - Enforce PIN block restrictions command (offset X'0350') in the active role to apply additional restrictions to PIN processing implemented in CCA 4.1.0, as follows:
   - Do not translate or reformat a non-ISO PIN block into an ISO PIN block. Specifically, do not allow an IBM 3624 PIN-block format in the output_PIN_profile variable when the PIN-block format in the input_PIN_profile variable is not IBM 3624.
   - Constrain use of ISO-2 PIN blocks to offline PIN verification and PIN change operations in integrated circuit card environments only. Specifically, do not allow ISO-2 input or output PIN blocks.
   - Do not translate or reformat a PIN-block format that includes a PAN into a PIN-block format that does not include a PAN. Specifically, do not allow an ISO-1 PIN-block format in the output_PIN_profile variable when the PIN-block format in the input_PIN_profile variable is ISO-0 or ISO-3.
   - Do not allow a change of PAN data. Specifically, when performing translations between PIN block formats that both include PAN data, do not allow the input_PAN_data and output_PAN_data variables to be different from the PAN data enciphered in the input PIN block.

   Note: A role with offset X'0350' enabled also affects access control of the Clear PIN Generate Alternate and the Secure Messaging for PINs verbs.

2. Enable the ANSI X9.8 PIN - Allow modification of PAN command (offset X'0351') in the active role to override the restriction to not allow a change of PAN data. This override is applicable only when either the ANSI X9.8 PIN - Enforce PIN block restrictions command (offset X'0350') or the ANSI X9.8 PIN - Allow only ANSI PIN blocks command (offset X'0352') or both are enabled in the active role. This override is to support account number changes in issuing environments. Offset X'0351' has no effect if neither offset X'0350' nor offset X'0352' is enabled in the active role.

   Note: A role with offset X'0351' enabled also affects access control of the Secure Messaging for PINs verbs.

3. Enable the ANSI X9.8 PIN - Allow only ANSI PIN blocks command (offset X'0352') in the active role to apply a more restrictive variation of the ANSI X9.8 PIN - Enforce PIN block restrictions command (offset X'0350'). In addition to the previously described restrictions of offset X'0350', this command also restricts the input_PIN_profile and the output_PIN_profile to contain only ISO-0,
ISO-1, and ISO-3 PIN block formats. Specifically, the IBM 3624 PIN-block format is not allowed with this command. Offset X'0352' overrides offset X'0350'.

**Note:** A role with offset X'0352' enabled also affects access control of the Secure Messaging for PINs verbs.

For more information, see “ANSI X9.8 PIN restrictions” on page 495.

### Usage notes

The usage notes for CSNBPTR.

Some PIN block formats are known by several names. The following table shows the additional names.

<table>
<thead>
<tr>
<th>PIN format</th>
<th>Additional name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO-0</td>
<td>ANSI X9.8, VISA format 1, ECI format 1</td>
</tr>
<tr>
<td>ISO-1</td>
<td>ECI format 4</td>
</tr>
</tbody>
</table>

### JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBPTRJ.

See “Building Java applications using the CCA JNI” on page 28.

### Format

```java
public native void CSNBPTRJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] input_PIN_encrypting_key_identifier,
    byte[] output_PIN_encrypting_key_identifier,
    byte[] input_PIN_profile,
    byte[] input_PAN_data,
    byte[] input_PIN_block,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] output_PIN_profile,
    byte[] output_PAN_data,
    hikmNativeNumber sequence_number,
    byte[] output_PIN_block);
```

### Encrypted PIN Translate Enhanced (CSNBPTRE)

The Encrypted PIN Translate Enhanced verb reformats a PIN into a different PIN-block format using an enciphered PAN field. You can use this verb in an interchange-network application, or to change the PIN block to conform to the format and encryption key used in a PIN-verification database.

The CSNBPTRE verb supports Visa Data Secure Platform (VDSP, formerly known as Visa Merchant Data Secure (VMDS)) processing. With this verb you can also use derived unique key per transaction (DUKPT) PIN-block encryption (ANS X9.24) for both input and output PIN blocks. The verb supports translation of PINs whose PAN information has been enciphered using the VDSP standard and Visa Format Preserving Encryption (VFPE) methods.
PIN blocks are sometimes formatted using the PAN information. For the CSNBPTRE verb, either the input PIN block profile or the output PIN block profile must specify a PIN block format that incorporates a PAN. The PIN block formats which incorporate a PAN are ISO-0, ISO-3, and Visa Format 4. VDSP enciphered PAN data can be enciphered using DUKPT key management or static TDES key management. The enciphered PAN could be enciphered with the CBC mode or the VFPE mode. VDSP requires that the same key management scheme and type of keys are used for both the PIN and PAN. For VDSP, the following pairings are supported:

<table>
<thead>
<tr>
<th>Function</th>
<th>Source</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Key management</td>
<td>VDSP option</td>
</tr>
<tr>
<td>Translation</td>
<td>DUKPT</td>
<td>Standard CBC</td>
</tr>
<tr>
<td>Static TDES non-DUKPT</td>
<td>Standard CBC</td>
<td>VFPE</td>
</tr>
</tbody>
</table>

**Terminology:** The VDSP specification speaks of two key management methods: **DUKPT** (derived unique key per transaction) and **Zone Encryption Keys**. The process for deriving these keys is documented in ANSI X9.24 Part 1. **Zone Encryption Keys** are called **static keys** in CCA. Static keys are presented for use and are not derived during verb processing. They are double length TDES keys for this service which are called **static TDES keys** in this document.

The verb operates in reformat mode. In reformat mode, the verb performs the translate-mode functions (changes the wrapping key) and, in addition, processes the cleartext information. Following the rules that you specify, the PIN is extracted from the recovered cleartext PIN block using the specified input PIN encrypting key and formatted into an output PIN block according to the output PIN profile for encryption. The PIN block is re-enciphered with the specified output PIN encrypting key. Change of PAN data is not allowed.

The Encrypted PIN Translate Enhanced verb performs the following processing:
- It decrypts the input PIN-block by using the supplied IPINENC key in ECB mode, or derives the decryption key using the specified KEYGENKY key and the current-key serial number (CKSN), and then uses ANSI X9.24-specified special decryption or the Triple-DES (TDES) method. The PAN must be deciphered using either the data decryption key derived from the base derivation key and Cksen or using the specified static TDES data decryption key (DECIPHER, CIPHER).
- Checks the control vector of the input PIN encryption key to ensure that for an IPINENC key the REFORMAT bit (CV bit 22) is set to B’1’ for reformat mode, or for a KEYGENKY key, that the UKPT bit (CV bit 18) is set to B’1’. Likewise the OPINENC key must have the REFORMAT bit set according to the requested mode.
- In reformat mode, performs these steps:
  - It extracts the PIN from the specified PIN-block format using the method specified by default or by a rule-array keyword. If required by the PIN-block format, PAN data is used in the extraction process.
Encrypted PIN Translate Enhanced (CSNBPTRE)

- Formats the extracted-PIN into the format declared for the output PIN-block. As required by the PIN-block format, the verb incorporates PAN data, sequence number, and pad character information in formatting the output.

- It enciphers the output PIN-block by using the supplied static OPINENC key in ECB mode, or derives the decryption key using the specified KEYGENKY key and the output current-key serial number (CKSN) from the output PIN profile and uses ANSI X9.24-specified special encryption or Triple-DES method. The REFORMAT bit must be set to B’1’ in the OPINENC control vector, or the UKPT bit must be set to B’1’ in the KEYGENKY control vector.

**Note:** This verb supports PCI-HSM 2016 compliant-tagged key tokens.

**Format**

The format of CSNBPTRE.

```c
CSNBPTRE(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    input_PIN_encrypting_key_identifier_length,
    input_PIN_encrypting_key_identifier,
    output_PIN_encrypting_key_identifier_length,
    output_PIN_encrypting_key_identifier,
    PAN_key_identifier_length,
    PAN_key_identifier,
    input_PIN_profile_length,
    input_PIN_profile,
    PAN_data_length,
    PAN_data,
    input_PIN_block_length,
    input_PIN_block,
    output_PIN_profile_length,
    output_PIN_profile,
    sequence_number,
    output_PIN_block_length,
    output_PIN_block,
    reserved1_length,
    reserved1,
    reserved2_length,
    reserved2)
```

**Parameters**

The parameters for CSNBPTRE.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- Direction: Input
- Type: Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. The value must be in the range 6 – 9.

**rule_array**
Encrypted PIN Translate Enhanced (CSNBPTRE)

Direction: Input
Type: String array

A pointer to a string variable containing an array of keywords. The keywords are 8 bytes in length, and are left-aligned and padded on the right with space characters. The returned rule array keywords express the contents of the token.

Table 170. Keywords for Encrypted PIN Translate Enhanced control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processing method</strong> (required)</td>
<td></td>
</tr>
<tr>
<td>VMDS</td>
<td>Specifies that the VDSP method (Visa Data Secure Platform method, formally known as the Visa Merchant Data Secure (VMDS) method) is to be used for processing.</td>
</tr>
<tr>
<td><strong>Mode</strong> (required)</td>
<td></td>
</tr>
<tr>
<td>REFORMAT</td>
<td>Specifies that either the PIN-block format and the PIN-block encryption, or both, are to be changed. If the PIN-extraction method is not chosen by default, another element in the rule array must specify one of the keywords that indicates a PIN-extraction method.</td>
</tr>
<tr>
<td><strong>Input PAN data key management method</strong> (one, required)</td>
<td></td>
</tr>
<tr>
<td>IN-DUKPT</td>
<td>Specifies that the key to be used to decrypt the PAN data is to be derived using the key specified in the input_PIN_encrypting_key_identifier. See the description of the input_PIN_encrypting_key_identifier for the requirements of the key. The DUKPT-BH or DUKPT-IP keyword is required.</td>
</tr>
<tr>
<td>OUTDUKPT</td>
<td>Specifies that the key to be used to decrypt the PAN data is to be derived using the key specified in the output_PIN_encrypting_key_identifier. See the description of the output_PIN_encrypting_key_identifier for the requirements of the key. The DUKPT-BH or DUKPT-OP keyword is required.</td>
</tr>
<tr>
<td>STATIC</td>
<td>Specifies the use of static double length (2-key) Triple-DES symmetric keys for the PAN.</td>
</tr>
<tr>
<td><strong>Input data algorithm</strong> (one, required)</td>
<td></td>
</tr>
<tr>
<td>TDES</td>
<td>Specifies that Triple-DES encryption was used for the PAN.</td>
</tr>
<tr>
<td><strong>Input data mode</strong> (one, required)</td>
<td></td>
</tr>
<tr>
<td>CBC</td>
<td>Specifies that CBC mode encryption was used for the PAN. This is the mode for the Standard Encryption option.</td>
</tr>
<tr>
<td>VFPE</td>
<td>Specifies that Visa format preserving encryption was used for the PAN.</td>
</tr>
<tr>
<td><strong>PAN input character set</strong> (one, required)</td>
<td></td>
</tr>
<tr>
<td>PAN4BITX</td>
<td>Specifies that the PAN data character set is 4-bit hex. Two digits per byte. Not valid with the CBC rule.</td>
</tr>
<tr>
<td>PAN8BITA</td>
<td>Specifies that the PAN data character set is normal ASCII, represented in binary format. Not valid with CBC rule.</td>
</tr>
<tr>
<td>PAN-EBLK</td>
<td>Specifies that the PAN data is in a CBC encrypted block. Valid only with CBC rule.</td>
</tr>
<tr>
<td><strong>PAN check digit compliance</strong> (one required if mode VFPE and PAN input character set keyword are present, otherwise not allowed)</td>
<td></td>
</tr>
<tr>
<td>CMPCKDGT</td>
<td>Last digit of the PAN contains a compliant check digit per ISO/IEC 7812-1.</td>
</tr>
<tr>
<td>NONCKDGT</td>
<td>Last digit of the PAN does not contain a compliant check digit per ISO/IEC 7812-1.</td>
</tr>
<tr>
<td><strong>Unique Key Per Transaction</strong> (one, optional). These keywords are for the PIN-encrypting keys.</td>
<td></td>
</tr>
<tr>
<td>DUKPT-BH</td>
<td>Specifies that the input and output PIN-encrypting keys are to be derived using the key-generating key specified in the respective parameters. See the descriptions of the input_PIN_key_identifier and output_PIN_key_identifier parameters for the requirements of the keys.</td>
</tr>
</tbody>
</table>
### Keywords for Encrypted PIN Translate Enhanced control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUKPT-IP</td>
<td>Specifies the use of DUKPT input-key derivation and PIN-block decryption, Triple-DES method. Specifies that the input PIN-encrypting key is to be derived using the key-generating key specified in the <code>input_PIN_key_identifier</code> parameter. See the description of the <code>input_PIN_key_identifier</code> parameter for the requirements of the key.</td>
</tr>
<tr>
<td>DUKPT-OP</td>
<td>Specifies that the output PIN-encrypting key is to be derived using the key-generating key specified in the <code>output_PIN_key_identifier</code> parameter. See the description of the <code>output_PIN_key_identifier</code> parameter for the requirements of the key.</td>
</tr>
</tbody>
</table>

### PIN-extraction method (one, optional).

If the PIN block format is provided, one of the PIN extraction method keywords shown in Table 143 on page 498 can be specified for the given PIN block format.

**Note:** Specify the **PIN block format** keyword in the **PIN_profile** variable (see “PIN block format” on page 498).

The following PIN-block formats are supported:

- 3624
- ISO-0
- ISO-1
- ISO-2
- ISO-3

The following S390 formats are also supported: VISA-2, VISA-3, VISA-4, OEM-1, ECI-2, ECI-3.

See “PIN block format and PIN extraction method keywords” on page 498 for additional information. If the default extraction method for a PIN block format is desired, specify the `rule_array_count` value as 1.

---

#### input_PIN_encrypting_key_identifier_length

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the `input_PIN_encrypting_key_identifier` variable. Set the value to 64.

#### input_PIN_encrypting_key_identifier

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing an operational fixed-length DES key-token or a key label of an operational fixed-length DES key-token record.

This is the identifier of the key to decrypt the input PIN block. Or it is the key-generating key to be used to derive the key to decrypt the input PIN block. The key-generating key can optionally be used to derive the key to decrypt the PAN data.

The key identifier is an operational token or the key label of an operational token in key storage. If you do not use the UKPT process or you specify the DUKPT-IP rule array keyword, the key token must contain the PIN-encrypting key to be used to decipher the input PIN block. The key algorithm must be DES, the key type must be IPIENC and the key usage REFORMAT bit must be enabled (B’1’).

If you use the UKPT process for the input PIN block by specifying the DUKPT-IP or the DUKPT-BH rule array keyword, the key token must contain the key-generating key to derive the PIN-encrypting key. If you have also
Encrypted PIN Translate Enhanced (CSNBPTRE)

specified the IN-DUKPT keyword, the key is used to derive the key to decrypt the PAN data. The key algorithm must be DES, the key type must be KEYGENKY and the key usage UKPT bit must be enabled.

output_PIN_encrypting_key_identifier_length
Direction: Input
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the output_PIN_encrypting_key_identifier variable. Set the value to 64.

output_PIN_encrypting_key_identifier
Direction: Input
Type: String

A pointer to a string variable containing an operational key token or a key label of an operational key token record in DES key storage.

The identifier of the key to encrypt the output PIN block or the key-generating key to be used to derive the key to encrypt the output PIN block. The key-generating key can optionally be used to derive the key to decrypt the PAN data.

If you do not use the UKPT process or you specify the DUKPT-IP rule array keyword, the key token must contain the PIN-encrypting key to be used to encipher the output PIN block. The key algorithm must be DES, the key type must be OPINENC and the key usage REFORMAT bit must be enabled.

If you use the UKPT process for the output PIN block by specifying the DUKPT-OP or the DUKPT-BH rule array keyword, the key token must contain the key-generating key to derive the PIN-encrypting key. If you have also specified the OUTDUKPT keyword, the key is used to derive the key to decrypt the PAN data. The key algorithm must be DES, the key type must be KEYGENKY and the key usage UKPT bit must be enabled (B'1').

PAN_key_identifier_length
Direction: Input
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the PAN_key_identifier variable. Set the value to 0 if the PAN key management method keyword specifies DUKPT. Set it to 64 if the PAN key management method specifies STATIC.

PAN_key_identifier
Direction: Input
Type: String

A pointer to a string variable containing an internal fixed-length DES key-token or the key label of such a record in DES key-storage. This token contains a double-length data decryption key if the input PAN data key management method is STATIC. If the key token contains a double-length data encryption key (Zone Encryption Key in the VDSP specification), the token must have a key type of CIPHER or DECIPHER.

The key is used to decipher the input PAN data. If IN-DUKPT is specified as the input PAN data key management method, the base derivation key in the input_PIN_encrypting_key_identifier parameter is used to create the decryption key for the PAN.
If OUTDUKPT is specified as the input PAN data key management method, the base derivation key in the **output_PIN_encrypting_key_identifier** parameter is used to create the decryption key for the PAN. This parameter is not used if IN-DUKPT or OUTDUKPT is specified.

The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm must be DES, the key type must be CIPHER or DECIPHER and the key must be a double-length key.

**Note:** DATA keys with ENC or DEC bits on are not supported. Also, zero CV data keys are not supported.

### input_PIN_profile_length

**Direction:** Input  
**Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the input_PIN_profile variable. Set the value to 24 if the profile does not contain a CKSN extension. Otherwise, set the value to 48.

### input_PIN_profile

**Direction:** Input  
**Type:** String

A pointer to a string variable containing three 8-byte character strings with information defining the PIN-block format and, optionally, an additional 24 bytes containing the input CKSN extension. The strings are equivalent to 24-byte or 48-byte strings.

### PAN_data_length

**Direction:** Input  
**Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the PAN_data parameter if the mode is CBC. If the mode is VFPE, this variable contains the number of PAN digits. The value is in the range 15 - 19 for VFPE. It is 16 if the standard encryption option is selected.

### PAN_data

**Direction:** Input  
**Type:** String

A pointer to a string variable containing the PAN data. For VFPE mode, if the PAN contains an odd number of 4-bit hex digits, the data must be left justified in the PAN variable and the right-most 4 bits are ignored. The verb uses this data to recover the PIN from the PIN block if you specify the REFORMAT keyword and the input PIN profile specifies the ISO-0, VISA-4 or ISO-3 keyword for the PIN-block format. If the output PIN profile specifies the ISO-0, VISA-4, or ISO-3 keyword for the PIN-block format, the 12 rightmost digits of the (decrypted) PAN, excluding the check digit, are used to format the output PIN block.

### input_PIN_block_length

**Direction:** Input  
**Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the encrypted input PIN block. The value must be 8.
input_PIN_block

Direction: Input
Type: String

A pointer to a string variable containing the encrypted PIN-block.

output_PIN_profile_length

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the output_PIN_profile variable. The value is 24 or 48.

output_PIN_profile

Direction: Input
Type: String

A pointer to a string variable containing three 8-byte character strings with information defining the PIN-block format and, optionally, an additional 24 bytes containing the output CKSN extension. The strings are equivalent to 24-byte or 48-byte strings.

sequence_number

Direction: Input
Type: Integer

A pointer to an integer variable containing the sequence number. Ensure that the referenced integer variable is valued to 99999 if the output PIN block format is 3621 or 4704-EPP. Otherwise, this parameter is ignored.

output_PIN_block_length

Direction: Input/Output
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the re-enciphered PIN block. The value must be at least 8.

output_PIN_block

Direction: Output
Type: String

A pointer to a string variable containing the re-enciphered and, optionally, reformatted PIN-block returned by the verb. The buffer can be larger on input. However, on output this field is updated to indicate the actual number of bytes returned by the card.

reserved1_length

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the reserved1 variable. This value must be zero.

reserved1

Direction: Input/Output
Type: String
A pointer to a string variable. This parameter is reserved for future use.

reserved2_length

Direction: Input/Output
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the reserved2 variable. This value must be zero.

reserved2

Direction: Output
Type: String

A pointer to a string variable. This parameter is reserved for future use.

**Restrictions**

The restrictions for CSNBPTRE.

None.

**Required commands**

The required commands for CSNBPTRE.

The Encrypted PIN Translate Enhanced verb requires the **Encrypted PIN Translate Enhanced** command (offset X'02D5') to be enabled in the active role. This ACP is ON by default. The following additional commands must be enabled depending on the capabilities that are requested:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Input profile format control keyword</th>
<th>Output profile format control keyword Explanation</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>REFORMAT</td>
<td>NONE</td>
<td>NONE</td>
<td>X'00B7'</td>
<td>Encrypted PIN Translate - Reformat</td>
</tr>
<tr>
<td></td>
<td>One or more of the following: IN-DUKPT, OUTDUKPT, DUKPT-OP, DUKPT-IP, or DUKPT-BH</td>
<td></td>
<td>X'001E'</td>
<td>Reencipher CKDS</td>
</tr>
</tbody>
</table>

An enhanced PIN security mode is available for extracting PINs from an IBM 3624 encrypted PIN-block and formatting an encrypted PIN block into 3621 or 3624 format using the PADDIGIT PIN-extraction method. This mode limits checking of the PIN to decimal digits, and a minimum PIN length of 4 is enforced. No other PIN-block consistency checking occurs. To activate this mode, enable the **Enhanced PIN Security** command (offset X'0313') in the active role.

The verb returns an error indicating that the PAD digit is not valid if all of these conditions are met:
1. The Enhanced PIN Security command is enabled in the active role.
2. The output PIN profile specifies 3624 as the PIN-block format.
3. The output PIN profile specifies a decimal digit (0 - 9) as the PAD digit.

Three additional commands should be considered (offsets X'0350', X'0351', and X'0352'). If enabled, these three commands affect how PIN processing by this and other verbs is performed:

1. Enable the ANSI X.9.8 PIN - Enforce PIN block restrictions command (offset X'0350') in the active role to apply additional restrictions to PIN processing as follows:
   - Do not translate or reformat a non-ISO PIN block into an ISO PIN block. Specifically, do not allow an IBM 3624 PIN-block format in the output_PIN_profile variable when the PIN-block format in the input_PIN_profile variable is not 3624.
   - Constrain use of ISO-2 PIN blocks to offline PIN verification and PIN change operations in integrated circuit card environments only. Specifically, do not allow ISO-2 input or output PIN blocks.
   - Do not translate or reformat a PIN-block format that includes a PAN into a PIN-block format that does not include a PAN. Specifically, do not allow an ISO-1 PIN-block format in the output_PIN_profile variable when the PIN-block format in the input_PIN_profile variable is ISO-0 or ISO-3.

2. Enable the ANSI X.9.8 PIN - Allow modification of PAN command (offset X'0351') in the active role to override the restriction to not allow a change of PAN data. This override is applicable only when either the ANSI X.9.8 PIN - Enforce PIN block restrictions command (offset X'0350') or the ANSI X.9.8 PIN - Allow only ANSI PIN blocks command (offset X'0352') or both are enabled in the active role. This override supports environments that issue account number changes. Offset X'0351' has no effect if neither offset X'0350' nor offset X'0352' is enabled in the active role. This rule does not apply for PTRE. Also, PAN changes are not allowed.

3. Enable the ANSI X.9.8 PIN - Allow only ANSI PIN blocks command (offset X'0352') in the active role to apply a more restrictive variation of the ANSI X.9.8 PIN - Enforce PIN block restrictions command (offset X'0350'). In addition to the previously described restrictions of offset X'0350', this command also restricts the input_PIN_profile and the output_PIN_profile parameters to contain only ISO-0, ISO-1, and ISO-3 PIN block formats. Specifically, the IBM 3624 PIN-block format is not allowed with this command. The command at offset X'0352' overrides the one at offset X'0350'.

Usage notes

The usage notes for CSNBPTRE.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBPTREJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBPTREJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
)```

554 Common Cryptographic Architecture Application Programmer's Guide
Encrypted PIN Translate Enhanced (CSNBPTRE)

Encrypted PIN Verify (CSNBPVR)

Use the Encrypted PIN Verify verb to verify that one of the customer selected trial PINs is valid.

Use the verb to verify that one of the following PINs is valid:

- IBM 3624 (IBM-PIN)
- IBM 3624 PIN offset (IBM-PINO)
- IBM German Bank Pool (GBP-PIN)
- VISA PIN validation value (VISA-PVV)
- VISA PIN validation value (VISAPVV4)
- Interbank PIN (INBK-PIN)

The unique-key-par-transaction key derivation for single and double-length keys is available for the `input_PIN_encrypting_key_identifier` parameter.

**Note:** This verb supports PCI-HSM 2016 compliant-tagged key tokens.
Encrypted PIN Verify (CSNPBVR)

Format

The format of CSNPBVR.

```
CSNPBVR(  
  return_code,  
  reason_code,  
  exit_data_length,  
  exit_data,  
  input_PIN_encrypting_key_identifier,  
  PIN_verifying_key_identifier,  
  input_PIN_profile,  
  PAN_data,  
  encrypted_PIN_block,  
  rule_array_count,  
  rule_array,  
  PIN_check_length,  
  data_array)
```

Parameters

The parameters for CSNPBVR.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

`input_PIN_encrypting_key_identifier`

Direction: Input/Output  
Type: String

The 64-byte key label or internal key token containing the PIN-encrypting key (IPINENC) that enciphers the PIN block. If keyword UKPTIPIN or DUKPT-IP is specified in the `rule_array`, the `input_PIN_encrypting_key_identifier` must specify a key token or key label of a KEYGENKY with the UKPT usage bit enabled.

`PIN_verifying_key_identifier`

Direction: Input/Output  
Type: String

The 64-byte key label or internal key token that identifies the PIN verify (PINVER) key.

`input_PIN_profile`

Direction: Input  
Type: String

The three 8-byte character elements that contain information necessary to either create a formatted PIN block or extract a PIN from a formatted PIN block. A particular PIN profile can be either an input PIN profile or an output PIN profile depending on whether the PIN block is being enciphered or deciphered by the verb. If you specify UKPTIPIN in the `rule_array` parameter, the `input_PIN_profile` is extended to a 48-byte field and must contain the current key serial number. See “The PIN profile” on page 497 for additional information.

If you specify DUKPT-IP in the `rule_array` parameter, the `input_PIN_profile` is extended to a 48-byte field and must contain the current key serial number. See “The PIN profile” on page 497 for additional information.
Encrypted PIN Verify (CSNBPR)

The pad digit is needed to extract the PIN from a 3624 or 3621 PIN block in the Encrypted PIN Verify verb.

The PINLENnn keywords are disabled for this verb by default. If these keywords are used, return code 8 with reason code 33 is returned. To enable them, the PTR Enhanced PIN Security access control point (bit X'0313') must be enabled using a TKE workstation.

**PAN_data**

- **Direction:** Input
- **Type:** String

The personal account number (PAN) is required for ISO-0, ISO-3, and VISA-4. Otherwise, this parameter is ignored. Specify 12 digits of account data in character format.

For ISO-0 or ISO-3, use the rightmost 12 digits of the PAN, excluding the check digit.

For VISA-4, use the leftmost 12 digits of the PAN, excluding the check digit.

**encrypted_PIN_block**

- **Direction:** Input
- **Type:** String

The 8-byte enciphered PIN block that contains the PIN to be verified.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1, 2, or 3.

**rule_array**

- **Direction:** Input
- **Type:** String array

The process rule for the PIN verify algorithm, described in Table 171.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithm value</strong> (One, required)</td>
<td></td>
</tr>
<tr>
<td>GBP-PIN</td>
<td>The IBM German Bank Pool PIN. It verifies the PIN entered by the customer and compares that PIN with the institution generated PIN by using an institution key.</td>
</tr>
<tr>
<td>IBM-PIN</td>
<td>The IBM 3624 PIN, which is an institution-assigned PIN. It does not calculate the PIN offset.</td>
</tr>
<tr>
<td>IBM-PINO</td>
<td>The IBM 3624 PIN offset, which is a customer-selected PIN and calculates the PIN offset.</td>
</tr>
<tr>
<td>INBK-PIN</td>
<td>The Interbank PIN verify algorithm.</td>
</tr>
<tr>
<td>VISA-PVV</td>
<td>The VISA PIN verify value.</td>
</tr>
<tr>
<td>VISAPVV4</td>
<td>The VISA PIN verify value. If the length is 4 digits, normal processing for VISA-PVV will occur.</td>
</tr>
</tbody>
</table>
### Encrypted PIN Verify (CSNBPVR)

**Table 171. Keywords for Encrypted PIN Verify control information (continued)**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PIN block format and PIN extraction method</strong> (Optional)</td>
<td>See “PIN block format and PIN extraction method keywords” on page 498 for additional information and a list of PIN block formats and PIN extraction method keywords. The PINLENnn keywords are disabled for this verb by default. If these keywords are used, return code 8 with reason code 33 is returned. To enable them, the PTR Enhanced PIN Security access control point (bit X'0313') must be enabled using a TKE workstation. <strong>Note:</strong> If a PIN extraction method is not specified, the first one listed in Table 143 on page 498 for the PIN block format will be the default.</td>
</tr>
<tr>
<td><strong>DUKPT keyword - single length key derivation</strong> (Optional)</td>
<td>The input_PIN_encrypting_key_identifier is derived as a single length key. The input_PIN_encrypting_key_identifier must be a KEYGENKY key with the UKPT usage bit enabled. The input_PIN_profile must be 48 bytes and contain the key serial number.</td>
</tr>
<tr>
<td><strong>DUKPT keyword - double length key derivation</strong> (Optional)</td>
<td>The input_PIN_encrypting_key_identifier is to be derived using the DUKPT algorithm. The input_PIN_encrypting_key_identifier must be a KEYGENKY key with the DUKPT usage bit enabled. The input_PIN_profile must be 48 bytes and contain the key serial number.</td>
</tr>
</tbody>
</table>

---

**PIN_check_length**

- **Direction:** Input
- **Type:** String

The PIN check length for the **IBM-PIN** or **IBM-PINO** process rules only. Otherwise, it is ignored. Specify the rightmost digits, 4 - 16, for the PIN to be verified.

**data_array**

- **Direction:** Input
- **Type:** Integer

Three 16-byte elements required by the corresponding rule_array parameter. The data array consists of three 16-byte fields whose specification depend on the process rule. If a process rule requires only one or two 16-byte fields, the rest of the data array is ignored by the verb. **Table 172** describes the array elements.

**Table 172. Array elements for Encrypted PIN Verify data_array parameter**

<table>
<thead>
<tr>
<th>Array element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimalization_table</td>
<td>Decimalization table for IBM and GBP only. Sixteen decimal digits of 0 - 9. If the <strong>ANSI X9.8 PIN - Use stored decimalization tables only</strong> command (offset X'0356') access control point is enabled in the active role, this table must match one of the active decimalization tables in the coprocessors.</td>
</tr>
<tr>
<td>PIN_offset</td>
<td>Offset data for <strong>IBM-PINO</strong>. One to twelve numeric characters, 0 - 9, left-aligned and padded on the right with blanks. For <strong>IBM-PINO</strong>, the PIN offset length is specified in the PIN_check_length parameter. For <strong>IBM-PIN</strong> and <strong>GBP-PIN</strong>, the field is ignored.</td>
</tr>
<tr>
<td>Trans_sec_parm</td>
<td>For VISA, only the leftmost twelve digits of the 16-byte field are used. These consist of the rightmost eleven digits of the personal account number (PAN) and a one-digit key index. The remaining four characters are ignored. For Interbank only, all 16 bytes are used. These consist of the rightmost eleven digits of the PAN, a constant of X'6', a one-digit key index, and three numeric digits of PIN validation data.</td>
</tr>
</tbody>
</table>
Encrypted PIN Verify (CSNBPVR)

Table 172. Array elements for Encrypted PIN Verify data_array parameter (continued)

<table>
<thead>
<tr>
<th>Array element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPVV</td>
<td>For VISA-PVV only, referenced PVV (four bytes) that is left-aligned. The rest of the field is ignored.</td>
</tr>
<tr>
<td>Validation_data</td>
<td>Validation data for IBM and GBP padded to 16 bytes. 1 - 16 characters of hexadecimal account data left-aligned and padded on the right with blanks.</td>
</tr>
</tbody>
</table>

Table 173 lists the data array elements required by the process rule (rule_array parameter). The numbers refer to the process rule’s position within the array.

Table 173. Array elements required by the process rule

<table>
<thead>
<tr>
<th>Process rule</th>
<th>IBM-PIN</th>
<th>IBM-PINO</th>
<th>GBP-PIN</th>
<th>VISA-PVV</th>
<th>INBK-PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimalization_table</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Validation_data</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIN_offset</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trans_sec_parm</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RPVV</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Restrictions

The restrictions for CSNBPVR.

None.

Required commands

The required commands for CSNBPVR.

This verb requires the following commands to be enabled in the active role:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM-PIN, IBM-PINO</td>
<td>X’00AB’</td>
<td>Encrypted PIN Verify - 3624</td>
</tr>
<tr>
<td>GBP-PIN</td>
<td>X’00AC’</td>
<td>Encrypted PIN Verify - GBP</td>
</tr>
<tr>
<td>VISA-PVV, VISAPVV4</td>
<td>X’00AD’</td>
<td>Encrypted PIN Verify - VISA PVV</td>
</tr>
<tr>
<td>INBK-PIN</td>
<td>X’00AE’</td>
<td>Encrypted PIN Verify - Interbank</td>
</tr>
</tbody>
</table>

This verb also requires the **UKPT - PIN Verify, PIN Translate** command (offset X’00E1’) to be enabled in the active role if you employ UKPT processing.

Note: A role with offset X’00E1’ enabled can also use the Encrypted PIN Translate verb with UKPT processing.

An enhanced PIN security mode is available for extracting PINs from an IBM 3624 encrypted PIN-block using the **PADDIGIT** PIN-extraction method. This mode limits checking of the PIN to decimal digits, and a minimum PIN length of four is enforced. No other PIN-block consistency checking will occur. To activate this mode, enable the **Enhanced PIN Security** command (offset X’0313’) in the active role.

Whenever the **ANSI X9.8 PIN - Use stored decimalization tables only** command (offset X’0356’) is enabled in the active role, the **Decimalization_table**
Encrypted PIN Verify (CSNBPRV)

Element of the data_array value must match one of the PIN decimalization tables that are in the active state on the coprocessor. Use of this command provides improved security and control for PIN decimalization tables. The VISA-PVV, VISAPVV4, and INBK-PIN PIN calculation methods do not have a Decimalization_table element and are therefore not affected by this command.

Usage notes

The usage notes for CSNBPRV.

None.

Related information

Additional information about CSNBPRV.

The algorithms are described in detail in Chapter 22, “PIN formats and algorithms,” on page 1005.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBPRVRJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

public native void CSNBPRVRJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    byte[] input_PIN_encrypting_key_identifier,
    byte[] PIN_verifying_key_identifier,
    byte[] input_PIN_profile,
    byte[] PAN_data,
    byte[] encrypted_PIN_block,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber PIN_check_length,
    byte[] data_array);

FPE Decipher (CSNBFPED)

The FPE Decipher verb is used to decrypt payment card data for the Visa Data Secure Platform (VDSP) processing.

The FPE Decipher verb uses the TDES (Triple-DES) algorithm with cipher block chaining (CBC) or Visa Format Preserving Encryption (VFPE) mode and a cipher key to decipher data called ciphertext. This verb returns data called plaintext.

The verb can be used to decipher one or all of the following fields:

- the primary account number (PAN),
- the cardholder name,
- the Track 1 Discretionary Data,
- the Track 2 Discretionary Data.

Also refer to “Visa Format-Preserving Encryption supporting information”
The CSNBFPED verb returns the enciphered fields and optionally the DUKPT PIN key, if DUKPT key management is selected.

**Terminology:** The VDSP specification speaks of two key management methods: DUKPT (derived unique key per transaction) and Zone Encryption Keys. The process for deriving these keys is documented in ANSI X9.24 Part 1. Zone Encryption Keys are called static keys in CCA. Static keys are presented for use and are not derived during verb processing. They are double length TDES keys for this service which are called static TDES keys in this document.

**Note:** This verb supports PCI-HSM 2016 compliant-tagged key tokens.

### Format

The format of CSNBFPED.

```c
CSNBFPED(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    enc_PAN_length,
    enc_PAN,
    enc_cardholder_name_length,
    enc_cardholder_name,
    enc_dtrack1_data_length,
    enc_dtrack1_data,
    enc_dtrack2_data_length,
    enc_dtrack2_data,
    key_identifier_length,
    key_identifier,
    derivation_data_length,
    derivation_data,
    clear_PAN_length,
    clear_PAN,
    clear_cardholder_name_length,
    clear_cardholder_name,
    clear_dtrack1_data_length,
    clear_dtrack1_data,
    clear_dtrack2_data_length,
    clear_dtrack2_data,
    DUKPT_PIN_key_identifier_length,
    DUKPT_PIN_key_identifier,
    reserved1_length,
    reserved1,
    reserved2_length,
    reserved2
)
```

### Parameters

The parameters for CSNBFPED.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the
FPE Decipher (CSNBFPED)

rule_array variable. The minimum value is 5. The maximum value is 10.

rule_array

Direction: Input
Type: String array

Keywords that provide control information to the verb. The rule_array keywords are described in Table 174.

Note: At least one character set keyword is required.

Table 174. Keywords for FPE Decipher control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing method</td>
<td>Specifies that the VDSP method (Visa Data Secure Platform method, formally known as the Visa Merchant Data Secure (VMDS) method) is to be used for processing.</td>
</tr>
<tr>
<td>VMDS</td>
<td>Specifies the use of double length (2-key) triple-DES symmetric keys. This is a non-DUKPT key.</td>
</tr>
<tr>
<td>DUKPT</td>
<td>Specifies the use of the transaction unique general purpose Data Encryption Keys generated by the DUKPT process at the point of service for data encryption. This is required if VFPE mode is specified. Otherwise, this is optional.</td>
</tr>
<tr>
<td>PRocessing method</td>
<td>Specifies the use of CBC mode triple-DES encryption.</td>
</tr>
<tr>
<td>TDES</td>
<td>Specifies the use of the standard encryption option.</td>
</tr>
<tr>
<td>MODE</td>
<td>Specifies the use of Visa format preserving encryption.</td>
</tr>
<tr>
<td>PAN input output character set</td>
<td>Specifies that the PAN data character set is ASCII represented in binary form, one character per byte. Valid ASCII values are in the range 0 - 9 (X'30' - X'39').</td>
</tr>
<tr>
<td>TK18BITA</td>
<td>Specifies that the track 1 discretionary data character set is ASCII represented in binary format, one character per byte. Valid ASCII values are in the range 0 - 9 (X'30' - X'39') and A - F (X'41' - X'46').</td>
</tr>
<tr>
<td>PIN encryption key output selection</td>
<td>Do not return a DUKPT PIN encryption key. This is the default.</td>
</tr>
<tr>
<td>PINKEY</td>
<td>Return a DUKPT PIN encryption key.</td>
</tr>
<tr>
<td>PAN check digit compliance</td>
<td>Specifies that the cardholder name character set is ASCII represented in binary format, one character per byte. Valid ASCII values are in the range 0 - 9 (X'30' - X'39').</td>
</tr>
<tr>
<td>CN8BITA</td>
<td>Specifies that the PAN data character set is ASCII represented in binary form, one character per byte. Valid ASCII values are in the range 0 - 9 (X'30' - X'39').</td>
</tr>
<tr>
<td>TK28BITA</td>
<td>Specifies that the track 2 discretionary data character set is ASCII represented in binary format, one character per byte. Valid ASCII values are in the range 0 - 9 (X'30' - X'39') and A - F (X'41' - X'46').</td>
</tr>
</tbody>
</table>
### Table 174. Keywords for FPE Decipher control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMPCKDGT</td>
<td>Last digit of the PAN contains a compliant check digit per ISO/IEC 7812-1.</td>
</tr>
<tr>
<td>NONCKDGT</td>
<td>Last digit of the PAN does not contain a compliant check digit per ISO/IEC 7812-1.</td>
</tr>
</tbody>
</table>

**enc_PAN_length**

- **Direction:** Input
- **Type:** Integer

Specifies the length of the `enc_PAN` parameter in bytes if the mode is CBC or the number of PAN digits if the mode is VFPE. The value is either 0 or in the range 15 - 19 for VFPE. The value must be 0 or 16 if the standard option with CBC mode is selected. The value is zero when the PAN has not been presented for decryption.

**enc_PAN**

- **Direction:** Input
- **Type:** String

The enciphered primary account number (PAN) that is to be decrypted. For VFPE mode, if the PAN contains an odd number of 4-bit digits, the data is left justified in the PAN variable and the right-most 4 bits are ignored.

**enc_cardholder_name_length**

- **Direction:** Input
- **Type:** Integer

Specifies the length of the `enc_cardholder_name` parameter in bytes. The input value is either 0 or in the range 1 - 32, inclusive for VFPE. For the standard method, the input value is either 0 or in the range 2 - 32 for VFPE. For CBC mode, the input value is 0, 16, 24, 32, or 40. The value is zero when the cardholder name has not been presented for decryption.

**enc_cardholder_name**

- **Direction:** Input
- **Type:** String

The enciphered cardholder full name that is to be decrypted. Only characters in Table 150 on page 505 are valid.

**enc_dtrack1_data_length**

- **Direction:** Input
- **Type:** Integer

Specifies the length of the `enc_dtrack1_data` parameter in bytes. The input value is either 0 or in the range 1 - 56, inclusive for VFPE. For the standard method, the input value is either 0 or in the range 1 - 56 for VFPE. For CBC mode, the input value is 0 or 16, 24, 32, 40, 48, 56, or 64. The value is zero when the track 1 discretionary data has not been presented for decryption.

**enc_dtrack1_data**

- **Direction:** Input
- **Type:** String
The encrypted track 1 data that is to be decrypted. Only characters in Table 150 on page 505 are valid.

**enc_dtrack2_data_length**

- **Direction:** Input
- **Type:** Integer

Specifies the length of the `enc_dtrack2_data` parameter in bytes. The input value is either 0 or in the range 1 - 19 for VFPE. For mode CBC, the input value is 0, 8, or 16. The value is zero when the track 2 discretionary data is not presented for decryption.

**enc_dtrack2_data**

- **Direction:** Input
- **Type:** String

The encrypted track 2 data that is to be decrypted.

**key_identifier_length**

- **Direction:** Input
- **Type:** Integer

Specifies the length of the `key_identifier` parameter in bytes. The value must be 64, because only fixed length DES tokens are supported as the key identifier.

**key_identifier**

- **Direction:** Input/Output
- **Type:** String

The identifier of the key that is used to either decrypt the card data (key management STATIC) or derive the DUKPT_PIN_key_identifier (key management DUKPT). The `key_identifier` is an operational token or the key label of an operational token in key storage.

For key management DUKPT, the key type must be KEYGENKY. In addition, it must have a control vector with bit 18 equal to B’1’ (UKPT). The base derivation key is the one from which the operational keys are derived using the DUKPT algorithm defined in ANS X9.24 Part 1.

For key management STATIC, (Zone Encryption Key in the VDSP specification), the key type must be either CIPHER or ENCIPHER. For production purposes, it is recommended that the key have left and right halves that are not equal.

**Note:** Data keys are not supported.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**derivation_data_length**

- **Direction:** Input
- **Type:** Integer

Specifies the length of the `derivation_data` parameter in bytes. The value must be 10 if a DUKPT key is specified in the `key_identifier` parameter. If a data encryption key is specified in the `key_identifier` parameter, this value must be set to zero.
derivation_data

Direction: Input
Type: String

Contains the 80 bit (10 byte) derivation data that is used as input to the DUKPT derivation process. The derivation data contains the current key serial number (CKSN), which is composed of the 59 bit initial key serial number, and concatenated with the 21 bit value of the current encryption counter, which the device increments for each new transaction. This field is in binary format.

clear_PAN_length

Direction: Input/Output
Type: Integer

Specifies the number of PAN digits in the clear_PAN parameter. This value must either be 0 or in the range 15 - 19, inclusive on output.

clear_PAN

Direction: Output
Type: String

This parameter returns the deciphered primary account number. The full account number, including check digit, is recovered. The data for this parameter is returned in binary format. It is the binary representation of 4-bit hex (keyword PAN4BITX) or ASCII (keyword PAN8BITA) as indicated by the supplied rule array keyword. The clear PAN is left justified in this field.

clear_cardholder_name_length

Direction: Input/Output
Type: Integer

Specifies the length of the clear_cardholder_name parameter in bytes. This output value is either 0 or in the range 2 - 32 on output. The variable can be larger on input. However, on output, this field is updated to indicate the actual number of bytes returned by the service.

clear_cardholder_name

Direction: Output
Type: String

This parameter returns the deciphered cardholder full name. The output data for this parameter is in binary format. It is the binary representation of ASCII as indicated by the supplied rule array keyword.

clear_dtrack1_data_length

Direction: Input/Output
Type: Integer

Specifies the length of the clear_dtrack1_data parameter in bytes. The output value is either 0 or in the range 1 - 56. The value can be larger on input. However, on output, this field is updated to indicate the actual number of bytes returned by the service.

clear_dtrack1_data

Direction: Output
Type: String
This parameter returns the deciphered discretionary track 1 data.

**clear_dtrack2_data_length**

- **Direction:** Input/Output
- **Type:** Integer

Specifies the length of the `clear_dtrack2_data` parameter in bytes. The output value is either 0 or in the range 1 - 19. The value can be larger on input. However, on output, this field is updated to indicate the actual number of bytes returned by the service.

**clear_dtrack2_data**

- **Direction:** Output
- **Type:** String

This parameter returns the deciphered discretionary track 2 data.

**DUKPT_PIN_key_identifier_length**

- **Direction:** Input/Output
- **Type:** Integer

Specifies the length of the `DUKPT_PIN_key_identifier` parameter in bytes. If the PINKEY rule-array keyword is specified, set this value to 64. Otherwise, set this value to 0. On output, the variable is updated with the length of the data returned in the `DUKPT_PIN_key_identifier` variable.

**DUKPT_PIN_key_identifier**

- **Direction:** Input/Output
- **Type:** String

On input, this parameter must contain a DES OPINENC or IPINENC skeleton token. On output, it contains the DES token with the derived DES OPINENC or IPINENC key.

**reserved1_length**

- **Direction:** Input
- **Type:** Integer

Specifies the length of the `reserved1` parameter in bytes. The value must be 0.

**reserved1**

- **Direction:** Input
- **Type:** String

This parameter is ignored.

**reserved2_length**

- **Direction:** Input/Output
- **Type:** Integer

Specifies the length of the `reserved2` parameter in bytes. The value must be 0.

**reserved2**

- **Direction:** Input
- **Type:** String

This parameter is ignored.
Restrictions
The restrictions for CSNBFPED.

None.

Required commands
The required commands for CSNBFPED.

This verb requires the FPE Decrypt command (offset X'02D0') to be enabled in the active role. This ACP is ON by default in z/OS. If DUKPT is specified, the verb also requires the UKPT - PIN Verify, PIN Translate command (offset X'00E1') to be enabled if you employ DUKPT processing.

Usage notes
The usage notes for CSNBFPED.

See “Usage notes for FPE Encipher (CSNBFPEE) and FPE Decipher (CSNBFPED) services” on page 508.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNBFPEDJ.

See “Building Java applications using the CCA JNI” on page 28.

Format
public native void CSNBFPEDJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber enc_PAN_length,
    byte[] enc_PAN,
    hikmNativeNumber enc_cardholder_name_length,
    byte[] enc_cardholder_name,
    hikmNativeNumber enc_dtrack1_data_length,
    byte[] enc_dtrack1_data,
    hikmNativeNumber enc_dtrack2_data_length,
    byte[] enc_dtrack2_data,
    hikmNativeNumber key_identifier_length,
    byte[] key_identifier,
    hikmNativeNumber derivation_data_length,
    byte[] derivation_data,
    hikmNativeNumber clear_PAN_length,
    byte[] clear_PAN,
    hikmNativeNumber clear_cardholder_name_length,
    byte[] clear_cardholder_name,
    hikmNativeNumber clear_dtrack1_data_length,
    byte[] clear_dtrack1_data,
    hikmNativeNumber clear_dtrack2_data_length,
    byte[] clear_dtrack2_data,
    hikmNativeNumber DUKPT_PIN_key_identifier_length,
    byte[] DUKPT_PIN_key_identifier,
    hikmNativeNumber reserved1_length,
    byte[] reserved1,
    hikmNativeNumber reserved2_length,
    byte[] reserved2);
The FPE Encipher verb is used to encrypt payment card data for the Visa Data Secure Platform (VDSP) processing.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.

Format

The format of CSNBFPEE.

```
CSNBFPEE(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  clear_PAN_length,
  clear_PAN,
  clear_cardholder_name_length,
  clear_cardholder_name,
  clear_dtrack1_data_length,
  clear_dtrack1_data,
  clear_dtrack2_data_length,
  clear_dtrack2_data,
  key_identifier_length,
  key_identifier,
  derivation_data_length,
  derivation_data,
  enc_PAN_length,
  enc_PAN,
  enc_cardholder_name_length,
  enc_cardholder_name,
  enc_dtrack1_data_length,
  enc_dtrack1_data,
  enc_dtrack2_data_length,
  enc_dtrack2_data,
  DUKPT_PIN_key_identifier_length,
  DUKPT_PIN_key_identifier,
  reserved1_length,
  reserved1,
  reserved2_length,
  reserved2)
```

Parameters

The parameters for CSNBFPEE.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

`rule_array_count`

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. The minimum value is 5. The maximum value is 10.

`rule_array`

- **Direction:** Input
Keywords that provide control information to the verb. The rule_array keywords are described in Table 175.

**Note:** At least one character set keyword is required.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processing method</strong> (required)</td>
<td></td>
</tr>
<tr>
<td>VMDS</td>
<td>Specifies that the VDSP method (Visa Data Secure Platform method, formally known as the Visa Merchant Data Secure (VMDS) method) is to be used for processing.</td>
</tr>
<tr>
<td><strong>Key management method</strong> (one required)</td>
<td></td>
</tr>
<tr>
<td>STATIC</td>
<td>Specifies the use of double length (2-key) triple-DES symmetric keys. This is a non-DUKPT key.</td>
</tr>
<tr>
<td>DUKPT</td>
<td>Specifies the use of the transaction unique general purpose Data Encryption Keys generated by the DUKPT process at the point of service for data encryption. This is required if VFPE mode is specified. Otherwise, this is optional.</td>
</tr>
<tr>
<td><strong>Algorithm</strong> (required)</td>
<td></td>
</tr>
<tr>
<td>TDES</td>
<td>Specifies the use of CBC mode triple-DES encryption.</td>
</tr>
<tr>
<td><strong>Mode</strong> (one required)</td>
<td></td>
</tr>
<tr>
<td>CBC</td>
<td>Specifies the use of CBC mode. This is the mode for the standard encryption option.</td>
</tr>
<tr>
<td>VFPE</td>
<td>Specifies the use of Visa format preserving encryption.</td>
</tr>
<tr>
<td><strong>PAN input output character set</strong> (one required if the clear_PAN_length variable is greater than 0. Otherwise, it is not allowed.)</td>
<td></td>
</tr>
<tr>
<td>PAN8BITA</td>
<td>Specifies that the PAN data character set is BASE-10 ASCII represented in binary form. Valid ASCII values are in the range 0 - 9 (X'30' - X'39').</td>
</tr>
<tr>
<td>PAN4BITX</td>
<td>Specifies that the PAN data character set is BASE-10 4-bit hex with two digits per byte. Valid 4-bit hexadecimal values are in the range X'0' - X'9'.</td>
</tr>
<tr>
<td><strong>Cardholder name input output character set</strong> (required if the clear_cardholder_name_length variable is greater than 0. Otherwise, it is not allowed.)</td>
<td></td>
</tr>
<tr>
<td>CN8BITA</td>
<td>Specifies that the cardholder name character set is ASCII represented in binary format, one character per byte. See Table 148 on page 504 for valid characters.</td>
</tr>
<tr>
<td><strong>Track_1 input output character set</strong> (required if the clear_dtrack1_data_length variable is greater than 0. Otherwise, it is not valid.)</td>
<td></td>
</tr>
<tr>
<td>TK18BITA</td>
<td>Specifies that the track 1 discretionary data character set is ASCII represented in binary format, one character per byte. See Table 148 on page 504 for valid characters.</td>
</tr>
<tr>
<td><strong>Track_2 input output character set</strong> (required if the clear_dtrack2_data_length variable is greater than 0. Otherwise, it is not valid.)</td>
<td></td>
</tr>
<tr>
<td>TK28BITA</td>
<td>Specifies that the track 2 discretionary data character set is ASCII represented in binary format, one character per byte. Valid ASCII values are in the range 0 - 9 (X'30' - X'39') and A - F (X'41' - X'46').</td>
</tr>
<tr>
<td><strong>PIN encryption key output selection</strong> (one, optional)</td>
<td></td>
</tr>
<tr>
<td>NOPINKEY</td>
<td>Do not return a DUKPT PIN encryption key. This is the default.</td>
</tr>
<tr>
<td>PINKEY</td>
<td>Return a DUKPT PIN encryption key.</td>
</tr>
<tr>
<td><strong>PAN check digit compliance</strong> (one required if mode VFPE and the pan input output character set keyword is present. Otherwise, it is not allowed.)</td>
<td></td>
</tr>
<tr>
<td>CMPCKDGT</td>
<td>Last digit of the PAN contains a compliant check digit per ISO/IEC 7812-1.</td>
</tr>
<tr>
<td>NONCKDGT</td>
<td>Last digit of the PAN does not contain a compliant check digit per ISO/IEC 7812-1.</td>
</tr>
</tbody>
</table>
FPE Encipher (CSNBFPEE)

**clear_PAN_length**

Direction: Input  
Type: Integer

Specifies the number of digits in the **clear_PAN** parameter. This value must be 0 if **clear_PAN** is not to be enciphered. The value must be in the range 15 - 19 if PAN data is presented for encryption.

**clear_PAN**

Direction: Input  
Type: String

Contains the account number with which the PIN is associated. The full account number, including check digit, should be included. The data for this parameter is in binary format. It is the binary representation of 4-bit hex (keyword PAN4BITX) or ASCII (keyword PAN8BITA). If the PAN contains an odd number of 4-bit digits, the data must be left justified in the PAN variable and the right-most 4 bits are ignored.

If the **clear_PAN_length** parameter is zero, this parameter is ignored.

**clear_cardholder_name_length**

Direction: Input  
Type: Integer

Specifies the length of the **clear_cardholder_name** parameter in bytes. This value must be 0 if the cardholder name is not to be enciphered. The value must be in the range 2 - 32 if the cardholder name data is presented for encryption.

**clear_cardholder_name**

Direction: Input  
Type: String

Contains the card holder's full name. The data for this parameter is in binary format. It is the binary representation of ASCII characters as defined in **Table 150 on page 505**. Only characters from this table are valid.

**clear_dtrack1_data_length**

Direction: Input  
Type: Integer

Specifies the length of the **clear_dtrack1_data** parameter in bytes. This value must be 0 if the track 1 discretionary data is not to be enciphered. The value must be in the range 1 - 56 if the track 1 discretionary data is presented for encryption.

**clear_dtrack1_data**

Direction: Input  
Type: String

Contains the discretionary data that is stored on track 1 of a magnetic stripe card. This data does not include the PAN, cardholder name, expiration date, or service code. The data for this parameter is in binary format. It is the binary representation of ASCII characters as defined in **Table 150 on page 505**. Only characters defined in this table are valid.

**clear_dtrack2_data_length**
Direction: Input
Type: Integer

Specifies the length of the clear_dtrack2_data parameter in bytes. This value must be 0 if the track 2 discretionary data is not to be enciphered. The value must be in the range 1 - 19 if the track 2 discretionary data is presented for encryption.

clear_dtrack2_data

Direction: Input
Type: String

Contains the discretionary data that is stored on track 2 of a magnetic stripe card. This data does not include the PAN, expiration date, or service code. The data for this parameter is in binary format. It is the binary representation of ASCII characters. The data for this parameter is in BASE-16 binary format. It is the binary representation of ASCII in the range X'30' - X'39' and X'41' - X'46' (ASCII: 0 - 9 and A - F).

key_identifier_length

Direction: Input
Type: Integer

Specifies the length of the key_identifier parameter in bytes. The value must be 64.

key_identifier

Direction: Input/Output
Type: String

The identifier of the key that is used to either encrypt the card data (key management STATIC) or derive the DUKPT_PIN_key_identifier (key management DUKPT). The key identifier is an operational token or the key label of an operational token in key storage.

For key management DUKPT, the key type must be KEYGENKY. In addition, it must have a control vector with bit 18 equal to B'1' (UKPT). The base derivation key is the one from which the operational keys are derived using the DUKPT algorithm defined in ANS X9.24 Part 1.

For key management STATIC, (Zone Encryption Key in the VDSP specification), the key type must be either CIPHER or ENCIPHER. For production purposes, it is recommended that the key have left and right halves that are not equal.

Note: Data keys are not supported.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

derivation_data_length

Direction: Input
Type: Integer

Specifies the length of the derivation_data parameter in bytes. The value must be 10 if a DUKPT key is specified in the key_identifier parameter. If a data encryption key is specified in the key_identifier parameter, this value must be set to zero.
derivation_data

Direction: Input
Type: String

Contains the 80 bit (10 byte) derivation data that is used as input to the DUKPT derivation process. The derivation data contains the current key serial number (CKSN), which is composed of the 59 bit initial key serial number concatenated with the 21 bit value of the current encryption counter, which the device increments for each new transaction. This field is in binary format.

derivation_data

enc_PAN_length

Direction: Input
Type: Integer

Specifies the number of PAN digits in the enc_PAN parameter. This value must either be 0 or in the range 15 - 19, inclusive on output.

dec_PAN

Direction: Output
Type: String

This parameter returns the enciphered primary account number. For VFPE mode, if the PAN contains an odd number of 4-bit digits, the data is left-justified in the PAN variable and the right-most 4 bits can be ignored.

dec_PAN_length

Direction: Input/Output
Type: Integer

Specifies the length of the dec_PAN parameter in bytes. This output value is either 0 or in the range 2 - 32 for VFPE. For CBC mode, the output value is in the range 16 - 40 and a multiple of 8 if the service is successful and cardholder name data is enciphered. The parameter can be larger on input. However, on output, this length is updated to indicate the actual number of bytes returned by the service.

dec_cardholder_name_length

Direction: Input/Output
Type: Integer

The field where the enciphered cardholder full name is returned.

dec_cardholder_name

Direction: Output
Type: String

FPE Encipher (CSNBFPEE)
This parameter returns the enciphered discretionary track 1 data.

**enc_dtrack2_data_length**

- **Direction:** Input/Output
- **Type:** Integer

Specifies the length of the `enc_dtrack2_data` parameter in bytes. The output value is either 0 or in the range 1 - 19 for mode VFPE. For mode CBC, the output value is 8 or 16 if the service is successful and the data is enciphered. The parameter can be larger on input. However, on output, this length is updated to indicate the actual number of bytes returned by the service.

**enc_dtrack2_data**

- **Direction:** Output
- **Type:** String

This parameter returns the enciphered discretionary track 2 data.

**DUKPT_PIN_key_identifier_length**

- **Direction:** Input/Output
- **Type:** Integer

Specifies the length of the `DUKPT_PIN_key_identifier` parameter in bytes. If the PINKEY rule-array keyword is specified, set this value to 64. Otherwise, set this value to 0. On output, the variable is updated with the length of the data returned in the `DUKPT_PIN_key_identifier` variable.

**DUKPT_PIN_key_identifier**

- **Direction:** Input/Output
- **Type:** String

On input, this parameter must contain a DES OPINENC or IPINENC skeleton token. On output, it contains the DES token with the derived DES OPINENC or IPINENC key.

**reserved1_length**

- **Direction:** Input
- **Type:** Integer

Specifies the length of the `reserved1` parameter in bytes. The value must be 0.

**reserved1**

- **Direction:** Input
- **Type:** String

This parameter is ignored.

**reserved2_length**

- **Direction:** Input
- **Type:** Integer

Length of the `reserved2` parameter in bytes. The value must be 0.

**reserved2**

- **Direction:** Input
- **Type:** String
FPE Encipher (CSNBFPEE)

This parameter is ignored.

**Restrictions**

The restrictions for CSNBFPEE.

None.

**Required commands**

The required commands for CSNBFPEE.

This verb requires the FPE Encrypt command (offset X'02CF') to be enabled in the active role. This ACP is ON by default in z/OS. If DUKPT is specified, the verb also requires the UKPT - PIN Verify, PIN Translate command (offset X'00E1') to be enabled if you employ DUKPT processing.

**Usage notes**

The usage notes for CSNBFPEE.

See “Usage notes for FPE Encipher (CSNBFPEE) and FPE Decipher (CSNBFPED) services” on page 508.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBFPEEJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBFPEEJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber clear_PAN_length,
    byte[] clear_PAN,
    hikmNativeNumber clear_cardholder_name_length,
    byte[] clear_cardholder_name,
    hikmNativeNumber clear_dtrack1_data_length,
    byte[] clear_dtrack1_data,
    hikmNativeNumber clear_dtrack2_data_length,
    byte[] clear_dtrack2_data,
    hikmNativeNumber key_identifier_length,
    byte[] key_identifier,
    hikmNativeNumber derivation_data_length,
    byte[] derivation_data,
    hikmNativeNumber enc_PAN_length,
    byte[] enc_PAN,
    hikmNativeNumber enc_cardholder_name_length,
    byte[] enc_cardholder_name,
    hikmNativeNumber enc_dtrack1_data_length,
    byte[] enc_dtrack1_data,
    hikmNativeNumber enc_dtrack2_data_length,
    byte[] enc_dtrack2_data,
    hikmNativeNumber DUKPT_PIN_key_identifier_length,
    byte[] DUKPT_PIN_key_identifier,
```
The FPE Translate verb is used to translate payment data from encryption under one key to encryption under another key with a possibly different format.

You should avoid having plaintext payment data in your environment. Translations can be performed with data that has been encrypted using the standard encryption option or with data that has been encrypted using the VFPE option. However, the target translation uses double length static TDES keys and the standard encryption option.

This service can be used to translate one or all of the following fields: the primary account number (PAN), the cardholder name, the track 1 discretionary data, or the track 2 discretionary data.

The following translation options are supported:
1. Translate standard option with CBC mode TDES and DUKPT keys.
2. Translate VFPE option with VFPE mode TDES and DUKPT keys.
3. Translate standard option with CBC mode TDES and static TDES keys.

To use this service, you must specify the following:
- the processing method, which is limited to Visa Data Secure Platform (VDSP)
- the key management method, either STATIC or DUKPT
- the algorithm, which is limited to TDES
- the mode, either CBC or Visa Format Preserving Encryption (VFPE) for the inbound data
- the ciphertext to be translated
- the character set of each field to be translated using rule-array keywords
- the base derivation key and key serial number if DUKPT key management is used, or a double-length TDES key if standard key management is used to recover the plaintext
- the double length static TDES key used to re-encrypt the data
- Optionally, a check digit compliance indicator if VFPE is specified.

The service returns the translated fields and optionally, the DUKPT PIN encryption key, if the DUKPT key management is selected and the PINKEY rule is specified.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.
FPE Translate (CSNBFPET)

Format

The format of CSNBFPET.

```
CSNBFPET(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    input_PAN_length,
    input_PAN,
    input_cardholder_name_length,
    input_cardholder_name,
    input_dtrack1_data_length,
    input_dtrack1_data,
    input_dtrack2_data_length,
    input_dtrack2_data,
    input_key_identifier_length,
    input_key_identifier,
    output_key_identifier_length,
    output_key_identifier,
    derivation_data_length,
    derivation_data,
    output_PAN_length,
    output_PAN,
    output_cardholder_name_length,
    output_cardholder_name,
    output_dtrack1_data_length,
    output_dtrack1_data,
    output_dtrack2_data_length,
    output_dtrack2_data,
    DUKPT_PIN_key_identifier_length,
    DUKPT_PIN_key_identifier,
    reserved1_length,
    reserved1,
    reserved2_length,
    reserved2)
```

Parameters

The parameters for CSNBFPET.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. The minimum value is 4. The maximum value is 10.

**rule_array**

- **Direction:** Input
- **Type:** String array

Keywords that provide control information to the verb. The `rule_array` keywords are described in Table 176 on page 577

**Note:** At least one character set keyword is required.
### Table 176. Keywords for FPE Translate control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processing method</strong> (required)</td>
<td></td>
</tr>
<tr>
<td>VMDS</td>
<td>Specifies that the VDSP method (Visa Data Secure Platform method, formally known as the Visa Merchant Data Secure (VMDS) method) is to be used for processing.</td>
</tr>
<tr>
<td><strong>Key management method</strong> (one required)</td>
<td></td>
</tr>
<tr>
<td>STATIC</td>
<td>Specifies the use of double length (2-key) triple-DES symmetric keys. This is a non-DUKPT key.</td>
</tr>
<tr>
<td>DUKPT</td>
<td>Specifies the use of the transaction unique general purpose Data Encryption Keys generated by the DUKPT process at the point of service for data encryption. This is required if VFPE mode is specified. Otherwise, this is optional.</td>
</tr>
<tr>
<td><strong>Algorithm</strong> (required)</td>
<td></td>
</tr>
<tr>
<td>TDES</td>
<td>Specifies the use of CBC mode triple-DES encryption.</td>
</tr>
<tr>
<td><strong>Mode</strong> (one required)</td>
<td></td>
</tr>
<tr>
<td>CBC</td>
<td>Specifies the use of CBC mode. This is the mode for the standard encryption option.</td>
</tr>
<tr>
<td>VFPE</td>
<td>Specifies the use of Visa format preserving encryption.</td>
</tr>
<tr>
<td><strong>PAN input output character set</strong> (one required if the clear_PAN_length variable is greater than 0. Otherwise, it is not allowed.)</td>
<td></td>
</tr>
<tr>
<td>PAN8BITA</td>
<td>Specifies that the PAN data character set is ASCII represented in binary form. Valid only for VFPE mode.</td>
</tr>
<tr>
<td>PAN4BITX</td>
<td>Specifies that the PAN data character set is 4-bit hex with two digits per byte. Valid only for VFPE mode.</td>
</tr>
<tr>
<td>PAN-EBLK</td>
<td>Specifies that the PAN data is in a CBC encrypted block. Valid only for CBC mode.</td>
</tr>
<tr>
<td><strong>Cardholder name input output character set</strong> (required if the clear_cardholder_name_length variable is greater than 0.)</td>
<td></td>
</tr>
<tr>
<td>CN8BITA</td>
<td>Specifies that the cardholder name character set is ASCII represented in binary format, one character per byte. See Table 148 on page 504 for valid characters.</td>
</tr>
<tr>
<td>CN-EBLK</td>
<td>Specifies that the cardholder name data is in a CBC-encrypted block.</td>
</tr>
<tr>
<td><strong>Track_1 input output character set</strong> (required if the clear_dtrack1_data_length variable is greater than 0. Otherwise, it is not valid.)</td>
<td></td>
</tr>
<tr>
<td>TK18BITA</td>
<td>Specifies that the track 1 discretionary data character set is ASCII represented in binary format, one character per byte. See Table 148 on page 504 for valid characters.</td>
</tr>
<tr>
<td>TK1-EBLK</td>
<td>Specifies that the track 1 discretionary data is in a CBC-encrypted block. Valid only for CBC mode.</td>
</tr>
<tr>
<td><strong>Track_2 input output character set</strong> (required if the clear_dtrack2_data_length variable is greater than 0. Otherwise, it is not valid.)</td>
<td></td>
</tr>
<tr>
<td>TK28BITA</td>
<td>Specifies that the track 2 discretionary data character set is ASCII represented in binary format. Valid only for VFPE mode.</td>
</tr>
<tr>
<td>TK2-EBLK</td>
<td>Specifies that the track 2 discretionary data is in a CBC encrypted block. Valid only for CBC mode.</td>
</tr>
<tr>
<td><strong>PIN encryption key output selection</strong> (one, optional, if DUKPT is specified. Otherwise, it is not valid.)</td>
<td></td>
</tr>
<tr>
<td>NOPINKEY</td>
<td>Do not return a DUKPT PIN encryption key. This is the default.</td>
</tr>
<tr>
<td>PINKEY</td>
<td>Return a DUKPT PIN encryption key.</td>
</tr>
<tr>
<td><strong>PAN check digit compliance</strong> (one required if mode VFPE and the PAN input character set keyword is present. Otherwise, it is not allowed.)</td>
<td></td>
</tr>
<tr>
<td>CMPCKDGT</td>
<td>Last digit of the PAN contains a compliant check digit per ISO/IEC 7812-1.</td>
</tr>
<tr>
<td>NONCKDGT</td>
<td>Last digit of the PAN does not contain a compliant check digit per ISO/IEC 7812-1.</td>
</tr>
</tbody>
</table>

_Please note: The table is a representation of the keywords used in FPE Translate control information._
FPE Translate (CSNBFPET)

**input_PAN**

**Direction:** Input  
**Type:** Integer

Specifies the length of the input_PAN parameter in bytes if the mode is CBC. Specifies the number of PAN digits if the mode is VFPE. The value is 0 if PAN data has not been presented for translation. Otherwise, the value is in the range 15 - 19 for VFPE. The value must be 16 if the standard option with CBC mode is selected.

**input_cardholder_name_length**

**Direction:** Input  
**Type:** Integer

Specifies the length of the input_cardholder_name parameter in bytes. This value must be 0 if cardholder name is not presented for translation. Otherwise, the value is in the range 1 - 32 for VFPE. For CBC mode, the input value is either 16, 24, 32, or 40.

**input_cardholder_name**

**Direction:** Input  
**Type:** String

The enciphered cardholder full name that is to be translated. Only characters in Table 150 on page 505 are valid.

**input_dtrack1_data_length**

**Direction:** Input  
**Type:** Integer

Specifies the length of the input_dtrack1_data parameter in bytes. This value must be 0 if track 1 discretionary data is not presented for translation. Otherwise, the value is in the range 1 - 56 for VFPE. For CBC mode, the input value is either 16, 24, 32, 40, 48, 56, or 64.

**input_dtrack1_data**

**Direction:** Input  
**Type:** String

The encrypted track 1 data that is to be translated. Only characters in Table 150 on page 505 are valid.

**input_dtrack2_data_length**

**Direction:** Input  
**Type:** Integer

Specifies the length of the input_dtrack2_data parameter in bytes. This value must be 0 if track 2 discretionary data is not presented for translation. Otherwise, the value is in the range 1 - 19 for VFPE. For CBC mode, the input value is either 8 or 16.
input_dtrack2_data

Direction: Input
Type: String

The encrypted track 2 data that is to be translated.

input_key_identifier_length

Direction: Input
Type: Integer

Specifies the length of the input_key_identifier parameter in bytes. The value must be 64, because only fixed length DES tokens are supported as the key identifier.

input_key_identifier

Direction: Input/Output
Type: String

The identifier of the key that is used to either decrypt the card data (key management STATIC) or derive the DUKPT_PIN_key_identifier (key management DUKPT). The key identifier is an operational token or the key label of an operational token in key storage.

For key management DUKPT, the key type must be KEYGENKY. In addition, it must have a control vector with bit 18 equal to B’1’ (UKPT). The base derivation key is the one from which the operational keys are derived using the DUKPT algorithm defined in ANSI X9.24 Part 1.

For key management STATIC, (Zone Encryption Key in the VDSP specification), the key type must be either CIPHER or ENCIPHER. For production purposes, it is recommended that the key have left and right halves that are not equal.

Note: Data keys are not supported.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

output_key_identifier_length

Direction: Input
Type: Integer

Specifies the length of the output_key_identifier parameter in bytes. The value must be 64, because only fixed length DES tokens are supported as the key identifier.

output_key_identifier

Direction: Input/Output
Type: String

The identifier of the key that is used to decrypt the output card data. The key identifier is an operational token or the key label of an operational token in key storage.

The key type must be either CIPHER or ENCIPHER. For production purposes, it is recommended that the key have left and right halves that are not equal.

Note: Data keys are not supported.
If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**derivation_data_length**
- **Direction:** Input
- **Type:** Integer

Specifies the length of the derivation_data parameter in bytes. The value must be 10 if a DUKPT key is specified in the key_identifier parameter. If a data encryption key is specified in the key_identifier parameter, this value must be set to zero.

**derivation_data**
- **Direction:** Input
- **Type:** String

Contains the 80 bit (10 byte) derivation data that is used as input to the DUKPT derivation process. The derivation data contains the current key serial number (CKSN), which is composed of the 59 bit initial key serial number concatenated with the 21 bit value of the current encryption counter, which the device increments for each new transaction. This field is in binary format.

**output_PAN_length**
- **Direction:** Input/Output
- **Type:** Integer

Specifies the number of bytes of data in the output_PAN parameter. This value is 0 or 16 on output.

**output_PAN**
- **Direction:** Output
- **Type:** String

This parameter returns the translated primary account number with which the PIN is associated. The full account number, including check digit, is translated. The data for this parameter is returned as TDES-encrypted data in binary format. The 16 byte output is left justified in this field.

**output_cardholder_name_length**
- **Direction:** Input/Output
- **Type:** Integer

 Specifies the length of the output_cardholder_name parameter in bytes. This output value is either 0 or 16, 24, 32, or 40 bytes on output. The variable can be larger on input. However, on output, this field is updated to indicate the actual number of bytes returned by the card.

**output_cardholder_name**
- **Direction:** Output
- **Type:** String

This parameter returns the translated cardholder full name. The data for this parameter is returned as TDES-encrypted data in binary format.

**output_dtrack1_data_length**
- **Direction:** Input/Output
- **Type:** Integer
Specifies the length of the output_dtrack1_data parameter in bytes. The output value is either 0 or 16, 24, 32, 40, 48, 56, or 64 bytes. The value can be larger on input. However, on output, this field is updated to indicate the actual number of bytes returned by the service.

**output_dtrack1_data**

Direction: Output  
Type: String

This parameter returns the translated discretionary track 1 data. This is the discretionary data from track 1 of a magnetic stripe card. The data for this parameter is returned as TDES-encrypted data in binary format.

**output_dtrack2_data_length**

Direction: Input/Output  
Type: Integer

Specifies the length of the output_dtrack2_data parameter in bytes. The output value is either 0, 8, or 16. The value can be larger on input. However, on output, this field is updated to indicate the actual number of bytes returned by the service.

**output_dtrack2_data**

Direction: Output  
Type: String

This parameter returns the translated discretionary track 2 data. This is the discretionary data from track 2 of a magnetic stripe card. The data for this parameter is returned as TDES-encrypted data in binary format.

**DUKPT_PIN_key_identifier_length**

Direction: Input/Output  
Type: Integer

Specifies the length of the DUKPT_PIN_key_identifier parameter in bytes. If the PINKEY rule-array keyword is specified, set this value to 64. Otherwise, set this value to 0. On output, the variable is updated with the length of the data returned in the DUKPT_PIN_key_identifier variable.

**DUKPT_PIN_key_identifier**

Direction: Input/Output  
Type: String

On input, this parameter must contain a DES OPINENC or IPINENC skeleton token. On output, it contains the DES token with the derived DES OPINENC or IPINENC key.

**reserved1_length**

Direction: Input  
Type: Integer

Specifies the length of the reserved1 parameter in bytes. The value must be 0.

**reserved1**

Direction: Input  
Type: String
FPE Translate (CSNBFPET)

This parameter is ignored.

reserved2_length

Direction:       Input
Type:           Integer

Specifies the length of the reserved2 parameter in bytes. The value must be 0.

reserved2

Direction:       Input
Type:           String

This parameter is ignored.

Restrictions

The restrictions for CSNBFPET.

None.

Required commands

The required commands for CSNBFPET.

This verb requires the FPE Translate command (offset X'02D1') to be enabled in the active role. This ACP is ON by default in z/OS. If DUKPT is specified, the verb also requires the UKPT - PIN Verify, PIN Translate command (offset X'00E1') to be enabled if you employ DUKPT processing.

Usage notes

The usage notes for CSNBFPET.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBFPETJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBFPETJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber input_PAN_length,
    byte[] input_PAN,
    hikmNativeNumber input_cardholder_name_length,
    byte[] input_cardholder_name,
    hikmNativeNumber input_dtrack1_data_length,
    byte[] input_dtrack1_data,
    hikmNativeNumber input_dtrack2_data_length,
    byte[] input_dtrack2_data,
    hikmNativeNumber input_key_identifier_length,
    byte[] input_key_identifier,
    hikmNativeNumber output_key_identifier_length,
    byte[] output_key_identifier
)
```
The PIN Change/Unblock verb is used to generate a special PIN block to change the PIN accepted by an integrated circuit card (smartcard).

The PIN Change/Unblock verb prepares an encrypted message-portion for communicating an original or replacement PIN for an EMV smart card. The verb embeds the PINs in an encrypted PIN-block from information that you supply. You incorporate the information created with the verb in a message sent to the smart card.

The processing is consistent with the specifications provided in these documents:
- EMV 2000 Integrated Circuit Card Specifications for Payment Systems Version 4.0 (EMV4.0)
- Visa Integrated Circuit Card Specification Manual, Version 1.4.0
- Integrated Circuit Card Specification (VIS) 1.4.0 Corrections

You specify the following information:
- Through the optional choice of one rule-array keyword, the key-diversification process to employ in deriving the session key used to encrypt the PIN block. See “Working with Europay-Mastercard-Visa Smart cards” on page 492 for processing details.

- **TDES-XOR**
  An exclusive-OR process described in “Deriving the CCA TDES-XOR session key” on page 585. It is the default.

- **TDESEMV2**
  The tree-based-diversification process with a branch factor of 2.

- **TDESEMV4**
  The tree-based-diversification process with a branch factor of 4.

- Through the required choice of one rule-array keyword, if you are providing a PIN for a smart card:

- **AMEXPCU1**
  For a card with a current PIN, provide the existing PIN in an encrypted PIN-block in the current_reference_PIN_block variable, and supply the new PIN-value in an encrypted PIN-block in the new_reference_PIN_block variable.
PIN Change/Unblock (CSNBPCU)

AMEXPCU2
For a card without a PIN, provide the new PIN in an encrypted PIN-block in the new_reference_PIN_block variable. The contents of the five current_reference_PIN_x variables are ignored.

VISAPCU1
For a card without a PIN, provide the new PIN in an encrypted PIN-block in the new_reference_PIN_block variable. The contents of the five current_reference_PIN_x variables are ignored.

VISAPCU2
For a card with a current PIN, provide the existing PIN in an encrypted PIN-block in the current_reference_PIN_block variable, and supply the new PIN-value in an encrypted PIN-block in the new_reference_PIN_block variable.

• Issuer-provided master-derivation keys (MDK). The card-issuer provides two keys for diversifying the same data:
  – The MAC-MDK key that you incorporate in the variable specified by the authentication_key_identifier parameter. The verb uses this key to derive an authentication value incorporated in the PIN block. The control vector for the MAC-MDK key must specify a DKYGENKY key type with DKYL0 (level-0), and DMAC or DALL permissions.
  – The ENC-MDK key that you incorporate in the variable specified by the encryption_key_identifier parameter. The verb uses this key to derive the PIN-block encryption key. The control vector for the ENC-MDK key must specify a DKYGENKY key type with DKYL0 (level-0), and DMPIN or DALL permissions.

• The diversification_data_length to indicate the sum of the lengths of:
  – Data, 8 or 16 bytes, encrypted by the verb using the MDK keys to generate the ENC-UDK and Unique DES Key.
  – The 2-byte application transaction counter (ATC). You receive the ATC value from the EMV smart card.
  – The optional 16-byte initial value used in the TDESEMVn processes. Valid lengths are 10, 18, 26, and 34 bytes.

• The diversification_data variable. Concatenate the 8-byte or 16-byte data, the ATC, and optionally the Initial Value.
  The 16-bit ATC counter is processed as a two-byte string, not as an integer value.

• The new-reference PIN in an encrypted PIN block. You provide:
  – the key to decrypt the PIN block
  – the PIN block
  – the format information that defines how to parse the PIN block
  – when using an ISO-0 or ISO-3 PIN-block format, primary account number (PAN) information to enable PIN recovery from the ISO-0 or ISO-3 PIN-block format.

• If you specified VISAPCU2 (because the target smart card already has a PIN), the current_reference_PIN in an encrypted PIN block with the associated decrypting key, PIN-block format, and PAN data. In any case, you must declare the five current_reference_PIN_x variables.

• The output_PIN_message variable to receive the encrypted PIN block for the smart card, and the length in bytes of the PIN block (16). The PIN-block format you specify (VISAPCU1 or VISAPCU2) corresponds to the one or two PIN values to be communicated to the smart card.
Two variables which are reserved for future use: output_PIN_data_length (valued to zero), and an output_PIN_data string variable (or set the associated parameter to a null pointer).

The PIN Change/Unblock verb:
- Decrypts the MDK keys and verifies the required control vector permissions.
- Diversifies the left-most eight bytes of data using the MAC-MDK key to obtain the authentication value for placement into the PIN block.
- Recovers the supplied PIN values provided that PIN-block encrypting keys are one of IPINENC or OPINENC type, and the use of the specific type is authorized with the appropriate access-control command.
- Constructs and pads the output PIN block to a 16-byte string.
- Generates the session key used to encrypt the output-PIN block using the ENC-MDK, the key_generation_data, the ATC counter value, and the optional Initial Value.
- Triple encrypts the 16-byte padded PIN-block in ECB mode.
- Returns the encrypted, padded PIN-block in the output_PIN_message variable.

The generating DKYGENKY cannot have replicated halves. The encryption_master_key is a DKYGENKY that permits generation of a SMPIN key. The authentication_master_key is also a DKYGENKY that permits generation of a double length MAC key.

**Deriving the CCA TDES-XOR session key**

In the Diversified_Key_Generate and PIN_Change/Unblock verbs, the TDES-XOR process first derives a smart-card-specific intermediate key from the issuer-supplied ENC-MDK key and card-specific data. (This intermediate key is also used in the TDESEMV2 and TDESEMV4 processes. See the next section.) The intermediate key is then modified using the application transaction counter (ATC) value supplied by the smart card.

The double-length session-key creation steps:

1. Obtain the left-half of an intermediate key by ECB-mode Triple-DES encrypting the (first) eight bytes of card specific data using the issuer-supplied ENC-MDK key.
2. Again using the ENC-MDK key, obtain the right-half of the intermediate key by ECB-mode Triple-DES encrypting with one of these methods:
   - The second 8 bytes of card-specific derivation data when 16 bytes have been supplied.
   - The exclusive-OR of the supplied 8 bytes of derivation data with X'FFFFFFFF FFFFFFFF'.
3. Pad the ATC value to the left with six bytes of X'00' and exclusive-OR the result with the left-half of the intermediate key to obtain the left-half of the session key.
4. Obtain the one’s complement of the ATC by exclusive-ORing the ATC with X'FFFF'. Pad the result on the left with six bytes of X'00'. Exclusive-OR the 8-byte result with the right-half of the intermediate key to obtain the right-half of the session key.

**Note:** This verb supports PCI-HSM 2016 compliant-tagged key tokens.
PIN Change/Unblock (CSNBPCU)

Format

The format of CSNBPCU.

```
CSNBPCU(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    authentication_master_key_length,
    authentication_master_key,
    encryption_master_key_length,
    encryption_master_key,
    key_generation_data_length,
    key_generation_data,
    new_reference_PIN_key_length,
    new_reference_PIN_key,
    new_reference_PIN_block,
    new_reference_PIN_profile,
    new_reference_PAN_data,
    current_reference_PIN_key_length,
    current_reference_PIN_key,
    current_reference_PIN_block,
    current_reference_PIN_profile,
    current_reference_PAN_data,
    output_PIN_data_length,
    output_PIN_data,
    output_PIN_profile,
    output_PIN_message_length,
    output_PIN_message)
```

Parameters

The parameters for CSNBPCU.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1 or 2.

**rule_array**

Direction: Input
Type: String array

Keywords that provide control information to the verb. The keywords are left-aligned in an 8-byte field and padded on the right with blanks. The keywords must be in contiguous storage. The `rule_array` keywords are described in Table 177.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm (One, optional)</td>
<td>Table 177. Keywords for PIN Change/Unblock control information</td>
</tr>
</tbody>
</table>

Table 177. Keywords for PIN Change/Unblock control information
Table 177. Keywords for PIN Change/Unblock control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDES-XOR</td>
<td>TDES encipher clear data to generate the intermediate (card-unique) key, followed by XOR of the final two bytes of each key with the ATC counter. This is the default.</td>
</tr>
<tr>
<td>TDESEMV2</td>
<td>Same processing as in the Diversified Key Generate verb.</td>
</tr>
<tr>
<td>TDESEMV4</td>
<td>Same processing as in the Diversified Key Generate verb.</td>
</tr>
<tr>
<td>PIN processing method (One, required)</td>
<td></td>
</tr>
<tr>
<td>AMEXPCU1</td>
<td>Form the new PIN from the new reference PIN, the smart-card-unique, intermediate key, and the current reference PIN.</td>
</tr>
<tr>
<td>AMEXPCU2</td>
<td>Form the new PIN from the new reference PIN and the smart-card-unique, intermediate key.</td>
</tr>
<tr>
<td>VISAPCU1</td>
<td>Form the new PIN from the new reference PIN and the intermediate (card-unique) key only.</td>
</tr>
<tr>
<td>VISAPCU2</td>
<td>Form the new PIN from the new reference PIN, the intermediate (card-unique) key and the current reference PIN.</td>
</tr>
</tbody>
</table>

**authentication_master_key_length**

- **Direction:** Input
- **Type:** Integer

  The length of the parameter. Currently, the value must be 64.

**authentication_master_key**

- **Direction:** Input/Output
- **Type:** String

  The label name or internal token of a **DKYGENKY** key type that is to be used to generate the card-unique diversified key. The control vector of this key must be a **DKYL0** key that permits the generation of a double-length MAC key (DMAC). This **DKYGENKY** might not have replicated key halves.

**encryption_master_key_length**

- **Direction:** Input
- **Type:** Integer

  The length of the **encryption_master_key** parameter. Currently, the value must be 64.

**encryption_master_key**

- **Direction:** Input/Output
- **Type:** String

  The label name or internal token of a **DKYGENKY** key type that is to be used to generate the card-unique diversified key and the secure messaging session key for the protection of the output PIN block. The control vector of this key must be a **DKYL0** key that permits the generation of a SMPIN key type. This **DKYGENKY** might not have replicated key halves.

**key_generation_data_length**

- **Direction:** Input
- **Type:** Integer

  The length of the **key_generation_data** parameter. This value must be 10, 18, 26, or 34 bytes.
PIN Change/Unblock (CSNBPCU)

**key_generation_data**

- **Direction:** Input
- **Type:** String

The data provided to generate the card-unique session key. For **TDES-XOR**, this consists of 8 or 16 bytes of data to be processed by TDES to generate the card-unique diversified key followed by a 16-bit ATC counter to offset the card-unique diversified key to form the session key. For **TDESEMv2** and **TDESEMv4**, this can be 10, 18, 26, or 34 bytes. See “Diversified Key Generate (CSNBDKG)” on page 183 for more information.

**new_reference_PIN_key_length**

- **Direction:** Input
- **Type:** Integer

The length of the **new_reference_PIN_key** parameter. Currently, the value must be 64.

**new_reference_PIN_key**

- **Direction:** Input/Output
- **Type:** String

The label name or internal token of a PIN encrypting key that is to be used to decrypt the **new_reference_PIN_block**. This must be an IPINENC or OPINENC key. If the label name is supplied, the name must be unique in the DES key storage file.

**new_reference_PIN_block**

- **Direction:** Input
- **Type:** String

This is an 8-byte field that contains the enciphered PIN block of the new PIN.

**new_reference_PIN_profile**

- **Direction:** Input
- **Type:** String

This is a 24-byte field that contains three 8-byte elements with a PIN block format keyword, a format control keyword (NONE), and a pad digit as required by certain formats.

**new_reference_PAN_data**

- **Direction:** Input
- **Type:** String

This is a 12-byte field containing the PAN in character format. This data might be needed to recover the new reference PIN if the format is ISO-0, ISO-3, or VISA-4. If neither is used, this parameter might be blanks.

For ISO-0 or ISO-3, use the rightmost 12 digits of the PAN, excluding the check digit. For VISA-4, use the leftmost 12 digits of the PAN, excluding the check digit.

**current_reference_PIN_key_length**

- **Direction:** Input
- **Type:** Integer
The length of the `current_reference_PIN_key` parameter. For the current implementation, the value must be 64. If the `rule_array` contains `VISAPCU1`, this value must be 0.

### current_reference_PIN_key

- **Direction:** Input/Output
- **Type:** String

The label name or internal token of a PIN encrypting key that is to be used to decrypt the `current_reference_PIN_block`. This must be an `IPINENC` or `OPINENC` key. If the label name is supplied, the name must be unique in the key storage. If the `rule_array` contains `VISAPCU1`, this value is ignored.

### current_reference_PIN_block

- **Direction:** Input
- **Type:** String

This is an 8-byte field that contains the enciphered PIN block of the new PIN. If the `rule_array` contains `VISAPCU1`, this value is ignored.

### current_reference_PIN_profile

- **Direction:** Input
- **Type:** String

This is a 24-byte field that contains three 8-byte elements with a PIN block format keyword, a format control keyword (NONE), and a pad digit as required by certain formats. If the `rule_array` contains `VISAPCU1`, this value is ignored.

### current_reference_PAN_data

- **Direction:** Input
- **Type:** String

This is a 12-byte field containing PAN in character format. This data might be needed to recover the new reference PIN if the format is `ISO-0`, `ISO-3`, or `VISA-4`. If neither is used, this parameter might be blanks. If the `rule_array` contains `VISAPCU1`, this value is ignored.

For `ISO-0` or `ISO-3`, use the rightmost 12 digits of the PAN, excluding the check digit. For `VISA-4`, use the leftmost 12 digits of the PAN, excluding the check digit.

### output_PIN_data_length

- **Direction:** Input
- **Type:** Integer

Currently this field is reserved. This value must be 0.

### output_PIN_data

- **Direction:** Input
- **Type:** String

This parameter is ignored.

### output_PIN_profile

- **Direction:** Input
- **Type:** String
PIN Change/Unblock (CSNBPCU)

This is a 24-byte field that contains three 8-byte elements with a PIN block format keyword (VISAPCU1 or VISCPU2), a format control keyword (NONE), and eight bytes of spaces.

**output_PIN_message_length**

- **Direction:** Input/Output
- **Type:** Integer

  The length of the `output_PIN_message` field. Currently the value must be a minimum of 16.

**output_PIN_message**

- **Direction:** Output
- **Type:** String

  The reformatted PIN block with the new reference PIN enciphered under the SMPIN session key.

**Restrictions**

The restrictions for CSNBPCU.

None.

**Required commands**

The required commands for CSNBPCU.

This verb requires the following commands to be enabled in the active role based on the permissible key-type, IPINENC or OPINENC, used in the decryption of the input PIN blocks.

<table>
<thead>
<tr>
<th>PIN-block encrypting key-type</th>
<th>Offset</th>
<th>Command</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPINENC</td>
<td>X'00BC'</td>
<td>PIN Change/Unblock - change EMV PIN with OPINENC</td>
<td>Required if either the <code>new_reference_PIN_key</code> or the <code>current_reference_PIN_key</code> are permitted to be an OPINENC key type.</td>
</tr>
<tr>
<td>IPINENC</td>
<td>X'00BD'</td>
<td>PIN Change/Unblock - change EMV PIN with IPINENC</td>
<td>Required if either the <code>new_reference_PIN_key</code> or the <code>current_reference_PIN_key</code> are permitted to be an IPINENC key type.</td>
</tr>
</tbody>
</table>

When a MAC-MDK or an ENC-MDK of key type DKYGENKY is specified with control vector bits (19 - 22) of B’1111’, the Diversified Key Generate - DKYGENKY - DALL command (offset X'0290') must also be enabled in the active role.

**Note:** A role with offset X'0290' enabled can also use the Diversified Key Generate verb with a DALL key.

An enhanced PIN security mode is available for extracting PINs from an IBM 3624 encrypted PIN-block using the PADDIGIT PIN-extraction method. This mode limits checking of the PIN to decimal digits, and a minimum PIN length of 4 is
enforced; no other PIN-block consistency checking will occur. To activate this mode, enable the **Enhanced PIN Security** command (offset X'0313') in the active role.

**Usage notes**

The usage notes for CSNBPCU.

There are additional access points for this verb.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBPCUJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBPCUJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber authentication_master_key_length,
    byte[] authentication_master_key,
    hikmNativeNumber encryption_master_key_length,
    byte[] encryption_master_key,
    hikmNativeNumber key_generation_data_length,
    byte[] key_generation_data,
    hikmNativeNumber new_reference_PIN_key_length,
    byte[] new_reference_PIN_key,
    byte[] new_reference_PIN_block,
    byte[] new_reference_PIN_profile,
    byte[] new_reference_PAN_data,
    hikmNativeNumber current_reference_PIN_key_length,
    byte[] current_reference_PIN_key,
    byte[] current_reference_PIN_block,
    byte[] current_reference_PIN_profile,
    byte[] current_reference_PAN_data,
    hikmNativeNumber output_PIN_data_length,
    byte[] output_PIN_data,
    byte[] output_PIN_profile,
    hikmNativeNumber output_PIN_message_length,
    byte[] output_PIN_message);
```

**Recover PIN from Offset (CSNBPFO)**

Use the Recover PIN from Offset verb to calculate the encrypted customer-entered PIN from a PIN generating key, account information, and an IBM-PINO Offset.

The customer-entered PIN is returned in a PIN block formatted to the specifications of the **PIN_profile** and **PAN_data** parameters, and encrypted with the key supplied in parameter **PIN_encryption_key_identifier** as described in “Parameters” on page 592.

**Note:** This verb supports PCI-HSM 2016 compliant-tagged key tokens.
Recover PIN From Offset (CSNBPFO)

Format

The format of CSNBPFO.

```
CSNBPFO(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    PIN_encryption_key_identifier_length,
    PIN_encryption_key_identifier,
    PIN_generation_key_identifier_length,
    PIN_generation_key_identifier,
    PIN_profile,
    PAN_data,
    offset,
    reserved_1,
    data_array,
    encrypted_PIN_block_length,
    encrypted_PIN_block)
```

Parameters

The parameters for CSNBPFO.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

`rule_array_count`

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0.

`rule_array`

Direction: Input
Type: String array

There are no keywords for this service. Therefore, this parameter is ignored.

`PIN_encryption_key_identifier_length`

Direction: Input
Type: Integer

Length of the `PIN_encryption_key_identifier` parameter in bytes. This value must be 64.

`PIN_encryption_key_identifier`

Direction: Input
Type: String

An internal key token or the label of the AES or DES key storage record containing an OPINENC key that is used to encrypt the `returned_encrypted_PIN_block`.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.
**PIN_generation_key_identifier_length**

**Direction:** Input  
**Type:** Integer

Length of the **PIN_generation_key_identifier** parameter in bytes. This value must be 64.

**PIN_generation_key_identifier**

**Direction:** Input  
**Type:** String

An internal key token or the label of the AES or DES key storage record containing a PINGEN key that is used to generate the bank reference PIN.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

**PIN_profile**

**Direction:** Input  
**Type:** String array

This parameter consists of three 8-byte character elements that contain information necessary to format a PIN block. The pad digit is needed to format an IBM 3624 PIN block. The format control constant must be **NONE**. The first element of the **PIN_profile** (PIN Block Format) determines the format of the output PIN block.

**PAN_data**

**Direction:** Input  
**Type:** String

A 12-byte personal account number (PAN) in character format. The PAN is used in formatting the PIN block if the PIN profile specifies ISO-0, ISO-3, or VISA-4 block formats. Otherwise, ensure that this parameter is a 12-byte variable in application storage. The information in this variable is ignored, but the variable must be specified.

**offset**

**Direction:** Input  
**Type:** String

A 16 byte area that contains the 4-byte PVV left-justified and padded with blanks. This is the value which was returned by a prior call to the **Clear PIN Generate Alternate** service (see "Clear PIN Generate Alternate (CSNCPA)" on page 518).

**data_array**

**Direction:** Input  
**Type:** String

A pointer to a string variable containing three 16-byte numeric character strings, which are equivalent to a single 48-byte string. The values in the **data_array** parameter depend on the keyword for the PIN-calculation method. Each element is not always used, but you must always declare a complete data array.
Recover PIN From Offset (CSNBPOFO)

<table>
<thead>
<tr>
<th>Array Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimalization_table</td>
<td>This element contains the decimalization table of 16 characters that are used to convert the hexadecimal digits X'0' to X'F' of the encrypted validation data to the decimal digits 0 to 9.</td>
</tr>
<tr>
<td>validation_data</td>
<td>This element contains 1 to 16 characters of account data, left justified and padded on the right with spaces.</td>
</tr>
<tr>
<td>reserved_field</td>
<td>Must be 16 bytes of blanks.</td>
</tr>
</tbody>
</table>

reserved_1

Direction: Input  
Type: Integer

The reserved_1 parameter must be zero.

encrypted_PIN_block_length

Direction: Input/Output  
Type: Integer

Length of the encrypted_PIN_block parameter in bytes.

encrypted_PIN_block

Direction: Input  
Type: String

This parameter is an 8-byte field that contains the encrypted customer PIN that was originally used in the Clear PIN Generate Alternate service (see “Clear PIN Generate Alternate (CSNBCPA)” on page 518).

Restrictions

The restrictions for CSNBPOFO.

None.

Required commands

The required commands for CSNBPOFO.

This verb requires the Recover PIN From Offset command (offset X'02B0') to be enabled in the active role.

When the ANSI X9.8 PIN - Enforce PIN block restrictions command (offset X'0350') is enabled in the active role, only a PIN-block format keyword of ISO-0 or ISO-3 is allowed in the input PIN_profile parameter. Note that offset X'0350' also affects access control of the Encrypted PIN Translate and the Secure Messaging for PINs verbs.

When the ANSI X9.8 PIN - Use stored decimalization tables only command (offset X'0356') is enabled in the active role, the Decimalization_table element of the data_array value must match one of the PIN decimalization tables that are in the active state on the coprocessor. Use of this command provides improved security and control for PIN PINs verbs.

An enhanced PIN security mode is available for formatting an encrypted PIN-block into IBM 3624 format. This mode limits checking of the PIN to decimal
digits; no other PIN-block consistency checking will occur. To activate this mode, enable the Enhanced PIN Security command (offset X’0313’) in the active role.

Usage notes

The usage notes for CSNBFPET.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBPFO.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBPFO(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber PIN_encryption_key_identifier_length,
    byte[] PIN_encryption_key_identifier,
    hikmNativeNumber PIN_generation_key_identifier_length,
    byte[] PIN_generation_key_identifier,
    byte[] PIN_profile,
    byte[] PAN_data,
    byte[] offset,
    hikmNativeNumber reserved_1,
    byte[] data_array,
    hikmNativeNumber encrypted_PIN_block_length,
    byte[] encrypted_PIN_block);
```

Secure Messaging for Keys (CSNBSKY)

The Secure Messaging for Keys verb will encrypt a text block including a clear key value decrypted from an internal or external DES token.

The text block is normally a "Value" field of a secure message TLV (Tag/Length/Value) element of a secure message. TLV is defined in ISO/IEC 7816-4.
Secure Messaging for Keys (CSNBSKY)

Format

The format of CSNBSKY.

```c
CSNBSKY(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    input_key_identifier,
    key_encrypting_key_identifier,
    secmsg_key_identifier,
    text_length,
    clear_text,
    initialization_vector,
    key_offset,
    key_offset_field_length,
    enciphered_text,
    output_chaining_vector)
```

Parameters

The parameters for CSNBSKY.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0 or 1.

**rule_array**

- **Direction:** Input
- **Type:** String array

Keywords that provide control information to the verb. The processing method is the encryption mode used to encrypt the message. The `rule_array` keywords are described in Table 178.

**Table 178. Keywords for Secure Messaging for Keys control information**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enciphering mode (One, optional)</td>
<td></td>
</tr>
<tr>
<td>TDES-CBC</td>
<td>Use CBC mode to encipher the message. This is the default.</td>
</tr>
<tr>
<td>TDES-ECB</td>
<td>Use EBC mode to encipher the message.</td>
</tr>
</tbody>
</table>

**input_key_identifier**

- **Direction:** Input/Output
- **Type:** String

The internal token, external token, or key label of an internal token of a double length DES key. The key is recovered in the clear and placed in the text to be encrypted. The control vector of the DES key must not prohibit export.
Secure Messaging for Keys (CSNBSKY)

**key_encrypting_key_identifier**

**Direction:** Input/Output  
**Type:** String

If the `input_key_identifier` is an external token, this parameter is the internal token or the key label of the internal token of IMPORTER or EXPORTER. If it is not, it is a null token. If a key label is specified, the key label must be unique.

**secmsg_key_identifier**

**Direction:** Input/Output  
**Type:** String

The internal token or key label of a secure message key for encrypting keys. This key is used to encrypt the updated `clear_text` containing the recovered DES key.

**text_length**

**Direction:** Input  
**Type:** Integer

The length of the `clear_text` parameter. Length must be a multiple of eight. Maximum length is 4096.

**clear_text**

**Direction:** Input  
**Type:** String

Cleartext that contains the recovered DES key at the offset specified and is then encrypted. Any padding or formatting of the message must be done by the caller on input.

**initialization_vector**

**Direction:** Input  
**Type:** String

The 8-byte supplied string for the TDES-CBC mode of encryption. The `initialization_vector` is XORed with the first eight bytes of `clear_text` before encryption. This field is ignored for TDES-ECB mode.

**key_offset**

**Direction:** Input  
**Type:** Integer

The offset within the `clear_text` parameter at `key_offset` where the recovered clear `input_key_identifier` value is to be placed. The first byte of the `clear_text` field is offset 0.

**key_offset_field_length**

**Direction:** Input  
**Type:** Integer

The length of the field within `clear_text` parameter at `key_offset` where the recovered clear `input_key_identifier` value is to be placed. Length must be a multiple of eight and is equal to the key length of the recovered key. The key must fit entirely within the `clear_text`. 

Chapter 12. Financial services  597
Secure Messaging for Keys (CSNBSKY)

**enciphered_text**

- **Direction:** Output
- **Type:** String

The field where the ciphertext is returned. The length of this field must be at least as long as the *clear_text* field.

**output_chaining_vector**

- **Direction:** Output
- **Type:** String

This field contains the last eight bytes of enciphered text and is used as the *initialization_vector* for the next encryption call if data needs to be chained for TDES-CBC mode. No data is returned for TDES-ECB.

**Restrictions**

The restrictions for CSNBSKY.

None.

**Required commands**

The required commands for CSNBSKY.

This verb requires the **Secure Messaging for Keys** command (offset X'0273') to be enabled in the active role.

**Usage notes**

The usage notes for CSNBSKY.

Keys appear in the clear only within the secure boundary of the cryptographic coprocessor, and never in host storage.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBSKYJ.

See [“Building Java applications using the CCA JNI” on page 28](#).

**Format**

```java
public native void CSNBSKYJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] input_key_identifier,
    byte[] key_encrypting_key,
    byte[] session_key,
    hikmNativeNumber text_length,
    byte[] clear_text,
    byte[] initialization_vector,
    hikmNativeNumber key_offset,
    hikmNativeNumber key_offset_field_length,
    byte[] cipher_text,
    byte[] output_chaining_value);
```
Secure Messaging for PINs (CSNBSPN)

The Secure Messaging for PINs verb will encrypt a text block including a clear PIN block recovered from an encrypted PIN block.

The input PIN block will be reformatted if the block format in the input_PIN_profile is different from the block format in the output_PIN_profile. The clear PIN block will only be self encrypted if the SELFENC keyword is specified in the rule_array. The text block is normally a "Value" field of a secure message TLV (Tag/Length/Value) element of a secure message. TLV is defined in ISO/IEC 7816-4.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.

Format

The format of CSNBSPN.

```
CSNBSPN(
   return_code,
   reason_code,
   exit_data_length,
   exit_data,
   rule_array_count,
   rule_array,
   input_PIN_block,
   PIN_encrypting_key_identifier,
   input_PIN_profile,
   input_PAN_data,
   secmsg_key_identifier,
   output_PIN_profile,
   output_PAN_data,
   text_length,
   clear_text,
   initialization_vector,
   PIN_offset,
   PIN_offset_field_length,
   enciphered_text,
   output_chaining_vector)
```

Parameters

The parameters for CSNBSPN.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

rule_array_count

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 0, 1, or 2.

rule_array

Direction: Input
Type: String array

Keywords that provide control information to the verb. The processing method is the algorithm used to create the generated key. The keywords are
Secure Messaging for PINs (CSNBSPN)

left-aligned and padded on the right with blanks. The rule_array keywords are described in Table 179.

Table 179. Keywords for Secure Messaging for PINs control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enciphering mode (One, optional)</td>
<td></td>
</tr>
<tr>
<td>TDES-CBC</td>
<td>Use CBC mode to encipher the message. This is the default.</td>
</tr>
<tr>
<td>TDES-ECB</td>
<td>Use EBC mode to encipher the message.</td>
</tr>
<tr>
<td>PIN encryption (One, optional)</td>
<td></td>
</tr>
<tr>
<td>CLEARPIN</td>
<td>Recovered clear input PIN block (might be reformatted) is placed in the clear in the message for encryption with the secure message key. This is the default.</td>
</tr>
<tr>
<td>SELFENC</td>
<td>Recovered clear input PIN block (might be reformatted) is self-encrypted and then placed in the message for encryption with the secure message key.</td>
</tr>
</tbody>
</table>

input_PIN_block

Direction: Input  
Type: String  
The 8-byte input PIN block that is to be recovered in the clear and, perhaps, reformatted and then placed in the clear_text to be encrypted.

PIN_encrypting_key_identifier

Direction: Input/Output  
Type: String  
The internal token or key label of the internal token of the PIN encrypting key used in encrypting the input_PIN_block. The key must be an IPINENC key.

input_PIN_profile

Direction: Input  
Type: String  
The three 8-byte character elements that contain information necessary to extract the PIN from a formatted PIN block. The valid input PIN formats are ISO-0, ISO-1, ISO-2, and ISO-3. See “The PIN profile” on page 497 for additional information.

input_PAN_data

Direction: Input  
Type: String  
The 12 digit personal account number (PAN) if the input PIN format is ISO-0 or ISO-3. Otherwise, the parameter is ignored.

For ISO-0 or ISO-3, use the rightmost 12 digits of the PAN, excluding the check digit.

secmsg_key_identifier

Direction: Input/Output  
Type: String  
The internal token or key label of an internal token of a secure message key for encrypting PINs. This key is used to encrypt the updated clear_text.

output_PIN_profile
The three 8-byte character elements that contain information necessary to create a formatted PIN block. If reformatting is not required, the input_PIN_profile and the output_PIN_profile must specify the same PIN block format. Output PIN block formats supported are ISO-0, ISO-1, ISO-2, and ISO-3.

output_PAN_data

The 12 digit personal account number (PAN) if the output PIN format is ISO-0 or ISO-3. Otherwise, this parameter is ignored.

For ISO-0 or ISO-3, use the rightmost 12 digits of the PAN, excluding the check digit.

text_length

The length of the clear_text parameter that follows. Length must be a multiple of eight. Maximum length is 4096.

clear_text

Cleartext that contains the recovered and/or reformatted/encrypted PIN at offset specified and then encrypted. Any padding or formatting of the message must be done by the caller on input.

initialization_vector

The 8-byte supplied string for the TDES-CBC mode of encryption. The initialization_vector is XORed with the first eight bytes of clear_text before encryption. This field is ignored for TDES-ECB mode.

PIN_offset

The offset within the clear_text parameter where the reformatted PIN block is to be placed. The first byte of the clear_text field is offset 0.

PIN_offset_field_length

The length of the field within clear_text parameter at PIN_offset where the recovered clear input_PIN_block value is to be placed. The PIN block might be self-encrypted if requested by the rule_array. Length must be eight. The PIN block must fit entirely within the clear_text.
Secure Messaging for PINs (CSNBSPN)

**enciphered_text**

- **Direction:** Output
- **Type:** String

The field where the ciphertext is returned. The length of this field must be at least as long as the `clear_text` field.

**output_chaining_vector**

- **Direction:** Output
- **Type:** String

This field contains the last eight bytes of ciphertext and is used as the `initialization_vector` for the next encryption call if data needs to be chained for TDES-CBC mode. No data is returned for TDES-ECB.

**Restrictions**

The restrictions for CSNBSPN.

None.

**Required commands**

The required commands for CSNBSPN.

This verb requires the Secure Messaging for PINs command (offset X'0274') to be enabled in the active role.

The following three commands at offsets X'0350', X'0351', and X'0352' affect how PIN processing is performed as described below:

1. Enable the **ANSI X9.8 PIN - Enforce PIN block restrictions** command (offset X'0350') in the active role to apply additional restrictions to PIN processing implemented in CCA 4.1.0, as follows:
   - Constrain use of ISO-2 PIN blocks to offline PIN verification and PIN change operations in integrated circuit card environments only. Specifically, do not allow ISO-2 input or output PIN blocks.
   - Do not reformat a PIN-block format that includes a PAN into a PIN-block format that does not include a PAN.
   - Do not allow a change of PAN data. Specifically, when performing translations between PIN block formats that both include PAN data, do not allow the `input_PIN_data` and `output_PIN_data` variables to be different from the PAN data enciphered in the input PIN block.

   **Note:** A role with offset X'0350' enabled also affects access control of the Clear PIN Generate Alternate and the Encrypted PIN Translate verbs.

2. Enable the **ANSI X9.8 PIN - Allow modification of PAN** command (offset X'0351') in the active role to override the restriction to not allow a change of PAN data. This override is applicable only when either the **ANSI X9.8 PIN - Enforce PIN block restrictions** command (offset X'0350') or the **ANSI X9.8 PIN - Allow only ANSI PIN blocks** command (offset X'0352') or both are enabled in the active role. This override is to support account number changes in issuing environments. Offset X'0351' has no effect if neither offset X'0350' nor offset X'0352' is enabled in the active role.
Note: A role with offset X'0351' enabled also affects access control of the Encrypted PIN Translate verbs.

3. Enable the **ANSI X9.8 PIN - Allow only ANSI PIN blocks** command (offset X'0352') in the active role to apply a more restrictive variation of the **ANSI X9.8 PIN - Enforce PIN block restrictions** command (offset X'0350'). In addition to the previously described restrictions of offset X'0350', this command also restricts the input_PIN_profile and the output_PIN_profile to contain only ISO-0, ISO-1, and ISO-3 PIN block formats. Specifically, the IBM 3624 PIN-block format is not allowed with this command. Offset X'0352' overrides offset X'0350'.

Note: A role with offset X'0352' enabled also affects access control of the Encrypted PIN Translate verbs.

For more information, see "ANSI X9.8 PIN restrictions" on page 495.

**Usage notes**

The usage notes for CSNBSPN.

Keys appear in the clear only within the secure boundary of the cryptographic coprocessors, and never in host storage.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBSPNJ.

See "Building Java applications using the CCA JNI" on page 28.

**Format**

```java
public native void CSNBSPNJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] input_PIN_block,
    byte[] PIN_encrypting_key_identifier,
    byte[] input_PIN_profile,
    byte[] input_PAN_data,
    byte[] secmg_key_identifier,
    byte[] output_PIN_profile,
    byte[] output_PAN_data,
    hikmNativeNumber text_length,
    byte[] clear_text,
    byte[] initialization_vector,
    hikmNativeNumber PIN_offset,
    hikmNativeNumber PIN_offset_field_length,
    byte[] enciphered_text,
    byte[] output_chaining_vector);
```

**Transaction Validation (CSNBTRV)**

The Transaction Validation verb supports the generation and validation of American Express card security codes (CSC).

This verb generates and verifies transaction values based on information from the transaction and a cryptographic key. You select the validation method, and either the generate or verify mode, through `rule_array` keywords.
For the American Express process, the control vector supplied with the cryptographic key must indicate a MAC or MACVER class key. The key can be single or double length. DATAM and DATAMV keys are not supported. The MAC generate control vector bit must be on (bit 20) if you request CSC generation and MAC verify bit (bit 21) must be on if you request verification.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.

**Format**

The format of CSNBTRV.

```c
CSNBTRV(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    transaction_key_length,
    transaction_key,
    transaction_info_length,
    transaction_info,
    validation_values_length,
    validation_values)
```

**Parameters**

The parameters for CSNBTRV.

For the definitions of the `return_code, reason_code, exit_data_length, and exit_data` parameters, see "Parameters common to all verbs" on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1 or 2.

**rule_array**

- **Direction:** Input
- **Type:** String array

Keywords that provide control information to the verb. The keywords are left-aligned in an 8-byte field and padded on the right with blanks. The keywords must be in contiguous storage. The `rule_array` keywords are described in [Table 180](#).

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Card security code algorithm</td>
<td>(One, optional) Specifies use of CSC version 1.0 algorithm for generating or verifying the validation values.</td>
</tr>
<tr>
<td>CSC-V1</td>
<td>Specifies use of CSC version 2.0 algorithm for generating or verifying the validation values.</td>
</tr>
<tr>
<td>American Express card security codes</td>
<td>(One, required) 3-digit card security code (CSC) located on the signature panel. VERIFY implied. This is the default.</td>
</tr>
</tbody>
</table>
### Table 180. Keywords for Transaction Validation control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSC-4</td>
<td>4-digit card security code (CSC) located on the signature panel. VERIFY implied.</td>
</tr>
<tr>
<td>CSC-5</td>
<td>5-digit card security code (CSC) located on the signature panel. VERIFY implied.</td>
</tr>
<tr>
<td>CSC-345</td>
<td>Generate 5-byte, 4-byte, or 3-byte values when given an account number and an expiration date. GENERATE implied.</td>
</tr>
</tbody>
</table>

**Operation (One, optional)**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERIFY</td>
<td>Specifies verification of the value presented in the validation values variable.</td>
</tr>
<tr>
<td>GENERATE</td>
<td>Specifies generation of the value presented in the validation values variable.</td>
</tr>
</tbody>
</table>

**transaction_key_length**

- **Direction:** Input
- **Type:** Integer

The length of the `transaction_key` parameter.

**transaction_key**

- **Direction:** Input
- **Type:** String

The label name or internal token of a MAC or MACVER class key. The key can be single or double length.

**transaction_info_length**

- **Direction:** Input
- **Type:** Integer

The length of the `transaction_info` parameter. For the American Express CSC codes, the length must be 19.

**transaction_info**

- **Direction:** Input
- **Type:** String

For American Express, this is a 19-byte field containing the concatenation of the 4-byte expiration data (in the format YYMM) and the 15-byte American Express account number. Provide the information in character format.

**validation_values_length**

- **Direction:** Input/Output
- **Type:** Integer

The length of the `validation_values` parameter. Maximum value for this field is 64.

**validation_values**

- **Direction:** Input
- **Type:** String

This variable contains American Express CSC values. The data is output for GENERATE and input for VERIFY. See Table 181 on page 606.
Transaction Validation (CSNBTRV)

Table 181. Values for Transaction Validation validation_values parameter

<table>
<thead>
<tr>
<th>Operation</th>
<th>Element description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERATE and CSC-345</td>
<td>555554444333 where:</td>
</tr>
<tr>
<td></td>
<td>55555 = CSC 5 value</td>
</tr>
<tr>
<td></td>
<td>4444 = CSC 4 value</td>
</tr>
<tr>
<td></td>
<td>333 = CSC 3 value</td>
</tr>
<tr>
<td>VERIFY and CSC-3</td>
<td>333 = CSC 3 value</td>
</tr>
<tr>
<td>VERIFY and CSC-4</td>
<td>4444 = CSC 4 value</td>
</tr>
<tr>
<td>VERIFY and CSC-5</td>
<td>55555 = CSC 5 value</td>
</tr>
</tbody>
</table>

Restrictions

The restrictions for CSNBTRV.

None.

Required commands

The required commands for CSNBTRV.

This verb requires the listed commands to be enabled in the active role, depending on the operation and card security code specified:

<table>
<thead>
<tr>
<th>Operation keyword</th>
<th>Card security code keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERATE</td>
<td>CSC-345</td>
<td>X'0291'</td>
<td>Transaction Validation - Generate</td>
</tr>
<tr>
<td>VERIFY</td>
<td>CSC-3</td>
<td>X'0292'</td>
<td>Transaction Validation - Verify CSC-3</td>
</tr>
<tr>
<td></td>
<td>CSC-4</td>
<td>X'0293'</td>
<td>Transaction Validation - Verify CSC-4</td>
</tr>
<tr>
<td></td>
<td>CSC-5</td>
<td>X'0294'</td>
<td>Transaction Validation - Verify CSC-5</td>
</tr>
</tbody>
</table>

Usage notes

The usage notes for CSNBTRV.

There are additional access control points for this verb.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBTRVJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBTRVJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber transaction_key_length,
    byte[] transaction_key,
    hikmNativeNumber transaction_info_length,
```

606 Common Cryptographic Architecture Application Programmer's Guide
byte[] transaction_info,
hikmNativeNumber validation_values_length,
byte[] validation_values;
Transaction Validation (CSNBTRV)
Chapter 13. Financial services for DK PIN methods

The German Banking Industry Committee, Deutsche Kreditwirtschaft (DK) specifies PIN methods and requirements for financial services.

DK is an association of the German banking industry. The intellectual property rights regarding the methods and specification belong to the German Banking Industry Committee.

Note: All cryptographic coprocessors must be loaded with the same level of code. There are several licensed internal code (LIC) releases in support of the DK PIN methods. Ensure that all of the coprocessors have the same LIC level to support the function you want to use.

The following financial services (verbs) are described in this topic:

- “DK Deterministic PIN Generate (CSNBDDPG)” on page 610
- “DK Migrate PIN (CSNBDMP)” on page 617
- “DK PAN Modify in Transaction (CSNBDPMT)” on page 623
- “DK PAN Translate (CSNBDPT)” on page 630
- “DK PIN Change (CSNBDPC)” on page 637
- “DK PIN Verify (CSNBDPV)” on page 649
- “DK PRW Card Number Update (CSNBDPNU)” on page 653
- “DK PRW CMAC Generate (CSNBDPCG)” on page 659
- “DK Random PIN Generate (CSNBDRPG)” on page 662
- “DK Regenerate PRW (CSNBDRP)” on page 668

Weak PIN table

The DK PIN methods support the use of a table of weak PINs.

Services that generate PINs compare the generated PIN against the table and if the PIN is in the table, the service generates a different PIN. Services that change PINs compare the new PIN against the table and if the new PIN is in the table, the service fails.

Weak PIN tables can be stored in the cryptographic coprocessors for use by callable services. Only tables that have been activated can be used. A TKE Workstation is required to manage the tables in the coprocessors.

A Trusted Key Entry workstation (TKE) is required to administer the weak PIN tables for each adapter. In the TKE documentation and user interface, each domain has a restricted PIN table. The corresponding tab is called Domain Restricted PINs. The user may activate, load, and remove PINs from the weak PIN tables on a per-domain basis.

Note: All coprocessors must have installed the same table of weak PINs.
DK PIN methods

The DK PIN methods use a PIN reference value or word (PRW) to verify PINs rather than regenerating the PIN from customer account data.

The PRW is generated by concatenating the customer PAN data, the issuer card data, the PIN length, the PIN, and a 4-byte random number. It is encrypted using a PRW key with the GENONLY key usage. The PRW and random number are the output of the generation. The PIN is verified by generating the PRW using a PRW key with the VERIFY key usage and comparing it against the supplied PRW and random number.

DK Deterministic PIN Generate (CSNBDDPG)

Use the DK Deterministic PIN Generate service to generate a PIN and PIN reference value or word (PRW) using an AES PIN calculation key. The PIN reference value is used to verify the PIN in other services.

**Note:** The PIN generation process uses the account information to calculate the PIN. If the process generates a PIN that appears in the weak PIN table, it rejects that PIN, increments the rightmost byte of the account information by 1, and generates another PIN. The PIN generation process continues to increment the byte until an acceptable PIN is generated. For additional information, see “Weak PIN table” on page 609.

You can use this service to perform the following tasks:

- Generate an encrypted PIN block in PBF-1 format with a PIN print key to be printed on a PIN mailer.
- Generate a PRW which can be used to verify the PIN.
- Optionally, generate an encrypted PIN block in PBF-1 format to be stored for later use in personalizing replacement cards.

**Weak PINs:** The PIN generation process uses the account information to calculate the PIN. During the PIN generation process, if a generated PIN is found in the weak PIN table, it is rejected; the process increments the rightmost byte of the account information by 1 and generates another PIN. The process repeats itself until a suitable PIN is generated.
DK Deterministic PIN Generate (CSNBDDPG)

Format

The format of CSNBDDPG.

```c
CSNBDDPG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    account_info_ER_length,
    account_info_ER,
    PAN_data_length,
    PAN_data,
    card_p_data_length,
    card_p_data,
    card_t_data_length,
    card_t_data,
    PIN_length,
    PIN_generation_key_identifier_length,
    PIN_generation_key_identifier,
    PRW_key_identifier_length,
    PRW_key_identifier,
    PIN_print_key_identifier_length,
    PIN_print_key_identifier,
    OPIN_encryption_key_identifier_length,
    OPIN_encryption_key_identifier,
    OEPB_MAC_key_identifier_length,
    OEPB_MAC_key_identifier,
    PIN_reference_value_length,
    PIN_reference_value,
    PRW_random_number_length,
    PRW_random_number,
    PIN_print_block_length,
    PIN_print_block,
    encrypted_PIN_block_length,
    encrypted_PIN_block,
    PIN_block_MAC_length,
    PIN_block_MAC
)
```

Parameters

The parameters for CSNBDDPG.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

Direction: Input  
Type: Integer  

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0 or 1.

**rule_array**

Direction: Input  
Type: String array  

Keywords that provide control information to the verb. The keywords are left-aligned in an 8-byte field and padded on the right with blanks. The
keywords must be in contiguous storage. The rule_array keywords are described in Table 182.

Table 182. Keywords for DK Deterministic PIN Generate control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIN Block output selection keyword</td>
<td>Return an encrypted PIN block (EPB) and a MAC of the encrypted PIN block.</td>
</tr>
<tr>
<td>NOEPB</td>
<td>Do not return an encrypted PIN block. This is the default value.</td>
</tr>
</tbody>
</table>

account_info_ER_length

- **Direction:** Input
- **Type:** Integer

Specifies the length in bytes of the account_info_ER parameter. The value must be 16.

account_info_ER

- **Direction:** Input
- **Type:** String

The 16-byte account information used to generate the PIN, right-aligned and padded on the left with binary zeros.

PAN_data_length

- **Direction:** Input
- **Type:** Integer

Specifies the length in bytes of the PAN_data parameter. The value must be in the range 10 - 19.

PAN_data

- **Direction:** Input
- **Type:** String

The PAN data to which the PIN is associated. The full account number, including check digit, should be included.

card_p_data_length

- **Direction:** Input
- **Type:** Integer

Specifies the length in bytes of the card_p_data parameter. The value must be in the range 2 - 256.

card_p_data

- **Direction:** Input
- **Type:** String

The time-invariant card data (CDp), determined by the card issuer, which is used to differentiate between multiple cards for one account.
Specifies the length in bytes of the `card_t_data` parameter. The value must be in the range 2 - 256.

**card_t_data**

**Direction:** Input  
**Type:** String

The time-sensitive card data, determined by the card issuer, which, together with the account number and the `card_p_data` value, specifies an individual card.

**PIN_length**

**Direction:** Input  
**Type:** Integer

Specifies the length of the PIN to be generated. This value must be in the range 4 - 12.

**PIN_generation_key_identifier_length**

**Direction:** Input  
**Type:** Integer

Specifies the length in bytes of the `PIN_generation_key_identifier` parameter. If the `PIN_generation_key_identifier` contains a label, the value must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**PIN_generation_key_identifier**

**Direction:** Input  
**Type:** String

The identifier of the PIN generating key. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINCALC, the key usage fields must indicate GENONLY, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**PRW_key_identifier_length**

**Direction:** Input  
**Type:** Integer

Specifies the length in bytes of the `PRW_key_identifier` parameter. If the `PRW_key_identifier` contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**PRW_key_identifier**

**Direction:** Input/Output  
**Type:** String

The identifier of the PRW generating key. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**PIN_print_key_identifier_length**
DK Deterministic PIN Generate (CSNBDDPG)

**PIN_print_key_identifier**

**Direction:** Input/Output  
**Type:** String  

The identifier of the key to wrap the PIN for printing. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOPP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**OPIN_encryption_key_identifier_length**

**Direction:** Input  
**Type:** Integer  

Specifies the length in bytes of the **OPIN_encryption_key_identifier** parameter. If the rule array indicates that no encrypted PIN block is to be returned, this value must be 0. If the **OPIN_encryption_key_identifier** contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**OPIN_encryption_key_identifier**

**Direction:** Input/Output  
**Type:** String  

The identifier of the key to wrap the PIN block. The key identifier is an operational token or the key label of an operational token in key storage. If the rule array indicates that no encrypted PIN block is to be returned, this parameter is ignored. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**OEPB_MAC_key_identifier_length**

**Direction:** Input  
**Type:** Integer  

Specifies the length in bytes of the **OEPB_MAC_key_identifier** parameter. If the rule array indicates that no encrypted PIN block MAC is to be returned, this value must be 0. If the **OEPB_MAC_key_identifier** contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**OEPB_MAC_key_identifier**

**Direction:** Input/Output  
**Type:** String  

The identifier of the key to generate the MAC of the PIN block. The key identifier is an operational token or the key label of an operational token in
If the rule array indicates that no encrypted PIN block is to be returned, this parameter is ignored. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate CMAC, GENONLY, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**PIN_reference_value_length**

Direction: Input/Output  
Type: Integer  

Specifies the length in bytes of the **PIN_reference_value** parameter. The value must be at least 16. On output, it is set to 16.

**PIN_reference_value**

Direction: Output  
Type: String  

The 16-byte calculated PIN reference value.

**PRW_random_number_length**

Direction: Input/Output  
Type: Integer  

Specifies the length in bytes of the **PRW_random_number** parameter. The value must be at least 4. On output, it is set to 4.

**PRW_random_number**

Direction: Output  
Type: String  

The 4-byte random number associated with the PIN reference value.

**PIN_print_block_length**

Direction: Input/Output  
Type: Integer  

Specifies the length in bytes of the **PIN_print_block** parameter. The value must be at least 32. On output, it is set to 32.

**PIN_print_block**

Direction: Output  
Type: String  

The 32-byte encrypted PIN block to be passed to the PIN mailer function.

**encrypted_PIN_block_length**

Direction: Input/Output  
Type: Integer  

Specifies the length in bytes of the **encrypted_PIN_block** parameter. If the rule array indicates that no encrypted PIN block should be returned, this value must be 0. Otherwise, it should be at least 32.

**encrypted_PIN_block**

Direction: Output
DK Deterministic PIN Generate (CSNBDDPG)

Type: String

The 32-byte encrypted PIN block in PBF-1 format. This parameter is ignored if no encrypted PIN block is returned.

PIN_block_MAC_length

Direction: Input/Output
Type: Integer

Specifies the length in bytes of the PIN_block_MAC parameter. If the rule_array indicates that no PIN block MAC should be returned, this value must be 0. Otherwise, it must be at least 8.

PIN_block_MAC

Direction: Output
Type: String

The 8-byte CMAC of the encrypted PIN block. This parameter is ignored if no encrypted PIN block is returned.

Restrictions

The restrictions for CSNBDDPG.

- Use of the Visa PVV PIN-calculation method always produces a four-digit output rather than padding the output with binary zeros to the length of the PIN.
- A key identifier length must be 64 for a key label.

Required commands

The required commands for CSNBDDPG.

The DK Deterministic PIN Generate verb requires the DK Deterministic PIN Generate command (offset X'02C6') to be enabled in the active role.

Usage notes

The usage notes for CSNBDDPG.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBDDPGJ.

See "Building Java applications using the CCA JNI" on page 28.

Format

```java
public native void CSNBDDPGJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber account_info_ER_length,
    byte[] account_info_ER,
    hikmNativeNumber PAN_data_length,
    byte[] PAN_data,
```
DK Migrate PIN (CSNBBDMP)

Use the DK Migrate PIN verb to generate the PIN reference value (PRW) for a specified user account. The PIN reference value is used to verify the PIN in other services.

An ISO-1 formatted PIN block is input to determine the value of the PIN for the account. The PIN is reformatted into a DK-defined PIN block and the PIN reference value is calculated using a PRW random value and other account information. The PIN reference value and associated PRW random value are returned to be used as input by other PIN processes to verify the PIN.

If validation of the PIN is desired to personalize smart cards, specify the EPB rule-array keyword. This keyword causes an output encrypted PIN block to be returned along with a PIN block MAC. The MAC is calculated over the output PIN block and additional card data using the block cipher-based MAC algorithm called CMAC (NIST SP 800-38B).

Note: This service does not test for weak PINs.
DK Migrate PIN (CSNBDMP)

Format

The format of CSNBDMP.

```c
CSNBDMP(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    PAN_data_length,
    PAN_data,
    card_p_data_length,
    card_p_data,
    card_t_data_length,
    card_t_data,
    ISO1_PIN_block_length,
    ISO1_PIN_block,
    IPIN_encryption_key_identifier_length,
    IPIN_encryption_key_identifier,
    PRW_key_identifier_length,
    PRW_key_identifier,
    OPIN_encryption_key_identifier_length,
    OPIN_encryption_key_identifier,
    OEPB_MAC_key_identifier_length,
    OEPB_MAC_key_identifier,
    PIN_reference_value_length,
    PIN_reference_value,
    PRW_random_number_length,
    PRW_random_number,
    encrypted_PIN_block_length,
    encrypted_PIN_block,
    PIN_block_MAC_length,
    PIN_block_MAC
)
```

Parameters

The parameters for CSNBDMP.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0 or 1.

**rule_array**

- **Direction:** Input
- **Type:** String array

Keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks. The `rule_array` keywords are described in Table 183 on page 619.
Table 183. Keywords for DK Migrate PIN control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIN Block output selection keyword (One, optional)</td>
<td></td>
</tr>
<tr>
<td>EPB</td>
<td>Return an encrypted PIN block and a MAC of the encrypted PIN block.</td>
</tr>
<tr>
<td>NOEPB</td>
<td>Do not return an encrypted PIN block (EPB). This is the default value.</td>
</tr>
</tbody>
</table>

**PAN_data_length**
- **Direction:** Input
- **Type:** Integer

Specifies the length in bytes of the **PAN_data** parameter. The value must be in the range 10 - 19.

**PAN_data**
- **Direction:** Input
- **Type:** String

The personal account number in character form which the PIN is associated. The primary account number, including check digit, should be included.

**card_p_data_length**
- **Direction:** Input
- **Type:** Integer

Specifies the length in bytes of the **card_p_data** parameter. The value must be in the range 2 - 256.

**card_p_data**
- **Direction:** Input
- **Type:** String

The time-invariant card data (CDp), determined by the card issuer, which is used to differentiate between multiple cards for one account.

**card_t_data_length**
- **Direction:** Input
- **Type:** Integer

Specifies the length in bytes of the **card_t_data** parameter. The value must be in the range 2 - 256.

**card_t_data**
- **Direction:** Input
- **Type:** String

The time-sensitive card data, determined by the card issuer, which, together with the account number and the **card_p_data**, specifies an individual card.

**ISO1_PIN_block_length**
- **Direction:** Input
- **Type:** Integer

Specifies the length in bytes of the **ISO1_PIN_block** parameter. This value must be 8.
ISO1_PIN_block

Direction: Input
Type: String

The 8-byte encrypted PIN block with the current PIN in ISO-1 format with the customer chosen PIN. This PIN is used to generate the PIN reference value.

IPIN_encryption_key_identifier_length

Direction: Input
Type: Integer

Specifies the length in bytes of the IPIN_encryption_key_identifier parameter. If the IPIN_encryption_key_identifier contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

IPIN_encryption_key_identifier

Direction: Input/Output
Type: String

The identifier of the key to decrypt the PIN block containing the IOS-1 PIN. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be DES and the key type must be IPINENC.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

PRW_key_identifier_length

Direction: Input
Type: Integer

Specifies the length in bytes of the PRW_key_identifier parameter. If PRW_key_identifier contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

PRW_key_identifier

Direction: Input/Output
Type: String

The identifier of the PRW generating key. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, and the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

OPIN_encryption_key_identifier_length

Direction: Input
Type: Integer

Specifies the length in bytes of the OPIN_encryption_key_identifier parameter. If the rule array indicates that no encrypted PIN block is to be returned, this value must be 0. If OPIN_encryption_key_identifier contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

OPIN_encryption_key_identifier
The identifier of the key to wrap the PIN block. The key identifier is an operational token or the key label of an operational token in key storage. If the rule array indicates that no encrypted PIN block is to be returned, this parameter is ignored. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**OEPB_MAC_key_identifier_length**

**Direction:** Input
**Type:** Integer

Specifies the length in bytes of the **OEPB_MAC_key_identifier** parameter. If the rule array indicates that no encrypted PIN block MAC is to be returned, this value must be 0. If **OEPB_MAC_key_identifier** contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**OEPB_MAC_key_identifier**

**Direction:** Input/Output
**Type:** String

The identifier of the key to generate the MAC of the PIN block. The key identifier is an operational token or the key label of an operational token in key storage. If the rule array indicates that no encrypted PIN block MAC is to be returned, this parameter is ignored. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**PIN_reference_value_length**

**Direction:** Input/Output
**Type:** Integer

Specifies the length in bytes of the **PIN_reference_value** parameter. This value must be 16. On output, **PIN_reference_value_length** is set to 16.

**PIN_reference_value**

**Direction:** Output
**Type:** String

The 16-byte calculated PIN reference value.

**PRW_random_number_length**

**Direction:** Input/Output
**Type:** Integer

Specifies the length in bytes of the **PRW_random_number** parameter. The value must be 4. On output, **PRW_random_number_length** is set to 4.

**PRW_random_number**

**Direction:** Output
DK Migrate PIN (CSNBDMP)

**Type:** String

The 4-byte random number associated with the PIN reference value.

**encrypted_PIN_block_length**

**Direction:** Input/Output

**Type:** Integer

Specifies the length in bytes of the encrypted_PIN_block parameter. If the rule array indicates that no encrypted PIN block should be returned, this value must be 0. Otherwise, it should be at least 32.

**encrypted_PIN_block**

**Direction:** Output

**Type:** String

The 32-byte encrypted PIN block in PBF-1 format. This parameter is ignored if no encrypted PIN block is returned.

**PIN_block_MAC_length**

**Direction:** Input/Output

**Type:** Integer

Specifies the length in bytes of the PIN_block_MAC parameter. If the rule_array indicates that no PIN block MAC should be returned, this value must be 0. Otherwise, it must be at least 8.

**PIN_block_MAC**

**Direction:** Output

**Type:** String

The 8-byte CMAC of the encrypted PIN block. This parameter is ignored if no encrypted PIN block is returned.

**Restrictions**

The restrictions for CSNBDMP.

A key identifier length must be 64 for a key label.

**Required commands**

The required commands for CSNBDMP.

The DK Migrate PIN verb requires the **DK Migrate PIN** command (offset X'02CE') to be enabled in the active role.

**Usage notes**

The usage notes for CSNBDMP.

None.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBDMPJ.

See “Building Java applications using the CCA JNI” on page 28.
DK Migrate PIN (CSNBDMJ)

Format

```java
public native void CSNBDMPJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber PAN_data_length,
    byte[] PAN_data,
    hikmNativeNumber card_p_data_length,
    byte[] card_p_data,
    hikmNativeNumber card_t_data_length,
    byte[] card_t_data,
    hikmNativeNumber ISO1_PIN_block_length,
    byte[] ISO1_PIN_block,
    hikmNativeNumber IPIN_encryption_key_identifier_length,
    byte[] IPIN_encryption_key_identifier,
    hikmNativeNumber PRW_key_identifier_length,
    byte[] PRW_key_identifier,
    hikmNativeNumber OPIN_encryption_key_identifier_length,
    byte[] OPIN_encryption_key_identifier,
    hikmNativeNumber OEPB_MAC_key_identifier_length,
    byte[] OEPB_MAC_key_identifier,
    hikmNativeNumber PIN_reference_value_length,
    byte[] PIN_reference_value,
    hikmNativeNumber PRW_random_number_length,
    byte[] PRW_random_number,
    hikmNativeNumber encrypted_PIN_block_length,
    byte[] encrypted_PIN_block,
    hikmNativeNumber PIN_block_MAC_length,
    byte[] PIN_block_MAC);
```

DK PAN Modify in Transaction (CSNBDPMT)

Use the DK PAN Modify in Transaction verb to obtain a new PIN reference value (PRW) for an existing PIN when a merger has occurred and the account information has changed.

The input includes the current PIN, the account information (PAN and card data) for the current, and the new account.

The DK PRW CMAC Generate service is called prior to this service to generate the MAC of the changed account information. If the MAC associated with the account information does not verify, the service fails.
DK PAN Modify in Transaction (CSNBDPMT)

Format

The format of CSNBDPMT.

```
CSNBDPMT(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    current_PAN_data_length,
    current_PAN_data,
    new_PAN_data_length,
    new_PAN_data,
    current_card_p_data_length,
    current_card_p_data,
    current_card_t_data_length,
    current_card_t_data,
    new_card_p_data_length,
    new_card_p_data,
    new_card_t_data_length,
    new_card_t_data,
    CMAC_FUS_length,
    CMAC_FUS,
    ISO_encrypted_PIN_block_length,
    ISO_encrypted_PIN_block,
    current_PIN_reference_value_length,
    current_PIN_reference_value,
    current_PRW_random_number_length,
    current_PRW_random_number,
    CMAC_FUS_key_identifier_length,
    CMAC_FUS_key_identifier,
    IPIN_encryption_key_identifier_length,
    IPIN_encryption_key_identifier,
    PRW_key_identifier_length,
    PRW_key_identifier,
    new_PRW_key_identifier_length,
    new_PRW_key_identifier,
    new_PIN_reference_value_length,
    new_PIN_reference_value,
    new_PRW_random_number_length,
    new_PRW_random_number)
```

Parameters

The parameters for CSNBDPMT.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

`rule_array_count`

**Direction:** Input

**Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0.

`rule_array`

**Direction:** Input

**Type:** String array

There are no keywords for this service.
**current_PAN_data_length**

Direction: Input  
Type: Integer

Specifies the length in bytes of the current_PAN_data parameter. The value must be in the range 10 - 19.

**current_PAN_data**

Direction: Input  
Type: String

The current PAN data associated with the PIN. The full account number, including check digit, should be included.

**new_PAN_data_length**

Direction: Input  
Type: Integer

Specifies the length in bytes of the new_PAN_data parameter. The value must be in the range 10 - 19.

**new_PAN_data**

Direction: Output  
Type: Integer

The new PAN data to be associated with the PIN. The full account number, including check digit, should be included.

**current_card_p_data_length**

Direction: Input  
Type: Integer

Specifies the length in bytes of the current_card_p_data parameter. The value must be in the range 2 - 256.

**current_card_p_data**

Direction: Input  
Type: String

The time-invariant card data (CDp) of the current account, determined by the card issuer.

**current_card_t_data_length**

Direction: Input  
Type: Integer

Specifies the length in bytes of the current_card_t_data parameter. The value must be in the range 2 - 256.

**current_card_t_data**

Direction: Input  
Type: String

The time-sensitive card data of the current account, determined by the card issuer.

**new_card_p_data_length**
DK PAN Modify in Transaction (CSNBDPMT)

**Direction:** Input  
**Type:** Integer

Specifies the length in bytes of the `new_card_p_data` parameter. The value must be in the range 2 - 256.

**new_card_p_data**

**Direction:** Input  
**Type:** String

The time-invariant card data (CDp) of the current account, determined by the card issuer.

**new_card_t_data_length**

**Direction:** Input  
**Type:** Integer

Specifies the length in bytes of the `new_card_t_data` parameter. The value must be in the range 2 - 256.

**new_card_t_data**

**Direction:** Input  
**Type:** String

The time-sensitive card data of the current account, determined by the card issuer.

**CMAC_FUS_length**

**Direction:** Input  
**Type:** Integer

Specifies the length in bytes of the `CMAC_FUS` parameter. The value must be in the range 8 - 16.

**CMAC_FUS**

**Direction:** Input  
**Type:** String

The 8-byte to 16-byte MAC that was generated from the current and new PANs and card data strings and PIN reference values. The MAC is generated using the DK PRW CMA C Generate service.

**ISO_encrypted_PIN_block_length**

**Direction:** Input  
**Type:** Integer

Specifies the length in bytes of the `ISO_encrypted_PIN_block` parameter. The value must be 8.

**ISO_encrypted_PIN_block**

**Direction:** Input  
**Type:** String

The 8-byte encrypted PIN block with the PIN in ISO-1 format.
DK PAN Modify in Transaction (CSNBDPMT)

```
Direction:    Input
Type:        Integer

Specifies the length in bytes of the current_PIN_reference_value parameter. The value must be 16.

**current_PIN_reference_value**

Direction:    Input
Type:        String

The 16-byte PIN reference value for comparison to the calculated value.

**current_PRW_random_number_length**

Direction:    Input
Type:        Integer

Specifies the length in bytes of the current_PRW_random_number parameter. The value must be 4.

**current_PRW_random_number**

Direction:    Input
Type:        String

The 4-byte random number associated with the PIN reference value.

**CMAC_FUS_key_identifier_length**

Direction:    Input
Type:        Integer

Specifies the length in bytes of the CMAC_FUS_key_identifier parameter. If CMAC_FUS_key_identifier contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**CMAC_FUS_key_identifier**

Direction:    Input
Type:        String

The identifier of the key to verify the CMAC_FUS value. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate VERIFY, CMAC, and DKPINAD2.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**IPIN_encryption_key_identifier_length**

Direction:    Input
Type:        Integer

Specifies the length in bytes of the IPIN_encryption_key_identifier parameter. If IPIN_encryption_key_identifier contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**IPIN_encryption_key_identifier**

Direction:    Input
Type:        String
```
The identifier of the key to decrypt the ISO_encrypted_PIN_block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be DES and the key type must be IPINENC.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**PRW_key_identifier_length**

- **Direction:** Input
- **Type:** Integer

Specifies the length in bytes of the PRW_key_identifier parameter. If PRW_key_identifier contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**PRW_key_identifier**

- **Direction:** Input
- **Type:** String

The identifier of the key to verify the input PRW. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, and the key usage fields must indicate VERIFY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**new_PRW_key_identifier_length**

- **Direction:** Input
- **Type:** Integer

Specifies the length in bytes of the new_PRW_key_identifier parameter. If new_PRW_key_identifier contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**new_PRW_key_identifier**

- **Direction:** Input
- **Type:** String

The identifier of the key to generate the new PRW. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, and the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**new_PIN_reference_value_length**

- **Direction:** Input
- **Type:** Integer

Specifies the length in bytes of the new_PIN_reference_value parameter. The value must be at least 16. On output, it is set to 16.

**new_PIN_reference_value**

- **Direction:** Input
- **Type:** String
The 16-byte new PIN reference value.

**new_PRW_random_number_length**

Direction: Input  
Type: Integer

Specifies the length in bytes of the `new_PRW_random_number` parameter. The value must be at least 4. On output, it is set to 4.

**new_PRW_random_number**

Direction: Input  
Type: String

The 4-byte random number associated with the new PIN reference value.

**Restrictions**

The restrictions for CSNBDPMT.

None.

**Required commands**

The required commands for CSNBDPMT.

The DK PAN Modify in Transaction verb requires the `DK PAN Modify in Transaction` command (offset X'02C5') to be enabled in the active role.

**Usage notes**

The usage notes for CSNBDPMT.

None.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBDPMTJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBDPMTJ(
    hikmNativeNumber    return_code,
    hikmNativeNumber    reason_code,
    hikmNativeNumber    exit_data_length,
    byte[]              exit_data,
    hikmNativeNumber    rule_array_count,
    byte[]              rule_array,
    hikmNativeNumber    current_PAN_data_length,
    byte[]              current_PAN_data,
    hikmNativeNumber    new_PAN_data_length,
    byte[]              new_PAN_data,
    hikmNativeNumber    current_card_p_data_length,
    byte[]              current_card_p_data,
    hikmNativeNumber    current_card_t_data_length,
    byte[]              current_card_t_data,
    hikmNativeNumber    new_card_p_data_length,
    byte[]              new_card_p_data,
    hikmNativeNumber    new_card_t_data_length,
    byte[]              new_card_t_data,
)
```
DK PAN Modify in Transaction (CSNBDPMT)

```c
hikmNativeNumber  CMAC_FUS_length,
byte[]            CMAC_FUS,
byte[]            ISO_encrypted_PIN_block_length,
byte[]            ISO_encrypted_PIN_block,
byte[]            current_PIN_reference_value_length,
byte[]            current_PIN_reference_value,
byte[]            current_PRW_random_number_length,
byte[]            current_PRW_random_number,
byte[]            CMAC_FUS_key_identifier_length,
byte[]            CMAC_FUS_key_identifier,
byte[]            IPIN_encryption_key_identifier_length,
byte[]            IPIN_encryption_key_identifier,
byte[]            PRW_key_identifier_length,
byte[]            PRW_key_identifier,
byte[]            new_PRW_key_identifier_length,
byte[]            new_PRW_key_identifier,
byte[]            new_PIN_reference_value_length,
byte[]            new_PIN_reference_value,
byte[]            new_PRW_random_number_length,
byte[]            new_PRW_random_number);
```

DK PAN Translate (CSNBDPT)

Use the DK PAN Translate verb to create an encrypted PIN block with the same PIN and a different PAN. The account data may change, but changing the PIN is to be avoided. This service creates a new encrypted PIN block and MAC on the encrypted PIN block that is used to accept the PAN change at an authorization node.

You can use this service to perform the following tasks:
- Generate an encrypted PIN block in PBF-1 format with a changed PAN to be used at the authorization node to create a PIN reference value.
- Generate a CMAC over the encrypted PIN block for validation.
Format

The format of CSNBDPT.

```c
CSNBDPT(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    card_p_data_length,
    card_p_data,
    card_t_data_length,
    card_t_data,
    new_PAN_data_length,
    new_PAN_data,
    new_card_p_data_length,
    new_card_p_data,
    PIN_reference_value_length,
    PIN_reference_value,
    PRW_random_number_length,
    PRW_random_number,
    current_encrypted_PIN_block_length,
    current_encrypted_PIN_block,
    current_PIN_block_MAC_length,
    current_PIN_block_MAC,
    PRW_key_identifier_length,
    PRW_key_identifier,
    IPIN_encryption_key_identifier_length,
    IPIN_encryption_key_identifier,
    IEPB_MAC_key_identifier_length,
    IEPB_MAC_key_identifier,
    OPIN_encryption_key_identifier_length,
    OPIN_encryption_key_identifier,
    OEPB_MAC_key_identifier_length,
    OEPB_MAC_key_identifier,
    new_encrypted_PIN_block_length,
    new_encrypted_PIN_block,
    new_PIN_block_MAC_length,
    new_PIN_block_MAC)
```

Parameters

The parameters for CSNBDPT.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

`rule_array_count`

Direction: Input  
Type: Integer  

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0.

`rule_array`

Direction: Input  
Type: String array  

There are no keywords for this service.

`card_p_data_length`
DK PAN Translate (CSNBDBPT)

Direction: Input  
Type: Integer

Specifies the length in bytes of the **card_p_data** parameter. The value must be in the range 2 - 256.

**card_p_data**

Direction: Input  
Type: String

The time-invariant card data (CDp), determined by the card issuer, which is used to differentiate between multiple cards for one account.

**card_t_data_length**

Direction: Input  
Type: Integer

Specifies the length in bytes of the **card_t_data** parameter. The value must be in the range 2 - 256.

**card_t_data**

Direction: Input  
Type: String

The time-sensitive card data, determined by the card issuer, which, together with the account number and the **card_p_data**, specifies an individual card.

**new_PAN_data_length**

Direction: Input  
Type: Integer

Specifies the length in bytes of the **new_PAN_data** parameter. The value must be in the range 10 - 19.

**new_PAN_data**

Direction: Input  
Type: String

The new personal account number (in character string form) to which the PIN is associated. The full account number, including check digit, should be included.

**new_card_p_data_length**

Direction: Input  
Type: Integer

Specifies the length in bytes of the **new_card_p_data** parameter. The value must be in the range 2 - 256.

**new_card_p_data**

Direction: Input  
Type: String

The time-invariant card data (CDp), determined by the card issuer, which is used to differentiate between multiple cards for one account.

**PIN_reference_value_length**
Direction: Input
Type: Integer

Specifies the length in bytes of the PIN_reference_value parameter. The value must be 16.

**PIN_reference_value**

Direction: Input
Type: String

The 16-byte PIN reference value for comparison to the calculated value.

**PRW_random_number_length**

Direction: Input
Type: Integer

Specifies the length in bytes of the PRW_random_number parameter. The value must be 4.

**PRW_random_number**

Direction: Input
Type: String

The 4-byte random number associated with the PIN reference value.

**current_encrypted_PIN_block_length**

Direction: Input
Type: Integer

Specifies the length in bytes of the current_encrypted_PIN_block parameter. The value must be 32.

**current_encrypted_PIN_block**

Direction: Input
Type: Integer

The 32-byte encrypted PIN block in PBF-1 format of the current PIN.

**current_PIN_block_MAC_length**

Direction: Input
Type: Integer

Specifies the length in bytes of the current_PIN_block_MAC parameter. The value must be 8.

**current_PIN_block_MAC**

Direction: Input
Type: String

The 8-byte MAC of the current encrypted PIN block and the card_p_data.

**PRW_key_identifier_length**

Direction: Input
Type: Integer

Specifies the length in bytes of the PRW_key_identifier parameter. If
**PRW_key_identifier** contains a label, the length must be 64. Otherwise, the value must at least the actual token length, up to 725.

**PRW_key_identifier**

Direction: Input  
Type: String

The identifier of the PRW verifying key. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, the key usage fields must indicate VERIFY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**IPIN_encryption_key_identifier_length**

Direction: Input  
Type: Integer

Specifies the length in bytes of the **IPIN_encryption_key_identifier** parameter. If the **IPIN_encryption_key_identifier** contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**IPIN_encryption_key_identifier**

Direction: Input/Output  
Type: String

The identifier of the key to decrypt the PIN block containing the current PIN. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate DECRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**IEPB_MAC_key_identifier_length**

Direction: Input  
Type: Integer

Specifies the length in bytes of the **IEPB_MAC_key_identifier** parameter. If the **IEPB_MAC_key_identifier** contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**IEPB_MAC_key_identifier**

Direction: Input/Output  
Type: String

The identifier of the key to verify MAC of the inbound encrypted PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate CMAC, VERIFY, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**OPIN_encryption_key_identifier_length**
Direction: Input
Type: Integer

Specifies the length in bytes of the `OPIN_encryption_key_identifier` parameter. If the `OPIN_encryption_key_identifier` contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**OPIN_encryption_key_identifier**

Direction: Input/Output
Type: String

The identifier of the key to encrypt the new PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINAD1.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**OEPB_MAC_key_identifier_length**

Direction: Input
Type: Integer

Specifies the length in bytes of the `OEPB_MAC_key_identifier` parameter. If the `OEPB_MAC_key_identifier` contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**OEPB_MAC_key_identifier**

Direction: Input/Output
Type: String

The identifier of the key to generate the MAC of the new encrypted PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate CMAC, GENONLY, and DKPINAD1.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**new_encrypted_PIN_block_length**

Direction: Input/Output
Type: Integer

Specifies the length in bytes of the `new_encrypted_PIN_block` parameter. The value must be at least 32. On output, it is set to 32.

**new_encrypted_PIN_block**

Direction: Output
Type: String

The 32-byte encrypted new PIN block.

**new_PIN_block_MAC_length**

Direction: Input/Output
Type: Integer
DK PAN Translate (CSNBDPT)

Specifies the length in bytes of the `new_PIN_block_MAC` parameter. The value must be at least 8.

`new_PIN_block_MAC`

Direction: Output
Type: String

The 8-byte MAC of the new encrypted PIN block.

Restrictions
The restrictions for CSNBDPT.

None.

Required commands
The required commands for CSNBDPT.

The DK PAN Translate verb requires the **DK PAN Translate** command (offset X'02C7') to be enabled in the active role.

Usage notes
The usage notes for CSNBDPT.

None.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNBDPTJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBDPTJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count
    byte[] rule_array,
    hikmNativeNumber card_p_data_length,
    byte[] card_p_data,
    hikmNativeNumber card_t_data_length,
    byte[] card_t_data,
    hikmNativeNumber new_PAN_data_length,
    byte[] new_PAN_data,
    hikmNativeNumber new_card_p_data_length,
    byte[] new_card_p_data,
    hikmNativeNumber PIN_reference_value_length,
    byte[] PIN_reference_value,
    hikmNativeNumber PRW_random_number_length,
    byte[] PRW_random_number,
    hikmNativeNumber current_encrypted_PIN_block_length,
    byte[] current_encrypted_PIN_block,
    hikmNativeNumber current_PIN_block_MAC_length,
    byte[] current_PIN_block_MAC,
    hikmNativeNumber PRW_MAC_key_identifier_length,
    byte[] PRW_MAC_key_identifier,
    hikmNativeNumber IPIN_encryption_key_identifier_length,
    byte[] IPIN_encryption_key_identifier,
```
DK PIN Change (CSNBBDPC)

Use the DK PIN Change verb to update the personal identification number (PIN) reference value or word (PRW) for a specified account when a cardholder uses a bank or credit card at an ATM or point-of-sale (POS) terminal to update a card with a new PIN, in other words, to change the current PIN to a customer-selected PIN.

When cardholders use a terminal to update a card with a new PIN, they enter the current PIN and the new PIN. At the terminal, the current PIN is formatted into an 8-byte ISO-1 PIN block and enciphered using a PIN encrypting key for the current PIN. Likewise, the new PIN is formatted into an 8-byte ISO-1 PIN block and enciphered using a PIN encrypting key for the new PIN.

The account of a cardholder is uniquely identified by a 10 - 19 digit primary account number (PAN) of the bank or credit card. Additional information normally available on the card is a time-sensitive card expiry date and a time-invariant (permanent) card sequence number. The verb verifies the current PIN by deciphering the current ISO-1 PIN block and uses the recovered PIN and other additional information to verify the current PIN. If the current PIN does not verify, the process is aborted and an error is returned. If it does verify, the new PIN is recovered from the new ISO-1 PIN block and is reformatted into a DK-defined PIN block that is used with a new PRW random value and other information to calculate a new PRW. The new PRW and associated new PRW random value are returned to be used as input later by other PIN processes for PIN verification.

A card script can be created and encrypted for use later to update a customer smart (chip) card. To create a TDES-encrypted card script, specify either the TDES-CBC or TDES-ECB script selection keyword in the rule array. Beginning with Release 4.4, to create an AES-encrypted card script, specify AES-CBC.

If validation of the PIN is desired to personalize a smart card, specify the EPB PIN block output selection rule-array keyword. This keyword causes an output encrypted PIN block to be returned along with a PIN block MAC. The MAC is calculated over the output PIN block and additional card data using the block cipher-based MAC algorithm, called CMAC (refer to NIST SP 800-38B).

**Note:** If the PIN recovered from the `new_ISO1_PIN_block` variable is found in the weak PIN table, it is rejected and an error is returned indicating that the selected PIN was in the weak PIN table.
DK PIN Change (CSNBDPC)

Format

The format of CSNBDPC.

```c
CSNBDPC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    PAN_data_length,
    PAN_data,
    card_p_data_length,
    card_p_data,
    card_t_data_length,
    card_t_data,
    cur_ISO1_PIN_block_length,
    cur_ISO1_PIN_block,
    new_ISO1_PIN_block_length,
    new_ISO1_PIN_block,
    card_script_data_length,
    card_script_data,
    script_offset,
    script_offset_field_length,
    script_initialization_vector_length,
    script_initialization_vector,
    output_PIN_profile,
    PIN_reference_value_length,
    PIN_reference_value,
    PRW_random_number_length,
    PRW_random_number,
    PRW_key_identifier_length,
    PRW_key_identifier,
    cur_IPIN_encryption_key_identifier_length,
    cur_IPIN_encryption_key_identifier,
    new_IPIN_encryption_key_identifier_length,
    new_IPIN_encryption_key_identifier,
    script_key_identifier_length,
    script_key_identifier,
    script_MAC_key_identifier_length,
    script_MAC_key_identifier,
    new_PRW_key_identifier_length,
    new_PRW_key_identifier,
    OPIN_encryption_key_identifier_length,
    OPIN_encryption_key_identifier,
    OEPB_MAC_key_identifier_length,
    OEPB_MAC_key_identifier,
    script_length,
    script,
    script_MAC_length,
    script_MAC,
    new_PIN_reference_value_length,
    new_PIN_reference_value,
    new_PRW_random_number_length,
    new_PRW_random_number,
    output_encrypted_PIN_block_length,
    output_encrypted_PIN_block,
    PIN_block_MAC_length,
    PIN_block_MAC)
```

Parameters

The parameters for CSNBDPC.
For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. The value must be 0, 1, 2, 3, 4, or 5.

**rule_array**

- **Direction:** Input
- **Type:** String array

Keywords that provide control information to the verb. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks. The rule_array keywords are described in Table 184.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PIN Block output selection keyword</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>EPB</td>
<td>Return an encrypted PIN block and a MAC to verify the encrypted PIN block.</td>
</tr>
<tr>
<td>NOEPB</td>
<td>Do not return an encrypted PIN block (EPB). This is the default value.</td>
</tr>
<tr>
<td><strong>Script selection algorithm and method</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>AES-CBC</td>
<td>Use CBC mode to AES encrypt the PIN block in the script.</td>
</tr>
<tr>
<td>NOSCRIPT</td>
<td>Do not return an encrypted SMPIN message with a MAC. This is the default value.</td>
</tr>
<tr>
<td>TDES-CBC</td>
<td>Use CBC mode to TDES encrypt the PIN block in the script.</td>
</tr>
<tr>
<td>TDES-ECB</td>
<td>Use ECB mode to TDES encrypt the PIN block in the script.</td>
</tr>
<tr>
<td><strong>PIN encryption keyword</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>Only valid if TDES-CBC or TDES-ECB is selected.</td>
<td></td>
</tr>
<tr>
<td>CLEARPIN</td>
<td>Do not encrypt the PIN prior to inserting in the script block. This is the default value.</td>
</tr>
<tr>
<td>SELF-ENC</td>
<td>Copy the PIN-block self-encrypted to the clear PIN block within the clear output message. Use this rule array keyword to specify that the 8-byte PIN block shall be used as a DES key to encrypt the PIN block. The service copies the self-encrypted PIN block to the clear PIN block in the output message.</td>
</tr>
<tr>
<td><strong>MAC Ciphering Method</strong> (One required for AES-CBC, one optional for TDES-CBC or TDES-ECB, otherwise not allowed.)</td>
<td></td>
</tr>
<tr>
<td>CMAC</td>
<td>Specifies to use the cipher-based MAC algorithm block cipher mode of operation for authentication, recommended in NIST SP 800-38B. Required for AES-CBC. Only valid with AES-CBC.</td>
</tr>
<tr>
<td>EMVMACD</td>
<td>Specifies the EMV-related message-packing and calculation method.</td>
</tr>
<tr>
<td>TDES-MAC</td>
<td>Specifies the ANSI X9.9 Option 1 (binary data) procedure and a CBC Triple-DES encryption of the data.</td>
</tr>
<tr>
<td>X9.19OPT</td>
<td>Specifies the ANSI X9.19 Optional Procedure. A double-length key is required. This is the default value.</td>
</tr>
<tr>
<td><strong>MAC Length and presentation</strong> (one optional, with keyword AES-CBC, TDES-CBC or TDES-ECB, otherwise not allowed.)</td>
<td></td>
</tr>
<tr>
<td>MACLEN8</td>
<td>Specifies an 8-byte MAC. This is the default for TDES-CBC and TDES-ECB.</td>
</tr>
<tr>
<td>MACLEN16</td>
<td>Specifies a 16-byte MAC. Only valid with CMAC. This is the default for AES-CBC.</td>
</tr>
</tbody>
</table>
DK PIN Change (CSNBDPDC)

**PAN_data_length**

Direction: Input  
Type: Integer

Specifies the length in bytes of the `PAN_data` parameter. The value must be in the range 10 - 19.

**PAN_data**

Direction: Input  
Type: String

The PAN data to which the PIN is associated. The full account number, including check digit, should be included.

**card_p_data_length**

Direction: Input  
Type: Integer

Specifies the length in bytes of the `card_p_data` parameter. The value must be in the range 2 - 256.

**card_p_data**

Direction: Input  
Type: String

The time-invariant card data (CDp), determined by the card issuer, which is used to differentiate between multiple cards for one account.

**card_t_data_length**

Direction: Input  
Type: Integer

Specifies the length in bytes of the `card_t_data` parameter. The value must be in the range 2 - 256.

**card_t_data**

Direction: Input  
Type: String

The time-sensitive card data, determined by the card issuer, which, together with the account number and the `card_p_data`, specifies an individual card.

**cur_ISO1_PIN_block_length**

Direction: Input  
Type: Integer

Specifies the length in bytes of the `cur_ISO1_PIN_block` parameter. The value must be 8.

**cur_ISO1_PIN_block**

Direction: Input  
Type: String

The 8-byte encrypted PIN block with the current PIN in ISO-1 format.
new_ISO1_PIN_block_length

Direction: Input
Type: Integer

Specifies the length in bytes of the new_ISO1_PIN_block parameter. The value must be 8.

new_ISO1_PIN_block

Direction: Input
Type: String

The new encrypted PIN block with the customer chosen PIN. The PIN block must be in ISO-1 format.

card_script_data_length

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the card_script_data variable. If the script selection of the rule array specifies to not return an encrypted SMPIN message with a PIN block MAC (that is, AES-CBC, TDES-CBC, or TDES-ECB is not specified), the value must be 0. Otherwise, set the value to a multiple of 16 and less than or equal to 4096 for AES_CBC, and set the value to a multiple of 8 and less than or equal to 4096 for TDES-CBC or TDES-ECB.

card_script_data

Direction: Input
Type: String

The cleartext string to be updated with the clear PIN block and encrypted.

script_offset

Direction: Input
Type: Integer

The offset to the location for the PIN block in the script. Specify the first byte of the cleartext as offset 0. This offset plus the value of the script_offset_field_length must be less than or equal to the card_script_data_length. If NOSCRIPT is specified in the rule array, this parameter is ignored.

script_offset_field_length

Direction: Input
Type: Integer

The length of the field within the card_script_data parameter at script_offset where the new PIN value is to be placed. Length must be 8. The PIN block must fit entirely within the card_script_data. If NOSCRIPT is specified in the rule array, this parameter is ignored.

script_initialization_vector_length

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the script_initialization_vector parameter. For script selection algorithm and
method keyword AES-CBC the value must be 16, and for TDES-CBC the value must be 8. Otherwise, set the value to 0.

**script_initialization_vector**

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing the initialization vector to use when encrypting the script in CBC mode. If the `script_initialization_vector_length` variable is 0 or if keyword TDES-ECB is specified, this parameter is ignored but must be declared. Otherwise, this parameter must point to a string of hexadecimal zeros.

**output_PIN_profile**

- **Direction:** Input
- **Type:** String

A 24-byte string containing the PIN profile, including the PIN block format for the script. See “The PIN profile” on page 497 for additional information. You can use PIN-block formats ISO-0, ISO-1, ISO-2, and ISO-3 with this service. If NOSCRIPT is specified in the rule array, this parameter is ignored.

**PIN_reference_value_length**

- **Direction:** Input
- **Type:** Integer

Specifies the length in bytes of the `PIN_reference_value` parameter. This value must be 16.

**PIN_reference_value**

- **Direction:** Input
- **Type:** String

The 16-byte PIN reference value of the current PIN for comparison to the calculated value.

**PRW_random_number_length**

- **Direction:** Input
- **Type:** Integer

Specifies the length in bytes of the `PRW_random_number` parameter. The value must be 4.

**PRW_random_number**

- **Direction:** Input
- **Type:** String

The 4-byte random number associated with the PIN reference value of the current PIN.

**PRW_key_identifier_length**

- **Direction:** Input
- **Type:** Integer

Specifies the length in bytes of the `PRW_key_identifier` parameter. If the `PRW_key_identifier` contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.
**PRW_key_identifier**

**Direction:** Input  
**Type:** String

The identifier of the key to verify the PRW of the current PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, and the key usage fields must indicate VERIFY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**cur_IPIN_encryption_key_identifier_length**

**Direction:** Input  
**Type:** Integer

Specifies the length in bytes of the `cur_IPIN_encryption_key_identifier` parameter. If the `cur_IPIN_encryption_key_identifier` contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**cur_IPIN_encryption_key_identifier**

**Direction:** Input/Output  
**Type:** String

The identifier of the key to decrypt the PIN_block containing the current PIN. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be DES and the key type must be IPINENC.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**new_IPIN_encryption_key_identifier_length**

**Direction:** Input  
**Type:** Integer

Specifies the length in bytes of the `new_IPIN_encryption_key_identifier` parameter. If the `new_IPIN_encryption_key_identifier` contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**new_IPIN_encryption_key_identifier**

**Direction:** Input/Output  
**Type:** String

The identifier of the key to decrypt the PIN_block containing the new PIN. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be DES and the key type must be IPINENC.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**script_key_identifier_length**

**Direction:** Input  
**Type:** Integer
A pointer to an integer variable containing the number of bytes of data in the `script_key_identifier` variable. If the `script_key_identifier` parameter identifies a key label, the length must be 64. Otherwise, for script selection algorithm and method keyword NOSCRIPT or its default, set the length to 0 or the length of a null key token, for AES-CBC set a maximum length of 725, and for TDES-CBC or TDES-ECB set the length to 64.

**`script_key_identifier`**

**Direction:** Input/Output  
**Type:** String

A pointer to a string variable containing an operational fixed-length DES key-token or the key label of such a record in DES key-storage. Beginning with Release 4.4, it can be a pointer to a string variable containing an operational variable-length AES key-token or the key label of such a record in AES key-storage. The type of key depends on the script selection algorithm and method of the rule array:

- If AES-CBC is specified to return an AES-enciphered SMPIN message with a PIN block MAC, the key must be contained in a variable-length symmetric key-token. The key must have a token algorithm of AES and a key type of SECMSG. In addition, the key usage fields must enable the encryption of PINs in an EMV secure message (SMPIN), and must allow the key to be used by the CSNBDPC verb (ANY-USE or DPC-ONLY).
- If TDES-ECB or TDES-CBC is specified to return a DES-enciphered SMPIN message with a PIN block MAC, the key must be contained in a fixed-length DES key token and have a key type of SECMSG. In addition, the control vector must enable the encryption of PINs (SMPIN bit 19 = B’1’).
- If NOSCRIPT or its default is specified to not return an enciphered SMPIN message with a PIN block MAC, the `script_key_identifier_length` variable should be set to 0. If the length is greater than 0, this parameter must identify a valid DES or, beginning with release 4.4, a valid AES key-token that is otherwise ignored.

**`script_MAC_key_identifier_length`**

**Direction:** Input  
**Type:** Integer

The script_MAC_key_identifier_length parameter is a pointer to an integer variable containing the number of bytes of data in the `script_MAC_key_identifier` variable. If the `script_MAC_key_identifier` parameter identifies a key label, the length must be 64. Otherwise, for script selection algorithm and method keyword NOSCRIPT or its default, set the value to 0 or the length of a null key token, for AES-CBC set a maximum length of 725, and for TDES-CBC or TDES-ECB set the length to 64.

**`script_MAC_key_identifier`**

**Direction:** Input/Output  
**Type:** String

A pointer to a string variable containing an operational fixed-length DES key-token or the key label of such a record in DES key-storage. Beginning with Release 4.4, it can be a pointer to a string variable containing an operational variable-length AES key-token or the key label of such a record in AES key-storage. The type of key depends on the script selection algorithm and method of the rule array:
• If AES-CBC is specified to return an AES-enciphered SMPIN message with a PIN block MAC, the key must be contained in a variable-length symmetric key-token. The key must have a token algorithm of AES and a key type of MAC. In addition, the key usage fields must have the MAC operation set so that the key can be used for generate (GENERATE or GENONLY), and the MAC mode must be CMAC.

• If TDES-ECB or TDES-CBC is specified to return a DES-enciphered SMPIN message with a PIN block MAC, the key must be double length and have a key type of MAC (generate is allowed). In addition, the control vector must have a subtype of ANY-MAC (bits 0-3 = 0000').

• If NOSCRIPT or its default is specified to not return an enciphered SMPIN message with a PIN block MAC, the script_key_identifier_length variable should be set to 0. If the length is greater than 0, this parameter must identify a valid DES or, beginning with Release 4.4, a valid AES key-token that is otherwise ignored.

new_PRW_key_identifier_length

Direction: Input
Type: Integer

Specifies the length in bytes of the new_PRW_key_identifier parameter. If the new_PRW_key_identifier contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

new_PRW_key_identifier

Direction: Input/Output
Type: String

The identifier of the key to verify the new PRW. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, and the key usage fields must indicate GENONLY, CMAc, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

OPIN_encryption_key_identifier_length

Direction: Input
Type: Integer

Specifies the length in bytes of the OPIN_encryption_key_identifier parameter. If the rule array indicates that no encrypted PIN block is to be returned, this value must be 0. If the OPIN_encryption_key_identifier contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

OPIN_encryption_key_identifier

Direction: Input/Output
Type: String

The identifier of the key to encrypt the new PIN block. The key identifier is an operational token or the key label of an operational token in key storage. If the OPIN_encryption_key_identifier_length is 0, this parameter is ignored. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.
**OEPB_MAC_key_identifier_length**

**Direction:** Input  
**Type:** Integer

Specifies the length in bytes of the `OEPB_MAC_key_identifier` parameter. If the rule array indicates that no encrypted PIN block is to be returned, this value must be 0. If the `OEPB_MAC_key_identifier` contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**OEPB_MAC_key_identifier**

**Direction:** Input/Output  
**Type:** String

The identifier of the key to generate the MAC of new PIN block. The key identifier is an operational token or the key label of an operational token in key storage. If the `OEPB_MAC_key_identifier_length` is 0, this parameter is ignored. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate CMAC, GENONLY, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**script_length**

**Direction:** Input/Output  
**Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the `script` variable. The value must be 0 if the script selection algorithm and method of the rule array specifies NOSCRIPT or its default. Otherwise, the value must be at least as long as the `card_script_data_length`.

**script**

**Direction:** Output  
**Type:** String

The encrypted output script. The length of the field must be at least as long as the input script.

**script_MAC_length**

**Direction:** Input/Output  
**Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the `script_MAC` variable. Set to 0 if script selection algorithm and method of the rule array specifies NOSCRIPT or its default. Otherwise, on input set the value to at least 8 for MAC length and presentation keyword MACLEN8, or at least 16 for MACLEN16. On output, the value is updated with the length of data returned in the `script_MAC` variable.

**script_MAC**

**Direction:** Output  
**Type:** String

A pointer to a string variable containing the MAC calculated on the script returned in the `script` variable. The value is left-aligned in the variable and is truncated on the right as needed to the length specified by the MAC length and presentation keyword.
**new_PIN_reference_value_length**

**Direction:** Input/Output  
**Type:** Integer

Specifies the length in bytes of the `new_PIN_reference_value` parameter. The value must be at least 16. On output, it is set to 16.

**new_PIN_reference_value**

**Direction:** Output  
**Type:** String

The 16-byte new PIN reference value of the new PIN block.

**new_PRW_random_number_length**

**Direction:** Input/Output  
**Type:** Integer

Specifies the length in bytes of the `new_PRW_random_number` parameter. The value must be at least 4. On output, it is set to 4.

**new_PRW_random_number**

**Direction:** Output  
**Type:** String

The 4-byte random number associated with the new PIN reference value.

**output_encrypted_PIN_block_length**

**Direction:** Input/Output  
**Type:** Integer

Specifies the length in bytes of the `output_encrypted_PIN_block` parameter. If the rule array indicates that no encrypted PIN block should be returned, this value must be 0. Otherwise, it must be at least 32. On output it is set to 32.

**output_encrypted_PIN_block**

**Direction:** Output  
**Type:** String

The 32-byte encrypted new PIN block. If the `output_encrypted_PIN_block_length` is 0, this parameter is ignored.

**PIN_block_MAC_length**

**Direction:** Input/Output  
**Type:** Integer

Specifies the length in bytes of the `PIN_block_MAC` parameter. If the rule_array indicates that no PIN block MAC should be returned, this value must be 0. Otherwise, it must be at least 8.

**PIN_block_MAC**

**Direction:** Output  
**Type:** String

The 8-byte MAC of the new encrypted PIN block. If the `PIN_block_MAC_length` is 0, this parameter is ignored.
Restrictions

The restrictions for CSNBDPC.

The following rule array keywords are not supported in releases before Release 4.4:

- Script selection algorithm and method keyword AES-CBC
- MAC ciphering method keyword CMAC
- MAC length and presentation keyword MACLEN16

The `script_MAC_key_identifier` parameter and the `script_key_identifier` parameter cannot identify an AES variable-length symmetric key-token in releases before Release 4.4.

Required commands

The required commands for CSNBDPC.

The DK PIN Change verb requires the **DK PIN Change** command (offset X'02C2') to be enabled in the active role.

Usage notes

The usage notes for CSNBDPC.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBDPCJ.

See "Building Java applications using the CCA JNI" on page 28.

Format

```java
public native void CSNBDPCJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber PAN_data_length,
    byte[] PAN_data,
    hikmNativeNumber card_p_data_length,
    byte[] card_p_data,
    hikmNativeNumber card_t_data_length,
    byte[] card_t_data,
    hikmNativeNumber cur_ISO1_PIN_block_length,
    byte[] cur_ISO1_PIN_block,
    hikmNativeNumber new_ISO1_PIN_block_length,
    byte[] new_ISO1_PIN_block,
    hikmNativeNumber card_script_data_length,
    byte[] card_script_data,
    hikmNativeNumber script_offset,
    hikmNativeNumber script_offset_field_length,
    hikmNativeNumber script_initialization_vector_length,
    byte[] script_initialization_vector,
    byte[] output_PIN_profile,
    hikmNativeNumber PIN_reference_value_length,
    byte[] PIN_reference_value,
    hikmNativeNumber PRW_random_number_length,
    byte[] PRW_random_number,
    hikmNativeNumber PRW_key_identifier_length,
);```

DK PIN Change (CSNBDPC)
DK PIN Change (CSNBDPC)

DK PIN Verify (CSNBDPV)

Use the DK PIN Verify verb to verify an ISO-1 format PIN in a transaction. The account, the card data, and the PRW are used to verify the PIN.

The input PIN is converted to PBF-0 format. A test PIN reference value (PRW) is created and that value is compared bit by bit to the input PRW.

Format

The format of CSNBDPV.

```
CSNBDPV(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    PAN_data_length,
    PAN_data,
    card_data_length,
    card_data,
    PIN_reference_value_length,
    PIN_reference_value,
    PRW_random_number_length,
    PRW_random_number,
    ISO_encrypted_PIN_block_length,
    ISO_encrypted_PIN_block,
    PRW_key_identifier_length,
    PRW_key_identifier,
    IPIN_encryption_key_identifier_length,
    IPIN_encryption_key_identifier
);`
Parameters
The parameters for CSNBDPV.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

rule_array_count
- Direction: Input
- Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 0.

rule_array
- Direction: Input
- Type: String array

There are no keywords for this service.

PAN_data_length
- Direction: Input
- Type: Integer

Specifies the length in bytes of the PAN_data parameter. The value must be in the range 10 - 19.

PAN_data
- Direction: Input
- Type: String

The PAN data which the PIN is associated. The full account number, including check digit, should be included.

card_data_length
- Direction: Input
- Type: Integer

Specifies the length in bytes of the card_data parameter. The value must be in the range 4 - 512.

card_data
- Direction: Input
- Type: String

The time-invariant card data (CDp) and the time-sensitive card data (CDt) which, together with the account number, specifies an individual card.

PIN_reference_value_length
- Direction: Input
- Type: Integer

Specifies the length in bytes of the PIN_reference_value parameter. This value must be 16.

PIN_reference_value
The 16-byte PIN reference value for comparison to the calculated value.

**PRW_random_number_length**

**Direction:** Input  
**Type:** Integer  

Specifies the length in bytes of the `PRW_random_number` parameter. The value must be 4.

**PRW_random_number**

**Direction:** Input  
**Type:** String  

The 4-byte random number associated with the PIN reference value.

**ISO_encrypted_PIN_block_length**

**Direction:** Input  
**Type:** Integer  

Specifies the length in bytes of the `ISO_encrypted_PIN_block` parameter. This value must be 8.

**ISO_encrypted_PIN_block**

**Direction:** Input  
**Type:** String  

The 8-byte encrypted PIN block in ISO-1 format.

**PRW_key_identifier_length**

**Direction:** Input  
**Type:** Integer  

Specifies the length in bytes of the `PRW_key_identifier` parameter. If the `PRW_key_identifier` contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**PRW_key_identifier**

**Direction:** Input/Output  
**Type:** String  

The identifier of the key to verify the PIN reference value. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, and the key usage fields must indicate VERIFY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is be returned encrypted under the current master key.

**IPIN_encryption_key_identifier_length**

**Direction:** Input  
**Type:** Integer  

Specifies the length in bytes of the `IPIN_encryption_key_identifier` parameter. If the `IPIN_encryption_key_identifier` contains a label, the length must be 64.
DK PIN Verify (CSNBDPV)

Otherwise, the value must be at least the actual token length, up to 725.

**IPIN_encryption_key_identifier**

**Direction:** Input/Output  
**Type:** String

The identifier of the key to decrypt the PIN_block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be DES and the key type must be IPINENC.

If the token supplied was encrypted under the old master key, the token is be returned encrypted under the current master key.

**Restrictions**

The restrictions for CSNBDPV.

None.

**Required commands**

The required commands for CSNBDPV.

The DK PIN Verify verb requires the DK PIN Verify command (offset X'02C1') to be enabled in the active role.

**Usage notes**

The usage notes for CSNBDPV.

None.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBDPVJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBDPVJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber PAN_data_length,
    byte[] PAN_data,
    hikmNativeNumber card_data_length,
    byte[] card_data,
    hikmNativeNumber PIN_reference_value_length,
    byte[] PIN_reference_value,
    hikmNativeNumber PRW_random_number_length,
    byte[] PRW_random_number,
    hikmNativeNumber ISO_encrypted_PIN_block_length,
    byte[] ISO_encrypted_PIN_block,
    hikmNativeNumber PRW_key_identifier_length,
    byte[] PRW_key_identifier,
    hikmNativeNumber IPIN_encryption_key_identifier_length,
    byte[] IPIN_encryption_key_identifier);```

652 Common Cryptographic Architecture Application Programmer’s Guide
DK PRW Card Number Update (CSNBDPNU)

Use the DK PRW Card Number Update verb to generate a PIN reference value (PRW) when a replacement card is being issued. The original primary account number (PAN) and PIN are used with new time-sensitive card data to generate the new PRW.

You can use this service to perform the following tasks:
- Generate a PRW that can be used to verify the PIN.
- Optionally, generate an encrypted PIN block in PBF-1 format to be stored for later use in personalizing replacement cards.

Format

The format of CSNBDPNU.

```
CSNBDPNU(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    card_p_data_length,
    card_p_data,
    card_t_data_length,
    card_t_data,
    encrypted_PIN_block_length,
    encrypted_PIN_block,
    PIN_block_MAC_length,
    PIN_block_MAC,
    PRW_key_identifier_length,
    PRW_key_identifier,
    IPIN_encryption_key_identifier_length,
    IPIN_encryption_key_identifier,
    IEPB_MAC_key_identifier_length,
    IEPB_MAC_key_identifier,
    OPIN_encryption_key_identifier_length,
    OPIN_encryption_key_identifier,
    OEPB_MAC_key_identifier_length,
    OEPB_MAC_key_identifier,
    PIN_reference_value_length,
    PIN_reference_value,
    PRW_random_number_length,
    PRW_random_number,
    new_encrypted_PIN_block_length,
    new_encrypted_PIN_block,
    new_PIN_block_MAC_length,
    new_PIN_block_MAC)
```

Parameters

The parameters for CSNBDPNU.

For the definitions of the `return_code, reason_code, exit_data_length,` and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

`rule_array_count`

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Integer</td>
</tr>
</tbody>
</table>
A pointer to an integer variable containing the number of elements in the
rule_array variable. This value must be 0 or 1.

**rule_array**

Direction: Input
Type: String array

Keywords that provide control information to the verb. The keywords are
left-aligned in an 8-byte field and padded on the right with blanks. The
keywords must be in contiguous storage. The rule_array keywords are
described in Table 185.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIN Block output selection keyword (One, optional)</td>
<td>Return an encrypted PIN block.</td>
</tr>
<tr>
<td>EPB</td>
<td>Do not return an encrypted PIN block (EPB). This is the default.</td>
</tr>
</tbody>
</table>

**card_p_data_length**

Direction: Input
Type: Integer

Specifies the length in bytes of the card_p_data parameter. The value must be
in the range 2 - 256.

**card_p_data**

Direction: Input
Type: String

The time-invariant card data (CDp), determined by the card issuer, which is
used to differentiate between multiple cards for one account.

**card_t_data_length**

Direction: Input
Type: Integer

Specifies the length in bytes of the card_t_data parameter. The value must be
in the range 2 - 256.

**card_t_data**

Direction: Input
Type: String

The time-sensitive card data, determined by the card issuer, which, together
with the account number and the card_p_data, specifies an individual card.

**encrypted_PIN_block_length**

Direction: Input
Type: Integer

Specifies the length in bytes of the encrypted_PIN_block parameter. The value
must be 32.

**encrypted_PIN_block**
Direction: Input
Type: String

The 32-byte input encrypted PIN block in PBF-1 format.

**PIN_block_MAC_length**

Direction: Input
Type: Integer

Specifies the length in bytes of the **PIN_block_MAC** parameter. The value must be 8.

**PIN_block_MAC**

Direction: Input
Type: String

The 8-byte CMAC of the encrypted PIN block.

**PRW_key_identifier_length**

Direction: Input
Type: Integer

Specifies the length in bytes of the **PRW_key_identifier** parameter. If the **PRW_key_identifier** contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**PRW_key_identifier**

Direction: Input/Output
Type: String

The identifier of the PRW generating key. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**IPIN_encryption_key_identifier_length**

Direction: Input
Type: Integer

Specifies the length in bytes of the **IPIN_encryption_key_identifier** parameter. If the **IPIN_encryption_key_identifier** contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**IPIN_encryption_key_identifier**

Direction: Input/Output
Type: String

The identifier of the key that encrypts the input PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate DECRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.
DK PRW Card Number Update (CSNBDPNU)

**IEPB_MAC_key_identifier_length**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Specifies the length in bytes of the **IEPB_MAC_key_identifier** parameter. If the **IEPB_MAC_key_identifier** contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**IEPB_MAC_key_identifier**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>String</td>
</tr>
</tbody>
</table>

The identifier of the CMAC verification key. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate CMAC, VERIFY, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**OPIN_encryption_key_identifier_length**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Specifies the length in bytes of the **OPIN_encryption_key_identifier** parameter. If the **OPIN_encryption_key_identifier** contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**OPIN_encryption_key_identifier**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>String</td>
</tr>
</tbody>
</table>

The identifier of the key to wrap the new PIN block. The key identifier is an operational token or the key label of an operational token in key storage. If the rule array indicates that no encrypted PIN block is to be returned, this value is ignored. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**OEPB_MAC_key_identifier_length**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Specifies the length in bytes of the **OEPB_MAC_key_identifier** parameter. If the rule array indicates that no encrypted PIN block MAC is to be returned, this value must be 0.

If the **OEPB_MAC_key_identifier** contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**OEPB_MAC_key_identifier**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>String</td>
</tr>
</tbody>
</table>

The identifier of the key to generate the CMAC of the new PRW. The key
identifier is an operational token or the key label of an operational token in key storage. If the rule array indicates that no encrypted PIN block MAC is to be returned, this parameter is ignored. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**PIN_reference_value_length**

Direction: Input/Output

Type: Integer

Specifies the length in bytes of the **PIN_reference_value** parameter. This value must be 16. On output, it is set to 16.

**PIN_reference_value**

Direction: Output

Type: String

The calculated 16-byte PIN reference value.

**PRW_random_number_length**

Direction: Input/Output

Type: Integer

Specifies the length in bytes of the **PRW_random_number** parameter. The value must be 4. On output, it is set to 4.

**PRW_random_number**

Direction: Output

Type: String

The 4-byte random number associated with the PIN reference value.

**new_encrypted_PIN_block_length**

Direction: Input/Output

Type: Integer

Specifies the length in bytes of the **new_encrypted_PIN_block** parameter. If the rule array indicates that no new encrypted PIN block should be returned, this parameter must be zero. Otherwise, the parameter should be at least 32.

**new_encrypted_PIN_block**

Direction: Output

Type: String

The new 32-byte encrypted PIN block. If the rule array indicates that no new encrypted PIN block should be returned, this parameter is ignored.

**new_PIN_block_MAC_length**

Direction: Input/Output

Type: Integer

Specifies the length in bytes of the **new_PIN_block_MAC** parameter. If the rule_array indicates that no new_PIN_block_MAC should be returned, this value must be zero. Otherwise, it must be at least 8.
DK PRW Card Number Update (CSNBDPNU)

new_PIN_block_MAC

Direction: Output
Type: String

The new 8-byte encrypted MAC of the new PIN block. If the rule array indicates that no new encrypted PIN block should be returned, this parameter is ignored.

Restrictions

The restrictions for CSNBDPNU.

None.

Required commands

The required commands for CSNBDPNU.

The DK PRW Card Number Update verb requires the DK PRW Card Number Update command (offset X'02C3') to be enabled in the active role.

Usage notes

The usage notes for CSNBDPNU.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBDPNUJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

public native void CSNBDPNUJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber card_p_data_length,
    byte[] card_p_data,
    hikmNativeNumber card_t_data_length,
    byte[] card_t_data,
    hikmNativeNumber encrypted_PIN_block_length,
    byte[] encrypted_PIN_block,
    hikmNativeNumber PIN_block_MAC_length,
    byte[] PIN_block_MAC,
    hikmNativeNumber PRW_key_identifier_length,
    byte[] PRW_key_identifier,
    hikmNativeNumber IPIN_encryption_key_identifier_length,
    byte[] IPIN_encryption_key_identifier,
    hikmNativeNumber IEPB_MAC_key_identifier_length,
    byte[] IEPB_MAC_key_identifier,
    hikmNativeNumber OPIN_encryption_key_identifier_length,
    byte[] OPIN_encryption_key_identifier,
    hikmNativeNumber OEPB_MAC_key_identifier_length,
    byte[] OEPB_MAC_key_identifier,
    hikmNativeNumber PIN_reference_value_length,
    byte[] PIN_reference_value,
    hikmNativeNumber PRW_random_number_length,
)
DK PRW Card Number Update (CSNBDPNU)

byte[] PRW_random_number,
hikmNativeNumber new_encrypted_PIN_block_length,
byte[] new_encrypted_PIN_block,
hikmNativeNumber new_PIN_block_MAC_length,
byte[] new_PIN_block_MAC;

DK PRW CMAC Generate (CSNBDPCG)

Use the DK PRW CMAC Generate verb to generate a message authentication code (MAC) over specific values involved in an account number change transaction. The input includes the current and new PAN and card data and the PIN reference value.

The output of this service is used as input to the DK PAN Modify in Transaction callable service, which will create the new PIN reference value (PRW) to be used to verify the PIN.

Format

The format of CSNBDPCG.

```
CSNBDPCG(  
  return_code,  
  reason_code,  
  exit_data_length,  
  exit_data,  
  rule_array_count,  
  rule_array,  
  current_PAN_data_length,  
  current_PAN_data,  
  new_PAN_data_length,  
  new_PAN_data,  
  current_card_data_length,  
  current_card_data,  
  new_card_data_length,  
  new_card_data,  
  PIN_reference_value_length,  
  PIN_reference_value,  
  CMAC_FUS_key_identifier_length,  
  CMAC_FUS_key_identifier,  
  CMAC_FUS_length,  
  CMAC_FUS)
```

Parameters

The parameters for CSNBDPCG.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see "Parameters common to all verbs" on page 22.

rule_array_count

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 0.

rule_array

Direction: Input
Type: String array
There are no keywords for this service.

**current_PAN_data_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the length in bytes of the `current_PAN_data` variable. This value must be in the range 10 - 19.

**current_PAN_data**

- **Direction:** Input
- **Type:** String

The current PAN data. The full account number, including check digit, should be included.

**new_PAN_data_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the length in bytes of the `new_PAN_data` variable. This value must be in the range 10 - 19.

**new_PAN_data**

- **Direction:** Input
- **Type:** String

The new PAN data. The full account number, including check digit, should be included.

**current_card_data_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the length in bytes of the `current_card_data` variable. This value must be in the range 4 - 512.

**current_card_data**

- **Direction:** Input/Output
- **Type:** String

The current card data, determined by the card issuer.

**new_card_data_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the length in bytes of the `new_card_data` variable. This value must be in the range 4 - 512.

**new_card_data**

- **Direction:** Input/Output
- **Type:** String

The new card data, determined by the card issuer.
PIN_reference_value_length

Direction: Input
Type: Integer

Specifies the length in bytes of the PIN_reference_value parameter. This value must be 16. On output, PIN_reference_value_length is set to 16.

PIN_reference_value

Direction: Input
Type: String

The 16-byte PIN reference value of the current PIN.

CMAC_FUS_key_identifier_length

Direction: Input
Type: Integer

Specifies the length in bytes of the CMAC_FUS_key_identifier variable. If the CMAC_FUS_key_identifier contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

CMAC_FUS_key_identifier

Direction: Input
Type: String

The identifier of the key to generate the MAC. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate GENONLY, CMAC, and DKPINAD2.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

CMAC_FUS_length

Direction: Input/Output
Type: Integer

A pointer to an integer variable that specifies the length in bytes of the CMAC_FUS variable. This value must be in the range 8 - 16.

CMAC_FUS

Direction: Output
Type: String

The MAC of the current and new PANs and card data strings.

Restrictions

The restrictions for CSNBDPCG.

None.

Required commands

The required commands for CSNBDPCG.

The DK PRW CMAC Generate verb requires the DK PRW CMAC Generate command (offset X'02C4') to be enabled in the active role.
DK PRW CMAC Generate (CSNBDPCG)

Usage notes

The usage notes for CSNBDPCG.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBDPCGJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBDPCGJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber current_PAN_data_length,
    byte[] current_PAN_data,
    hikmNativeNumber new_PAN_data_length,
    byte[] new_PAN_data,
    hikmNativeNumber current_card_data_length,
    byte[] current_card_data,
    hikmNativeNumber new_card_data_length,
    byte[] new_card_data,
    hikmNativeNumber PIN_reference_value_length,
    byte[] PIN_reference_value,
    hikmNativeNumber CMAC_FUS_key_identifier_length,
    byte[] CMAC_FUS_key_identifier,
    hikmNativeNumber CMAC_FUS_length,
    byte[] CMAC_FUS);
```

DK Random PIN Generate (CSNBDRPG)

Use the DK Random PIN Generate verb to generate a PIN and a PIN reference value using the random process. After the PIN is generated, a PIN reference value (PRW) is created. The PIN reference value is used to verify the PIN in other processes. An optional encrypted PIN block is generated for printing.

**Note:** If the generated PIN appears in the weak PIN table, the generation process is modified and re-tried until a valid PIN is generated.

You can use this service to perform the following tasks:

- Generate an encrypted PIN block in PBF-1 format with a PIN print key to be printed on a PIN mailer.
- Generate a PIN reference value which can be used to verify the PIN.
- Optionally, generate an encrypted PIN block in PBF-1 format to be stored for later use in personalizing replacement cards, along with a verifying CMAC over the encrypted block and additional card data.
DK Random PIN Generate (CSNBDRPG)

Format

The format of CSNBDRPG.

```c
CSNBDRPG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    PAN_data_length,
    PAN_data,
    card_p_data_length,
    card_p_data,
    card_t_data_length,
    card_t_data,
    PIN_length,
    PRW_key_identifier_length,
    PRW_key_identifier,
    PIN_print_key_identifier_length,
    PIN_print_key_identifier,
    OPIN_encryption_key_identifier_length,
    OPIN_encryption_key_identifier,
    OEPB_MAC_key_identifier_length,
    OEPB_MAC_key_identifier,
    PIN_reference_value_length,
    PIN_reference_value,
    PRW_random_number_length,
    PRW_random_number,
    PIN_print_block_length,
    PIN_print_block,
    encrypted_PIN_block_length,
    encrypted_PIN_block,
    PIN_block_MAC_length,
    PIN_block_MAC)
```

Parameters

The parameters for CSNBDRPG.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

**rule_array_count**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Integer</td>
</tr>
</tbody>
</table>

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0 or 1.

**rule_array**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>String array</td>
</tr>
</tbody>
</table>

Keywords that provide control information to the verb. The keywords are left-aligned in an 8-byte field and padded on the right with blanks. The keywords must be in contiguous storage. The `rule_array` keywords are described in Table 186 on page 664.
Table 186. Keywords for DK Random PIN Generate control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIN Block output selection keyword</td>
<td>(One, optional) Return an encrypted PIN block.</td>
</tr>
<tr>
<td>EPB</td>
<td>Return an encrypted PIN block.</td>
</tr>
<tr>
<td>NOEPB</td>
<td>Do not return an encrypted PIN block (EPB). This is the default.</td>
</tr>
</tbody>
</table>

**PAN_data_length**
- Direction: Input
- Type: Integer

Specifies the length in bytes of the **PAN_data** parameter. The value must be in the range 10 - 19.

**PAN_data**
- Direction: Input
- Type: String

The personal account number in character form to which the PIN is associated. The primary account number, including check digit, should be included.

**card_p_data_length**
- Direction: Input
- Type: Integer

Specifies the length in bytes of the **card_p_data** parameter. The value must be in the range 2 - 256.

**card_p_data**
- Direction: Input
- Type: String

The time-invariant card data (CDp), determined by the card issuer, which is used to differentiate between multiple cards for one account.

**card_t_data_length**
- Direction: Input
- Type: Integer

Specifies the length in bytes of the **card_t_data** parameter. The value must be in the range 2 - 256.

**card_t_data**
- Direction: Input
- Type: String

The time-sensitive card data, determined by the card issuer, which, together with the account number and the **card_p_data**, specifies an individual card.

**PIN_length**
- Direction: Input
- Type: Integer

 Specifies the length of the PIN to be generated. This value must be in the range 4 - 12.
**PRW_key_identifier_length**

**Direction:** Input
**Type:** Integer

Specifies the length in bytes of the `PRW_key_identifier` parameter. If the `PRW_key_identifier` contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**PRW_key_identifier**

**Direction:** Input/Output
**Type:** String

The identifier of the key to verify the PRW of the current PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, and the key usage fields must indicate VERIFY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**PIN_print_key_identifier_length**

**Direction:** Input
**Type:** Integer

Specifies the length in bytes of the `PIN_print_key_identifier` parameter. If the `PIN_print_key_identifier` contains a label, the value must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**PIN_print_key_identifier**

**Direction:** Input/Output
**Type:** String

The identifier of the key to wrap the PIN for printing. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOPP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**OPIN_encryption_key_identifier_length**

**Direction:** Input
**Type:** Integer

Specifies the length in bytes of the `OPIN_encryption_key_identifier` parameter. If the `OPIN_encryption_key_identifier` contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**OPIN_encryption_key_identifier**

**Direction:** Input/Output
**Type:** String

The identifier of the key to wrap the new PIN block. The key identifier is an operational token or the key label of an operational token in key storage. If the rule array indicates that no encrypted PIN block is to be returned, this value is ignored. The key algorithm of this key must be AES, the key type must be
DK Random PIN Generate (CSNBDRPG)

PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**OEPB_MAC_key_identifier_length**

| Direction: Input |
| Type: Integer |

Specifies the length in bytes of the **OEPB_MAC_key_identifier** parameter. If the rule array indicates that no encrypted PIN block MAC is to be returned, this value must be 0.

If the **OEPB_MAC_key_identifier** contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**OEPB_MAC_key_identifier**

| Direction: Input/Output |
| Type: String |

The identifier of the key to generate the CMAC of the new PRW. The key identifier is an operational token or the key label of an operational token in key storage. If the rule array indicates that no encrypted PIN block MAC is to be returned, this parameter is ignored. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate GENONLY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**PIN_reference_value_length**

| Direction: Input/Output |
| Type: Integer |

Specifies the length in bytes of the **PIN_reference_value** parameter. This value must be 16. On output, it is set to 16.

**PIN_reference_value**

| Direction: Output |
| Type: String |

The calculated 16-byte PIN reference value.

**PRW_random_number_length**

| Direction: Input/Output |
| Type: Integer |

Specifies the length in bytes of the **PRW_random_number** parameter. The value must be 4. On output, it is set to 4.

**PRW_random_number**

| Direction: Output |
| Type: String |

The 4-byte random number associated with the PIN reference value.

**PIN_print_block_length**
DK Random PIN Generate (CSNBDRPG)

**Direction:** Input/Output  
**Type:** Integer

Specifies the length in bytes of the `PIN_print_block` parameter. The value must be at least 32. On output, it is set to 32.

**PIN_print_block**

**Direction:** Output  
**Type:** String

The 32-byte encrypted PIN block to be passed to the PIN mailer function.

**encrypted_PIN_block_length**

**Direction:** Input/Output  
**Type:** Integer

Specifies the length in bytes of the `encrypted_PIN_block` parameter. If the rule array indicates that no encrypted PIN block should be returned, this value must be 0. Otherwise, it should be at least 32.

**encrypted_PIN_block**

**Direction:** Output  
**Type:** String

The 32-byte encrypted PIN block in PBF-1 format. This parameter is ignored if no encrypted PIN block is returned.

**PIN_block_MAC_length**

**Direction:** Input/Output  
**Type:** Integer

Specifies the length in bytes of the `PIN_block_MAC` parameter. If the rule_array indicates that no PIN block MAC should be returned, this value must be 0. Otherwise, it must be at least 8.

**PIN_block_MAC**

**Direction:** Output  
**Type:** String

The 8-byte CMAC of the encrypted PIN block. This parameter is ignored if no encrypted PIN block is returned.

**Restrictions**

The restrictions for CSNBDRPG.

None.

**Required commands**

The required commands for CSNBDRPG.

The DK Random PIN Generate verb requires the **DK Random PIN Generate** command (offset X'02C0') to be enabled in the active role.
DK Random PIN Generate (CSNBDRPG)

Usage notes

The usage notes for CSNBDRPG.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBDRPGJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBDRPGJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber PAN_data_length,
    byte[] PAN_data,
    hikmNativeNumber card_p_data_length,
    byte[] card_p_data,
    hikmNativeNumber card_t_data_length,
    byte[] card_t_data,
    hikmNativeNumber PIN_length,
    hikmNativeNumber PRW_key_identifier_length,
    byte[] PRW_key_identifier,
    hikmNativeNumber PIN_print_key_identifier_length,
    byte[] PIN_print_key_identifier,
    hikmNativeNumber OPIN_encryption_key_identifier_length,
    byte[] OPIN_encryption_key_identifier,
    hikmNativeNumber OEPB_MAC_key_identifier_length,
    byte[] OEPB_MAC_key_identifier,
    hikmNativeNumber PIN_reference_value_length,
    byte[] PIN_reference_value,
    hikmNativeNumber PRW_random_number_length,
    byte[] PRW_random_number,
    hikmNativeNumber PIN_print_block_length,
    byte[] PIN_print_block,
    hikmNativeNumber encrypted_PIN_block_length,
    byte[] encrypted_PIN_block,
    hikmNativeNumber PIN_block_MAC_length,
    byte[] PIN_block_MAC);
```

DK Regenerate PRW (CSNBDRP)

The DK Regenerate PRW verb generates a new PIN reference value for a changed account number.

You can use this service to perform the following tasks:

- Generate a PIN reference value over the existing PIN and new PAN, which can be used to verify transactions.
- Generate an encrypted PIN block in PBF-1 format to be stored for later use in personalization of smart cards.
Format

The format of CSNBDRP.

```c
CSNBDRP(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    card_p_data_length,
    card_p_data,
    card_t_data_length,
    card_t_data,
    encrypted_PIN_block_length,
    encrypted_PIN_block,
    PIN_block_MAC_length,
    PIN_block_MAC,
    PRW_key_identifier_length,
    PRW_key_identifier,
    IPIN_encryption_key_identifier_length,
    IPIN_encryption_key_identifier,
    IEPB_MAC_key_identifier_length,
    IEPB_MAC_key_identifier,
    OPIN_encryption_key_identifier_length,
    OPIN_encryption_key_identifier,
    OEPB_MAC_key_identifier_length,
    OEPB_MAC_key_identifier,
    PIN_reference_value_length,
    PIN_reference_value,
    PRW_random_number_length,
    PRW_random_number,
    new_encrypted_PIN_block_length,
    new_encrypted_PIN_block,
    new_PIN_block_MAC_length,
    new_PIN_block_MAC)
```

Parameters

The parameters for CSNBDRP.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** String array

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0.

**rule_array**

- **Direction:** Input
- **Type:** String array

There are no keywords for this service.

**card_p_data_length**

- **Direction:** Input
- **Type:** Integer
DK Regenerate PRW (CSNBDRP)

Specifies the length in bytes of the card_p_data parameter. The value must be in the range 2 - 256.

card_p_data
Direction: Input
Type: String

The time-invariant card data (CDp), determined by the card issuer, which is used to differentiate between multiple cards for one account.

card_t_data_length
Direction: Input
Type: Integer

Specifies the length in bytes of the card_t_data parameter. The value must be in the range 2 - 256.

card_t_data
Direction: Input
Type: String

The time-sensitive card data, determined by the card issuer, which, together with the account number and the card_p_data, specifies an individual card.

encrypted_PIN_block_length
Direction: Input/Output
Type: Integer

Specifies the length in bytes of the encrypted_PIN_block parameter. If the rule array indicates that no encrypted PIN block should be returned, this value must be 0. Otherwise, it should be at least 32.

encrypted_PIN_block
Direction: Input
Type: String

The 32-byte encrypted PIN block in PBF-1 format. This parameter is ignored if no encrypted PIN block is returned.

PIN_block_MAC_length
Direction: Input/Output
Type: Integer

Specifies the length in bytes of the PIN_block_MAC parameter. If the rule array indicates that no PIN block MAC should be returned, this value must be 0. Otherwise, it must be at least 8.

PIN_block_MAC
Direction: Input
Type: String

The 8-byte CMAC of the encrypted PIN block. This parameter is ignored if no encrypted PIN block is returned.

PRW_key_identifier_length
Direction: Input
DK Regenerate PRW (CSNBDPRP)

Type: Integer

Specifies the length in bytes of the PRW_key_identifier parameter. If PRW_key_identifier contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

PRW_key_identifier

Direction: Input/Output
Type: String

The identifier of the key to verify the PRW of the current PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPRW, and the key usage fields must indicate VERIFY, CMAC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

IPIN_encryption_key_identifier_length

Direction: Input
Type: Integer

Specifies the length in bytes of the IPIN_encryption_key_identifier parameter. If IPIN_encryption_key_identifier contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

IPIN_encryption_key_identifier

Direction: Input
Type: String

The identifier of the key to decrypt the PIN block containing the current PIN. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate DECRYPT, CBC, and DKPINAD1.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

IEPB_MAC_key_identifier_length

Direction: Input
Type: Integer

Specifies the length in bytes of the IEPB_MAC_key_identifier parameter. If IEPB_MAC_key_identifier contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

IEPB_MAC_key_identifier

Direction: Input/Output
Type: String

The identifier of the key to verify MAC of the inbound encrypted PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate CMAC, VERIFY, and DKPINAD1.
If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**OPIN_encryption_key_identifier_length**

**Direction:** Input
**Type:** Integer

Specifies the length in bytes of the **OPIN_encryption_key_identifier** parameter. If **OPIN_encryption_key_identifier** contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**OPIN_encryption_key_identifier**

**Direction:** Input/Output
**Type:** String

The identifier of the key to encrypt the new PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be PINPROT, and the key usage fields must indicate ENCRYPT, CBC, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**OEPB_MAC_key_identifier_length**

**Direction:** Input
**Type:** Integer

Specifies the length in bytes of the **OEPB_MAC_key_identifier** parameter. If **OEPB_MAC_key_identifier** contains a label, the length must be 64. Otherwise, the value must be at least the actual token length, up to 725.

**OEPB_MAC_key_identifier**

**Direction:** Input/Output
**Type:** String

The identifier of the key to generate the MAC of the new encrypted PIN block. The key identifier is an operational token or the key label of an operational token in key storage. The key algorithm of this key must be AES, the key type must be MAC, and the key usage fields must indicate CMAC, GENONLY, and DKPINOP.

If the token supplied was encrypted under the old master key, the token is returned encrypted under the current master key.

**PIN_reference_value_length**

**Direction:** Input
**Type:** Integer

Specifies the length in bytes of the **PIN_reference_value** parameter. The value must be 16. On output, it is set to 16.

**PIN_reference_value**

**Direction:** Input
**Type:** String

The 16-byte PIN reference value for comparison to the calculated value.

**PRW_random_number_length**
DK Regenerate PRW (CSNBDRP)

**Direction:** Input
**Type:** Integer

Specifies the length in bytes of the PRW_random_number parameter. The value must be 4. On output, it is set to 4.

**PRW_random_number**

**Direction:** Input
**Type:** String

The 4-byte random number associated with the PIN reference value.

**new_encrypted_PIN_block_length**

**Direction:** Input/Output
**Type:** Integer

Specifies the length in bytes of the new_encrypted_PIN_block parameter. The value must be at least 32. On output, it is set to 32.

**new_encrypted_PIN_block**

**Direction:** Output
**Type:** String

The 32-byte encrypted new PIN block.

**new_PIN_block_MAC_length**

**Direction:** Input/Output
**Type:** Integer

Specifies the length in bytes of the new_PIN_block_MAC parameter. The value must be at least 8.

**new_PIN_block_MAC**

**Direction:** Output
**Type:** String

The 8-byte MAC of the new encrypted PIN block.

**Restrictions**

The restrictions for CSNBDRP.

None.

**Required commands**

The required commands for CSNBDRP.

The DK Regenerate PRW verb requires the DK Regenerate PRW command (offset X'02C8') to be enabled in the active role.

**Usage notes**

The usage notes for CSNBDRP.

None.
JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBDRPJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBDRPJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber card_p_data_length,
    byte[] card_p_data,
    hikmNativeNumber card_t_data_length,
    byte[] card_t_data,
    hikmNativeNumber encrypted_PIN_block_length,
    byte[] encrypted_PIN_block,
    hikmNativeNumber PIN_block_MAC_length,
    byte[] PIN_block_MAC,
    hikmNativeNumber PRW_key_identifier_length,
    byte[] PRW_key_identifier,
    hikmNativeNumber IPIN_encryption_key_identifier_length,
    byte[] IPIN_encryption_key_identifier,
    hikmNativeNumber IEPB_MAC_key_identifier_length,
    byte[] IEPB_MAC_key_identifier,
    hikmNativeNumber OPIN_encryption_key_identifier_length,
    byte[] OPIN_encryption_key_identifier,
    hikmNativeNumber OEPB_MAC_key_identifier_length,
    byte[] OEPB_MAC_key_identifier,
    hikmNativeNumber PIN_reference_value_length,
    byte[] PIN_reference_value,
    hikmNativeNumber PRW_random_number_length,
    byte[] PRW_random_number,
    hikmNativeNumber new_encrypted_PIN_block_length,
    byte[] new_encrypted_PIN_block,
    hikmNativeNumber new_PIN_block_MAC_length,
    byte[] new_PIN_block_MAC);
```
Chapter 14. Using digital signatures

Use the CCA verbs described in this topic to support digital signatures to authenticate messages.

- “Digital Signature Generate (CSNDDSG)”
- “Digital Signature Verify (CSNDDSV)” on page 680

Digital Signature Generate (CSNDDSG)

This verb generates a digital signature from hashed input data using an RSA or ECC private key.

This verb supports the following signature formatting methods:

- ANSI X9.62 (ECDSA)
- ANSI X9.31 (RSA)
- ISO 9796-1 (RSA)
- RSA DSI PKCS 1.0 and 1.1 (RSA)
- RSA PKCS-PSS (RSA)
- Padding on the left with zeros (RSA)

Note: The maximum signature length is 512 bytes (4096 bits).

The input text should have been previously hashed using either the One-Way Hash verb or the MDC Generate verb. If the signature formatting algorithm specifies ANSI X9.31, you must specify the hash algorithm used to hash the text (SHA-1 or RPMD-160). See “Formatting hashes and keys in public-key cryptography” on page 1044.

You select the method of formatting the text through the rule_array parameter.

If the PKA_private_key_identifier specifies an RSA private key, you select the method of formatting the text through the rule_array parameter. If the PKA_private_key_identifier specifies an ECC private key, the ECC signature generated is according to ANSI X9.30.

Note: For PKCS the message digest and the message-digest algorithm identifier are combined into an ASN.1 value of type DigestInfo, which is BER-encoded to give an octet string D (see Table 187 on page 676). D is the text string supplied in the hash variable.
Format

The format of CSNDDSG.

```c
CSNDDSG(    return_code,    
           reason_code,    
           exit_data_length,    
           exit_data,    
           rule_array_count,    
           rule_array,    
           PKA_private_key_identifier_length,    
           PKA_private_key_identifier,    
           hash_length,    
           hash,    
           signature_field_length,    
           signature_bit_length,    
           signature_field)
```

Parameters

The parameter definitions for CSNDDSG.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- Direction: Input
- Type: Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0, 1, 2, or 3.

**rule_array**

- Direction: Input
- Type: String array

Keywords that provide control information to the verb. A keyword specifies the method for calculating the digital signature. Each keyword is left-aligned in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage. The `rule_array` keywords are described in Table 187.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ISO-9796</strong></td>
<td>Calculate the digital signature on the <code>hash</code> according to ISO-9796-1. Any hash method is allowed. This is the default.</td>
</tr>
<tr>
<td><strong>PKCS-1.0</strong></td>
<td>Calculate the digital signature on the BER-encoded ASN.1 value of the type DigestInfo containing the hash according to the RSA Data Security, Inc. Public Key Cryptography Standards #1 block type 00. The text must have been hashed and BER-encoded before input to this service.</td>
</tr>
<tr>
<td><strong>PKCS-1.1</strong></td>
<td>Calculate the digital signature on the BER-encoded ASN.1 value of the type DigestInfo containing the hash according to the RSA Data Security, Inc. Public Key Cryptography Standards #1 block type 01. The text must have been hashed and BER-encoded before input to this service.</td>
</tr>
<tr>
<td><strong>PKCS-PSS</strong></td>
<td>Specifies to format the digital signature on the string supplied in the hash variable as defined in the RSA PKCS #1 v2.2 standard for the RSASSA-PSS signature scheme.</td>
</tr>
<tr>
<td><strong>ZERO-PAD</strong></td>
<td>Format the hash by padding it on the left with binary zeros to the length of the RSA key modulus. Any supported hash function is allowed.</td>
</tr>
</tbody>
</table>
Table 187. Keywords for Digital Signature Generate control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X9.31</td>
<td>Format according to the ANSI X9.31 standard. The input text must have been previously hashed with one of the hash algorithms specified below.</td>
</tr>
<tr>
<td><strong>Data input type</strong> (One, optional.)</td>
<td></td>
</tr>
<tr>
<td>HASH</td>
<td>Process the text supplied in the hash variable as a hash digest that is to be digitally signed by the verb. This is the default.</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>Process the text supplied in the hash variable as a message that is to be hashed using the specified hashing-method to produce a hash digest, then use the resultant hash digest to produce a digital signature. The text to be hashed must be a multiple of eight bits and byte-aligned.</td>
</tr>
<tr>
<td><strong>Hash method specification</strong> (One required, if any of the X9.31, MESSAGE, or PKCS-PSS keywords are specified. Otherwise such a specification is not allowed.)</td>
<td></td>
</tr>
<tr>
<td>RPMD-160</td>
<td>Hash the input text from the hash parameter using the RIPEMD-160 hash method.</td>
</tr>
<tr>
<td>SHA-1</td>
<td>Hash the input text from the hash parameter using the SHA-1 hash method.</td>
</tr>
<tr>
<td>SHA-224</td>
<td>Hash the input text from the hash parameter using the SHA-224 hash method. Not valid with X9.31.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Hash the input text from the hash parameter using the SHA-256 hash method.</td>
</tr>
<tr>
<td>SHA-384</td>
<td>Hash the input text from the hash parameter using the SHA-384 hash method.</td>
</tr>
<tr>
<td>SHA-512</td>
<td>Hash the input text from the hash parameter using the SHA-512 hash method.</td>
</tr>
<tr>
<td><strong>Token algorithm</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>ECDSA</td>
<td>Generate an ECC digital signature. This keyword was introduced with CCA 4.1.0. When specified, this is the only keyword permitted in the rule_array.</td>
</tr>
<tr>
<td>RSA</td>
<td>Generate an RSA digital signature. This is the default.</td>
</tr>
</tbody>
</table>

**PKA_private_key_identifier_length**

Direction: Input  
Type: Integer  

The length of the PKA_private_key_identifier field. The maximum size is 3500 bytes.

**PKA_private_key_identifier**

Direction: Input  
Type: String  

An internal token or label of the RSA private key or retained key. If the signature format is X9.31, the modulus of the RSA key must have a minimum length of 1024 bits or greater. If the signature algorithm is ECDSA, this parameter must be a token or label of an ECC private key.

If the digital-signature hash formatting method is PKCS-PSS, the RSA key-token must have a private key section ID of X'30' or X'31'. Refer to “PKA Key Translate (CSNDPKT)” on page 713 to translate an RSA key-token as needed to the format desired.

**hash_length**

Direction: Input  
Type: Integer  

The length of the hash parameter in bytes. It must be the exact length of the text to sign. The maximum size is 512 bytes. If you specify ZERO-PAD in the rule_array parameter, the length is restricted to 36 bytes unless the RSA key is a
Digital Signature Generate (CSNDDSG)

signature-only key. In this case, the maximum length is 512 bytes.

The hash length limit is controlled by an access control point. Only RSA key
management keys are affected by this access control point. The limit for RSA
signature-only keys is 512 bytes. This access control point is always disabled in
the default role. You must have a TKE workstation to enable it.

If you specify PKCS-PSS, the first four bytes must contain the salt length. The
remaining bytes of the parameter must contain the hash or message. The value
will be $4 + \text{length of the hash or message}$.

If MESSAGE is not specified, the maximum length is 68 bytes. If MESSAGE is
specified, the value is the length of the message to be hashed. The maximum
length is in the range 64 bytes - 32 MB.

**hash**

**Direction:** Input

**Type:** String

A pointer to a string variable containing the application-supplied text on which
to generate the signature.

The parameter can identify the hash to be signed or the message to be hashed
and signed, depending on the specified rule-array keyword. If the MESSAGE
keyword is specified, the input data is treated as data that must be hashed by
the verb and then signed. Otherwise, if HASH is specified, the input data is
treated as hash digest that is to be signed.

For PKCS formatting, it must be BER-encoded as previously described. For
X9.31, the hash algorithms must have been either SHA-1 or RPMD-160. See
the rule_array parameter for more information.

If PKCS-PSS is specified, the data variable includes an additional 4 byte field
that contains the salt length in big endian format. This length must be
 prepended to the hash or message. To allow use of a salt length less than the
hash length, but not zero, enable the Allow Small Salt command (offset
X'033C') in the active role. A salt length value can be either 0, or be in the
range from 20 through the size allowed by the RSA modulus in bytes. The size
of the data to be signed is still governed by the size of the RSA modulus. The
salt length cannot exceed 490 bytes. The modulus size and hash length affect
the maximum salt length for a given key modulus size and specified hash. The
maximum salt length equals $\text{modulus-size}/8 - \text{hash-length} - 2$. For example,
with a 4096 bit modulus key and SHA-1 hash, the maximum salt length
becomes $(4096/8) - 20 - 2 = 490$.

Also, if PKCS-PSS is specified, you must specify the hashing method
specification keyword indicating the hash algorithm used for the hash data
identified by the data parameter.

**signature_field_length**

**Direction:** Input/Output

**Type:** Integer

The length in bytes of the signature_field to contain the generated digital
signature. The maximum size is 512 bytes.

For RSA, this must be at least the RSA modulus size (rounded up to a multiple
of 32 bytes for the X9.31 signature format, or one byte for all other signature
formats).
For RSA, this field is updated with the minimum byte length of the digital signature.

For the ECDSA signature algorithm, R concatenated with S is the digital signature. The maximum output value will be 1042 bits (131 bytes). The size of the signature is determined by the size of P. Both R and S will have size P. For prime curves, the maximum size is 2 * 521 bits. For Brainpool curves, the maximum size is 2 * 512 bits.

**signature_bit_length**

**Direction:** Output  
**Type:** Integer

The bit length of the digital signature generated. For ISO-9796 this is 1 less than the modulus length. For other RSA processing methods, this is the modulus length.

**signature_field**

**Direction:** Output  
**Type:** String

The digital signature generated is returned in this field. The digital signature is in the low-order bits (right-aligned) of a string whose length is the minimum number of bytes that can contain the digital signature. This string is left-aligned within the `signature_field`. Any unused bytes to the right are undefined.

**Restrictions**

The restrictions for CSNDDSG.

Although ISO-9796 does not require the input hash to be an integral number of bytes in length, this verb requires you to specify the `hash_length` in bytes.

X9.31 requires the RSA token to have a minimum modulus bit length of 1024 bits, and the length must also be a multiple of 256 bits (or 32 bytes).

The length of the `hash` parameter in bytes must be the exact length of the text to sign. The maximum size is 256 bytes. If you specify `ZERO-PAD` in the `rule_array` parameter, the length is restricted to 36 bytes unless the RSA key is a signature only key, then the maximum length is 256 bytes.

The hash length limit is controlled by an access control point. If OFF (disabled), the maximum hash length limit for `ZERO-PAD` is the modulus length of the PKA private key. If ON (enabled), the maximum hash length limit for `ZERO-PAD` is 36 bytes. Only RSA key management keys are affected by this access control point. The limit for RSA signature use only keys is 256 bytes. This new access control point is always disabled in the default role. You must have a TKE workstation to enable it.

**Required commands**

The required commands for CSNDDSG.

This verb requires the Digital Signature Generate command (offset X’0100’) to be enabled in the active role.
Digital Signature Generate (CSNDDSG)

With the use of the **DSG ZERO-PAD unrestricted hash length** command (offset X'030C'), the hash-length restriction does not apply when using ZERO-PAD formatting.

**Usage notes**

The usage notes for CSNDDSG.

None.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNDDSGJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNDDSGJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber PKA_private_key_identifier_length,
    byte[] PKA_private_key_identifier,
    hikmNativeNumber hash_length,
    byte[] hash,
    hikmNativeNumber signature_field_length,
    hikmNativeNumber signature_bit_length,
    byte[] signature_field);
```

Digital Signature Verify (CSNDDSV)

This verb verifies a digital signature using an RSA or ECC public key in a CCA key token or in an X.509 certificate.

This verb verifies digital signatures generated with these methods:

- ANSI X9.62 (ECDSA)
- ANSI X9.31 (RSA)
- ISO 9796-1 (RSA)
- RSA DSI PKCS 1.0 and 1.1 (RSA)
- RSA PKCS-PSS (RSA)
- Padding on the left with zeros (RSA)

This verb can use the RSA or ECC public key, depending on the digital signature algorithm used to generate the signature.

This verb can also use the public keys that are contained in trusted blocks, regardless of whether the block also contains rules to govern its use when generating or exporting keys with the Remote Key Export verb. The format of the trusted block enables Digital Signature Verify to distinguish it from other RSA key tokens, and therefore no special rule array keyword or other parameters are required in order to indicate that the trusted block is being used. However, if the Digital Signature Generate verb is used with the TPK-ONLY keyword in the `rule_array`, an error will occur if the `PKA_private_key_identifier` does not contain a trusted block.
If the input text is specified to be a hash, then it should have been previously hashed. You can use the One-Way Hash verb. See also “Formatting hashes and keys in public-key cryptography” on page 1044.

An X.509 certificate couples an identity to a PKA (RSA or ECC) public key. As of CCA 6.0, such an X.509 certificate can be used by the CSNDDSV verb instead of a PKA (RSA or ECC) public key token. Because X.509 certificates require authentication against a PKI, the TKE workstation must be used to install the trust roots that will be used to validate your operational X.509 certificates at run-time.

If the input text is specified to be a message (by the MESSAGE key word), or if PKCS-PSS is specified, then apply a required hashing-method specification keyword. If the PKCS-PSS keyword is specified, optionally specify a signature checking rule to indicate whether the derived salt length is an exact match to the input salt length (the default), or is not an exact match (requires special authorization).

See also “Formatting hashes and keys in public-key cryptography” on page 1044.

Note: The maximum signature length is 256 bytes (2048 bits).

EC signature verification update

Beginning with CCA Release 5.0 of CSNDDSV for the CEX5C, checking of EC signatures is strengthened with new hardware support. For most invalid signatures the typical response of return code 4, reason code 429 is still returned. However, for some cases where the signature value r or value s is mathematically off the curve described by the q value from the EC key, that is, when r > (q-1) or s > (q-1), then return code 12 with reason code 769 is returned, indicating a rejection from the EC hardware layer. The decision to return the different error for the more serious case is reached when the host library loads (at application startup) by the value of a new environment variable: CSU_EC_CHECKCURVE

For appropriate failure cases, if the value of CSU_EC_CHECKCURVE is 1, then the new return/reason code 12/769 is returned. Otherwise, 4/429 is returned. The default value of CSU_EC_CHECKCURVE is 0, to maintain compatibility with prior releases. IBM recommends that customers prepare their applications and set the new environment variable to 1. Use the following command to set the variable (also to set it in a profile):

```
export CSU_EC_CHECKCURVE=1
```

Some error path test cases with pre-figured values may need to be updated, and the new return/reason code handling may need to be added to your application as a different type of signature verification failure. The adapter is functioning normally and no service action is required. Note that this change does not narrow valid verification cases or add restrictions. All valid signatures when passed to CSNDDSV with the correct key still verify with return code of 0, reason code of 0.

Advantages of the new verification granularity

The new use of return code/reason code 12/769 can help you with problem determination for the following practical cases:

- Check the bytes of the ECDSA signature. The r value is mathematically determined from the order of G of the curve and a random number. If the value of r or the value of s equals or exceeds the order of G of the curve, then the signature has been corrupted. No value or range of values can be statically ruled
Digital Signature Verify (CSNDDSV)

out for any bytes of r or s because of the random components. However you may be able to detect a corruption pattern by inspection. The s value is the same byte length as r (which is the same length as the curve order of G). The s value immediately follows r in the signature.

• Certain types of key corruption may also cause this error - forcing the key to be off the curve. Since public keys are passed in the clear, this is hard to detect until the public key is actually used, but again inspection of the key data may help.
• While using the wrong key type on the same curve size (example: substituting a P384 key for a BP384 key) will probably not generate 12/769 instead of 4/429, using a key for a smaller curve may generate 12/769 (example: substituting a P224 key for a P256 key).

Format

The format of CSNDDSV.

```
CSNDDSV(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    PKA_public_key_identifier_length,  
    PKA_public_key_identifier,  
    hash_length,  
    hash,  
    signature_field_length,  
    signature_field)
```

Parameters

The parameters for CSNDDSV.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 0, 1, 2, or 3.

**rule_array**

- **Direction:** Input
- **Type:** String array

Keywords that provide control information to the verb. A keyword specifies the method to use to verify the digital signature. Each keyword is left-aligned in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage. The rule_array keywords are described in Table 188.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital signature formatting method</td>
<td>(Optional and not valid with ECDSA keyword.)</td>
</tr>
</tbody>
</table>

Table 188. Keywords for Digital Signature Verify control information
### Table 188. Keywords for Digital Signature Verify control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO-9796</td>
<td>Verify the digital signature on the hash according to ISO-9796-1. Any hash method is allowed. This is the default.</td>
</tr>
<tr>
<td>PKCS-1.0</td>
<td>Verify the digital signature on the BER-encoded ASN.1 value of the type DigestInfo as specified in the RSA Data Security, Inc. Public Key Cryptography Standards #1 block type 00. The text must specify BER encoded hash text.</td>
</tr>
<tr>
<td>PKCS-1.1</td>
<td>Verify the digital signature on the BER-encoded ASN.1 value of the type DigestInfo as specified in the RSA Data Security, Inc. Public Key Cryptography Standards #1 block type 01. The text must specify BER encoded hash text.</td>
</tr>
<tr>
<td>PKCS-PSS</td>
<td>Verify the digital signature on the string supplied in the hash variable as defined in the RSA PKCS #1 v2.2 standard for the RSASSA-PSS signature scheme.</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>Format the hash by padding it on the left with binary zeros to the length of the PKA key modulus. Any supported hash function is allowed.</td>
</tr>
<tr>
<td>X9.31</td>
<td>Format according to ANSI X9.31 standard.</td>
</tr>
<tr>
<td><strong>Data input type</strong> (One, optional).</td>
<td></td>
</tr>
<tr>
<td>HASH</td>
<td>Process the text supplied in the hash variable as a hash digest that is to be digitally signed by the verb. This is the default.</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>Process the text supplied in the hash variable as a message. This means, that the text is first hashed using the specified hashing-method to produce a hash digest. Then the resulting hash digest is used to produce a digital signature. The text to be hashed, identified by the hash parameter, must be a multiple of eight bits and byte-aligned.</td>
</tr>
<tr>
<td><strong>Hash method specification</strong> (One required when the MESSAGE keyword is specified or one required when the HASH keyword is specified and the hash formatting method is PKCS-PSS).</td>
<td></td>
</tr>
<tr>
<td>RPMD-160</td>
<td>The input value supplied in the hash parameter is generated using the RIPEMD-160 hash method. Not valid with the PKCS-1.0, PKCS-1.1, or PKCS-PSS signature methods.</td>
</tr>
<tr>
<td>SHA-1</td>
<td>The input value supplied in the hash parameter is generated using the SHA-1 hash method.</td>
</tr>
<tr>
<td>224</td>
<td>The input value supplied in the hash parameter is generated using the 224 hash method. Not valid with the X9.31 signature method.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>The input value supplied in the hash parameter is generated using the SHA-256 hash method.</td>
</tr>
<tr>
<td>SHA-384</td>
<td>The input value supplied in the hash parameter is generated using the SHA-384 hash method.</td>
</tr>
<tr>
<td>SHA-512</td>
<td>The input value supplied in the hash parameter is generated using the SHA-512 hash method.</td>
</tr>
<tr>
<td><strong>Signature checking rule</strong> (One optional. Valid only with the PKCS-PSS digital signature hash formatting method.)</td>
<td></td>
</tr>
<tr>
<td>EXMATCH</td>
<td>Specifies that the 4-byte salt length prepended to the data identified by the hash parameter must be an exact match to the salt length derived from the signature. This is the default.</td>
</tr>
<tr>
<td>NEXMATCH</td>
<td>Specifies that the 4-byte salt length prepended to the data identified by the hash parameter does not have to be an exact match to the salt length derived from the signature.</td>
</tr>
<tr>
<td><strong>Note:</strong> The derived salt length cannot be less than the prepended length.</td>
<td></td>
</tr>
<tr>
<td><strong>Trusted public key restriction</strong> (Optional. Not valid with ECDSA keyword. Valid only with trusted blocks. See “Trusted blocks” on page 968.)</td>
<td></td>
</tr>
<tr>
<td>TPK-ONLY</td>
<td>Permits the use of only public keys contained in trusted blocks. By specifying this keyword, the use of regular CCA RSA key tokens is rejected and only the use of a (trusted) public key supplied by the PKA_public_key_identifier parameter can be used to verify the digital signature, thus assuring a sensitive signature verification operation is limited to trusted public keys.</td>
</tr>
<tr>
<td></td>
<td>If TPK-ONLY is specified, the PKA_public_key_identifier parameter must identify a trusted block that contains two sections after the trusted block token header: (1) trusted block trusted RSA public key (section X'11'), and (2) trusted block information (section X'14'). Section X'14' is required for all trusted blocks. Section X'11' contains the trusted public key, and its usage rules must indicate it can be used in digital signature operations.</td>
</tr>
</tbody>
</table>
### Table 188. Keywords for Digital Signature Verify control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Token algorithm</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>ECDSA</td>
<td>Verify an ECC digital signature. When specified, this is the only keyword permitted in the rule_array.</td>
</tr>
<tr>
<td>RSA</td>
<td>Verify an RSA digital signature. This is the default.</td>
</tr>
<tr>
<td><strong>Certificate validation method</strong> (One required if the token or label in <code>PKA_public_key_identifier</code> is a certificate. Otherwise, a validation method must not be specified.)</td>
<td></td>
</tr>
<tr>
<td>RFC-2459</td>
<td>Validate the certificate using the semantics of RFC-2459.</td>
</tr>
<tr>
<td>RFC-3280</td>
<td>Validate the certificate using the semantics of RFC-3280.</td>
</tr>
<tr>
<td>RFC-5280</td>
<td>Validate the certificate using the semantics of RFC-5280.</td>
</tr>
<tr>
<td>RFC-ANY</td>
<td>Attempt to validate the certificate by first using the semantics of RFC-2459, then the semantics of RFC-3280, and finally, the semantics of RFC-5280.</td>
</tr>
</tbody>
</table>

**Tip:**

- Use the SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512 hashing method to create the hash digest for PKCS-1.0 or PKCS-1.1.
- Also, use one of these hashing methods to create the hash digest for ECDSA and PKCS-PSS.
- Use any hashing method to create the hash digest for ISO-9796 and ZERO-PAD.
- Use RIPEMD-160, SHA-1, SHA-256, SHA-384, or SHA-512 hashing method to create the hash digest for X9.31.
- Refer to "Formatting hashes and keys in public-key cryptography" on page 1044 for a discussion of hash formatting methods.

**PKA_public_key_identifier_length**

- **Direction:** Input
- **Type:** Integer

  The length of the `PKA_public_key_identifier` field containing the public key token or label. The maximum size is 3500 bytes.

**PKA_public_key_identifier**

- **Direction:** Input
- **Type:** String

  A token or label of the RSA public key or internal trusted block. If this parameter contains a token or the label of an internal trusted block, the rule_array parameter must specify `TPK-ONLY`. If the signature algorithm is `ECDSA`, this must be a token label or an ECC public key.

**hash_length**

- **Direction:** Input
- **Type:** Integer

  The length of the `hash` parameter in bytes. It must be the exact length of the text to sign. The maximum size is 512 bytes. If you specify `ZERO-PAD` in the `rule_array` parameter, the length is restricted to 36 bytes unless the RSA key is a signature-only key. In this case, the maximum length is 512 bytes.
The hash length limit is controlled by an access control point. Only RSA key management keys are affected by this access control point. The limit for RSA signature-only keys is 512 bytes. This access control point is always disabled in the default role. You must have a TKE workstation to enable it.

If you specify PKCS-PSS, the first four bytes must contain the salt length. The remaining bytes of the parameter must contain the hash or message. The value will be 4 + length of the hash or message.

If MESSAGE is not specified, the maximum length is 68 bytes. If MESSAGE is specified, the value is the length of the message to be hashed. The maximum length is in the range 64 bytes - 32 MB.

hash

| Direction: Input |
| Type: String |

The data parameter is a pointer to a string variable containing the hash to be signed. This verb can identify the hash to be verified or the message to be hashed and verified, depending on whether the HASH or MESSAGE input type keyword specified. If MESSAGE is specified, the input data is treated as data that must be hashed by the verb and then verified, otherwise the input data is treated as hash information that is to be signed.

If PKCS-PSS is specified (Release 5.3 or later), the data variable includes an additional 4 byte field that contains the salt length in big endian format. This length must be prepended to the hash or message. To allow use of a salt length less than the hash length, but not zero, enable the Allow Small Salt command (offset X'033C') in the active role. A salt length value can be 0 and 20 through the size allowed by the RSA modulus in bytes. Please note that the size of the data to be signed is still governed by the size of the RSA modulus. Currently, the salt length cannot exceed 490 bytes. The modulus size and hash length affect the maximum salt length for a given key modulus size and specified hash. The maximum salt length equals \( \text{modulus} \text{size}/8 - \text{hash\_length} - 2 \). For example, with a 4096 bit modulus key and SHA-1 hash, the maximum salt length becomes \( (4096/8) - 20 - 2 = 490 \).

If PKCS-PSS is specified for verification with normal processing, the salt length derived from the signature must be an exact match (keyword EXMATCH, the default) for the 4-byte salt length that is prepended in big endian format to the beginning of this parameter. With the Allow Not Exact Salt Length command (offset X'033B') enabled in the active role, keyword NEXMATCH can be specified to allow signature verification when the salt length derived from the signature is not an exact match for the salt length specified with the data parameter. Salt lengths derived from the signature are not allowed to be less than the value specified with the data parameter. Specifying the keyword and enabling the access control point in the active role will allow an additional level of checking. Currently, the salt length cannot exceed 490 bytes.

signature_field_length

| Direction: Input |
| Type: Integer |

The length in bytes of the signature_field parameter. The maximum size is 512 bytes.

signature_field
Digital Signature Verify (CSNDDSV)

**Direction:** Input  
**Type:** String

This field contains the digital signature to verify. The digital signature is in the low-order bits (right-aligned) of a string whose length is the minimum number of bytes that can contain the digital signature. This string is left-aligned within the `signature_field`.

**Restrictions**

The restrictions for CSNDDSV.

The ability to recover a message from a signature (which ISO-9796 allows but does not require) is **not** supported.

The exponent of the RSA public key must be odd.

Although ISO-9796 does not require the input hash to be an integral number of bytes in length, this service requires you to specify the `hash_length` in bytes.

X9.31 requires the RSA token to have a minimum modulus bit length of 1024, and the length must also be a multiple of 256 bits (or 32 bytes).

**Required commands**

The required commands for CSNDDSV.

This verb requires the **Digital Signature Verify** command (offset X'0101') to be enabled in the active role.

**Usage notes**

The usage notes for CSNDDSV.

None.

**Related information**

Additional information for CSNDDSV.

Trusted Block Create (CSNDTBC), Remote Key Export (CSNDRKX)

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNDDSVJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNDDSVJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber PKA_public_key_identifier_length,
    byte[] PKA_public_key_identifier,
    hikmNativeNumber hash_length,
```
Digital Signature Verify (CSNDDSV)

byte[] hash,
hkmNativeNumber signature_field_length,
byte[] signature_field);
Digital Signature Verify (CSNDDSV)
Chapter 15. Managing PKA cryptographic keys

Use these verbs to generate and manage PKA keys.

- “PKA Key Generate (CSNDPKG)”
- “PKA Key Import (CSNDPKI)” on page 696
- “PKA Key Token Build (CSNDPKB)” on page 699
- “PKA Key Token Change (CSNDKTC)” on page 710
- “PKA Key Translate (CSNDPKT)” on page 713
- “PKA Public Key Extract (CSNDPKX)” on page 720
- “Remote Key Export (CSNDRKX)” on page 722
- “Trusted Block Create (CSNDTBC)” on page 734
- “Public Infrastructure Certificate (CSNDPIC)” on page 738
- “Public Infrastructure Manage (CSNDPIM)” on page 744

PKA Key Generate (CSNDPKG)

Use the PKA Key Generate verb to generate an RSA public-private key-pair for use with the RSA algorithm. You can also use the verb to generate an ECC public-private key pair for use with the ECC algorithm.

Input to the PKA Key Generate verb is either a skeleton key token that has been built by the PKA Key Token Build verb, or a valid internal token. In the case of a valid internal token, the verb will generate a key with the same modulus length and the same exponent. In the case of a valid internal ECC token, PKA Key Generate generates a key based on the curve type and size. Internal tokens with an 'X'09' section are not supported.

The PKA input skeleton determines the following characteristics of the generated key-pair:

- the key type: RSA or ECC
- the RSA key length (modulus size) or ECC Brainpool or Prime curve size of \( p \) in bits: 192, 224, 256, 384 or 521 for Prime curves and 160, 192, 224, 256, 320, 384, or 512 for Brainpool curves
- The RSA public exponent: valued to 3, 65537, or random. Beginning with Release 5.2, a value of 5, 17, or 257 is valid. See “Restrictions” on page 694.

Note: The value 3, 5, 17, 257, and 65537 are the first Fermat numbers. Fermat numbers take the form \( F_n = 2^{2^n} + 1 \), where \( n \) is a non-negative integer. The first five Fermat numbers are known to be prime.

- any RSA private-key optimization (Modulus-Exponent versus Chinese-Remainder Theorem format)
- any signatures and signature-information that should be associated with the public key.
- for EXX key generation: Key usage information and optionally, application associated data

An ECC key is always randomly generated by this verb. Normally an RSA key is randomly generated. However, an RSA key can be derived using regeneration data. By providing regeneration data for an RSA key, a seed can be supplied so that the
same value of the generated key can be obtained in multiple instances. This can be useful in testing situations or where the regeneration data can be securely held for key generation. The process for generating a particular key pair from regeneration data might vary between products. Therefore, do not expect to obtain the same key pair for a given regeneration data string between products.

The generated private-key can be returned in one of three forms for RSA and ECC:

**Unenciphered**

When rule-array keyword CLEAR is specified, the PKA private key is returned in cleartext form. The clear key is returned in an external PKA key-token.

**Enciphered using a master key**

When rule-array keyword MASTER is specified, a master key is used to protect the private key or its OPK if the private key has one. The enciphered key is returned in an internal PKA key-token. ECC keys (section X'20') and RSA keys in section X'30' and X'31' have an OPK that is enciphered by the APKA master key. All other RSA keys are enciphered by the PKA master key.

**Enciphered using a transport key**

When rule-array keyword XPORT is specified, a transport key (key-encrypting key) is used to protect the private key or its object protection key (OPK) if the private key has one. The enciphered key is returned in an external PKA key-token. By definition, an ECC private-key section always has an OPK, while an RSA private-key section either has an OPK or it does not, depending on the section identifier. The OPK of an external ECC private-key is enciphered under a variable-length AES EXPORTER or IMPORTER key-encrypting key after it is used to encipher the private key. Beginning with Release 4.4, there are two new private key sections defined, X'30' and X'31'. When the key token is external, the OPK data contained in either private-key section X'30' or X'31' is enciphered under a variable-length AES EXPORTER or IMPORTER key-encrypting key after it is used to encipher the private key. An external RSA private-key that does not have a private-key section of X'30' or X'31' is enciphered under a fixed-length DES EXPORTER or IMPORTER key-encrypting key.

**Note:** A private key enciphered by an EXPORTER key can be imported onto a node where the corresponding IMPORTER key is installed. In contrast, a private key enciphered by an IMPORTER key can be imported onto the generating node.

With the exception of RSA private key sections X'30' and X'31', use the RETAIN rule-array keyword to cause an RSA private key to be retained within the coprocessor. Incorporate the key label to be used later to reference the newly generated key in the key name section of the skeleton key-token. Later, use this label to employ the key in verbs such as Digital Signature Generate, Symmetric Key Import, SET Block Decompose, and PKA Decrypt.

On output, the verb returns an external key-token containing the public key in the `generated_key_identifier` variable. This variable returned by the verb does not contain the private key.
Note: When using the RETAIN private-key encryption option, the key label supplied in the skeleton key-token references the key storage within the coprocessor, and, in this case, must not reference a record in the host-system PKA key-storage.

The rule array keyword CLONE flags a generated and retained RSA private-key as usable in an engine cloning process. Cloning is a technique for copying sensitive coprocessor information from one coprocessor to another (see "Understanding and managing master keys," on page 83). ECC private keys and RSA keys in private key sections X'30' or X'31' are not usable in an engine cloning process.

If you include an RSA public-key certificate section within the PKA skeleton key-token, the cryptographic engine signs a certificate with the key that is designated in the RSA public-key certificate signature subsection. This technique causes the cryptographic engine to sign the newly generated RSA public key using another key that has been retained within the engine, including the newly generated key (producing a self-signature). You can obtain more than one signature on the public key when you include multiple signature subsections in the skeleton key token. See “PKA public-key certificate section” on page 885.

Tip: The verb returns a section X'06' RSA private-key token 1024-bit Modulus-Exponent with OPK format when you request a Modulus Exponent internal key even though you have specified a type X'02' RSA private-key 1024-bit Modulus Exponent skeleton key token.

Format
The format of CSNDPKG.

```c
CSNDPKG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    regeneration_data_length,
    regeneration_data,
    skeleton_key_identifier_length,
    skeleton_key_identifier,
    transport_key_identifier,
    generated_key_identifier_length,
    generated_key_identifier)
```

Parameters
The parameter definitions for CSNDPKG.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

rule_array_count
Direction:     Input
Type:          Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 1, 2, or 3.
PKA Key Generate (CSNDPKG)

**rule_array**

- **Direction:** Input
- **Type:** String array

A keyword that provides control information to the verb. A keyword is left-aligned in an 8-byte field and padded on the right with blanks. The *rule_array* keywords are described in Table 189.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private key encryption (One, required)</td>
<td></td>
</tr>
<tr>
<td>CLEAR</td>
<td>Return the private key in cleartext. The private key in cleartext is an external token. This keyword is valid only for RSA and ECC keys.</td>
</tr>
<tr>
<td>MASTER</td>
<td>Encipher the private key or OPK using the PKA master-key for an RSA key, or the OPK using the APKA master-key for an ECC key. The <em>transport_key_identifier</em> parameter should specify a null key-token. The keyword is not supported if a skeleton token with a 09 section is provided.</td>
</tr>
</tbody>
</table>
| RETAIN | Retains the private key within the cryptographic engine and returns the public key. This is only valid for RSA signature keys. Because of this, the RETAIN keyword is not supported for:
  - A skeleton token with a X'09' section provided.
  - An ECC token.

  Before using this keyword, see the information about retained keys in “Using retained keys” on page 485.

  Note: Take special notice on the types of skeleton key tokens that can be passed. The PKA Key Token Build verb will, of course, let you create many more types of skeleton key tokens than can be used to generate retained keys, because this is the minority of supported function. |
| XPORT | Enciphers the private key under the IMPORTER or EXPORTER key-encrypting-key identified by the *transport_key_identifier* parameter. For an RSA key, this is an EXPORTER or IMPORTER transport key in a fixed-length operational DES key-token. For an ECC key, this is an EXPORTER or IMPORTER transport key in an operational variable-length AES key-token. This keyword is valid only for RSA and ECC keys. |

| RETAIN option (one, optional). Valid only with the RETAIN keyword. | |
| CLONE | Mark a generated and retained private key as usable in cryptographic engine cloning process. This keyword is supported only if RETAIN is also specified. Only valid for RSA keys. The keyword is not supported for:
  - A skeleton token with a X'09' section provided
  - An ECC token |

| Regeneration data option (One, optional) | |
| ITER-38 | Force 38 iterations of tests for primality, as required by ANSI X9.31 for the Miller-Rabin primality tests. This option produces a more secure key, but it is labor intensive. This keyword is invalid for ECC key generation. This keyword was introduced with CCA 4.1.0. |

| Transport key-type (one, optional; one required if *transport_key_identifier* is a label). If this keyword is specified, it must match the type of key to be transported, whether the identifier is a label or not. | |
| OKEK-AES | The outbound key-encrypting key represents an AES key-token. |
| OKEK-DES | The outbound key-encrypting key represents a DES key-token. This is the default. |

**regeneration_data_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the *regeneration_data* variable. This parameter must be 0 for ECC tokens. For RSA tokens, the value must be 8 - 512.
If the value is 0, the generated keys are based on a random-seed value. If this value is between 8 - 256, the regeneration data is hashed to form a seed value used in the key generation process to provide a means for recreating a public-private key pair.

**regeneration_data**

**Direction:** Input  
**Type:** String

This field points to a string variable containing a string used as the basis for creating a particular public-private key pair in a repeatable manner. The regeneration data is hashed to form a seed value used in the key generation process and provides a means for recreating a public-private key pair.

**skeleton_key_identifier_length**

**Direction:** Input  
**Type:** Integer

The length of the `skeleton_key_identifier` parameter in bytes. The maximum allowed value is 3500 bytes.

**skeleton_key_identifier**

**Direction:** Input  
**Type:** String

A pointer to the application-supplied skeleton key token generated by PKA Key Token Build, or the label of the token that contains the required modulus length and public exponent for RSA key generation, or the required curve type and bit length for ECC key generation.

If RETAIN was specified and the `skeleton_key_identifier` is a label, the label must match the private key name of the key. For RSA keys, the `skeleton_key_identifier` parameter must contain a token that specifies a modulus length in the range 512 - 4096 bits.

**transport_key_identifier**

**Direction:** Input  
**Type:** String

A pointer to a string variable containing an operational AES or DES key-encrypting-key token, a null key-token, or a key label of such a key. Use an IMPORTER key to encipher a private key to be used at this node. Use an EXPORTER key to encipher a private key to be used at another node. Choose one of the following:

- When generating an ECC key with the XPORT rule-array keyword, provide the variable-length symmetric IMPORTER or EXPORTER key-token to be used to wrap the generated ECC key. Key bit lengths of 128, 192, and 256 are supported. If this parameter points to a key label, specify rule-array keyword OKEK-AES to indicate that the AES key-storage dataset contains the key token.
- When generating an RSA key with the XPORT rule-array keyword, provide the fixed-length DES IMPORTER or EXPORTER key-token to be used to wrap the generated RSA key. If this parameter points to a key label, specify rule-array keyword OKEK-DES to indicate that the DES key-storage dataset contains the key token.
PKA Key Generate (CSNDPKG)

- If the XPORT rule-array keyword is not specified, specify a null key-token. If this parameter points to a key label, specify keyword OKEK-AES for an ECC key or keyword OKEK-DES for an RSA key.

**generated_key_identifier_length**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Integer</td>
</tr>
</tbody>
</table>

The length of the generated key token. The field is checked to ensure that it is at least equal to the size of the token being returned. The maximum size is 3500 bytes. On output, this field is updated with the actual token length.

**generated_key_identifier**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>String</td>
</tr>
</tbody>
</table>

The internal key token or key label of the generated RSA or ECC key. When generating an RSA retained key, on output the verb returns the public token in this variable.

If the key label identifies a key record in PKA key-storage:
- A record must already exist in the PKA key storage file with this same label or the verb will fail
- The generated key token replaces any key token associated with the label.
- the generated_key_token_length returned to the application will be the same as the input length.

If the first byte of the identified string does not indicate a key label (that is, not in the range X'20' - X'FF'), and the variable is of sufficient length to receive the result, then the generated key token is returned in the identified variable.

**Restrictions**

The restrictions for CSNDPKG.

- The maximum public exponent is 17 bits for any key that has a modulus greater than 2048 bits.
- Not all IBM implementations of CCA support a CRT form of the RSA private key; check the product-specific literature. The IBM implementations support an optimized RSA private key (a key in Chinese Remainder Theorem format). The formats vary between versions.
- See "PKA key tokens" on page 880 for the formats used when generating the various forms of key tokens.
- When generating a key for use with ANSI X9.31 digital signatures, the modulus length must be: 1024, 1280, 1536, 1792, 2048, or 4096 bits.
- The key label used for a retained key must not exist in the external PKA key-storage held on the hard disk drive.
- Due to potential loss of a retained private key within the cryptographic engine, retained keys should be avoided for key management purposes.
- 2048-bit RSA keys may have a public exponent in the range of 1 - 256 bytes.
- 4096-bit RSA key public exponents are restricted to the values 3 and 65537.

RSA key generation has the following restrictions:
- For Modulus-Exponent, there are restrictions on the modulus, public exponent, and private exponent.
- For CRT, there are restrictions on \(dp, dq, U\), and the public exponent.
See the Key value structure in “PKA Key Token Build (CSNDPKB)” on page 699 for a summary of restrictions.

### Required commands

The required commands for CSNDPKG.

- This verb requires the PKA Key Generate command (offset X'0103') to be enabled in the active role.
- With the CLONE rule-array keyword, enable the PKA Key Generate - Clone command (offset X'0204')
- With the CLEAR rule-array keyword, enable the PKA Key Generate - Clear RSA Key command (offset X'0205') in the hardware.
- To generate ECC keys with the CLEAR rule-array keyword, this verb requires the PKA Key Generate - Clear ECC keys command (offset X'0326') to be enabled in the active role.
- To generate keys based on the value supplied in the regeneration_data variable, you must enable one of these commands:
  - When not using the RETAIN keyword, enable the PKA Key Generate - Permit Regeneration Data command (offset X'027D').
  - When using the RETAIN keyword, enable the PKA Key Generate - Permit Regeneration Data Retain command (offset X'027E').
- To disallow the wrapping of a key with a weaker key-encrypting key, enable the Prohibit weak wrapping - Transport keys command (offset X'0328') in the active role. This command affects multiple verbs. See Chapter 24, "Access control points and verbs," on page 1047.
- To receive a warning when wrapping a key with a weaker key-encrypting key, enable the Warn when weak wrap - Transport keys command (offset X'032C') in the active role. The Prohibit weak wrapping - Transport keys command (offset X'0328') overrides this command.

### Usage notes

Usage notes for CSNDPKG.

None.

### JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNDPKGJ.

See “Building Java applications using the CCA JNI” on page 28.

### Format

public native void CSNDPKGJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber regeneration_data_length,
    byte[] regeneration_data,
    hikmNativeNumber skeleton_key_identifier_length,
    byte[] skeleton_key_identifier,
    hikmNativeNumber generated_key_identifier_length,
    byte[] generated_key_identifier);
PKA Key Import (CSNDPKI)

This verb imports an external PKA or ECC private key token. (This consists of a PKA or ECC private key and public key.)

The secret values of the key can be:
- Clear
- Encrypted under a limited-authority DES importer key if the source_key_identifier is an RSA token
- Encrypted under an AES Key Encryption Key if the source_key_identifier is an ECC token

This verb can also import a clear PKA key. The PKA Key Token Build verb creates a clear PKA key token.

This verb can also import an external trusted block token for use with the Remote Key Export verb.

Output of this verb is a CCA internal token of the RSA or ECC private key or trusted block.

Format

The format of CSNDPKI.

```
CSNDPKI(
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    source_key_identifier_length,  
    source_key_identifier,  
    importer_key_identifier,  
    target_key_identifier_length,  
    target_key_identifier)
```

Parameters

The parameter definitions for CSNDPKI.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see "Parameters common to all verbs" on page 22.

rule_array_count

Direction: Input  
Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. This value must be 0 or 1.

rule_array

Direction: Input  
Type: String array

The rule_array parameter is a pointer to a string variable containing a
keyword. The keyword is 8 bytes in length and must be left-aligned and padded on the right with space characters. The rule array keywords are described in Table 190.

Table 190. Keywords for PKA Key Import control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token type</td>
<td>(One, optional)</td>
</tr>
<tr>
<td>ECC</td>
<td>Specifies that the key being imported is an ECC key.</td>
</tr>
<tr>
<td>RSA</td>
<td>Specifies that the key being imported is an RSA key or a trusted block. This is the default.</td>
</tr>
<tr>
<td>Transport key type</td>
<td>(optional)</td>
</tr>
<tr>
<td>IKEK-AES</td>
<td>The importer_key_identifier is an AES key.</td>
</tr>
<tr>
<td>IKEK-DES</td>
<td>The importer_key_identifier is a DES key. This is the default.</td>
</tr>
</tbody>
</table>

source_key_identifier_length

- **Direction:** Input
- **Type:** Integer

The length of the source_key_identifier parameter. The maximum size is 3500 bytes.

source_key_identifier

- **Direction:** Input
- **Type:** String

Contains an external key token or key label of a PKA private key, without section identifier X'14' (Trusted Block Information), or the trusted block in external form as produced by the Trusted Block Create verb with the ACTIVATE keyword.

If a PKA private key without the section identifier X'14' is passed in:
- There are no qualifiers. A retained key can not be used.
- The key token must contain both public-key and private-key information. The private key can be in cleartext or it can be enciphered. ECC tokens must contain a private key in cleartext.
- This is the output of the PKA Key Generate (CSNDPKG) verb or the PKA Key Token Build (CSNDPKB) verb.
- If encrypted, the key was created on another platform.

If a PKA private key with the section identifier X'14' is passed in:
- This verb is used to encipher the MAC key within the trusted block under the PKA master key instead of the IMP-PKA key-encrypting key.
- The importer_key_identifier must contain an IMP-PKA KEK.

importer_key_identifier

- **Direction:** Input/Output
- **Type:** String

A variable-length field containing an AES or DES key identifier used to wrap the imported key. For RSA keys and trusted blocks, this must be a DES limited authority transport key (IMP-PKA). For ECC keys, this must be an AES transport key.

This parameter contains one of the following:
- 64-byte label of a key storage record that contains the transport key.
PKA Key Import (CSNDPKI)

- 64-byte DES internal key token containing the transport key.
- A variable-length AES internal key token containing the transport key.

This parameter is ignored for clear tokens.

target_key_identifier_length

Direction: Input/Output
Type: Integer

The length of the target_key_identifier parameter. The maximum size is 3500 bytes. On output, and if the size is of sufficient length, the variable is updated with the actual length of the target_key_identifier field.

target_key_identifier

Direction: Input/Output
Type: String

This field contains the internal token or label of the imported PKA private key or a trusted block. If a label is specified on input, a PKA key storage record with this label must exist. The PKA key storage record with this label will be overwritten with the imported key unless the existing record is a retained key. If the record is a retained key, the import will fail. A retained key record cannot be overwritten. If no label is specified on input, this field is ignored and should be set to binary zeros on input.

Restrictions

The restrictions for CSNDPKI.

This verb imports RSA keys of up to 4096 bits. However, the hardware configuration sets the limits on the modulus size of keys for digital signatures and key management; thus, the key can be successfully imported but fail when used if the limits are exceeded.

The importer_key_identifier parameter is a limited-authority key-encrypting key.

CRT form tokens with a private section ID of X'05' cannot be imported.

Required commands

The required commands for CSNDPKI.

This verb requires the PKA Key Import command (offset X'0104') to be enabled in the active role. If the source_key_token parameter points to a trusted block, also enable the PKA Key Import - Import an external trusted block command (offset X'0311').

To disable the wrapping of a key with a weaker master key, the Prohibit weak wrapping - Master keys command (offset X'0333') must be enabled in the active role.

To receive a warning when wrapping a key with a weaker master key, enable the Warn when weak wrap - Master keys command (offset X'0332') in the active role. The Prohibit weak wrapping - Master keys command overrides this command.

Usage notes

The usage notes for CSNDPKI.
This verb imports keys of any modulus size up to 2048 bits. However, the hardware configuration sets the limits on the modulus size of keys for digital signatures and key management; thus, the key can be successfully imported but fail when used if the limits are exceeded.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNDPKIJ.

This verb has a Java Native Interface (JNI) version, which is named CSNDPKIJ. See “Building Java applications using the CCA JNI” on page 28.

**Format**

```
public native void CSNDPKIJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber source_key_identifier_length,
    byte[] source_key_identifier,
    byte[] transport_key_identifier,
    hikmNativeNumber target_key_identifier_length,
    byte[] target_key_identifier);
```

**PKA Key Token Build (CSNDPKB)**

Use this verb to build external PKA key tokens containing unenciphered private RSA or ECC keys.

You can use this token as input to the PKA Key Import verb to obtain an operational internal token containing an enciphered private key. This verb builds a skeleton token that you can use as input to the PKA Key Generate verb (see Table 189 on page 692). You can also input to this verb a clear unenciphered public RSA or ECC key and return the public key in a token format that other PKA verbs can use directly.

This verb is used to create the following:

- A skeleton key token for use with the PKA Key Generate verb.
- A key token with a public key that has been obtained from another source.
- A key token with a clear private-key and the associated public key.
- A key token for an RSA private key in optimized Chinese Remainder Theorem (CRT) format.
- An RSA token with X'09' section identifier using the RSAMEVAR keyword to obtain a token for a key in Modulus-Exponent format that is variable length.

ECC key generation requires this information in the skeleton token:

- The key type: ECC
- The type of curve: Prime or Brainpool
- The size of \( p \) in bits: 192, 224, 256, 384 or 521 for Prime curves and 160, 192, 224, 256, 320, 384, or 521 for Brainpool curves
- Key usage information
- Optionally, application associated data
PKA Key Token Build (CSNDPKB)

Format

The format of CSNDPKB.

```
CSNDPKB(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_value_structure_length,
    key_value_structure,
    private_key_name_length,
    private_key_name,
    user_definable_associated_data_length,
    user_definable_associated_data,
    reserved_2_length,
    reserved_2,
    reserved_3_length,
    reserved_3,
    reserved_4_length,
    reserved_4,
    reserved_5_length,
    reserved_5,
    key_token_length,
    key_token)
```

Parameters

The parameters for CSNDPKB.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1, 2, or 3.

**rule_array**

Direction: Input
Type: String array

A pointer to a string variable containing an array of keywords for the verb. The keywords must be in contiguous storage with each of the keywords left-aligned in its own 8-byte location and padded on the right with blanks. The `rule_array` keywords are described in **Table 191**.

**Table 191. Keywords for PKA Key Token Build control information**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Token type</strong> (One, required)</td>
<td></td>
</tr>
<tr>
<td>ECC-PAIR</td>
<td>This keyword indicates building a token containing both public and private ECC key information. The parameter <code>key_value_structure</code> identifies the input key values, if supplied.</td>
</tr>
<tr>
<td>ECC-PUBL</td>
<td>This keyword indicates building a token containing public ECC key information. The parameter <code>key_value_structure</code> identifies the input values, if supplied.</td>
</tr>
</tbody>
</table>
Table 191. Keywords for PKA Key Token Build control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA-AESC</td>
<td>Create a key token for an RSA public key and an RSA private key in Chinese-Remainder Theorem (CRT) format with an AES-encrypted OPK. Starting with CCA 6.0, specify a format restriction keyword to restrict the private key to a particular digital-signature hash formatting method, or none (the default, see Format restriction section later in this table).</td>
</tr>
<tr>
<td>RSA-AESM</td>
<td>Create a key token for an RSA public key and an RSA private key in Modulus-Exponent format with an AES-encrypted OPK. Starting with CCA 6.0, specify a format restriction keyword to restrict the private key to a particular digital-signature hash formatting method, or none (the default, see Format restriction section later in this table).</td>
</tr>
<tr>
<td>RSA-CRT</td>
<td>This keyword indicates building a token containing an RSA private key in the optimized Chinese Remainder Theorem (CRT) format. The parameter key_value_structure identifies the input key values, if supplied.</td>
</tr>
<tr>
<td>RSA-PRIV</td>
<td>This keyword indicates building a token containing both public and private RSA key information. The parameter key_value_structure identifies the input key values, if supplied.</td>
</tr>
<tr>
<td>RSA-PUBL</td>
<td>This keyword indicates building a token containing public RSA key information. The parameter key_value_structure identifies the input values, if supplied.</td>
</tr>
<tr>
<td>RSAMEVAR</td>
<td>This keyword indicates RSA-Modulus Exponent-Variant (RSAMEVAR), a type X’09’ key token for RSA, named VAR_OPK. Note: Key tokens created with this key type cannot be passed to the PKA Key Generate verb for creating RETAIN (retained) keys.</td>
</tr>
</tbody>
</table>

**Key usage control** (One, optional)

| KEY-MGMT    | Indicates that an RSA or ECC private key can be used in both the Symmetric Key Import and the Digital Signature Generate verbs. Note: Key tokens created with this key usage cannot be passed to the PKA Key Generate verb for creating RETAIN (retained) keys. |
| KM-ONLY     | Indicates that an RSA or ECC private key can be used only in symmetric key distribution. Note: Key tokens created with this key usage cannot be passed to the PKA Key Generate verb for creating RETAIN (retained) keys. |
| SIG-ONLY    | Indicates that an RSA or ECC private key cannot be used in symmetric key distribution. This is the default. Note: Only a skeleton key-token created from PKA Key Token Build with this key usage type can be passed to PKA Key Generate to create a RETAIN (retained) key. |

**Translate control** (One, optional)

| NO-XLATE    | The RSA or ECC key cannot be used as a key-encrypting-key for “PKA Key Translate (CSNDPKT)” on page 713. Note: Use of this keyword does not matter when creating a skeleton key-token for a later retained key generation operation. It is redundant to the necessary SIG-ONLY keyword. |
| XLATE-OK    | The RSA or ECC key can be used as a key-encrypting-key for “PKA Key Translate (CSNDPKT)” on page 713. Note: Key tokens created with this keyword cannot be passed to the PKA Key Generate verb for creating RETAIN (retained) keys. |

**Format restriction** (One, optional). As of release 6.0, only valid with token type keyword RSA-AESC or RSA-AESM.

| FR-NONE     | Specifies to not restrict the private key to be used by a particular digital-signature hash formatting method. The key is usable for any method. This is the default. |
| FR-I9796    | Specifies to render the private key usable only with the digital-signature hash formatting method ISO-9796. |
PKA Key Token Build (CSNDPKB)

Table 191. Keywords for PKA Key Token Build control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR-X9.31</td>
<td>Specifies to render the private key usable only with the digital-signature hash formatting method X9.31.</td>
</tr>
<tr>
<td>FR-ZPAD</td>
<td>Specifies to render the private key usable only with the digital-signature hash formatting method ZERO-PAD.</td>
</tr>
<tr>
<td>FR-PK10</td>
<td>Specifies to render the private key usable only with the digital-signature hash formatting method PKCS-1.0.</td>
</tr>
<tr>
<td>FR-PK11</td>
<td>Specifies to render the private key usable only with the digital-signature hash formatting method PKCS-1.1.</td>
</tr>
<tr>
<td>FR-PSS</td>
<td>Specifies to render the private key usable only with the digital-signature hash formatting method PKCS-PSS.</td>
</tr>
</tbody>
</table>

ECC token version (One, optional). Release 5.2 or later. Only valid with token type ECC-PAIR.

| ECC-VER0 | Build an ECC private-key section (X'20') using the format section version number X'00'. This is the default. Note: Use of this option is provided for backward compatibility and its use is discouraged. The preferred format is Version X'01'. |
| ECC-VER1 | Build an ECC private-key section (X'20') using the format section version number X'01'. This keyword is required if key derivation data is provided. Version X'01' key token enhancements include a pedigree field, a section hash tag-length-value (TLV) object (X'60') that gets included in the IBM extended associated data (IEAD) with a hash digest of all optional sections up to the IEAD. The Version 1 key token also supports the ECC key-derivation information section (X'23') that is required by the Elliptic Curve Diffie-Hellman verb to derive one element of any key pair using the ANSI-X963-KDF key derivation function. |

key_value_structure_length

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the key_value_structure variable. The length depends on the key type parameter in the rule_array and on the actual values input. The length is in bytes. For maximum values, see Table 192.

Table 192. PKA Key Token Build - Key value structure length maximum values

<table>
<thead>
<tr>
<th>Key type</th>
<th>Key value structure maximum value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC-PAIR</td>
<td>207</td>
</tr>
<tr>
<td>ECC-PUBL</td>
<td>139</td>
</tr>
<tr>
<td>RSA-CRT, RSAMEVAR</td>
<td>3500</td>
</tr>
<tr>
<td>RSA-PRIV</td>
<td>648</td>
</tr>
<tr>
<td>RSA-PUBL</td>
<td>520</td>
</tr>
</tbody>
</table>

key_value_structure

Direction: Input
Type: String

This is a segment of contiguous storage containing a variable number of input clear key values and the lengths of these values in bits or bytes, as specified. The structure elements are ordered, of variable length, and the input key values must be right-aligned within their respective structure elements and padded on the left with binary zeros. If the leading bits of the modulus are
Table 193. PKA Key Token Build - Key value structure elements, ECC keys

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key value structure (ECC-PAIR)</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 000 | 001 | Curve type:  
X'00'  Prime curve  
X'01'  Brainpool curve |
| 001 | 001 | Reserved X'00'
| 002 | 002 | Length of \( p \) in bits  
X'00A0'  Brainpool P-160  
X'00C0'  Prime P-192, Brainpool P-192  
X'00E0'  Prime P-224, Brainpool P-224  
X'0100'  Prime P-256, Brainpool P-256  
X'0140'  Brainpool P-320  
X'0180'  Prime P-384, Brainpool P-384  
X'0200'  Brainpool P-512  
X'0209'  Prime P-521 |
| 004 | 002 | **ddd** - this field is the length of the private key \( d \) in bytes. This value can be zero if the key token is used as a skeleton key-token in the PKA Key Generate verb. The maximum value is 66 bytes. |
| 006 | 002 | **xxx** - this field is the length of the public key \( Q \) in bytes. This value can be zero if the key token is used as a skeleton key-token in the PKA Key Generate verb. The maximum value is 133 bytes, which includes one byte to indicate if the value is compressed. |
| 008 | 002 | **ddd**  Private key, \( d \)  
008 + **ddd**  Public key, \( Q \) |
| **Key value structure (ECC-PUBL)** | | |
| 000 | 001 | Curve type:  
X'00'  Prime curve  
X'01'  Brainpool curve |
| 001 | 001 | Reserved X'00'
| 002 | 002 | Length of \( p \) in bits  
X'00A0'  Brainpool p-160  
X'00C0'  Prime P-192, Brainpool P-192  
X'00E0'  Prime P-224, Brainpool P-224  
X'0100'  Prime P-256, Brainpool P-256  
X'0140'  Brainpool P-320  
X'0180'  Prime P-384, Brainpool P-384  
X'0200'  Brainpool P-512  
X'0209'  Prime P-521 |
| 004 | 002 | **xxx** - this field is the length of the public key \( Q \) in bytes. This value can be zero if the key token is used as a skeleton key-token in the PKA Key Generate verb. The maximum value is 133 bytes, which includes one byte to indicate if the value is compressed. |
| 006 | 002 | **xxx**  Public key, \( Q \) |
| **Key value structure (Optimized RSA, Chinese Remainder Theorem format, RSA-CRT)** | | |
| 000 | 002 | Modulus length in bits (512 - 2048). This is required. |
| 002 | 002 | Modulus field length in bytes, \( mnn \). This value can be zero if the key token is used as a skeleton key-token in the PKA Key Generate verb. This value must not exceed 256. |
| 004 | 002 | Public exponent field length in bytes, \( eee \). This value can be zero if the key token is used as a skeleton key-token in the PKA Key Generate verb. |
### PKA Key Token Build (CSNDPKB)

**Table 193. PKA Key Token Build - Key value structure elements, ECC keys (continued)**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>006</td>
<td>002</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>008</td>
<td>002</td>
<td>Length of the prime number $p$, in bytes, $ppp$. This value can be zero if the key token is used as a skeleton key-token in the PKA Key Generate verb. Maximum size of $p + q$ is 256 bytes.</td>
</tr>
<tr>
<td>010</td>
<td>002</td>
<td>Length of the prime number $q$, in bytes, $qqq$. This value can be zero if the key token is used as a skeleton key-token in the PKA Key Generate verb. Maximum size of $p + q$ is 256 bytes.</td>
</tr>
<tr>
<td>012</td>
<td>002</td>
<td>Length of $d_p$, in bytes, $rrr$. This value can be zero if the key token is used as a skeleton key-token in the PKA Key Generate verb. Maximum size of $d_p + d_q$ is 256 bytes.</td>
</tr>
<tr>
<td>014</td>
<td>002</td>
<td>Length of $d_q$, in bytes, $sss$. This value can be zero if the key token is used as a skeleton key-token in the PKA Key Generate verb. Maximum size of $d_p + d_q$ is 256 bytes.</td>
</tr>
<tr>
<td>016</td>
<td>002</td>
<td>Length of $U$, in bytes, $uuu$. This value can be zero if the key token is used as a skeleton key-token in the PKA Key Generate verb. Maximum size of $U$ is 256 bytes.</td>
</tr>
<tr>
<td>018</td>
<td>$mmm$</td>
<td>Modulus, $n$.</td>
</tr>
<tr>
<td>018 + $mmm$ + $eee$</td>
<td>$ppp$</td>
<td>Prime number $p$.</td>
</tr>
<tr>
<td>018 + $mmm$ + $eee$ + $ppp$</td>
<td>$qqq$</td>
<td>Prime number $q$.</td>
</tr>
<tr>
<td>018 + $mmm$ + $eee$ + $ppp$ + $qqq$ + $rrr$</td>
<td>$rrr$</td>
<td>$d_p = d \mod(p-1)$.</td>
</tr>
<tr>
<td>018 + $mmm$ + $eee$ + $ppp$ + $qqq$ + $rrr$ + $sss$</td>
<td>$sss$</td>
<td>$d_q = d \mod(q-1)$.</td>
</tr>
<tr>
<td>018 + $mmm$ + $eee$ + $ppp$ + $qqq$ + $rrr$ + $sss$</td>
<td>$uuu$</td>
<td>$U = q^r \mod(p)$.</td>
</tr>
</tbody>
</table>

**Key value structure** (RSA private, RSA private variable, or RSA public)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Modulus length in bits. This is required. When building a skeleton key-token, the modulus length in bits must be greater than or equal to 512 bits.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Modulus field length in bytes, $XXX$. This value can be zero if you are using the key token as a skeleton in the PKA Key Generate verb. This value must not exceed 256 when the RSA-PUBL keyword is used and must not exceed 128 when the RSA-PRIV keyword is used. This verb can build a key token for a public RSA key with a 2048-bit modulus length or it can build a key token for a 1024-bit modulus length private key.</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Public exponent field length in bytes, $YYY$. This value must not exceed 256 when the RSA-PUBL keyword is used and must not exceed 128 when the RSA-PRIV keyword is used. This value can be zero if you are using the key token as a skeleton key-token in the PKA Key Generate verb. In this case, a random exponent is generated. To obtain a fixed, predetermined public key exponent, you can supply this field and the public exponent as input to the PKA Key Generate verb.</td>
</tr>
</tbody>
</table>
### Table 193. PKA Key Token Build - Key value structure elements, ECC keys (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>006</td>
<td>002</td>
<td>Private exponent field length in bytes, ZZZ. This field can be zero, indicating that private key information is not provided. This value must not exceed 128 bytes. This value can be zero if you are using the key token as a skeleton in the PKA Key Generate verb.</td>
</tr>
<tr>
<td>008</td>
<td>XXX</td>
<td>Modulus, n. This is an integer such that (1 &lt; n &lt; 2^{1048}). The (n) is the product of (p) and (q) for primes (p) and (q).</td>
</tr>
<tr>
<td>008 + XXX</td>
<td>YYY</td>
<td>RSA public exponent, e. This is an integer such that (1 &lt; e &lt; n). (e) must be odd. When you are building a skeleton key_token to control the generation of an RSA key pair, the public key exponent can be one of the following values: 3, 65537 ((2^{16} + 1)), or 0 to indicate that a full random exponent should be generated. The exponent field can be a null-length field if the exponent value is 0.</td>
</tr>
<tr>
<td>008 + XXX + YYY</td>
<td>ZZZ</td>
<td>RSA secret exponent (d). This is an integer such that (1 &lt; d &lt; n). The value of (d) is (e^{1} \mod(p-1)(q-1)). You need not specify this value if you specify RSA-PUBL in the rule_array parameter.</td>
</tr>
</tbody>
</table>

### Table 194. PKA Key Token Build - Key value structure elements, RSA keys

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Length of the modulus in bits RSA-AESM (section X'30') 512 - 4096 RSA-PRIV (section X'02') 512 - 1024 RSA-PUBL (section X'04') 512 - 4096 RSAMEVAR (section X'09') 512 - 4096</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the modulus field (n), in bytes: (nnn). This value must not exceed 512 for a 4096-bit-length key. This value should be zero when preparing a skeleton key-token for use with the PKA Key Generate verb.</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Length of the public exponent field (e), in bytes: (eee). This value should be zero when preparing a skeleton key-token to generate a random-exponent public key in the PKA Key Generate verb. This value must not exceed 512.</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>Private exponent field length in bytes, (ddd). This value can be zero indicating that private key information is not provided. This value must not exceed 512.</td>
</tr>
<tr>
<td>008</td>
<td>(nnn)</td>
<td>Modulus (n), integer value, (1 &lt; n &lt; 2^{4096}). (n = pq) for prime (p) and prime (q).</td>
</tr>
<tr>
<td>8 + (nnn)</td>
<td>(eee)</td>
<td>Public exponent field (e), integer value, (1 &lt; e &lt; n). (e) must be odd. When you are building a skeleton key_token to control the generation of an RSA key pair, the public key exponent must be one of the following values: 0 (full-random), 3, or 65537. Beginning with Release 5.2, you can also specify a value of 5, 17, or 257. The exponent field can be a null-length field when preparing a skeleton_key_token.</td>
</tr>
<tr>
<td>8 + (nnn) + (eee)</td>
<td>(ddd)</td>
<td>Private exponent (d), integer value, (1 &lt; d &lt; n), (d = e^{-1} \mod(p-1)(q-1)).</td>
</tr>
</tbody>
</table>

RSA key-values structure, Chinese-Remainder Theorem format (RSA-AESC or RSA-CRT)
## PKA Key Token Build (CSNDPKB)

**Table 194. PKA Key Token Build - Key value structure elements, RSA keys (continued)**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Length of the modulus in bits:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>RSA-AESC (section X'31')</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>512 - 4096</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>RSA-CRT (section X'30')</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>512 - 4096</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the modulus field (n), in bytes: (nnn). This value must not exceed (4096/8 = 512). This value should be zero when preparing a skeleton_key_token for use with the PKA Key Generate verb.</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Length of the public exponent field (e), in bytes: (eee). This value should be zero when preparing a skeleton key-token to generate a random-exponent public key in the PKA Key Generate verb. This value must not exceed 512.</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>008</td>
<td>002</td>
<td>Length of the prime number field (p), in bytes: (ppp). Should be zero in a skeleton key-token. The maximum value of (ppp+qqq) is 512 bytes.</td>
</tr>
<tr>
<td>010</td>
<td>002</td>
<td>Length of the prime number field (q), in bytes: (qqq). Should be zero in a skeleton key-token. The maximum value of (ppp+qqq) is 512 bytes.</td>
</tr>
<tr>
<td>012</td>
<td>002</td>
<td>Length of the (d_p) field, in bytes: (rrr). Should be zero in a skeleton key-token. The maximum value of (rrr+sss) is 512 bytes.</td>
</tr>
<tr>
<td>014</td>
<td>002</td>
<td>Length of the (d_q) field, in bytes: (sss). Should be zero in a skeleton key-token. The maximum value of (rrr+sss) is 512 bytes.</td>
</tr>
<tr>
<td>016</td>
<td>002</td>
<td>Length of the (U) field, in bytes: (uuu). Should be zero in a skeleton key-token. The maximum length of (U) is 256 bytes.</td>
</tr>
<tr>
<td>018</td>
<td>(nnn)</td>
<td>Modulus (n).</td>
</tr>
<tr>
<td>018 + (nnn)</td>
<td>(eee)</td>
<td>Public exponent field (e), integer value, (1 &lt; e &lt; n), (e) must be odd. When you are building a skeleton key-token to control the generation of an RSA key pair, the public key exponent must be one of the following values: 0 (full-random), 3, or 65537. Beginning with Release 5.2, you can also specify a value of 5, 17, or 257. The exponent field can be a null-length field when preparing a skeleton key-token.</td>
</tr>
<tr>
<td>018 + (nnn)</td>
<td>(eee) + (ppp)</td>
<td>Prime number (p).</td>
</tr>
<tr>
<td>018 + (nnn)</td>
<td>(eee) + (ppp) + (qqq)</td>
<td>Prime number (q).</td>
</tr>
<tr>
<td>018 + (nnn)</td>
<td>(eee) + (ppp) + (qqq) + (rrr)</td>
<td>(d_p = d \text{ mod}(p-1)).</td>
</tr>
<tr>
<td>018 + (nnn)</td>
<td>(eee) + (ppp) + (qqq) + (rrr) + (sss)</td>
<td>(d_q = d \text{ mod}(q-1)).</td>
</tr>
<tr>
<td>018 + (nnn)</td>
<td>(eee) + (ppp) + (qqq) + (rrr) + (sss) + (uuu)</td>
<td>(U = q^{-1} \text{ mod}(p)).</td>
</tr>
</tbody>
</table>

**Note:** All length fields are in binary, and all binary fields (exponents, lengths, and so on) are stored with the high-order byte first (big-endian format).

**Note:**

1. All length fields are in binary.
2. All binary fields (exponent, lengths, modulus, and so on) are stored with the high-order byte field first. This integer number is right-aligned within the key structure element field.

3. You must supply all values in the structure to create a token containing an RSA or ECC private key for input to the PKA Key Import verb.

**private_key_name_length**

**Direction:** Input  
**Type:** Integer

The length can be 0 or 64.

**private_key_name**

**Direction:** Input  
**Type:** String

This field contains the name of a private key. The name must conform to CCA key label syntax rules. That is, allowed characters are alphanumeric, national (@, #, $) or period (.). The first character must be alphabetic or national. The name is folded to upper case and converted to ASCII characters. ASCII is the permanent form of the name because the name should be independent of the platform. The name is then cryptographically coupled with clear private key data before encryption of the private key. Because of this coupling, the name can never change after the key token is imported. The parameter is valid only with key type RSA-CRT.

**user_definable_associated_data_length**

**Direction:** Input  
**Type:** Integer

Length in bytes of the user_definable_associated_data parameter. This parameter is valid only for a key type of ECC-PAIR, and must be set to 0 for all other key types. The maximum value is 100.

**user_definable_associated_data**

**Direction:** Input  
**Type:** String

The user_definable_associated_data parameter identifies a string variable containing the associated data that will be placed following the IBM associated data in the token. The associated data is data whose integrity, but not whose confidentiality, is protected by a key wrap mechanism. The user_definable_associated_data can be used to bind usage control information.

This parameter is valid only for a key type of ECC-PAIR.

**reserved_2_length**

**Direction:** Input  
**Type:** Integer

Length in bytes of a reserved parameter. You must set this variable to 0.

**reserved_2**

**Direction:** Input  
**Type:** String
PKA Key Token Build (CSNDPKB)

The reserved_2 parameter identifies a string that is reserved. The verb ignores it.

reserved_3_length

Direction: Input
Type: Integer

Length in bytes of a reserved parameter. You must set this variable to 0.

reserved_3

Direction: Input
Type: String

The reserved_3 parameter identifies a string that is reserved. The verb ignores it.

reserved_4_length

Direction: Input
Type: Integer

Length in bytes of a reserved parameter. You must set this variable to 0.

reserved_4

Direction: Input
Type: String

The reserved_4 parameter identifies a string that is reserved. The verb ignores it.

reserved_5_length

Direction: Input
Type: Integer

Length in bytes of a reserved parameter. You must set this variable to 0.

reserved_5

Direction: Input
Type: String

The reserved_5 parameter identifies a string that is reserved. The verb ignores it.

key_token_length

Direction: Input/Output
Type: Integer

Length of the returned key token. The verb checks the field to ensure that it is at least equal to the size of the token to return. On return from this verb, this field is updated with the exact length of the key_token created. On input, a size of 3500 bytes is sufficient to contain the largest key_token created.

key_token

Direction: Output
Type: String

The returned key token containing an unenciphered private or public key. The private key is in an external form that can be exchanged with different CCA
PKA systems. You can use the public key token directly in appropriate CCA signature verification or key management services.

**Restrictions**

The restrictions for CSNDPKB.

- The RSA key length is limited to the range of 512 - 4096 bits, with specific formats restricted to a maximum of 1024 or 2048 bits.
- When generating a key for use with ANSI X9.31 digital signatures, the key length must be 1024, 1280, 1536, 1792, 2048, or 4096 bits.
- Allowable ECC key bit lengths are based on curve type. For Brainpool, the key length must be 160, 192, 224, 256, 320, 384, or 512. For Prime, it must be 192, 224, 256, 384, or 521.
- Rule array keywords ECC-VER0 and ECC-VER1 are not supported in releases before Release 5.2.
- Optional ECC key-derivation information section (X’23’) for an ECC private-key is not supported in releases before Release 5.2.
- RSA public exponent value of 5, 17, and 257 are not supported in releases before Release 5.2.

**Required commands**

The required commands for CSNDPKB.

None.

**Usage notes**

The usage notes for CSNDPKB.

None.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNDPKBJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNDPKBJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber key_values_structure_length,
    byte[] key_values_structure,
    hikmNativeNumber private_key_name_length,
    byte[] private_key_name,
    hikmNativeNumber user_definable_associated_data_length,
    byte[] user_definable_associated_data,
    hikmNativeNumber reserved_2_length,
    byte[] reserved_2,
    hikmNativeNumber reserved_3_length,
    byte[] reserved_3,
    hikmNativeNumber reserved_4_length,
    byte[] reserved_4,
)
```
PKA Key Token Change (CSNDKTC)

The PKA Key Token Change verb changes PKA key tokens (RSA or ECC) or trusted block key tokens, from encipherment under old ASYM-MK or APKA-MK, to encipherment under the current ASYM-MK or APKA-MK master key.

IMPORTANT

Two problems have been discovered with the CCA microcode related to the reenciphering of master keys. Although similar, the two problems are slightly different and exist in different levels of the microcode. These problems could lead to a loss of operational private keys after a master key change. Symmetric keys are not affected. Although it is expected few customers will be impacted this document describes the problems and how to recover.

For details, see http://www.ibm.com/support/techdocs/atmastr.nsf/WebIndex/FLASH10764.

The PKA Key Token Change (CSNDKTC) verb has been changed to not permit the use of the RTNRMK keyword for processor firmware levels that have this problem.

- For RSA key tokens - Key tokens must be private internal PKA key tokens in order to be changed by this verb. PKA private keys encrypted under the Key Management Master Key (KMMK) cannot be reenciphered using this service unless the KMMK has the same value as the Signature Master Key (SMK).
- For trusted block key tokens - Trusted block key tokens must be internal.
- For ECC key tokens - Key tokens must be private internal ECC key tokens encrypted under the APKA-MK.

Format

The format of CSNDKTC.

```
CSNDKTC(  return_code,  reason_code,  exit_data_length,  exit_data,  rule_array_count,  rule_array,  key_identifier_length,  key_identifier)
```

Parameters

The parameters for CSNDKTC.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

Direction: Input  
Type: Integer

A pointer to an integer variable containing the number of elements in the
The process rule for the verb. The keyword must be in eight bytes of contiguous storage, left-aligned, and padded on the right with blanks. The *rule_array* keywords are described in Table 195.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Token type</strong> (One, optional)</td>
<td></td>
</tr>
<tr>
<td>ECC</td>
<td>Specifies that the key being changed is an ECC key. This keyword was introduced with CCA 4.1.0.</td>
</tr>
<tr>
<td>RSA</td>
<td>Specifies that the key being changed is an RSA key or a trusted block. This is the default. This keyword was introduced with CCA 4.1.0.</td>
</tr>
<tr>
<td><strong>Reencipherment method</strong> (One, required)</td>
<td></td>
</tr>
<tr>
<td>RTCMK</td>
<td>If the <em>key_identifier</em> is an RSA key token, the verb will change an RSA private key from encipherment with the old ASYM-MK to encipherment with the current ASYM-MK. If the <em>key_identifier</em> is a trusted block token, the verb will change the trusted block's embedded MAC key from encipherment with the old ASYM-MK to encipherment with the current ASYM-MK. If the <em>key_identifier</em> is an ECC key token, the verb will change an ECC private key from encipherment with the old APKA-MK to encipherment with the current APKA-MK.</td>
</tr>
<tr>
<td>RTNMK</td>
<td>Re-enciphers a private (internal) RSA or ECC key to the new master key. A key enciphered under the new master key is not usable. It is expected that the user will use this keyword (RTNMK) to take a preparatory step in re-enciphering an external key store that they manage themselves to a new master-key, before the set operation has occurred. Note also that the new master-key register must be full; it must have had the last key part loaded and therefore not be empty or partially full (partially full means that one or more key parts have been loaded but not the last key part). The 'SET' operation makes the new master-key operational, moving it to the current master-key register, and the current master-key is displaced into the old master-key register. When this happens, all the keys that were re-enciphered to the new master-key are now usable, because the new master-key is not 'new' any more, it is 'current'. Because the RTNRMK keyword is added primarily for support of externally managed key storage (see &quot;Key Storage on z/OS (RTNRMK-focused)&quot; on page 453, it is not valid to pass a <em>key_identifier</em> when the RTNRMK keyword is used. Only a full internal key token (encrypted under the current master-key) can be passed for re-encipherment with the RTNRMK keyword. When a key LABEL is passed along with the RTNRMK keyword, the error return code 8 with reason code 63 will be returned. For more information, see &quot;Key storage with Linux on Z, in contrast to z/OS&quot; on page 451.</td>
</tr>
<tr>
<td>VALIDATE</td>
<td>Validate an internal PKA key token which is under the current master key (same processing as RTNRMK without checking the new master key or actually re-enciphering the token).</td>
</tr>
</tbody>
</table>

**key_identifier_length**

- **Direction:** Input
- **Type:** Integer

The length of the *key_identifier* parameter. The maximum size is 3500 bytes.

**key_identifier**
PKA Key Token Change (CSNDKTC)

Direction: Input/Output
Type: String

Contains an internal RSA key-token, 134 CCA Basic Services November, 2016 ECC key-token, or an active trusted block, or a key label identifying such a key-record in PKA key-storage. The master-key encrypted data within the RSA key-token or trusted block is securely reenciphered under the current PKA master key, and any such key token records are stored in PKA key-storage. Beginning with Release 4.4, the master-key enciphered OPK contained in an RSA private-key section with section identifier of X'30' or X'31' is securely reenciphered under the current APKA master key, and any such key token records are stored in AES key-storage. The enciphered OPK of an ECC private key-token is securely reenciphered under the current APKA master key. ECC key-token records are stored in PKA key-storage.

Restrictions
The restrictions for CSNDKTC.
None.

Required commands
The required commands for CSNDKTC.

This verb requires the PKA Key Token Change RTCMK command (offset X'0102') to be enabled in the active role.

To disable the wrapping of a key with a weaker master key, the Prohibit weak wrapping - Master keys command (offset X'0333') must be enabled in the active role.

To receive a warning when wrapping a key with a weaker master key, enable the Warn when weak wrap - Master keys command (offset X'0332') in the active role. The Prohibit weak wrapping - Master keys command overrides this command.

Usage notes
The usage notes for CSNDKTC.
None.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNDKTCJ.

See “Building Java applications using the CCA JNI” on page 28.

Format
public native void CSNDKTCJ(
  hikmNativeNumber return_code,
  hikmNativeNumber reason_code,
  hikmNativeNumber exit_data_length,
  byte[] exit_data,
  hikmNativeNumber rule_array_count,
  byte[] rule_array,
  hikmNativeNumber key_identifier_length,
  byte[] key_identifier);
PKA Key Translate (CSNDPKT)

Use the PKA Key Translate verb to translate an RSA key in a PKA key-token using an output format specified by the input rule array. The RSA key to be translated is provided in a source PKA key-token that contains a private-key section, and the translated key is returned in the buffer identified by the target_key_token parameter. If the source key is in an external key-token, the source transport key must be in an operational fixed-length DES key-token.

This verb changes only Private Internal PKA Key Tokens.

The source CCA RSA key token must be wrapped with a transport key encrypting key (KEK). The XLATE bit must also be turned on in the key usage byte of the source token. The source token is unwrapped using the specified source transport KEK. The target key token will be wrapped with the specified target transport KEK. Existing information in the target token is overwritten.

There are three types of output formatting available described in the subsequent sections. The first type is an external-to-external translation of an RSA key to one of three smart card formats. The second is an external-to-external or internal-to-internal translation of an RSA key to a target PKA key-token that is protected by an AES transport key or an APKA master key. The third is an external-to-external translation of an RSA key to one of three EMV formats.

**Target smart card formats**

To use this verb to translate an RSA key into a smart-card format, do the following:

- Specify one of the following rule-array keywords for the smart-card format to apply to the target key:

  **SCVISA**
  Specifies translation of an RSA private-key to the Visa proprietary format. This format is defined in *Visa Smart Debit/Credit Technical Guide to Visa Applets for GlobalPlatform Cards* and *Visa Smart Debit/Credit Personalization Guide for GlobalPlatform Cards*.

  **SCCOMME**
  Specifies translation of an RSA private-key to the common Modulus-Exponent (M-E) format. This format is defined in *Visa Smart Debit/Credit Technical Guide to Visa Applets for GlobalPlatform Cards*.

  **SCCOMCRT**
  Specifies translation of an RSA private-key to the common Chinese-Remainder Theorem (CRT) format. This format is defined in *Visa Smart Debit/Credit Technical Guide to Visa Applets for GlobalPlatform Cards*.

**Note:**

1. Translation from an M-E format to a CRT format is not supported.
2. Translation from a CRT format to an M-E format is not supported.

- Specify the source key identifier of an external PKA key-token that has been protected by a fixed-length DES transport key to be translated. The RSA private key for smart card output formats SCVISA, SCCOMME, and SCCOMCRT must have translation control of XLATE-OK (offset 50 in the private-key section).
- Specify the source transport key identifier of an operational fixed-length DES transport key (EXPORTER or IMPORTER) to be used to unwrap the source key.
PKA Key Translate (CSNDPKT)

- Specify the target transport key identifier of an operational fixed-length DES transport key to be used to wrap the unwrapped source key. The control vector of the target transport key must have CV bit 22 = B’1’ (XLA).

**Note:** Translation using an EXPORTER source transport key and an IMPORTER target transport key is not allowed.

- Specify the buffer of the target key token, to receive the returned external TDES-wrapped PKA key-token.

The verb builds the target external key-token using the chosen smart card format as follows:

1. The external RSA source key-token is unwrapped using the operational fixed-length DES source transport key.
2. The unwrapped key material is formatted into the specified target smart-card format.
3. The formatted key-material is TDES encrypted in ECB mode using the operational fixed-length DES target transport key.
4. The formatted and encrypted key material is written to the buffer identified by the `target_key_token` parameter, and the target key token length is updated.

**Target AES-protected formats**

To use this verb to translate an RSA key into an AES-protected format, specify:

- One of the following rule-array keywords for the AES-protected format to apply to the target key:

  **EXTDWAKW**
  specifies translation of an RSA key in an external TDES-wrapped (DES transport key) RSA key-token into an external AESKW-wrapped RSA key-token

  **INTDWAKW**
  specifies translation of an RSA key in an internal TDES-wrapped (PKA master key) RSA key-token into an internal AESKW-wrapped (APKA master key) PKA key-token.

  **Note:** An AESKW-wrapped key is protected at a higher level than a TDES-wrapped key.

- The source key identifier of a PKA key-token to be translated, either in an external key-token that has been protected by a fixed-length DES transport key to be translated, or an internal key-token that has been protected by a PKA master key.

- Source key is in an external PKA key-token:
  The source transport key identifier of an operational fixed-length DES transport key (EXPORTER or IMPORTER) to be used to unwrap the source key. The control vector of the source transport key does not require the XLA TE bit on.
  The target transport key identifier of an operational variable-length AES transport key to be used to wrap the OPK data of the target key-token. In addition, the key usage fields must have the algorithm wrap control set so that the key can wrap or unwrap RSA keys (WR-RSA).
  The buffer of the target key-token, to receive the external RSA key-token with AES-wrapped OPK.

- Source key is in an internal PKA key-token:
Set the source transport key identifier length to 0 or identify a null key-token as the source transport key.
Set the target transport key identifier length to 0 or identify a null key-token as the target transport key.
The buffer of the target key-token, to receive the internal PKA key-token with APKA-wrapped OPK.

The verb builds the target external key-token using the chosen AES-protected format as follows:
1. The source key-token is unwrapped using the source transport key or a PKA master-key, as appropriate.
2. The unwrapped key material is wrapped using the AES OPK of the target key-token. The OPK in turn is wrapped by the AES key from the target transport key if the target key is external, or the APKA master key if the target key is internal.
3. The completed external or internal key-token is written to the key token buffer identified by the target_key_token parameter, and the target key token length is updated.

**Target EMV formats**

To use this verb to translate an RSA key into an EMV format, specify one of the following rule_array keywords for the EMV format to apply to the target key:

**EMVCRT**
specifies the translation of an RSA CRT private-key to an EMV CRT format and wrapped using TDES-ECB

**EMVDDA**
specifies the translation of an RSA CRT private-key to an EMV DDA format and wrapped using TDES-CBC

**EMVDDAE**
specifies the translation of an RSA CRT private-key to an EMV DDA format and wrapped using TDES-ECB.

**Note:** The PKA source key-token must have a private key section of X'08', and the bit length of the modulus must be 512 - 2040.

**Format**
The format of CSNDPKT.

```
CSNDPKT(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    source_key_identifier_length,
    source_key_identifier,
    source_transport_key_identifier_length,
    source_transport_key_identifier,
    target_transport_key_identifier_length,
    target_transport_key_identifier,
    target_key_token_length,
    target_key_token)
```
PKA Key Translate (CSNDPKT)

Parameters

The parameters for CSNDPKT.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

`rule_array_count`

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1.

`rule_array`

- **Direction:** Input
- **Type:** String array

The process rule for the verb. The keyword must be in eight bytes of contiguous storage, left-aligned, and padded on the right with blanks. The `rule_array` keywords are described in Table 196.

Table 196. Keywords for PKA Key Translate control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output format</strong> (One, required)</td>
<td></td>
</tr>
<tr>
<td>EXTDWAKW</td>
<td>Specifies that the source key is an external DES wrapped token to be converted to an AESKW wrapped token.</td>
</tr>
<tr>
<td>INTDWAKW</td>
<td>Specifies that the source key is an internal DES wrapped token to be converted to an AESKW wrapped token.</td>
</tr>
<tr>
<td>EMVDDA</td>
<td>This keyword indicates translating an external RSA CRT key into EMV DDA format and wrapping with TDES-CBC. The XLA TE bit (bit 22) must be set in the target_transport_key control vector.</td>
</tr>
<tr>
<td>EMVDDAE</td>
<td>This keyword indicates translating an external RSA CRT key into EMV DDAE format and wrapping with TDES-ECB. The XLA TE bit (bit 22) must be set in the target_transport_key control vector.</td>
</tr>
<tr>
<td>EMVCRT</td>
<td>This keyword indicates translating an external RSA CRT key into EMV CRT format and wrapping with TDES-ECB. The XLA TE bit (bit 22) must be set in the target_transport_key control vector.</td>
</tr>
<tr>
<td>SCCOMCRT</td>
<td>This keyword indicates translating the key into the smart card Chinese Remainder Theorem format.</td>
</tr>
<tr>
<td>SCCOMME</td>
<td>This keyword indicates translating the key into the smart card Modulus-Exponent format.</td>
</tr>
<tr>
<td>SCVISA</td>
<td>This keyword indicates translating the key into the smart card Visa proprietary format.</td>
</tr>
<tr>
<td><strong>Format restriction</strong> (One, optional). As of release 6.0 or later, only valid with token type keyword EXTDWAKW or INTDWAKW.</td>
<td></td>
</tr>
<tr>
<td>FR-NONE</td>
<td>Specifies to not restrict the private key to be used by a particular digital-signature hash formatting method. The key is usable for any method. This is the default.</td>
</tr>
<tr>
<td>FR-I9796</td>
<td>Specifies to render the private key usable only with the digital-signature hash formatting method ISO-9796.</td>
</tr>
<tr>
<td>FR-X9.31</td>
<td>Specifies to render the private key usable only with the digital-signature hash formatting method X9.31.</td>
</tr>
<tr>
<td>FR-ZPAD</td>
<td>Specifies to render the private key usable only with the digital-signature hash formatting method ZERO-PAD.</td>
</tr>
</tbody>
</table>
Table 196. Keywords for PKA Key Translate control information  (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR-PK10</td>
<td>Specifies to render the private key usable only with the digital-signature hash formatting method PKCS-1.0.</td>
</tr>
<tr>
<td>FR-PK11</td>
<td>Specifies to render the private key usable only with the digital-signature hash formatting method PKCS-1.1.</td>
</tr>
<tr>
<td>FR-PSS</td>
<td>Specifies to render the private key usable only with the digital-signature hash formatting method PKCS-PSS.</td>
</tr>
</tbody>
</table>

**source_key_length**

- **Direction:** Input
- **Type:** Integer

The length of the **source_key** parameter. The maximum size is 3500 bytes.

**source_key**

- **Direction:** Input
- **Type:** String

This field contains either a key label identifying an RSA private key, or an external public-private key token. The private key must be wrapped with a key encrypting key.

**source_transport_key_length**

- **Direction:** Input
- **Type:** Integer

Length in bytes of the **source_transport_key** parameter. This value must be 64.

**source_transport_key**

- **Direction:** Input/Output
- **Type:** String

This field contains an internal token or label of a DES key-encrypting key. This key is used to unwrap the input RSA key token specified with parameter **source_key**. See “Usage notes” on page 719 for details on the type of transport key that can be used.

**target_transport_key_length**

- **Direction:** Input
- **Type:** Integer

Length in bytes of the **target_transport_key** parameter. This value must be 64.

**target_transport_key**

- **Direction:** Input/Output
- **Type:** String

This field contains an internal token or label of a DES key-encrypting key. This key is used to wrap the output RSA key returned with the **target_key_token** parameter. See “Usage notes” on page 719 for details on the type of transport key that can be used.

**target_key_token_length**
PKA Key Translate (CSNDPKT)

**Direction:** Input
**Type:** Integer

Length in bytes of the `target_key_token` parameter. On output, the value in this variable is updated to contain the actual length of the `target_key_token` produced by the verb. The maximum length is 3500 bytes.

**target_key_token**

**Direction:** Output
**Type:** String

This field contains the RSA key in the smartcard format specified in the `rule_array` parameter, and is protected by the key-encrypting key specified in the `target_transport_key` parameter. This is not a CCA token, and cannot be stored in the key storage.

**Restrictions**

The restrictions for CSNDPKT.

CCA RSA Modulus-Exponent tokens will not be translated to the SCCOMCRT format. CCA RSA Chinese Remainder Theorem tokens will not be translated to the SCCOMME format. SCVISA supports only Modulus-Exponent (ME) keys.

**Required commands**

The required commands for CSNDPKT.

This verb requires the following commands to be enabled in the active role based on the keyword:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMVCRT</td>
<td>X'033A'</td>
<td>PKA Key Translate - from CCA RSA CRT to EMV CRT format</td>
</tr>
<tr>
<td>EMVDDA</td>
<td>X'0338'</td>
<td>PKA Key Translate - from CCA RSA CRT to EMV DDA format</td>
</tr>
<tr>
<td>EMVDDAE</td>
<td>X'0339'</td>
<td>PKA Key Translate - from CCA RSA CRT to EMV DDAE format</td>
</tr>
<tr>
<td>EXTDWAKW</td>
<td>X'00FF'</td>
<td>PKA Key Translate - Translate external key token</td>
</tr>
<tr>
<td>INTDWAKW</td>
<td>X'00FE'</td>
<td>PKA Key Translate - Translate internal key token</td>
</tr>
<tr>
<td>SCVISA</td>
<td>X'0318'</td>
<td>PKA Key Translate - from CCA RSA to SC Visa Format</td>
</tr>
<tr>
<td>SCCOMME</td>
<td>X'0319'</td>
<td>PKA Key Translate - from CCA RSA to SC ME Format</td>
</tr>
<tr>
<td>SCCOMCRT</td>
<td>X'031A'</td>
<td>PKA Key Translate - from CCA RSA to SC CRT Format</td>
</tr>
</tbody>
</table>

These commands must also be enabled to allow the key type combinations shown in this table:

<table>
<thead>
<tr>
<th>Source transport key type</th>
<th>Target transport key type</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPORTER</td>
<td>EXPORTER</td>
<td>X'031B'</td>
<td>PKA Key Translate - from source EXP KEK to target EXP KEK</td>
</tr>
<tr>
<td>IMPORTER</td>
<td>EXPORTER</td>
<td>X'031C'</td>
<td>PKA Key Translate - from source IMP KEK to target EXP KEK</td>
</tr>
</tbody>
</table>
PKA Key Translate (CSNDPKT)

<table>
<thead>
<tr>
<th>Source transport key type</th>
<th>Target transport key type</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMPORTER</td>
<td>IMPORTER</td>
<td>X'031D'</td>
<td>PKA Key Translate - from source IMP KEK to target IMP KEK</td>
</tr>
<tr>
<td>EXPORTER</td>
<td>IMPORTER</td>
<td>N/A</td>
<td>This key type combination is not allowed.</td>
</tr>
</tbody>
</table>

The following access control points control the use of weak transport keys:

- To disable the wrapping of a key with a weaker transport key, the Prohibit weak wrapping - Transport keys command (offset X'0328') must be enabled in the active role.
- To receive an informational message when wrapping a key with a weaker key-encrypting key, enable the Warn when weak wrap - Transport keys command (offset X'032C') in the active role. The Prohibit weak wrapping - Transport keys command overrides this command.

The following access control points control the use of weak master keys:

- To disable the wrapping of a key with a weaker master key, the Prohibit weak wrapping - Master keys command (offset X'0333') must be enabled in the active role.
- To receive a warning when wrapping a key with a weaker master key, enable the Warn when weak wrap - Master keys command (offset X'0332') in the active role. The Prohibit weak wrapping - Master keys command overrides this command.

Usage notes

The usage notes for CSNDPKT.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNDPKTJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNDPKTJ (  
    hikmNativeNumber return_code,  
    hikmNativeNumber reason_code,  
    hikmNativeNumber exit_data_length,  
    byte[] exit_data,  
    hikmNativeNumber rule_array_count,  
    byte[] rule_array,  
    hikmNativeNumber source_key_identifier_length,  
    byte[] source_key_identifier,  
    hikmNativeNumber source_transport_key_identifier_length,  
    byte[] source_transport_key_identifier,  
    hikmNativeNumber target_transport_key_identifier_length,  
    byte[] target_transport_key_identifier,  
    hikmNativeNumber target_key_token_length,  
    byte[] target_key_token);
```

Chapter 15. Managing PKA cryptographic keys 719
PKA Public Key Extract (CSNDPKX)

Use the PKA Public Key Extract verb to extract a PKA public key token from a supplied PKA internal or external private key token.

This verb performs no cryptographic verification of the PKA private token. You can verify the private token by using it in a verb such as Digital Signature Generate.

Format

The format of CSNDPKX.

```
CSNDPKX(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    source_key_identifier_length,
    source_key_identifier,
    target_public_key_identifier_length,
    target_public_key_identifier)
```

Parameters

The parameters for CSNDPKX.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

`rule_array_count`

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 0.

`rule_array`

Direction: Input
Type: String array

This parameter is ignored.

`source_key_identifier_length`

Direction: Input
Type: Integer

The length of the `source_key_identifier` parameter. The maximum size is 3500 bytes. When the `source_key_identifier` parameter is a key label, this field specifies the length of the label.

`source_key_identifier`

Direction: Input/Output
Type: String

The internal or external token of a PKA private key or the label of a PKA private key. This can be the input or output from the PKA Key Import or PKA...
Key Generate verbs. This verb supports:
• RSA private key token formats supported on the CEX*C. If the source_key_identifier specifies a label for a private key that has been retained within a CEX*C, this verb extracts only the public key section of the token.
• ECC private key token formats supported starting with CEX3C.

target_public_key_identifier_length
Direction: Input/Output
Type: Integer

The length of the target_public_key_identifier parameter. The maximum size is 2500 bytes. On output, this field will be updated with the actual byte length of the target_public_key_token.

target_public_key_identifier
Direction: Output
Type: String

This field contains the token of the extracted PKA public key.

Restrictions
The restrictions for CSNDPKX.

None.

Required commands
The required commands for CSNDPKX.

None.

Usage notes
The usage notes for CSNDPKX.

This verb extracts the public key from the internal or external form of a private key. However, it does not check the cryptographic validity of the private token.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNDPKXJ.

See “Building Java applications using the CCA JNI” on page 28.

Format
public native void CSNDPKXJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber source_key_identifier_length,
    byte[] source_key_identifier,
    hikmNativeNumber target_public_key_identifier_length,
    byte[] target_public_key_identifier);
Remote Key Export (CSNDRKX)

This verb is used as a method of secured transport of DES keys using asymmetric techniques from a security module (for example, the CEX3C) to a remote device such as an Automated Teller Machine (ATM).

The DES keys to be transported are either key encrypting keys that are generated within the coprocessor or, alternately, operational keys or replacement KEKs enciphered under a KEK currently installed in a remote device.

Generating and exporting DES keys

This verb uses a trusted block to generate or export DES keys.

To create a trusted block, see “Trusted Block Create (CSNDTBC)” on page 734. Remote Key Export accepts as input parameters a trusted block, a public-key certificate and certificate parameters, a transport key, a rule ID to identify the appropriate rule section to be used within a trusted block, an importer key, a source key, optional extra data that can be used as part of the OAEP key-wrapping process, and key-check parameters used to calculate an optional key-check value.

This verb validates all input parameters for generate and export operations. After the verb performs the input parameter validation, the remaining steps depend on whether the generate option or the export option is specified in the selected rule of the trusted block.

This is a high-level description of the remaining processing steps for generate and export.

Processing for generate operation

The verb performs these steps for the generate operation:

1. Generates a random value for the generated key, K. The generated key length specified by the selected rule determines the key length.
2. XORs the output key variant with the randomly generated key K from the previous step, if the selected rule contains a common export key parameters subsection and the output key variant length is greater than zero. Adjusts the result to have valid DES key parity.
3. Continues with “Final processing common to generate and export operations” on page 723.

Processing for export operation

The verb performs these steps for the export operation:

1. If the selected rule contains a transport key rule reference subsection, verifies that the rule ID in the transport key rule reference subsection matches the rule ID in the token identified by the transport_key_identifier parameter, provided that the token is an RKX key-token. For more information on RKX key tokens, see “External RKX DES key tokens” on page 851.
2. Verifies that the length of the transport key variant in the transport key variant subsection of the selected rule is greater than or equal to the length of the key identified by the transport_key_identifier parameter.
3. Verifies that the key token identified by the importer_key_identifier parameter is of key type IMPORTER, if the source_key_identifier parameter identifies an external CCA DES key-token.
4. Recovers the clear value of the source key, K, identified by the `source_key_identifier` parameter.

5. Verifies that the length of key K is between the export key minimum length and export key maximum length specified in the common export key parameters subsection of the selected rule.

6. XORs the output key variant with the randomly generated key K from the previous step, if the selected rule contains a common export key parameters subsection and the output key variant length is greater than zero. Adjusts the result to have valid DES key parity.

7. Uses the public key in the trusted block to verify the digital signature embedded in the certificate variable if the `certificate_length` variable is greater than zero. Any necessary certificate objects are located with information from the `certificate_parms` variable. Returns an error if the signature verification fails.

8. XORs the transport key variant with the clear value of the transport key (recovered in the previous step) if the selected rule contains a transport key variant subsection and the output key variant length is greater than zero. Adjusts the result to have valid DES key parity.

9. Continues with “Final processing common to generate and export operations.”

**Final processing common to generate and export operations**

1. Based on the symmetric encrypted output key format flag of the selected rule, returns the encrypted result in the token identified by the `sym_encrypted_key_identifier` parameter:
   - of “Processing for generate operation” on page 722, step 2, or of “Processing for export operation” on page 722, step 6, into an RKX key-token, if the flag indicates to return an RKX key-token.
   - using the resulting key from “Processing for export operation” on page 722, step 6 into a CCA DES key-token and returns it in the token identified by the `sym_encrypted_key_identifier` parameter, if the flag indicates to return a CCA DES key-token.

2. Encrypts the key result from “Processing for generate operation” on page 722, step 2 or from “Processing for export operation” on page 722, step 6, with the format specified, if the asymmetric encrypted output key format flag of the selected rule indicates to output an asymmetric encrypted key and return it to the `asym_encrypted_key` parameter.

3. Returns the computed key-check value as determined by the key-check algorithm identifier if the key-check algorithm identifier in the specified rule indicates to compute a key-check value. The value is returned in the `key_check_value` variable.
Remote Key Export (CSNDRKX)

Format

The format of CSNDRKX.

```c
CSNDRKX(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    trusted_block_identifier_length,
    trusted_block_identifier,
    certificate_length,
    certificate,
    certificateParms_length,
    certificateParms,
    transport_key_identifier_length,
    transport_key_identifier,
    rule_id_length,
    rule_id,
    importer_key_identifier_length,
    importer_key_identifier,
    source_key_identifier_length,
    source_key_identifier,
    asym_encrypted_key_length,
    asym_encrypted_key,
    sym_encrypted_key_identifier_length,
    sym_encrypted_key_identifier,
    extra_data_length,
    extra_data,
    key_check_parameters_length,
    key_check_parameters,
    key_check_value_length,
    key_check_value
)
```

Parameters

The parameters for CSNDRKX.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

**rule_array_count**

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of keywords in the `rule_array` variable. This number must be 0, 1 or 2.

**rule_array**

Direction: Output
Type: String array

The `rule_array` is an array of keywords. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks. The `rule_array` keywords are described in Table 197 on page 725.
Table 197. Keywords for Remote Key Export control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key wrapping method</strong> (Optional)</td>
<td></td>
</tr>
<tr>
<td>USECONFG</td>
<td>This is the default. Specifies to wrap the key using the configuration setting for the default wrapping method. The default wrapping method configuration setting may be changed using the TKE. This keyword is ignored for AES keys.</td>
</tr>
<tr>
<td>WRAP-ENH</td>
<td>Specifies that the new enhanced wrapping method is to be used to wrap the key.</td>
</tr>
<tr>
<td>WRAP-ECB</td>
<td>Specifies that the original wrapping method is to be used.</td>
</tr>
<tr>
<td><strong>Translation control</strong> (Optional, valid only for enhanced wrapping)</td>
<td></td>
</tr>
<tr>
<td>ENH-ONLY</td>
<td>Specify this keyword to indicate that the key once wrapped with the enhanced method cannot be wrapped with the original method. This restricts translation to the original method.</td>
</tr>
</tbody>
</table>

**trusted_block_identifier_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the `trusted_block_identifier` variable. The maximum length is 3500 bytes.

**trusted_block_identifier**

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing a trusted block key-token of an internal trusted block, or the key label of a trusted block key-token record of an internal trusted block. It is used to validate the public-key certificate and to define the rules for key generation and key export.

**certificate_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the `certificate` variable. The maximum length is 5000 bytes.

It is an error if the `certificate_length` variable is 0 and the trusted block’s asymmetric encrypted output key format in the rule section selected by the `rule_id` variable indicates PKCS-1.2 output format or RSA-OAEP output format.

If the `certificate_length` variable is 0 or the trusted block’s asymmetric encrypted output key format in the rule section selected by the `rule_id` variable indicates no asymmetric key output, the certificate is ignored.

**certificate**

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing a public-key certificate. The certificate must contain the public-key modulus and exponent in binary form, as well as a digital certificate. The certificate must verify using the root public key that is in the trusted block pointed to by the `trusted_block_identifier` parameter.

**Note:** After the hash is computed over the certificate data specified by offsets 28 and 32, the hash is BER encoded by pre-pending these bytes:
Remote Key Export (CSNDRKX)

X'30213009 06052B0E 03021A05 000 414'

See “PKCS #1 hash formats” on page 1044.

certificate_Parms_length

Direction:  Input
Type:  Integer

A pointer to an integer variable containing the number of bytes of data in the
certificate_Parms variable. The length must be 36 bytes if the certificate_length
variable is 0, else the length must be 0.

certificate_Parms

Direction:  Input
Type:  String

A pointer to a string variable containing a structure for identifying the location
and length of values within the public-key certificate pointed to by the
certificate parameter. If the value of the certificate_length variable is 0, then the
information in this variable is ignored but the variable must be declared. The
format of the certificate_Parms variable is defined in Table 198.

Table 198. Keywords for Remote Key Export certificate_Parms parameter

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>Offset of modulus</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Length of modulus</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>Offset of public exponent</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>Length of public exponent</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>Offset of digital signature</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>Length of digital signature</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>Identifier for hash algorithm. The following values are defined:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Identifier</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Hash algorithm</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>Identifier for digital signature hash formatting method used. The following values are defined:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Identifier</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Hash formatting method</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'04'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'05'</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>Reserved, must be binary zeros</td>
</tr>
<tr>
<td>28</td>
<td>4</td>
<td>Offset of first byte of certificate data hashed to compute the digital signature</td>
</tr>
<tr>
<td>32</td>
<td>4</td>
<td>Length of certificate data hashed to compute the digital signature</td>
</tr>
</tbody>
</table>
Note: The modulus, exponent, and signature values can have bit lengths that are not multiples of 8; each of these values is right-aligned and padded on the left with binary zeroes to make it an even number of bytes in length.

**transport_key_identifier_length**

Direction: Input  
Type: Integer  

A pointer to an integer variable containing the number of bytes of data in the `transport_key_identifier` variable. The length must be 0 or 64 bytes.

**transport_key_identifier**

Direction: Input  
Type: String  

A pointer to a string variable containing a KEK key-token, or a key label of a KEK key-token record. The KEK is either an internal CCA DES key-token (key type IMPORTER or EXPORTER), or an external version X'10' (RKX) DES key-token. It is used to encrypt a key exported by the verb.

When the symmetric encrypted output key format flag of the selected rule indicates return an RKX key-token, this parameter is ignored but must be declared.

- If this parameter points to a CCADES key-token, the token must be of key type IMPORTER or EXPORTER.
- If the `source_key_identifier` parameter identifies an internal CCA DES key-token, the token must be of key type EXPORTER.

For more information on RKX key tokens, see ["External RKX DES key tokens"](on page 851).

**rule_id_length**

Direction: Input  
Type: Integer  

A pointer to an integer variable containing the number of bytes of data in the `rule_id` variable. The length must be eight bytes.

**rule_id**

Direction: Input  
Type: String  

A pointer to a string variable that identifies the rule in the trusted block to be used to control key generation or export. The trusted block can contain multiple rules, each of which is identified by a unique rule ID value.

**importer_key_identifier_length**

Direction: Input  
Type: Integer  

A pointer to an integer variable containing the number of bytes of data in the `importer_key_identifier` variable. The length must be 0 or 64 bytes.

**importer_key_identifier**

Direction: Input  
Type: String
Remote Key Export (CSNDRKX)

A pointer to a string variable containing an IMPORTER KEK key-token or a label of an IMPORTER KEK key-token record. This KEK is used to decipher the key pointed to by the source_key_identifier parameter.

This variable is ignored if the verb is used to generate a new key, or the source_key_identifier variable contains either an RKX key token or an internal CCA DES key-token. For more information on RKX key tokens, see "External RKX DES key tokens" on page 851.

source_key_identifier_length

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the source_key_identifier variable. The length must be 0 or 64 bytes.

source_key_identifier

Direction: Input
Type: String

A pointer to a string variable containing a DES key-token or a label of a DES key-token record. The key token contains the key to be exported, and must meet one of these criteria:

- It is a single-length or double-length external CCA DES key-token.
- It is a single-length or double-length internal CCA DES key-token.
- It is a single-length, double-length, or triple-length RKX key-token.

Note:
1. If the key token is a CCA DES key-token, its XPORT-OK control vector bit (bit 17) must be B'1', or else the export will not be allowed.
2. If a DES key-token has three 8-byte key parts, the parts are considered unique if any two of the three key parts differ.

asym_encrypted_key_length

Direction: Input/Output
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the asym_encrypted_key variable. On output, the variable is updated with the actual length of the asym_encrypted_key variable. The input length must be at least the length of the modulus in bytes of the public-key in the certificate variable.

asym_encrypted_key

Direction: Output
Type: String

A pointer to a string variable containing a generated or exported clear key returned by the verb. The clear key is encrypted by the public (asymmetric) key provided by the certificate variable.

sym_encrypted_key_identifier_length

Direction: Input/Output
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the sym_encrypted_key_identifier variable. On output, the variable is updated with
the actual length of the `sym_encrypted_key_identifier` variable. The input length must be a minimum of 64 bytes.

**sym_encrypted_key_identifier**

**Direction:** Output  
**Type:** String

A pointer to a string variable. On input, the `sym_encrypted_key_identifier` variable must contain either a key label of a CCA DES key-token record or an RKX key-token record, or be filled with binary zeros.

On output, the verb produces a CCA DES key-token or an RKX key-token, depending on the value of the symmetric encrypted output key format value of the rule section within the `trusted_block_identifier` variable. The key token produced contains either a generated or exported key encrypted using the key-encrypting key provided by the `transport_key_identifier` variable.

- If the output is an external CCA DES key-token:
  1. If a common export key parameters subsection (X'0003') is present in the selected rule, the control vector (CV) is copied from the subsection into the output CCA DES key-token. Otherwise, the CV is copied from source key-token.
  2. If a transport key variant subsection (X'0001') is present in the selected rule, the key is multiply enciphered under the transport key XORed with the transport key variant from the subsection. Otherwise, the key is multiply enciphered under the transport key XORed with binary zero.
  3. XORs the CV in the token with the encrypted result from the previous step.
  4. Stores the previous result in the token and updates the TVV.
- If the output is an (external) RKX key-token:
  1. Encrypts the key using a variant of the trusted block MAC key.
  2. Builds the token with the encrypted key and the `rule_id` variable.
  3. Calculates the MAC of the token contents and stores the result in the token.

If the `sym_encrypted_key_identifier` variable is a key label on input, on output the key token produced by the verb is stored in DES key-storage and the variable remains the same. Otherwise, on output the variable is updated with the key token produced by the verb, provided the field is of sufficient length.

**extra_data_length**

**Direction:** Input  
**Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the `extra_data` variable. The length must be less than or equal to the byte length of the certificate public key modulus minus the generated/exported key length minus 42 (X'2A'), which is the OAEP overhead. For example, if the public key in the certificate has a modulus length of 1024 bits (128 bytes), and the exported key is single length, then the extra data length must be less than or equal to 128 minus 8 minus 42, which equals 78.

**extra_data**

**Direction:** Input  
**Type:** String

A pointer to a string variable containing extra data to be used as part of the
Remote Key Export (CSNDRKX)

OAEP key-wrapping process. The extra_data variable is used when the output format for the RSA-encrypted key that is returned in the asym_encrypted_key variable is RSA-OAEP; otherwise, it is ignored.

**Note:** The RSA-OAEP format is specified as part of the rule in the trusted block.

**key_check_parameters_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the key_check_parameters variable. The length must be 0.

**key_check_parameters**

- **Direction:** Input
- **Type:** String

Reserved for future use.

**key_check_value_length**

- **Direction:** Input/Output
- **Type:** Integer

A pointer to a string variable containing the number of bytes of data in the key_check_value variable. On output, and if the field is of sufficient length, the variable is updated with the actual length of the key_check_value variable.

**key_check_value**

- **Direction:** Output
- **Type:** String

A pointer to a string variable containing the result of the key-check algorithm chosen in the rule section of the selected trusted block. See "Encrypt zeros DES-key verification algorithm" on page 1024 and "Modification Detection Code calculation" on page 1024. When the selected key-check algorithm is to encrypt an 8-byte block of binary zeros with the key, and the generated or exported key is:

- **Single length**
  1. A value of 0, 1, or 2 is considered insufficient space to hold the output encrypted result, and the verb returns an error.
  2. A value of 3 returns the leftmost three bytes of the encrypted result if the key_check_value_length variable is 3 or greater. Otherwise, an error is returned.
  3. A value of 4 - 8 returns the leftmost four bytes of the encrypted result if the key_check_value_length variable is 4 or greater. Otherwise, an error is returned.

- **Double length or triple length**
  The verb returns the entire 8-byte result of the encryption in the key_check_value variable if the key_check_value_length variable is 8 or more. Otherwise, an error is returned.

When the selected key-check algorithm is to compute the MDC-2 hash of the key, and the generated or exported key is single length, the 8-byte key is made into a double-length key by replicating the key halves. This is because the MDC-2 calculation method does no padding, and requires that the data be a
minimum of 16 bytes and a multiple of eight bytes. If the generated or exported key is double length or triple length, the key is processed as is. The verb returns the 16-byte hash result of the key in the key_check_value variable if the key_check_value_length variable is large enough, else an error is returned.

Restrictions

The restrictions for CSNDRKX.

- AES keys are not supported by this verb.
- Keys with a modulus length greater than 2048 bits are not supported in releases before Release 3.30.
- The maximum public exponent is 17 bits for any key that has a modulus greater than 2048 bits.
- A key identifier length must be 64 for a key label.
- Key-wrapping method keywords are not supported in releases before Release 4.4.
- A key-wrapping method rule-array keyword cannot be specified if the symmetric encrypted output key format flag is not set to return a CCA fixed-length DES key-token.

Required commands

The required commands for CSNDRKX.

This verb requires the **Remote Key Export - Gen or export a non-CCA node key** command (offset X'0312') to be enabled in the active role.

The verb also requires the **Key Generate - SINGLE-R** command (offset X'00DB') to be enabled to replicate a single-length source key (either from a CCA DES key-token or an RKX key-token). If authorized, key replication occurs if all of the following are true:

1. The key token returned using the sym_encrypted_key_identifier parameter is a CCA DES key-token, as defined in the rule section identified by the rule_id parameter.
2. The rule section identified by the rule_id parameter has a common export key parameters subsection defined, and the control vector in the subsection is 16 bytes in length with key-form bits of B'010' for the left half and B'001' for the right half.
3. The token identified by the source_key_identifier parameter is single length, and is either a CCA DES key-token or an RKX key-token.

**Note:** A role with X'00DB' enabled can also use the Key Generate verb with the SINGLE-R key-length keyword.

To enable the use of key-encrypting-keys with the NOCV option for export, this verb requires the **NOCV KEK usage for export-related functions** command (offset X'0300') to be enabled in the active role.

To enable the use of key-encrypting-keys with the NOCV option for import, this verb requires the **NOCV KEK usage for import-related functions** command (offset X'030A') to be enabled in the active role.

This verb also requires the following commands to be enabled in the active role:
Remote Key Export (CSNDRKX)

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>USECONFIG</td>
<td>X'013F'</td>
<td>Remote Key Export - include RKX in default wrap config</td>
</tr>
<tr>
<td>WRAP-ECB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRAP-ENH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENH-ONLY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRAP-ECB or WRAP-ENH and</td>
<td>X'02BA'</td>
<td>Remote Key Export - Allow wrapping override keywords</td>
</tr>
<tr>
<td>default key-wrapping method setting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>does not match keyword</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Commands related to wrapping DES keys coincide with the addition of rule array keywords to specify key-wrapping method and translation control:

- **Remote Key Export - include RKX in default wrap config** (offset X'013F')
  
  Enable this command in the active role to allow any rule array keywords to be specified or, in the absence of a key-wrapping method keyword, to have the verb wrap the DES key using the default key-wrapping configuration (USECONFIG) setting. Enabling X'013F' makes the verb behave in the same manner as other verbs that accept key-wrapping method and translation control keywords.

  **Note:** The purpose of offset X'013F' is to provide a way to maintain backward compatibility with the key-wrapping method used in releases before Release 4.4. To maintain backward compatibility with releases that do not accept key-wrapping method keywords, do not enable X'013F' in the active role.

- **Remote Key Export - Allow wrapping override keywords** (offset X'02BA')
  
  This command requires offset X'013F' to be enabled in the active role. Enable offset X'02BA' in the active role to allow a key-wrapping keyword to be specified that overrides the default key-wrapping configuration.

When an existing key is exported into a CCA fixed-length DES key-token (that is, offset 18 in the selected trusted block rule section is X'01') and the **Remote Key Export - include RKX in default wrap config** command (offset X'013F') is not enabled in the active role (or the release is before Release 4.4), the DES key identified by the `sym_encrypted_key_identifier` parameter is wrapped based on the following criteria depending on whether the source key is an RKX key-token or a CCA fixed-length DES key-token:

- **Source key is an RKX key-token:**

<table>
<thead>
<tr>
<th>Configuration setting of default wrapping method</th>
<th>CV bit 56 of rule subsection X'0003' of selected trusted block rule section</th>
<th>Wrapping method (and CV bit 56) of exported DES key</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRAP-ECB (legacy)</td>
<td>No CV (CV length = 0)</td>
<td>WRAP-ECB (B'0')</td>
</tr>
<tr>
<td></td>
<td>B'0' (not ENH-ONLY)</td>
<td>WRAP-ECB (B'0')</td>
</tr>
<tr>
<td></td>
<td>B'1' (ENH-ONLY)</td>
<td>Error</td>
</tr>
<tr>
<td>WRAP-ENH (enhanced)</td>
<td>No CV</td>
<td>WRAP-ENH (B'0')</td>
</tr>
<tr>
<td></td>
<td>B'0'</td>
<td>WRAP-ENH (B'0')</td>
</tr>
<tr>
<td></td>
<td>B'1'</td>
<td>WRAP-ENH (B'1')</td>
</tr>
</tbody>
</table>

- **Source key is a CCA fixed-length DES key-token:**
Remote Key Export (CSNDRKX)

<table>
<thead>
<tr>
<th>Source key wrapping method</th>
<th>CV bit 56 of source key</th>
<th>CV bit 56 of rule subsection X'0003' of selected trusted block rule section</th>
<th>Wrapping method (and CV bit 56) of exported DES key</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRAP-ECB (legacy)</td>
<td>B'0' (not ENH-ONLY)</td>
<td>No CV (CV length = 0)</td>
<td>WRAP-ECB (B'0')</td>
</tr>
<tr>
<td></td>
<td>B'0' (not ENH-ONLY)</td>
<td></td>
<td>WRAP-ECB (B'0')</td>
</tr>
<tr>
<td></td>
<td>B'1' (ENH-ONLY)</td>
<td></td>
<td>Error</td>
</tr>
<tr>
<td>WRAP-ENH (enhanced)</td>
<td>B'0'</td>
<td>No CV</td>
<td>WRAP-ENH (B'0')</td>
</tr>
<tr>
<td></td>
<td>B'0'</td>
<td></td>
<td>WRAP-ENH (B'0')</td>
</tr>
<tr>
<td></td>
<td>B'1'</td>
<td></td>
<td>WRAP-ENH (B'1')</td>
</tr>
<tr>
<td></td>
<td>B'1' (ENH-ONLY)</td>
<td>No CV</td>
<td>WRAP-ENH (B'1')</td>
</tr>
<tr>
<td></td>
<td>B'0'</td>
<td></td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td>B'1'</td>
<td></td>
<td>WRAP-ENH (B'1')</td>
</tr>
</tbody>
</table>

Usage notes
The usage notes for CSNDRKX.

None.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNDRKXJ.

See “Building Java applications using the CCA JNI” on page 28.

Format
public native void CSNDRKXJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber trusted_block_length,
    byte[] trusted_block_identifier,
    hikmNativeNumber certificate_length,
    byte[] certificate,
    hikmNativeNumber certificate_parms_length,
    byte[] certificate_parms,
    hikmNativeNumber transport_key_identifier_length,
    byte[] transport_key_identifier,
    hikmNativeNumber rule_id_length,
    byte[] rule_id,
    hikmNativeNumber importer_key_identifier_length,
    byte[] importer_key_identifier,
    hikmNativeNumber source_key_identifier_length,
    byte[] source_key_identifier,
    hikmNativeNumber asym_encrypted_key_length,
    byte[] asym_encrypted_key,
    hikmNativeNumber sym_encrypted_key_identifier_length,
    byte[] sym_encrypted_identifierkey,
    hikmNativeNumber extra_data_length,
    byte[] extra_data,
    hikmNativeNumber key_check_parameters_length,
    byte[] key_check_parameters,
    hikmNativeNumber key_check_value_length,
    byte[] key_check_value);
Trusted Block Create (CSNDTBC)

The verb creates an external trusted block under dual control. A trusted block is an extension of CCA PKA key tokens using new section identifiers.

Trusted blocks are an integral part of a remote key-loading process. They contain various items, some of which are optional, and some of which can be present in different forms. Tokens are composed of concatenated sections. For a detailed description of a trusted block, including its format and field values, see “Trusted blocks” on page 968.

Creating an external trusted block: Create an active external trusted block in two steps:

1. Create an inactive external trusted block using the INACTIVE rule_array keyword. This step requires the Trusted Block Create - Create Block in inactive form command (offset X'030F') to be enabled in the active role.

2. Complete the creation process by activating (promoting) an inactive external trusted block using the ACTIVE rule_array keyword. This step requires the Trusted Block Create - Activate an inactive block command (offset X'0310') to be enabled in the active role. Changing an external trusted block from inactive to active effectively approves the trusted block for further use.

Note: Authorize each command in a different role to enforce a dual-control policy.

The creation of an external trusted block typically takes place in a highly secure environment. Use “PKA Key Import (CSNDPKI)” on page 696 to import an active external trusted block into the desired node. The imported internal trusted block can then be used as input to “Remote Key Export (CSNDRKX)” on page 722 in order to generate or export DES keys.

Creating an inactive external trusted block: To create an inactive external trusted block, use a rule_array_count of 1 and a rule_array keyword of INACTIVE. Identify the input trusted block using the input_block_identifier parameter, and set the input_block_identifier_length variable to the length of the key label or the key token of the input block. The input block can be any one of these forms:

- An uninitialized trusted block. The trusted block is complete except that it does not have MAC protection.

- An inactive trusted block. The trusted block is external, and it is in inactive form. MAC protection is present due to recycling of an existing inactive trusted block.

- An active trusted block. The trusted block is internal or external, and it is in active form. MAC protection is present due to recycling of an existing active trusted block.

Note: The MAC key is replaced with a new MAC key, and any RKX key-token created with the input trusted block cannot be used with the output trusted block.

This verb randomly generates a confounder and triple-length MAC key, and uses a variant of the MAC key to calculate an ISO 16609 CBC mode TDES MAC of the trusted block contents. To protect the MAC key, the verb encrypts the confounder and MAC key using a variant of an IMP-PKA key. The calculated MAC and the encrypted confounder and MAC key are embedded in the output trusted block. Use the transport_key_identifier parameter to identify the key token that contains the IMP-PKA key.
On input, set the `trusted_block_identifier_length` variable to the length of the key label or at least the size of the output trusted block. The output trusted block is returned in the key-token identified by the `trusted_block_identifier` parameter, and the verb updates the `trusted_block_identifier_length` variable to the size of the key token if a key label is not specified.

**Creating an active external trusted block:** To create an active external trusted block, use a `rule_array_count` of 1 and a `rule_array` keyword of `ACTIVE`. Identify the input trusted block using the `input_block_identifier` parameter, and set the `input_block_identifier_length` variable to the length of the key label or the key token of the input block. The input block must be an inactive external trusted block that was created using the `INACTIVE` `rule_array` keyword.

Use the `transport_key_identifier` parameter to identify the key token that contains the IMP-PKA key.

On input, set the `trusted_block_identifier_length` variable to the length of the key label or at least the size of the output trusted block. The verb returns an error if the input trusted block is not valid. Otherwise, it changes the flag in the trusted block information section from the inactive state to the active state, recalculates the MAC, and embeds the updated MAC value in the output trusted block.

The output trusted block is returned in the key-token identified by the `trusted_block_identifier` parameter, and the verb updates the `trusted_block_identifier_length` variable to the size of the key token if a key label is not specified.

**Format**

The format of CSNDTBC.

```
CSNDTBC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    input_block_identifier_length,
    input_block_identifier,
    transport_key_identifier,
    trusted_block_identifier_length,
    trusted_block_identifier)
```

**Parameters**

The parameters for CSNDTBC.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “[Parameters common to all verbs” on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1.

**rule_array**
trusted_block_identifier_length

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of bytes in the input_block_identifier variable. The maximum length is 3500 bytes.

input_block_identifier

Direction: Input
Type: String

A pointer to a string variable containing a trusted block key-token or the key label of a trusted block key-token that has been built according to the format specified in "Trusted blocks" on page 968. The trusted block key-token will be updated by the verb and returned in the trusted_block_identifier variable.

When the operation is INACTIVE, the trusted block can have MAC protection (for example, due to recycling of an existing trusted block), but typically it does not.

transport_key_identifier

Direction: Input
Type: String

A pointer to a string variable containing an operational CCA DES key-token or the key label of an operational CCA DES key-token record. The key token must be of type IMP-PKA.

An IMP-PKA key type is an IMPORTER key-encrypting key with only its IMPORT key-usage bit (bit 21) on; its other key-usage bits (IMEX, OPIM, IMIM, and XLA) must be off.

Note: An IMP-PKA control vector can be built using "Control Vector Generate (CSNBCVG)" on page 170 with a key type of IMPORTER and a rule_array keyword of IMPORT.

trusted_block_identifier_length

Direction: Input/Output
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the

Table 199. Keywords for Trusted Block Create control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>(One required)</td>
</tr>
<tr>
<td>INACTIVE</td>
<td>Create an external trusted block, based on the input_block_identifier variable, and set the active flag to B'0'. This makes the trusted block unusable in any other CCA services.</td>
</tr>
<tr>
<td>ACTIVE</td>
<td>Create an external trusted block, based on the token identified by the input_block_identifier parameter, and change the active flag from B'0' to B'1'. This makes the trusted block usable in other CCA services</td>
</tr>
</tbody>
</table>

Table 199. Keywords for Trusted Block Create control information
trusted_block_identifier variable. The maximum length is 3500 bytes. The output trusted block token can be up to seven bytes longer than the input trusted block token due to padding.

trusted_block_identifier

Direction: Output
Type: String

A pointer to a string variable containing a trusted block token or a label of a trusted block token returned by the verb.

Restrictions

The restrictions for CSNDRKX.

AES keys are not supported by this verb.

Required commands

The required commands for CSNDTBC.

The verb requires the following commands to be enabled in the active role based on the keyword specified for the operation rule:

<table>
<thead>
<tr>
<th>rule_array</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>INACTIVE</td>
<td>X'030F'</td>
<td>Trusted Block Create - Create Block in inactive form</td>
</tr>
<tr>
<td>ACTIVE</td>
<td>X'0310'</td>
<td>Trusted Block Create - Activate an inactive block</td>
</tr>
</tbody>
</table>

To prevent a weaker transport key (key-encrypting key) from being used to encipher a triple-length MAC key into an external trusted block, enable the TBC - Disallow triple-length MAC key command (offset X'032E') to be enabled in the active role.

Usage notes

The usage notes for CSNDRKX.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNDTBCJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNDTBCJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber input_block_identifier_length,
    byte[] input_block_identifier,
    byte[] transport_key_identifier,
    hikmNativeNumber trusted_block_identifier_length,
    byte[] trusted_block_identifier);
```
Use the Public Infrastructure Certificate verb to create a self-signed PKCS #10 certificate signing request (CSR) based on an existing RSA or ECC pair of private key and public key. The self-signed PKCS #10 request for the input public key is signed by the input private key.

For a PKCS #10 CSR, the input private key can either be an internal RSA or ECC token, or the label of the private key, or it can be the label of an RSA retained key token. In addition to the input private key, the user must specify extra input parameters that specify the following:
- The subject’s distinguished name.
- The key usage and constraints indicators.
- The signature algorithm and hashing-method.
- The certificate extensions.
- The output format of the CSR.

Format

The format of CSNDPIC.

```
CSNDPIC(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  subject_private_key_identifier_length,
  subject_private_key_identifier,
  subject_name_length,
  subject_name,
  extensions_length,
  extensions,
  reserved1_length,
  reserved1,
  reserved2_length,
  reserved2,
  reserved3_length,
  reserved3,
  reserved4_length,
  reserved4,
  reserved5_length,
  reserved5,
  reserved6_length,
  reserved6,
  certificate_length,
  certificate)
```

Parameters

The parameters for CSNDPIC.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

`rule_array_count`

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the
rule_array variable. This value must be in the range of 6 -14.

**rule_array**

- **Direction:** Input
- **Type:** String array

Keywords that provide control information to the verb. Each keyword is left-aligned in 8-byte fields, and padded on the right with blanks. All keywords must be in contiguous storage.

The rule_array keywords are described in Table 200.

**Table 200. Keywords for Public Infrastructure Certificate control information**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK10SNRQ</td>
<td>Specifies to create a PKCS #10 CSR request from the input private key (which always includes a public key section).</td>
</tr>
<tr>
<td>Issuer modifier</td>
<td>One required. Specifies how the issuer and issuer’s distinguished name will be determined).</td>
</tr>
<tr>
<td>SELFSIGN</td>
<td>Specifies that the CSR is for a self-signed certificate. The issuer’s distinguished name is the value that is passed in the subject_name parameter.</td>
</tr>
<tr>
<td>Input subject name format indicator</td>
<td>One required. Specifies the format of the input subject_name parameter.</td>
</tr>
<tr>
<td>SDNDER</td>
<td>Specifies that the input subject_name is ASN.1 DER encoded.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SDNCLEAR</td>
<td>Specifies that the input subject name is specified as a series of X.509 attribute-value pairs that are separated by commas. For example: cn=Thomas Watson,ou=Endicott,c=US</td>
</tr>
</tbody>
</table>

**Identifier**

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>countryName</td>
</tr>
<tr>
<td>O</td>
<td>organizationName</td>
</tr>
<tr>
<td>OU</td>
<td>organizationalUnitName</td>
</tr>
<tr>
<td>CN</td>
<td>commonName</td>
</tr>
<tr>
<td>SN</td>
<td>surname</td>
</tr>
<tr>
<td>L</td>
<td>localityName</td>
</tr>
<tr>
<td>ST</td>
<td>stateOrProvinceName</td>
</tr>
<tr>
<td>SP</td>
<td>stateOrProvinceName</td>
</tr>
<tr>
<td>S</td>
<td>stateOrProvinceName</td>
</tr>
<tr>
<td>T</td>
<td>title</td>
</tr>
<tr>
<td>PC</td>
<td>postalCode</td>
</tr>
<tr>
<td>EMAIL</td>
<td>emailAddress</td>
</tr>
<tr>
<td>E</td>
<td>emailAddress</td>
</tr>
<tr>
<td>EMAILADDRESS</td>
<td>emailAddress</td>
</tr>
<tr>
<td>STREET</td>
<td>streetAddress</td>
</tr>
<tr>
<td>DC</td>
<td>domainComponent</td>
</tr>
<tr>
<td>MAIL</td>
<td>mail</td>
</tr>
<tr>
<td>NAME</td>
<td>name</td>
</tr>
<tr>
<td>GIVENNAME</td>
<td>givenName</td>
</tr>
<tr>
<td>INITIALS</td>
<td>initials</td>
</tr>
<tr>
<td>GENERATIONQUALIFIER</td>
<td>generationQualifier</td>
</tr>
<tr>
<td>DNQUALIFIER</td>
<td>dnQualifier</td>
</tr>
<tr>
<td>SERIALNUMBER</td>
<td>serialNumber</td>
</tr>
</tbody>
</table>

To specify a comma within an attribute value, escape the comma with the back slash character (\). For example, an organization name of IBM,Poughkeepsie would be specified as: OU=IBM\,Poughkeepsie

**Output format indicator** (One required. Specifies the format of the data returned in the certificate parameter).

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DER-FMT</td>
<td>Specifies that the output in the certificate parameter object are DER encoded according to the X.509 standard.</td>
</tr>
<tr>
<td>PEM-FMT</td>
<td>Specifies that the output in the certificate parameter are encoded using Base64 encoding according to RFC 7468. The encoded stream will consist of EBCDIC printable characters with one line feed (X'25') inserted after each group of 64 encoded characters and one line feed at the end of the encoded stream.</td>
</tr>
</tbody>
</table>
Table 200. Keywords for Public Infrastructure Certificate control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key usage and constraint indicators.</strong> Specifies key usage indicators that are encoded as allowed in the Key Usage extension in the data that is returned in the output certificate parameter.</td>
<td></td>
</tr>
<tr>
<td>One to seven of these rules are required if the extensions parameter does not specify any allowed usages for Key Usage or Extended Key Usage. None of these rules are allowed if the extensions parameter is specified and does specify usages for Key Usage or Extended Key Usage. Also, U-DECONL and U-ENCONL requires U-KEYAGR to be specified. U-DECONL cannot be combined with U-ENCONL.</td>
<td></td>
</tr>
<tr>
<td>U-DIGSIG</td>
<td>Specifies that digitalSignature is allowed.</td>
</tr>
<tr>
<td>U-NONRPD</td>
<td>Specifies that nonRepudiation is allowed.</td>
</tr>
<tr>
<td>U-DATEENC</td>
<td>Specifies that keyEncipherment is allowed.</td>
</tr>
<tr>
<td>U-KEYAGR</td>
<td>Specifies that keyAgreement is allowed.</td>
</tr>
<tr>
<td>U-KCRTSN</td>
<td>Specifies that keyCertSign is allowed.</td>
</tr>
<tr>
<td>U-CRLSN</td>
<td>Specifies that cRLSign is allowed.</td>
</tr>
<tr>
<td>U-ENCONL</td>
<td>Specifies that encipherOnly is allowed.</td>
</tr>
<tr>
<td>U-DECONL</td>
<td>Specifies that decipherOnly is allowed.</td>
</tr>
<tr>
<td><strong>Signature algorithm specification</strong> (One required. Specifies the signature algorithm is to be used in creating the data returned in the certificate parameter. A hash method rule must also be specified.</td>
<td></td>
</tr>
<tr>
<td>RSA</td>
<td>Specifies to use the RSA signature algorithm. The hash method that is used must be SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.</td>
</tr>
<tr>
<td>ECDSA</td>
<td>Specifies to use the ECDSA signature algorithm. The hash method that is used must be SHA-224, SHA-256, SHA-384, or SHA-512.</td>
</tr>
<tr>
<td><strong>Hash-method specification</strong> (One required. Specifies the hashing method that is to be used in conjunction with the required signature algorithm in creating the data return in the certificate parameter.</td>
<td></td>
</tr>
<tr>
<td>SHA-1</td>
<td>The hash method to be used is SHA-1. Not allowed with the ECDSA signature algorithm.</td>
</tr>
<tr>
<td>SHA-224</td>
<td>The hash method to be used is SHA-224.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>The hash method to be used is SHA-256.</td>
</tr>
<tr>
<td>SHA-384</td>
<td>The hash method to be used is SHA-384.</td>
</tr>
<tr>
<td>SHA-512</td>
<td>The hash method to be used is SHA-512.</td>
</tr>
</tbody>
</table>

subject_private_key_identifier_length

- **Direction:** Input
- **Type:** Integer

The length of the subject_private_key_identifier parameter in bytes. The maximum value is 3500.

subject_private_key_identifier

- **Direction:** Input
- **Type:** String

Contains an internal token or label of a private RSA or ECC key or the label of an RSA retained key token. The key must be able to generate digital signatures. In addition, the key usage attributes must not conflict with the key usage keywords.

subject_name_length

- **Direction:** Input
Type: Integer

The length of the subject_name parameter in bytes. The maximum value is 400.

subject_name

Direction: Input
Type: String

The subject distinguished name (SDN) that is used for creating the output in the certificate parameter. When the SELFSIGN rule is specified, the value of subject_name is also used for the issuer distinguished name (IDN) in the output certificate parameter.

extensions_length

Direction: Input
Type: Integer

The length of the extensions parameter in bytes. The maximum value is 1000.

extensions

Direction: Input
Type: String

Specifies a DER encoded set of x.509 extensions. If any of the extensions specify key usage or extended key usage indicators, then no key usage or constrain indicator rules can be specified.

reserved1_length

Direction: Input
Type: Integer

This parameter must be zero.

reserved1

Direction: Input
Type: String

This field is not used.

reserved2_length

Direction: Input
Type: Integer

This parameter must be zero.

reserved2

Direction: Input
Type: String

This field is not used.

reserved3_length

Direction: Input
Type: Integer

This parameter must be zero.
Public Infrastructure Certificate (CSNDPIC)

reserved3
Direction: Input
Type: String
This field is not used.

reserved4
Direction: Input
Type: Integer
This parameter must be zero.

reserved5
Direction: Input
Type: Integer
This parameter must be zero.

reserved6_length
Direction: Input
Type: Integer
This parameter must be zero.

reserved6
Direction: Input
Type: String
This field is not used.

certificate_length
Direction: Input/Output
Type: Integer
On input, the length in bytes of the buffer for the certificate parameter. The maximum value is 3500.
On output, the length in bytes of the data returned in the certificate parameter.

certificate
Direction: Input/Output
Type: String
On input, specifies the buffer to be used for the output certificate. On output, contains the data that is requested by the Requested Action rule.

Restrictions
The restrictions for CSNDPIC.
None.
Public Infrastructure Certificate (CSNDPIC)

Required commands

The required commands for CSNDPIC.

This verb requires the following commands to be enabled in the active role:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
<th>Function control</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>X'0070'</td>
<td>Public Infrastructure Certificate</td>
<td>Allows the service to be used.</td>
</tr>
<tr>
<td>PK10SNRQ</td>
<td>X'007C'</td>
<td>Public Infrastructure Certificate - PK10SNRQ</td>
<td>Allows the PK10SNRQ keyword to be used with the service.</td>
</tr>
</tbody>
</table>

Usage notes

The usage notes for CSNDPIC.

None.

JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNDPICJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNDPICJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber subject_private_key_identifier_length,
    byte[] subject_private_key_identifier,
    hikmNativeNumber subject_name_length,
    byte[] subject_name,
    hikmNativeNumber extensions_length,
    byte[] extensions,
    hikmNativeNumber reserved1_length,
    byte[] reserved1,
    hikmNativeNumber reserved2_length,
    byte[] reserved2,
    hikmNativeNumber reserved3_length,
    byte[] reserved3,
    hikmNativeNumber reserved4,
    hikmNativeNumber reserved5,
    hikmNativeNumber reserved6_length,
    byte[] reserved6,
    hikmNativeNumber certificate_length,
    byte[] certificate
);
```

Public Infrastructure Manage (CSNDPIM)

Use the CSNDPIM verb to manage the public key infrastructure stored inside the adapter.
You can manage a public key infrastructure inside the HSM to enable the use of standard-compliant X.509 certificates with CCA. For such certificates, you use a so-called trust parent, which is the certificate installed from the Trusted Key Entry workstation (TKE). The trust parent is the root of the trust chain of certificates that validate an operational X.509 certificate. The trust parent could be an actual certificate authority (CA) root certificate, or it could be a sub-CA certificate. The security is the same from the perspective of the adapter because the trust parent is installed under dual control security by administrators of the adapter.

Thus, the management of trust parents requires dual-control administrative capabilities through the TKE, while management of operational certificates is similar to existing public key tokens. The CSNDPIMOK interface to the public key infrastructure (PKI) in the HSM allows some limited management and query functions that do not require administrative authentication.

You supply the following information:

- To import a trust parent on the TKE:
  - Supply a X.509 certificate that is DER or PEM encoded (binary DER), 64-byte label to initiate import.
  - Generate a hash of the X.509 certificate, a 64-byte label and a keyword to complete import.
  - A trust parent from a CCA point of view may actually be a sub-CA certificate when considered from the view of an external PKI:
    - The distinction for CCA is that a trust parent:
      - establishes a new line of trust in the HSM, much like a master wrapping key
      - is loaded under dual control: load of a trust root is protected by two access control points, one for each phase of the process.
    - Loading a sub-CA certificate as a trust parent is expected for some workflows:
      - CCA does not implement a full PKI hierarchy internally. So any operational X.509 certificate should have its trust parent, namely the certificate for the issuer, loaded to the HSM.
      - This is reasonable from a security perspective given that a sub-CA loaded as a trust parent has been loaded by two administrators, and so the same level of trust is justified.

  **Note:** As a PKI participant, CCA cannot validate the signature on a sub-CA certificate loaded as a trust parent, since the issuer certificate is not part of the internal PKI.

  - The HSM has no independent access to a network and thus:
    - cannot use NTP to maintain accurate timing for trust parent expiration
    - cannot interrogate external revocation lists.

- To list trust parents (by label):
  - Supply a keyword.
  - Supply a buffer to hold the information.
Public Infrastructure Manage (CSNDPIM)

Format

The format of CSNDPIM.

```c
CSNDPIM(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    certificate_length,
    certificate,
    label_length,
    label,
    hash_length,
    hash,
    input_data_length,
    input_data,
    output_data_length,
    output_data,
    reserved1_length,
    reserved1)
```

Parameters

The parameters for CSNDPIM.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

**rule_array_count**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Integer</td>
</tr>
</tbody>
</table>

A pointer to an integer variable containing the number of elements in the `rule_array` variable. The value must be 1, 2, 3 or 4.

**rule_array**

<table>
<thead>
<tr>
<th>Direction:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>String array</td>
</tr>
</tbody>
</table>

A pointer to a string variable containing an array of keywords. The keywords are 8 bytes in length and must be left-aligned and padded on the right with space characters. The `rule_array` keywords are described in Table 201.

Table 201. Keywords for CSNDPIM control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requested action</td>
<td>(One, required)</td>
</tr>
</tbody>
</table>
### Table 201. Keywords for CSNDPIM control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
</table>
| RNW-CERT  | Specifies to load the input certificate in place of a previously imported certificate. The new certificate must have the same public key, subject name, and subject key identifier as the old certificate to be a valid replacement. This operation is expected for expiration updates or other operations that change certificate parameters without changing the subject public key.  
  
  **Input:**  
  - **label** 64-byte label  
  - **certificate**  
    X.509 certificate, binary DER encoded, or in PEM format  
  - **rule array**  
    Hashing method keyword (optional). For available methods (keywords) see section *Hashing method* later in this table. If not passed, then no hash is returned.  
  
  **Output:**  
  - A status indication is returned.  
  - If a hashing method had been specified, on success, the hash of the certificate is returned for verification with a tool. The size of the hash is determined by the hashing method. |
| CHGLABEL  | Specifies to change the label for a previously imported certificate. This might be used after RNW-CERT to update the label in some matching updated certificate characteristics. This command does not change any security aspects for the referenced certificate or the certificates that depend on it.  
  
  **Input:**  
  - **label** current 64-byte label  
  - **input_data**  
    new 64-byte label  
  
  As output, a status indication is returned. |
| VAL-CERT  | Specifies to validate the input certificate against the public key infrastructure stored in the HSM. This is a method of testing the certificate to see if an operational request with the certificate will succeed.  
  
  **Input:**  
  - **certificate**  
    X.509 certificate, binary DER encoded or PEM format  
  
  **Output:**  
  - **status indication**  
    Either a success indication is returned or an informational error about why the certificate validation failed (for example, issuer not found). The returned status indicates only one error, while there may be more than one issues with a particular certificate.  
  - **output_data**  
    The 64-byte label is returned for the certificate in the adapter that is used to validate the input operational certificate. This returned label may be used as input with the LSTISSUR keyword to obtain the issuer of the validating certificate. It can also be used with GET-CERT to obtain the full certificate for the validating certificate. This allows the user to re-construct the validation certificate chain used for their operational certificate. This is necessary if a user needs to pass a certification chain to a partner.  
  
  **Note:** This keyword does not require a signature on a Connectivity Programming Request/Reply Block (CPRB) of type T2 (T2 CPRB). |
Table 201. Keywords for CSNDPIM control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET-CERT</td>
<td>Given an input 64-byte label for a particular certificate, this keyword returns the full X.509 certificate for that certificate and the HSM-internal state of the certificate. Optionally the HSM-computed hash of the certificate is also returned.</td>
</tr>
</tbody>
</table>

**Input:**
- **label**: 64-byte label
- **rule array**: Hashing method keyword (optional). For available methods (keywords) see section Hashing method later in this table. If not passed, then no hash is returned.

**Output:**
On success, the following output is returned. On failure, only an error indication is returned.

- **output_data**: X.509 certificate, binary DER encoded, or in PEM format, matching the input 64-byte label
  - See the SIZEDATA keyword definition for alternate output.
  - See the Output format indicator section later in this table.
- **hash**: If a hashing method had been specified, the hash of the certificate referred to by the input label is returned.

**rule array**
State of the certificate, specified as a single 8-byte rule-array keyword, to be interpreted as follows:
1. The first left-most byte X'BBxxxxxxxxxxxxx' is reserved.
2. The second left-most byte X'xxBBxxxxxxxxxxxx' indicates the activity status of a certificate. Valid values are:
   - X'xx30xxxxxxxxxxxx': The certificate is not active.
     - For a root certificate this could mean that the certificate has only been loaded, but not activated.
     - For any certificates (root or non-root) this could mean the certificate has expired (check other bytes).
   - X'xx31xxxxxxxxxxxx': The certificate is active.
     - An active installed certificate is used by the HSM to validate other certificates.
3. The third left-most byte X'xxxxBBxxxxxxxxxxxx' indicates the expiration status of a certificate. Valid values are:
   - X'xxxx30xxxxxxxxxxxx': The certificate is not expired.
   - X'xxxx31xxxxxxxxxxxx': The certificate is expired.
4. The fourth left-most byte X'xxxxxxBBxxxxxxxxxxxx' is reserved.
5. The fifth left-most byte X'xxxxxxxx8Bxxxxxxx' indicates the hash method specified when a certificate was loaded. Valid values are:
   - X'xxxxxx32xxxxxxx': The SHA-256 method was used to load the certificate.
   - X'xxxxxx34xxxxxxx': The SHA-384 method was used to load the certificate.
   - X'xxxxxx38xxxxxxx': The SHA-512 method was used to load the certificate.

**Note:** This keyword does not require a signature on a Connectivity Programming Request/Reply Block (CPRB) of type T2 (T2 CPRB).
Table 201. Keywords for CSNDPIM control information  (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTROOTS</td>
<td>Returns a list of 64-byte labels for installed trust parent certificates.</td>
</tr>
<tr>
<td></td>
<td>There is no input.</td>
</tr>
<tr>
<td></td>
<td>Output:</td>
</tr>
<tr>
<td></td>
<td><strong>status indication</strong></td>
</tr>
<tr>
<td></td>
<td>A status indication is returned.</td>
</tr>
<tr>
<td></td>
<td><strong>output_data</strong></td>
</tr>
<tr>
<td></td>
<td>On success, an array of 64 byte labels representing all the loaded root certificates is returned.</td>
</tr>
<tr>
<td></td>
<td>The returned buffer size indicates the count of certificates (as a multiple of 64).</td>
</tr>
<tr>
<td></td>
<td>• See the SIZEDATA keyword definition for alternate output.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This keyword does not require a signature on a Connectivity Programming Request/Reply Block (CPRB) of type T2 (T2 CPRB).</td>
</tr>
</tbody>
</table>

**Action modifier** (One, optional)

| SIZEDATA  | Specifies to return only the size of the data returned by the requested action. Used with VAL-CERT, GET-CERT, and LSTROOTS keywords. |
|           | The size returned is placed in the **output_data_length** parameter. It refers to the data that would have been placed in the **output_data** parameter if the SIZEDATA keyword was not passed. |

**Certificate validation method** (One, required for VAL-CERT)

| RFC-2459  | Attempt to validate the certificate using the semantics of RFC-2459.         |
| RFC-3280  | Attempt to validate the certificate using the semantics of RFC-3280.         |
| RFC-5280  | Attempt to validate the certificate using the semantics of RFC-5280.         |
| RFC-ANY   | Attempt to validate the certificate using first the semantics of RFC-2459, then RFC-3280, and then RFC-5280. If the certificate is not compliant with any RFC, the first error encountered (from RFC-2459 processing) is returned. |

**Hashing method** (One, optional with RNW-CERT and GET-CERT)

| SHA-256   | The hash variable (either input or output, depending on the action) is calculated with the SHA-256 hash method. Size of the variable is 32 bytes. |
| SHA-384   | The hash variable (either input or output, depending on the action) is calculated with the SHA-384 hash method. Size of the variable is 48 bytes. |
| SHA-512   | The hash variable (either input or output, depending on the action) is calculated with the SHA-512 hash method. Size of the variable is 64 bytes. |

**Output format indicator** (one, required with GET-CERT and not allowed otherwise) (One, optional with RNW-CERT and GET-CERT)

| DER-FMT   | Specifies that the output data object should be DER encoded.                |
| PEM-FMT   | Specifies that the output data object should be PEM (Base64) encoded.       |

**Certificate_length**

- **Direction:** Input  
- **Type:** Integer  

Pointer to an integer variable containing the number of bytes of data in the certificate variable. The maximum length is 3500.

**Certificate**

- **Direction:** Input  
- **Type:** String
A pointer to a string variable containing an X.509 certificate. The data must be DER or PEM encoded.

**label_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the label variable. This value must be 64 or 0 if no label processing is expected.

**label**

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing a label for the input certificate specified with the `certificate` parameter. The label is 64-byte ASCII data buffer, with the following format characteristics:
  - If the label is less than 64 characters, it must be left-justified in the 64 character buffer, and it must be padded on the right with spaces.
  - The first character must be either alphanumeric, or one member from the following set: @, $, or #.
  - All other characters must be either alphanumeric, a period, or one of the set: @, $, or #.
  - The input string is not null-terminated. It is a fixed length string (64 characters) with no special terminators.

**hash_length**

- **Direction:** Input/Output
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the hash variable. This value depends on the specified hashing method keyword. It must be 32 for SHA-256, 48 for SHA-384, and 64 for SHA-512.

**hash**

- **Direction:** Input/Output
- **Type:** String

A pointer to a string variable containing a hash of the referenced X.509 certificate. The hash is either input or output, depending on the requested action keyword. The format of the hash data is raw binary data of the size indicated by the hashing method keyword.

**input_data_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the `input_data` variable. The maximum length depends on the usage.

**input_data**

- **Direction:** Input
- **Type:** String

A pointer to a string variable with contents that depend on the rule array keyword for the requested action:
Parameter input_data shall contain the new 64-byte label for the internally stored certificate.

all others
Parameter input_data_length should be zero and this parameter should be NULL.

output_data_length
Direction: Input/Output
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the output_data variable. On input, it must indicate the size of the buffer available. On output, if the size is sufficient, the variable contains the actual length of the data returned by the service. If the SIZEDATA keyword is passed, the size returned refers to the amount of data that is returned if the SIZEDATA keyword is not passed.

output_data
Direction: Output
Type: String

A pointer to a string variable containing the output data from the service, with contents that depends on the rule array keyword for the requested action:

LISTROOTS
Parameter output_data contains the array of 64 byte labels returned.

GET-CERT
Parameter output_data contains the returned X.509 certificate corresponding to the input label.

VAL-CERT
Parameter output_data contains the returned label from the validating certificate for the input certificate.

all others
Parameter output_data_length is zero and this parameter is NULL.

reserved1_length
Direction: Input/Output
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the reserved1 variable. This parameter must be a null pointer or point to a value of 0.

reserved1
Direction: Output
Type: String

A pointer to a string variable reserved for this verb.

Restrictions
The restrictions for CSNDPIM.

- PKCS#7 certificate containers are currently not supported by this service.
However it is possible to install all of the certificates from a PKCS#7 container, given that they are pulled from the container and loaded to the HSM as sub-CA certificates one at a time, in dependency order. The trust anchor for the chain should be loaded first using the platform-appropriate administration tool, as required. This is still done as a ‘load-root’ from a TKE command perspective, however the adapter PKI will recognize the parent relationship and automatically chain them.

- An end-entity may not be loaded as a sub-CA certificate.
- A key identifier length must be 64 for a key label.
- The following signature algorithms are supported:

<table>
<thead>
<tr>
<th>Signature Algorithm</th>
<th>OID</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>sha1-with-rsa-signature</td>
<td>1.2.840.113549.1.1.5</td>
<td><a href="http://oid-info.com/get/1.2.840.113549.1.1.5">http://oid-info.com/get/1.2.840.113549.1.1.5</a></td>
</tr>
<tr>
<td>sha224WithRSAEncryption</td>
<td>1.2.840.113549.1.1.14</td>
<td><a href="http://oid-info.com/get/1.2.840.113549.1.1.14">http://oid-info.com/get/1.2.840.113549.1.1.14</a></td>
</tr>
<tr>
<td>sha256WithRSAEncryption</td>
<td>1.2.840.113549.1.1.11</td>
<td><a href="http://oid-info.com/get/1.2.840.113549.1.1.11">http://oid-info.com/get/1.2.840.113549.1.1.11</a></td>
</tr>
<tr>
<td>sha384WithRSAEncryption</td>
<td>1.2.840.113549.1.1.12</td>
<td><a href="http://oid-info.com/get/1.2.840.113549.1.1.12">http://oid-info.com/get/1.2.840.113549.1.1.12</a></td>
</tr>
<tr>
<td>sha512WithRSAEncryption</td>
<td>1.2.840.113549.1.1.13</td>
<td><a href="http://oid-info.com/get/1.2.840.113549.1.1.13">http://oid-info.com/get/1.2.840.113549.1.1.13</a></td>
</tr>
<tr>
<td>ecdsa-with-SHA1</td>
<td>1.2.840.10045.4.1</td>
<td><a href="http://oid-info.com/get/1.2.840.10045.4.1">http://oid-info.com/get/1.2.840.10045.4.1</a></td>
</tr>
<tr>
<td>ecdsa-with-SHA224</td>
<td>1.2.840.10045.4.3.1</td>
<td><a href="http://oid-info.com/get/1.2.840.10045.4.3.1">http://oid-info.com/get/1.2.840.10045.4.3.1</a></td>
</tr>
<tr>
<td>ecdsa-with-SHA256</td>
<td>1.2.840.10045.4.3.2</td>
<td><a href="http://oid-info.com/get/1.2.840.10045.4.3.2">http://oid-info.com/get/1.2.840.10045.4.3.2</a></td>
</tr>
<tr>
<td>ecdsa-with-SHA384</td>
<td>1.2.840.10045.4.3.3</td>
<td><a href="http://oid-info.com/get/1.2.840.10045.4.3.3">http://oid-info.com/get/1.2.840.10045.4.3.3</a></td>
</tr>
<tr>
<td>ecdsa-with-SHA512</td>
<td>1.2.840.10045.4.3.4</td>
<td><a href="http://oid-info.com/get/1.2.840.10045.4.3.4">http://oid-info.com/get/1.2.840.10045.4.3.4</a></td>
</tr>
</tbody>
</table>

- Concerning the signature algorithm, it is recommended to use a hash method of commensurate security to the curve of the key being wrapped. The IETF [RFC5753 document](http://oid-info.com/get/1.2.840.10045.4.3.1) provides the following information:
  - Implementations that support SignedData with ECDSA:
    - must support ECDSA with SHA-256
    - may support ECDSA with SHA-1, ECDSA with SHA-224, ECDSA with SHA-384, and ECDSA with SHA-512. Other digital signature algorithms may also be supported.
  - When using ECDSA, to promote interoperability it is recommended that
    - the P-192, P-224, and P-256 curves are to be used with SHA-256
    - the P-384 curve is to be used with SHA-384
    - the P-521 curve is to be used with SHA-512.

**Required commands**

The required commands for CSNDPIM.

This verb requires the following commands to be enabled in the active role:

<table>
<thead>
<tr>
<th>Rule-array keyword or verb</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command required for complete verb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNDPIM</td>
<td>0x01B1</td>
<td>Public Infrastructure Management</td>
</tr>
<tr>
<td>Command required for requested action</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RNW-CERT</td>
<td>0x01B4</td>
<td>PIM: Renew Certificate</td>
</tr>
</tbody>
</table>
Usage notes
The usage notes for CSNDPIM.
None.

JNI version
This verb has a Java Native Interface (JNI) version, which is named CSNDPIMJ.

See “Building Java applications using the CCA JNI” on page 28.

Format
public native void CSNDPIMJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber certificate_length,
    byte[] certificate,
    hikmNativeNumber label_length,
    byte[] label,
    hikmNativeNumber hash_length,
    byte[] hash,
    hikmNativeNumber input_data_length,
    byte[] input_data,
    hikmNativeNumber output_data_length,
    byte[] output_data,
    hikmNativeNumber reserved1_length,
    byte[] reserved1
);
Public Infrastructure Manage (CSNDPIM)
Chapter 16. TR-31 symmetric key management verbs

These verbs support TR-31 symmetric key management.

- “Key Export to TR31 (CSNBT31X)”
- “TR31 Key Import (CSNBT31I)” on page 781
- “TR31 Key Token Parse (CSNBT31P)” on page 804
- “TR31 Optional Data Build (CSNBT31O)” on page 808
- “TR31 Optional Data Read (CSNBT31R)” on page 811

Key Export to TR31 (CSNBT31X)

Use the Key Export to TR31 verb to convert a proprietary CCA external or internal symmetric key-token and its attributes into a non-proprietary key block that is formatted under the rules of TR-31.

After being exported into a TR-31 key block, the key and its attributes are ready to be interchanged with any outside third party who uses TR-31. The verb takes as input either an external or internal fixed-length DES key-token that contains a DES or Triple-DES (TDES) key, along with an internal DES EXPORTER or OKEYXLAT key-encrypting key used to wrap the external TR-31 key block.

The Key Export to TR31 verb is analogous to the Key Export verb, except that Key Export to TR31 accepts an external or internal fixed-length DES key-token as input, instead of only an internal fixed-length DES key-token, and it translates the key to an external non-CCA format instead of an external fixed-length DES key-token. The purpose of both verbs is to export a DES key to another party.

An external-to-external translation would not normally be called an export or import operation. Instead, it would be called a key translation, and would be handled by a verb such as Key Translate2. For practical reasons, the export of an external CCA DES key-token to external TR-31 format is supported by the Key Export to TR31 verb, and the import of an external TR-31 key block to an external CCA DES key-token is supported by the TR31 Key Import verb.

Note that the Key Export to TR31 verb does not support the translation of an external key from encipherment under one key-encrypting key to encipherment under a different key-encrypting key. When converting an external DES key to an external TR-31 format, the key-encrypting key used to wrap the external source key must be the same as the one used to wrap the TR-31 key block. If a translation of an external DES key from encipherment under one key-encrypting to a different key-encrypting key is desired, use the Key Translate or Key Translate2 verbs.

Both CCA and TR-31 define key attributes that control key usage. In both cases, the usage information is securely bound to the key so that the attributes cannot be changed in unintended or uncontrolled ways. CCA maintains its DES key attributes in a control vector (CV), while a TR-31 key block uses fields: key usage, algorithm, mode of use, and exportability.

Each attribute in a CCA control vector falls under one of these categories:
1. There is a one-to-one correspondence between the CV attribute and the TR-31 attribute. For these attributes, conversion is straightforward.
2. There is not a one-to-one correspondence between the CV attribute and the TR-31 attribute, but the attribute can be automatically translated when performing this export operation.

3. There is not a one-to-one correspondence between the CV attribute and the TR-31 attribute, in which case a rule-array keyword is defined to specify which attribute is used in the TR-31 key block.

4. Category (1), (2), or (3) applies, but there are some attributes that are lost completely on translation (for example, key-generating bits in key-encrypting keys).

5. None of the above categories applies, because the key type, its attributes, or both simply cannot be reasonably translated into a TR-31 key block.

The control vector is always checked for compatibility with the TR-31 attributes. It is an error if the specified TR-31 attributes are in any way incompatible with the control vector of the input key. In addition, access control points are defined that can be used to restrict the permitted attribute conversions.

The TR-31 key block has a header that can contain optional blocks. Optional blocks become securely bound to the key by virtue of the MAC on the TR-31 key block. The opt_blocks parameter is provided to allow a complete and properly formatted optional block structure to be included as part of the TR-31 key block that is returned by the verb. The TR31 Optional Data Build (CSNBT31O) verb can be used to construct an optional block structure, one optional block at a time.

An optional block has a 2-byte ASCII block ID value that determines the use of the block. The use of a particular optional block is either defined by TR-31, or it has a proprietary use. An optional block that has a block ID with a numeric value is a proprietary block. IBM has its own proprietary optional block to contain a CCA control vector. See "TR-31 optional block data" on page 967 for a description of the IBM-defined data.

To include a copy of the control vector from the DES source key in an optional block of the TR-31 key block, specify the ATTR-CV or INCL-CV control vector transport control keyword in the rule array. If either optional keyword is specified, the verb copies the single-length or double-length control vector field from the source key into the optional data field of the TR-31 header. The TR31 Key Import verb can later extract this data and use it as the control vector for the CCA key that it creates when importing the TR-31 key block. This method provides a way to use TR-31 for transport of CCA keys and to make the CCA key have identical control vectors on the sending and receiving nodes.

The ATTR-CV and INCL-CV keywords both cause the control vector to be included in a TR-31 optional block, but each has a different purpose:

**ATTR-CV**

Causes a copy of the control vector to be included, but both the TR-31 usage and mode of use fields in the non-optional part of the TR-31 key block header are set to IBM proprietary values. These values, described in "TR-31 optional block data" on page 967, indicate that the usage and mode information are specified in the control vector of the optional block and not in the TR-31 header. The restrictions imposed by the setting of the relevant access control points are bypassed, and any CCA key can be exported as long as the export control fields in the control vector allow it.

**INCL-CV**

Causes a copy of the control vector to be included as additional detail. The
resulting attributes set in the non-optional part of the TR-31 key block header are identical to not using either keyword, except that the value for the number of optional blocks is increased by one. The export operation is still subject to the restrictions imposed by the settings of the relevant access control points.

**Note:** This verb supports PCI-HSM 2016 compliant-tagged key tokens.

With PCI-HSM 2016, CCA introduces compliance-tagged key support for domains that enter compliance mode. Compliance-tagged TDES keys may be converted to or from TR-31 key blocks. The wrapping key encrypting key (KEK) must be compliance-tagged as well as the key to be exported or imported. A compliance-tagged KEK cannot be used to import or export a key that is not compliance-tagged.

### Format

The format of CSNBT31X.

```plaintext
CSNBT31X(
   return_code,
   reason_code,
   exit_data_length,
   exit_data,
   rule_array_count,
   rule_array,
   key_version_number,
   key_field_length,
   source_key_identifier_length,
   source_key_identifier,
   unwrap_kek_identifier_length,
   unwrap_kek_identifier,
   wrap_kek_identifier_length,
   wrap_kek_identifier,
   opt_blocks_length,
   opt_blocks,
   tr31_key_block_length,
   tr31_key_block)
```

### Parameters

The parameters for CSNBT31X.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. The value must be 2, 3, 4 or 5.

**rule_array**

- **Direction:** Input
- **Type:** String array

A pointer to a string variable containing an array of keywords. The keywords are 8 bytes in length and must be left-aligned and padded on the right with
Key Export to TR31 (CSNB31X)

space characters. The rule_array keywords for this verb are shown in Table 203.

Table 203. Keywords for Key Export to TR31 control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key block protection method</strong> (one required). Specifies which version of the TR-31 key block to use for exporting the CCA DES key. The version defines the method by which the key block is cryptographically protected and the content and layout of the block.</td>
<td></td>
</tr>
<tr>
<td>VARXOR-A</td>
<td>Specifies to use the Key Variant Binding Method 2005 Edition. Corresponds to TR-31 Key Block Version ID of &quot;A&quot; (X'41').</td>
</tr>
<tr>
<td>VARDRV-B</td>
<td>Specifies to use the Key Derivation Binding Method 2010 Edition. Corresponds to TR-31 Key Block Version ID of &quot;B&quot; (X'42').</td>
</tr>
<tr>
<td>VARXOR-C</td>
<td>Specifies to use the Key Variant Binding Method 2010 Edition. Corresponds to TR-31 Key Block Version ID of &quot;C&quot; (X'43').</td>
</tr>
<tr>
<td><strong>Control vector transport control</strong> (one, optional). If no keyword from this group is provided, or keyword INCL-CV is specified, the control vector in the CCA key token identified by the source_key_identifier parameter is verified to agree with the TR-31 key usage and mode of key use keywords specified from the groups below.</td>
<td></td>
</tr>
<tr>
<td>INCL-CV</td>
<td>Specifies to copy the control vector from the CCA key-token into an optional proprietary block that is included in the TR-31 key block header. See Table 279 on page 968. The TR-31 key usage and mode of use fields indicate the key attributes. Those attributes, as derived from the keywords specified, must be compatible with the ones in the included CV. In addition, the export of the key must meet the translation and ACP authorizations indicated in the export translation table for the specified keywords. A key usage keyword and a mode of use keyword are required when this keyword is specified.</td>
</tr>
<tr>
<td>ATTR-CV</td>
<td>Same as keyword INCL-CV, except that the key usage field of the TR-31 key block (byte number 5 - 6) is set to the proprietary value &quot;10&quot; (X'3130'), and the mode of use field (byte number 8) is set to the proprietary value &quot;1&quot; (X'31'). These proprietary values indicate that the key usage and mode of use attributes are specified by the CV in the optional block. For this option, only the general ACPs related to export are checked, not the ones relating to specific CCA to TR-31 translations. No key usage or mode of use keywords are allowed when this keyword is specified.</td>
</tr>
</tbody>
</table>

**TR-31 key usage value for output key** (one required). Not valid if ATTR-CV keyword is specified. Only those TR-31 modes of key use shown are supported.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>TR-31 modes of key use</th>
<th>CCA usage keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDK</td>
<td>&quot;B0&quot;</td>
<td>KEYGENKY</td>
<td>Specifies to export to a TR-31 base derivation key (BDK). You must select one mode of use keyword from Table 204 on page 762 with this usage keyword. The table shows all of the supported translations for key usage keyword BDK. It also shows the access control commands that must be enabled in the active role in order to use the combination of inputs shown.</td>
</tr>
<tr>
<td>CVK</td>
<td>&quot;C0&quot;</td>
<td>MAC or DATA</td>
<td>Specifies to export to a TR-31 CVK card verification key. You must select one mode of use keyword from Table 205 on page 764 with this usage keyword. The table shows all of the supported translations for key usage keyword CVK. It also shows the access control commands that must be enabled in the active role in order to use the combination of inputs shown.</td>
</tr>
</tbody>
</table>
### Table 203. Keywords for Key Export to TR31 control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENC &quot;D0&quot; ENCIPHER, DECIPHER, CIPHER, or DATA</td>
<td>Specifies to export to a TR-31 data encryption key. You must select one mode of use keyword from Table 206 on page 765 with this usage keyword. The table shows all of the supported translations for key usage keyword ENC. It also shows the access control commands that must be enabled in the active role in order to use the combination of inputs shown.</td>
<td></td>
</tr>
<tr>
<td>KEK &quot;K0&quot; EXPORTER or OKEYXLAT</td>
<td>Specifies to export to a TR-31 key-encryption or wrapping key. You must select one mode of use keyword from Table 207 on page 766 with this usage keyword. The table shows all of the supported translations for key usage keyword KEK. It also shows the access control commands that must be enabled in the active role in order to use the combination of inputs shown.</td>
<td></td>
</tr>
<tr>
<td>KEK-WRAP &quot;K1&quot; IMPORTER or IKEEXLAT</td>
<td>Specifies to export to a TR-31 key block protection key. You must select one mode of use keyword from Table 207 on page 766 with this usage keyword. The table shows all of the supported translations for key usage keyword KEK-WRAP. It also shows the access control commands that must be enabled in the active role in order to use the combination of inputs shown.</td>
<td></td>
</tr>
<tr>
<td>ISOMAC0 &quot;M0&quot; MAC, MACVER, DATA, DATAM, or DATAMV</td>
<td>Specifies to export to a TR-31 ISO 16609 MAC algorithm 1 (using TDEA) key. You must select one mode of use keyword from Table 208 on page 767 with this usage keyword. The table shows all of the supported translations for key usage keyword ISOMAC0. It also shows the access control commands that must be enabled in the active role in order to use the combination of inputs shown.</td>
<td></td>
</tr>
<tr>
<td>ISOMAC1 &quot;M1&quot; MAC, MACVER, DATA, DATAM, or DATAMV</td>
<td>Specifies to export to a TR-31 ISO 9797-1 MAC algorithm 1 key. You must select one mode of use keyword from Table 208 on page 767 with this usage keyword. The table shows all of the supported translations for key usage keyword ISOMAC1. It also shows the access control commands that must be enabled in the active role in order to use the combination of inputs shown.</td>
<td></td>
</tr>
<tr>
<td>ISOMAC3 &quot;M3&quot; MAC, MACVER, DATA, DATAM, or DATAMV</td>
<td>Specifies to export to a TR-31 ISO 9797-1 MAC algorithm 3 key. You must select one mode of use keyword from Table 208 on page 767 with this usage keyword. The table shows all of the supported translations for key usage keyword ISOMAC3. It also shows the access control commands that must be enabled in the active role in order to use the combination of inputs shown.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 203. Keywords for Key Export to TR31 control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PINENC</td>
<td>&quot;P0&quot; OPINENC or IPINENC</td>
<td>Specifies to export to a TR-31 PIN encryption key. You must select one mode of use keyword from Table 210 on page 770 with this usage keyword. The table shows all of the supported translations for key usage keyword PINENC. It also shows the access control commands that must be enabled in the active role in order to use the combination of inputs shown.</td>
</tr>
<tr>
<td>PINVO</td>
<td>&quot;V0&quot; PINGEN or PINVER</td>
<td>Specifies to export to a TR-31 PIN verification key or other algorithm. You must select one mode of use keyword from Table 210 on page 770 with this usage keyword. The table shows all of the supported translations for key usage keyword PINVO. It also shows the access control commands that must be enabled in the active role in order to use the combination of inputs shown.</td>
</tr>
<tr>
<td>PINV3624</td>
<td>&quot;V1&quot; PINGEN or PINVER</td>
<td>Specifies to export to a TR-31 PIN verification, IBM 3624 key. You must select one mode of use keyword from Table 210 on page 770 with this usage keyword. The table shows all of the supported translations for key usage keyword PINV3624. It also shows the access control commands that must be enabled in the active role in order to use the combination of inputs shown.</td>
</tr>
<tr>
<td>VISAPVV</td>
<td>&quot;V2&quot; PINGEN or PINVER</td>
<td>Specifies to export to a TR-31 PIN verification, VISA PVV key. You must select one mode of use keyword from Table 210 on page 770 with this usage keyword. The table shows all of the supported translations for key usage keyword VISAPVV. It also shows the access control commands that must be enabled in the active role in order to use the combination of inputs shown.</td>
</tr>
<tr>
<td>EMVACMK</td>
<td>&quot;E0&quot; DKEYGENKY</td>
<td>Specifies to export to a TR-31 EMV/chip issuer master key: application cryptograms key. You must select one mode of use keyword from Table 211 on page 773 with this usage keyword. The table shows all of the supported translations for key usage keyword EMVACMK. It also shows the access control commands that must be enabled in the active role in order to use the combination of inputs shown.</td>
</tr>
<tr>
<td>EMVSCMK</td>
<td>&quot;E1&quot; DKEYGENKY</td>
<td>Specifies to export to a TR-31 EMV/chip issuer master key: secure messaging for confidentiality key. You must select one mode of use keyword from Table 211 on page 773 with this usage keyword. The table shows all of the supported translations for key usage keyword EMVSCMK. It also shows the access control commands that must be enabled in the active role in order to use the combination of inputs shown.</td>
</tr>
</tbody>
</table>
### Table 203. Keywords for Key Export to TR31 control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>TR-31 mode</th>
<th>Usage keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMVSIMK</td>
<td>&quot;E2&quot;</td>
<td>DKYGENKY</td>
<td>Specifies to export to a TR-31 EMV/chip issuer master key: secure messaging for integrity key. You must select one mode of use keyword from Table 211 on page 773 with this usage keyword. The table shows all of the supported translations for key usage keyword EMVSIMK. It also shows the access control commands that must be enabled in the active role in order to use the combination of inputs shown.</td>
</tr>
<tr>
<td>EMVDAMK</td>
<td>&quot;E3&quot;</td>
<td>DATA, MAC, CIPHER, or ENCIPHER</td>
<td>Specifies to export to a TR-31 EMV/chip issuer master key: data authentication code key. You must select one mode of use keyword from Table 211 on page 773 with this usage keyword. The table shows all of the supported translations for key usage keyword EMVDAMK. It also shows the access control commands that must be enabled in the active role in order to use the combination of inputs shown.</td>
</tr>
<tr>
<td>EMVDNMK</td>
<td>&quot;E4&quot;</td>
<td>DKYGENKY</td>
<td>Specifies to export to a TR-31 EMV/chip issuer master key: dynamic numbers key. You must select one mode of use keyword from Table 211 on page 773 with this usage keyword. The table shows all of the supported translations for key usage keyword EMVDNMK. It also shows the access control commands that must be enabled in the active role in order to use the combination of inputs shown.</td>
</tr>
<tr>
<td>EMVCPMK</td>
<td>&quot;E5&quot;</td>
<td>DKYGENKY</td>
<td>Specifies to export to a TR-31 EMV/chip issuer master key: card personalization key. You must select one mode of use keyword from Table 211 on page 773 with this usage keyword. The table shows all of the supported translations for key usage keyword EMVCPMK. It also shows the access control commands that must be enabled in the active role in order to use the combination of inputs shown.</td>
</tr>
</tbody>
</table>

**TR-31 mode of key use** (one required). Not valid if ATTR-CV keyword is specified. Only those TR-31 modes shown are supported.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>TR-31 mode</th>
<th>Usage keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCDEC</td>
<td>&quot;B&quot;</td>
<td>ENC, KEK, KEK-WRAP</td>
<td>Specifies both encrypt and decrypt, wrap and unwrap.</td>
</tr>
<tr>
<td>DEC-ONLY</td>
<td>&quot;D&quot;</td>
<td>ENC, KEK, KEK-WRAP, PINENC</td>
<td>Specifies to decrypt and unwrap only.</td>
</tr>
<tr>
<td>ENC-ONLY</td>
<td>&quot;E&quot;</td>
<td>ENC, PINENC</td>
<td>Specifies to encrypt and wrap only.</td>
</tr>
<tr>
<td>GENVER</td>
<td>&quot;C&quot;</td>
<td>CVK, ISOMAC0, ISOMAC1, ISOMAC3, PINVO, PINV3624, VISAPVV</td>
<td>Specifies to both generate and verify.</td>
</tr>
<tr>
<td>GEN-ONLY</td>
<td>&quot;G&quot;</td>
<td>CVK, ISOMAC0, ISOMAC1, ISOMAC3, PINVO, PINV3624, VISAPVV</td>
<td>Specifies to generate only.</td>
</tr>
</tbody>
</table>
### Table 203. Keywords for Key Export to TR31 control information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>VER-ONLY</td>
<td>&quot;V&quot; CVK, ISOMAC0, ISOMAC1, ISOMAC3, PINVO, PINV3624, VISAPVV</td>
</tr>
<tr>
<td></td>
<td>Specifies to verify only.</td>
</tr>
<tr>
<td>DERIVE</td>
<td>&quot;X&quot; BDK, EMVACMK, EMVSCMK, EMVSIMK, EMVDAMK, EMVDNMK, EMVCPMK</td>
</tr>
<tr>
<td></td>
<td>Specifies that key is used to derive other keys.</td>
</tr>
<tr>
<td>ANY</td>
<td>&quot;N&quot; BDK, PINVO, PINV3624, VISAPVV, EMVACMK, EMVSCMK, EMVSIMK, EMVDAMK,</td>
</tr>
<tr>
<td></td>
<td>EMVDNMK, EMVCPMK</td>
</tr>
<tr>
<td></td>
<td>Specifies no special restrictions (other than restrictions implied by</td>
</tr>
<tr>
<td></td>
<td>the key usage).</td>
</tr>
</tbody>
</table>

**TR-31 exportability** (one, optional). Use to set exportability field in TR-31 key block. Defines whether the key may be transferred outside the cryptographic domain in which the key is found.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>TR-31 byte</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXP-ANY</td>
<td>&quot;E&quot;</td>
<td>Specifies that the key in the TR-31 key block is exportable under a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>key-encrypting key in a form that meets the requirements of X9.24 Parts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 or 2. This is the default. Note: A TR-31 key block with a key block</td>
</tr>
<tr>
<td></td>
<td></td>
<td>version ID of &quot;B&quot; or &quot;C&quot; and an exportability field value of &quot;E&quot; cannot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>be wrapped by a key-encrypting key that is wrapped in ECB mode (legacy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>wrap mode). This limitation is because ECB mode does not comply with</td>
</tr>
<tr>
<td>EXP-TRST</td>
<td>&quot;S&quot;</td>
<td>Specifies that the key in the TR-31 key block is sensitive, exportable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>under a key-encrypting key in a form not necessarily meeting the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>requirements of X9.24 parts 1 or 2.</td>
</tr>
<tr>
<td>EXP-NONE</td>
<td>&quot;N&quot;</td>
<td>Specifies that the key in the TR-31 key block is non-exportable.</td>
</tr>
</tbody>
</table>

**Note:**
1. These keys are the base keys from which derived unique key per transaction (DUKPT) initial keys are derived for individual devices such as PIN pads.
2. This table defines the only supported translations for this TR-31 usage. Usage must be the following value:
   "B0" BDK base derivation key.
3. KEYGENKY keys are double length only.

### Table 204. Export translation table for a TR-31 BDK base derivation key (BDK)

<table>
<thead>
<tr>
<th>Key usage keyword</th>
<th>Key block protection method keyword</th>
<th>CCA key type and required control vector attributes</th>
<th>Mode of use keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDK (&quot;B0&quot;)</td>
<td>VARXOR-A</td>
<td>KEYGENKY, double length, UKPT (CV bit 18 = B'1)</td>
<td>ANY (&quot;N&quot;)</td>
<td>X'0180'</td>
<td>TR31 Export - Permit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KEYGENKY:UKPT to B0</td>
</tr>
<tr>
<td>BDK (&quot;B0&quot;)</td>
<td>VARDRV-B, VARXOR-C</td>
<td>KEYGENKY, double length, UKPT (CV bit 18 = B'1)</td>
<td>DERIVE (&quot;X&quot;)</td>
<td>X'0180'</td>
<td>TR31 Export - Permit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KEYGENKY:UKPT to B0</td>
</tr>
</tbody>
</table>

**Security considerations:**
1. There is asymmetry in the translation from a CCA DATA key to a TR-31 key. The asymmetry results from CCA DATA keys having attributes of both data encryption keys and MAC keys, while TR-31 separates data encryption keys from MAC keys. A CCA DATA key can be exported to a TR-31 "C0" key, if one or both applicable MAC generate and MAC verify control vector bits are on. However, a TR-31 "C0" key cannot be imported to the lower-security CCA DATA key, it can be imported only to a CCA key type of MAC or MACVER. This restriction eliminates the ability to export a CCA MAC or MACVER key to a TR-31 key and re-importing it back as a CCA DATA key with the capability to Encipher, Decipher, or both.

2. Since the translation from TR-31 usage "C0" is controlled by rule array keywords when using the CSNBT31I verb, it is possible to convert an exported CCA CVVKEY-A or CVVKEY-B key into an AMEX-CSC key or the other way around. This conversion can be restricted by not enabling offsets X'015A' (TR31 Import - Permit C0 to MAC/MACVER:CVVKEY-A) and X'015B' (TR31 Import - Permit C0 to MAC/MACVER:AMEXSC) at the same time. However, if both CVVKEY-x and AMEX-CSC translation types are required, then offsets X'015A' and X'015B' must be enabled. In this case, control is up to the development, deployment, and execution of the applications themselves.

Note:
1. Card verification keys are used for computing or verifying (against supplied value) a card verification code with the CVV, CVC, CVC2, and CVV2 algorithms. In CCA, these keys correspond to keys used with two algorithms:
   - Visa CVV and MasterCard CVC codes are generated and verified using the CVV Generate and CVV Verify verbs. These verbs require a key type of DATA or MAC/MACVER with a subtype extension (CV bits 0 - 3) of ANY-MAC, single-length CVVKEY-A and single-length CVVKEY-B, a double-length CVVKEY-A (see CVV Key Combine verb). The MAC generate and the MAC verify (CV bits 20 - 21) key usage values must be set appropriately.
   - American Express CSC codes are generated and verified using the Transaction Validation verb. This verb requires a key type of MAC or MACVER with a subtype extension of ANY-MAC or AMEX-CSC.
2. This table defines the only supported translations for this TR-31 usage. Usage must be the following value:
   "C0"  CVK card verification key.
3. CCA and TR-31 represent CVV keys differently. These differences make representations between CCA and TR-31 incompatible. CCA represents the key-A and key-B keys as two 8-byte (single length) keys, while TR-31 represents these keys as one 16-byte (double length) key. Visa standards now require one 16-byte key. The CVV Generate and CVV Verify verbs have support added to accept one 16-byte CVV key, using left and right key parts as key-A and key-B. See "CVV Key Combine (CSNBCKC)" on page 526. This new verb provides a way to combine two single-length MAC-capable keys into one double-length CVV key.
4. Import and export of 8-byte CVVKEY-A and CVVKEY-B MAC/MACVER keys is allowed only using the IBM proprietary TR-31 usage and mode values ("10" and "1", respectively) to indicate encapsulation of the IBM control vector in an optional block, since the 8-byte CVVKEY-A is meaningless and useless as a TR-31 "C0" usage key of any mode.
### Table 205. Export translation table for a TR-31 CVK card verification key (CVK)

<table>
<thead>
<tr>
<th>Key usage keyword</th>
<th>Key block protection method keyword</th>
<th>CCA key type and required control vector attributes</th>
<th>Mode of use keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVK (&quot;C0&quot;)</td>
<td>VARXOR-A, VARDRV-B, VARXOR-C</td>
<td>MAC, single or double length, AMEX-CSC (CV bits 0 - 3 = B'0100')</td>
<td>GEN-ONLY (&quot;G&quot;)</td>
<td>X'0181'</td>
<td>TR31 Export - Permit MAC/MACVER:AMEX-CSC to C0:G/C/V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAC generate on, MAC verify off (CV bits 20 - 21 = B'010')</td>
<td>VER-ONLY (&quot;V&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAC generate on, MAC verify on (CV bits 20 - 21 = B'01')</td>
<td>GENVER (&quot;C&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAC, double length, CVVKEY-A (CV bits 0 - 3 = B'0010')</td>
<td></td>
<td>MAC generate on, MAC verify off (CV bits 20 - 21 = B'10')</td>
<td>GEN-ONLY (&quot;G&quot;)</td>
<td>X'0182'</td>
<td>TR31 Export - Permit MAC/MACVER:CVV-KEYA to C0:G/C/V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAC generate on, MAC verify off (CV bits 20 - 21 = B'010')</td>
<td>VER-ONLY (&quot;V&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAC generate on, MAC verify on (CV bits 20 - 21 = B'01')</td>
<td>GENVER (&quot;C&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAC, double length, ANY-MAC (CV bits 0 - 3 = B'0000')</td>
<td></td>
<td>MAC generate on, MAC verify off (CV bits 20 - 21 = B'10')</td>
<td>GEN-ONLY (&quot;G&quot;)</td>
<td>X'0183'</td>
<td>TR31 Export - Permit MAC/MACVER:ANY-MAC to C0:G/C/V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAC generate on, MAC verify off (CV bits 20 - 21 = B'010')</td>
<td>VER-ONLY (&quot;V&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAC generate on, MAC verify on (CV bits 20 - 21 = B'01')</td>
<td>GENVER (&quot;C&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA, double length</td>
<td></td>
<td>MAC generate on, MAC verify off (CV bits 20 - 21 = B'10')</td>
<td>GEN-ONLY (&quot;G&quot;)</td>
<td>X'0184'</td>
<td>TR31 Export - Permit DATA to C0:G/C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAC generate on, MAC verify off (CV bits 20 - 21 = B'11')</td>
<td>GENVER (&quot;C&quot;)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Security consideration:** There is asymmetry in the translation from a CCA DATA key to a TR-31 key. The asymmetry results from CCA DATA keys having attributes of both data encryption keys and MAC keys, while TR-31 separates data encryption keys from MAC keys. A CCA DATA key can be exported to a TR-31
"D0" key, if one or both applicable Encipher or Decipher control vector bits are on. However, a TR-31 "D0" key cannot be imported to the lower-security CCA DATA key, it can be imported only to a CCA key type of ENCRYPTER, DECRYPTER, or CIPHER. This restriction eliminates the ability to export a CCA DATA key to a TR-31 key and re-importing it back as a CCA DATA key with the capability to MAC generate and MAC verify.

Note:
1. Data encryption keys are used for the encryption and decryption of data.
2. This table defines the only supported translations for this TR-31 usage. Usage must be the following value:
   "D0"  Data encryption

<table>
<thead>
<tr>
<th>Key usage keyword</th>
<th>Key block protection method keyword</th>
<th>CCA key type and required control vector attributes</th>
<th>Mode of use keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENC (&quot;D0&quot;)</td>
<td>VARXOR-A, VARDRV-B, VARXOR-C</td>
<td>ENCIPHER, single or double length</td>
<td>ENC-ONLY (&quot;E&quot;)</td>
<td>X'0185'</td>
<td>TR31 Export - Permit ENCIPHER/DECRYPTER/ CIPHER to D0:E/D/B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DECRYPTER, single or double length</td>
<td>DEC-ONLY (&quot;D&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CIPHER, single or double length</td>
<td>ENCDEC (&quot;B&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA, single or double length, Encipher on, Decipher on (CV bits 18 - 19 = B'11')</td>
<td>ENCDEC (&quot;B&quot;)</td>
<td>X'0186'</td>
<td>TR31 Export - Permit DATA to D0:B</td>
</tr>
</tbody>
</table>

Security consideration: The CCA OKEYXLAT, EXPORTER, IKEYXLAT, or IMPORTER KEK translation to a TR-31 "K0" key with mode "B" (both wrap and unwrap) is not allowed for security reasons. Even with access-control point control, this capability would give an immediate path to turn a CCA EXPORTER key into a CCA IMPORTER key, and the other way around.

Note:
1. Key encryption or wrapping keys are used only to encrypt or decrypt other keys, or as a key used to derive keys that are used for that purpose.
2. This table defines the only supported translations for this TR-31 usage. Usage must be one of the following values:
   "K0"  Key encryption or wrapping
   "K1"  TR-31 key block protection key
3. CCA mode support is the same for version IDs "B" and "C", because the distinction between TR-31 "K0" and "K1" does not exist in CCA keys. CCA does not distinguish between targeted protocols, and so there is no good way to represent the difference. Also note that most wrapping mechanisms now involve derivation or key variation steps.
### Table 207. Export translation table for a TR-31 key encryption or wrapping, or key block protection key (KEK or KEK-WRAP)

<table>
<thead>
<tr>
<th>Key usage keyword</th>
<th>Key block protection method keyword</th>
<th>CCA key type and required control vector attributes</th>
<th>Mode of use keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEK (&quot;K0&quot;)</td>
<td>VARXOR-A, VARDRV-B, VARXOR-C</td>
<td>EXPORTER, double length, EXPORT on (CV bit 21 = B'1')</td>
<td>ENC-ONLY (&quot;E&quot;)</td>
<td>X'0187'</td>
<td>TR31 Export - Permit EXPORTER/OKEYXLAT to K0:E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OKEYXLAT, double length</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>IMPORTER, double length, IMPORT on (CV bit 21 = B'1')</td>
<td>DEC-ONLY (&quot;D&quot;)</td>
<td>X'0188'</td>
<td>TR31 Export - Permit IMPORTER/IKEYXLAT to K0:D</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OKEYXLAT, double length</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEK-WRAP (&quot;K1&quot;)</td>
<td>VARDRV-B, VARXOR-C</td>
<td>EXPORTER, double length, EXPORT on (CV bit 21 = B'1')</td>
<td>ENC-ONLY (&quot;E&quot;)</td>
<td>X'0189'</td>
<td>TR31 Export - Permit EXPORTER/OKEYXLAT to K1:E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OKEYXLAT, double length</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>IMPORTER, double length, IMPORT on (CV bit 21 = B'1')</td>
<td>DEC-ONLY (&quot;D&quot;)</td>
<td>X'018A'</td>
<td>TR31 Export - Permit IMPORTER/IKEYXLAT to K1:D</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OKEYXLAT, double length</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Security consideration:** There is asymmetry in the translation from a CCA DATA key to a TR-31 key. The asymmetry results from CCA DATA keys having attributes of both data encryption keys and MAC keys, while TR-31 separates data encryption keys from MAC keys. A CCA DATA key can be exported to a TR-31 "M0", "M1", or "M3" key, if one or both applicable MAC generate and MAC verify control vector bits are on. However, a TR-31 "M0", "M1", or "M3" key cannot be imported to the lower-security CCA DATA key, it can be imported only to a CCA key type of MAC or MACVER. This restriction eliminates the ability to export a CCA MAC or MACVER key to a TR-31 key and re-importing it back as a CCA DATA key with the capability to Encipher, Decipher, or both.

**Note:**
1. MAC keys are used to compute or verify a code for message authentication.
2. This table defines the only supported translations for this TR-31 usage. Usage must be one of the following values:

   **"M0"**  ISO 16609 MAC algorithm 1, TDEA
   - The ISO 16609 MAC algorithm 1 is based on ISO 9797. It is identical to "M1", except that it does not support 8-byte DES keys.

   **"M1"**  ISO 9797 MAC algorithm 1
   - The ISO 9797 MAC algorithm 1 is identical to "M0", except that it also supports 8-byte DES keys.

   **"M3"**  ISO 9797 MAC algorithm 3
   - The X9.19 style of Triple-DES MAC.
3. A CCA control vector has no bits defined to limit key usage by algorithm, such as CBC MAC (TR-31 usage "M0" and "M1") or X9.19 (TR-31 usage "M3"). When importing a TR-31 key block, the resulting CCA key token deviates from the restrictions of usages "M0", "M1", and "M3". Importing a TR-31 key block which allows MAC generation ("G" or "C") results in a control vector with the ANY-MAC attribute rather than for the restricted algorithm that is set in the TR-31 key block. The ANY-MAC attribute provides the same restrictions as what CCA currently uses for generating and verifying MACs.
### Table 208. Export translation table for a TR-31 ISO MAC algorithm key (ISOMACn)

<table>
<thead>
<tr>
<th>Key usage keyword</th>
<th>Key block protection method keyword</th>
<th>CCA key type and required control vector attributes</th>
<th>Mode of use keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISOMAC0 (&quot;M0&quot;)</td>
<td>VARXOR-A, VARDRV-B, VARXOR-C</td>
<td>MAC, double length, MAC generate on (CV bit 20 = B'1') DATA, double length, MAC generate on (CV bit 20 = B'1') MAC, double length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11') DATAM, double length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11') DATA, double length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11') MACVER, double length, MAC generate off, MAC verify on (CV bits 20 - 21 = B'01') DATAMV, double length, MAC generate off, MAC verify on (CV bits 20 - 21 = B'01')</td>
<td>GEN-ONLY (&quot;G&quot;) GENVER (&quot;C&quot;)</td>
<td>X'018B'</td>
<td>TR31 Export - Permit MAC/DATA/DATAM to M0:G/C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAC, single or double length, MAC generate on (CV bit 20 = B'1') DATA, single or double length, MAC generate on (CV bit 20 = B'1') MAC, single or double length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11') DATAM, single or double length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11') DATA, single or double length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11') MACVER, single or double length, MAC generate off, MAC verify on (CV bits 20 - 21 = B'01') DATAMV, single or double length, MAC generate off, MAC verify on (CV bits 20 - 21 = B'01')</td>
<td>GEN-ONLY (&quot;G&quot;) GENVER (&quot;C&quot;)</td>
<td>X'018D'</td>
<td>TR31 Export - Permit MAC/DATA/DATAM to M1:G/C</td>
</tr>
<tr>
<td>ISOMAC1 (&quot;M1&quot;)</td>
<td>VARXOR-A, VARDRV-B, VARXOR-C</td>
<td>MAC, single or double length, MAC generate on (CV bit 20 = B'1') DATA, single or double length, MAC generate on (CV bit 20 = B'1') MAC, single or double length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11') DATAM, single or double length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11') DATA, single or double length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11') MACVER, single or double length, MAC generate off, MAC verify on (CV bits 20 - 21 = B'01') DATAMV, single or double length, MAC generate off, MAC verify on (CV bits 20 - 21 = B'01')</td>
<td>GEN-ONLY (&quot;G&quot;) GENVER (&quot;C&quot;)</td>
<td>X'018E'</td>
<td>TR31 Export - Permit MACVER/DATAMV to M1:V</td>
</tr>
</tbody>
</table>
### Table 208. Export translation table for a TR-31 ISO MAC algorithm key (ISOMACn) (continued)

<table>
<thead>
<tr>
<th>Key usage keyword</th>
<th>Key block protection method keyword</th>
<th>CCA key type and required control vector attributes</th>
<th>Mode of use keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISOMAC3 (&quot;M3&quot;)</td>
<td>VARXOR-A, VARDRV-B, VARXOR-C</td>
<td>MAC, single or double length, MAC generate on (CV bit 20 = B'1)</td>
<td>GEN-ONLY (&quot;G&quot;)</td>
<td>X'018F'</td>
<td>TR31 Export - Permit MAC/DATA/DATAM to M3:G/C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA, single or double length, MAC generate on (CV bit 20 = B'1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAC, single or double length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11')</td>
<td>GENVER (&quot;C&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATAM, single or double length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11')</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA, single or double length, MAC generate on, MAC verify on (CV bits 20 - 21 = B'11')</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MACVER, single or double length, MAC generate off, MAC verify on (CV bits 20 - 21 = B'01')</td>
<td>VER-ONLY (&quot;V&quot;)</td>
<td>X'0190'</td>
<td>TR31 Export - Permit MACVER/DATAMV to M3:V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATAMV, single or double length, MAC generate off, MAC verify on (CV bits 20 - 21 = B'01')</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Security considerations:

1. It is highly recommended that the INCL-CV keyword be used when exporting PINGEN, PINVER, IPINENC, or OPINENC keys. Using this keyword ensures that importing the TR-31 key block back into CCA will have the desired attributes.

2. TR-31 key blocks that are protected under legacy version ID "A" (keyword VARXOR-A, using the Key Variant Binding Method 2005 Edition) use the same mode of use "N" (keyword ANY) for PINGEN and PINVER keys. For version ID "A" keys only, for a given PIN key usage, enabling both the PINGEN and PINVER access-control points at the same time while enabling offset X'01B0' (for mode "N") is NOT recommended. In other words, for a particular PIN verification usage, you should not simultaneously enable the four commands shown below for that usage:

### Table 209. Commands

<table>
<thead>
<tr>
<th>Key type, mode, or version</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;V0&quot;: For usage V0, a user with the following four commands enabled in the active role can change a PINVER key into a PINGEN key and the other way round. Avoid simultaneously enabling these four commands.</td>
<td>X'0193'</td>
<td>TR31 Export - Permit PINVER:NO-SPEC to V0</td>
</tr>
<tr>
<td>Key type PINVER</td>
<td>X'0193'</td>
<td>TR31 Export - Permit PINVER:NO-SPEC to V0</td>
</tr>
<tr>
<td>Key type PINGEN</td>
<td>X'0194'</td>
<td>TR31 Export - Permit PINGEN:NO-SPEC to V0</td>
</tr>
<tr>
<td>Mode ANY</td>
<td>X'01B0'</td>
<td>TR31 Export - Permit PINGEN/PINVER to V0/V1/V2:N</td>
</tr>
<tr>
<td>Version VARXOR-A</td>
<td>X'014D'</td>
<td>TR31 Export - Permit Version A TR-31 Key Blocks</td>
</tr>
</tbody>
</table>
Table 209. Commands (continued)

<table>
<thead>
<tr>
<th>Key type, mode, or version</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;V1&quot;: For usage V1, a user with the following four commands enabled in the active role can change a PINVER key into a PINGEN key and the other way around. <strong>Avoid simultaneously enabling these four commands.</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key type PINVER</td>
<td>X'0195'</td>
<td>TR31 Export - Permit PINVER:NO-SPEC/IBM-PIN/IBM-PINO to V1</td>
</tr>
<tr>
<td>Key type PINGEN</td>
<td>X'0196'</td>
<td>TR31 Export - Permit PINGEN:NO-SPEC/IBM-PIN/IBM-PINO to V1</td>
</tr>
<tr>
<td>Mode ANY</td>
<td>X'01B0'</td>
<td>TR31 Export - Permit PINGEN/PINVER to V0/V1/V2:N</td>
</tr>
<tr>
<td>Version VARXOR-A</td>
<td>X'014D'</td>
<td>TR31 Export - Permit Version A TR-31 Key Blocks</td>
</tr>
</tbody>
</table>

"V2": For usage V2, a user with the following four commands enabled in the active role can change a PINVER key into a PINGEN key and the other way around. **Avoid simultaneously enabling these four commands.**

| Key type PINVER           | X'0197' | TR31 Export - Permit PINVER:NO-SPEC/VISA-PVV to V2 |
| Key type PINGEN           | X'0198' | TR31 Export - Permit PINGEN:NO-SPEC/VISA-PVV to V2 |
| Mode ANY                  | X'01B0' | TR31 Export - Permit PINGEN/PINVER to V0/V1/V2:N |
| Version VARXOR-A          | X'014D' | TR31 Export - Permit version A TR-31 key blocks |

Failure to comply with this recommendation allows changing PINVER keys into PINGEN and the other way around.

**Note:**

1. PIN encryption keys are used to protect PIN blocks. PIN verification keys are used to generate or verify a PIN using a particular PIN-calculation method for that key type.
2. This table defines the only supported translations for this TR-31 usage. Usage must be one of the following values:
   - "P0" PIN encryption
   - "V0" PIN verification, KPV, other algorithm
     Usage "V0" is intended to be a PIN-calculation method "other" than those methods defined for "V1" or "V2". Because CCA does not have a PIN-calculation method of "other" defined, it maps usage "V0" to the subtype extension of NO-SPEC (CV bits 0 - 3 = B'0000'). Be aware that NO-SPEC allows any method, including "V1" and "V2", and that this mapping is suboptimal.
   - "V1" PIN verification, IBM 3624
   - "V2" PIN verification, Visa PVV
3. Mode must be one of the following values:
   - "E" Encrypt/unwrap only
     This mode restricts PIN encryption keys to encrypting a PIN block. May be used to create or reencipher an encrypted PIN block (for key-to-key translation).
   - "D" Decrypt/unwrap only
     This mode restricts PIN encryption keys to decrypting a PIN block. Generally used in a PIN translation to decrypt the incoming PIN block.
   - "N" No special restrictions (other than restrictions implied by the key usage)
     This mode is used by several vendors for a PIN generate or PIN verification key when the key block version ID is "A".
Key Export to TR31 (CSNBT31X)

"G" Generate only

This mode is used for a PINGEN key that may not perform a PIN verification. This mode is the only mode available when the control vector in the CCA key-token (applicable when INCL-CV keyword is not provided) does NOT have the EPINVER control vector bit on.

"V" Verify only

This mode is used for PIN verification only. This mode is the only mode available when the control vector in the CCA key-token (applicable when INCL-CV is not provided) ONLY has the EPINVER control vector usage bit on (CV bits 18 - 22 = B'00001').

"C" Both generate and verify (combined)

This mode is the only output mode available for TR-31 when any of the CCA key-token PIN generating bits are on in the control vector (CPINGENA, EPINGENA, EPINGEN, or CPINGENA) in addition to the EPINVER bit.

Table 210. Export translation table for a TR-31 PIN encryption or PIN verification key (PINENC, PINVO, PINV3624, VISAPVV)

<table>
<thead>
<tr>
<th>Key usage keyword</th>
<th>Key block protection method keyword</th>
<th>CCA key type and required control vector attributes</th>
<th>Mode of use keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>PINENC (&quot;P0&quot;)</td>
<td>VARXOR-A, VARDRV-B, VARXOR-C</td>
<td>OPINENC, double length</td>
<td>ENC-ONLY (&quot;E&quot;)</td>
<td>X'0191'</td>
<td>TR31 Export - Permit OPINENC to P0:E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IPINENC, double length</td>
<td>DEC-ONLY (&quot;D&quot;)</td>
<td>X'0192'</td>
<td>TR31 Export - Permit IPINENC to P0:D</td>
</tr>
<tr>
<td>PINVO (&quot;V0&quot;)</td>
<td>VARXOR-A</td>
<td>PINVER, double length, NO-SPEC (CV bits 0 - 4 = B'0000')</td>
<td>ANY (&quot;N&quot;) (requires both commands)</td>
<td>X'0193'</td>
<td>TR31 Export - Permit PINVER:NO-SPEC to V0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X'01B0'</td>
<td>TR31 Export - Permit PINVER:EPINENC/PINVER to V0/V1/V2:N</td>
</tr>
<tr>
<td></td>
<td>VARXOR-A, VARDRV-B, VARXOR-C</td>
<td>PINVER, double length, NO-SPEC (CV bits 0 - 4 = B'0000'), CPINGEN off, EPINGENA off, EPINGEN off, CPINGEN off (CV bits 18 - 21 = B'0000')</td>
<td>VER-ONLY (&quot;V&quot;)</td>
<td>X'0193'</td>
<td>TR31 Export - Permit PINVER:NO-SPEC to V0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARXOR-A</td>
<td>PINGEN, double length, NO-SPEC (CV bits 0 - 4 = B'0000')</td>
<td>ANY (&quot;N&quot;) (requires both commands)</td>
<td>X'0194'</td>
<td>TR31 Export - Permit PINGEN:NO-SPEC to V0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X'01B0'</td>
<td>TR31 Export - Permit PINGEN/PINVER to V0/V1/V2:N</td>
</tr>
<tr>
<td></td>
<td>VARXOR-A, VARDRV-B, VARXOR-C</td>
<td>PINGEN, double length, NO-SPEC (CV bits 0 - 4 = B'0000'), EPINVER off (CV bit 22 = B'0')</td>
<td>GEN-ONLY (&quot;G&quot;)</td>
<td>X'0194'</td>
<td>TR31 Export - Permit PINGEN:NO-SPEC to V0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PINGEN, double length, NO-SPEC (CV bits 0 - 4 = B'0000'), EPINVER on (CV bit 22 = B'1)</td>
<td>GENVER (&quot;C&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key usage keyword</td>
<td>Key block protection method keyword</td>
<td>CCA key type and required control vector attributes</td>
<td>Mode of use keyword</td>
<td>Offset</td>
<td>Command</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------</td>
<td>-----------------------------------------------</td>
<td>------------------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>PINV3624 (&quot;V1&quot;)</td>
<td>VARXOR-A</td>
<td>PINVER, double length, NO-SPEC or IBM-PIN/IBM-PINO (CV bits 0 - 4 = B'0000' or B'0001')</td>
<td>ANY (&quot;N&quot;) (requires both commands)</td>
<td>X'0195'</td>
<td>TR31 Export - Permit PINVER:NO-SPEC/IBM-PIN/IBM-PINO to V1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X'01B0'</td>
<td>TR31 Export - Permit PINGEN/PINVER to V0/V1/V2:N</td>
</tr>
<tr>
<td>VARXOR-A,</td>
<td>VARDRV-B, VARXOR-C</td>
<td>PINVER, double length, NO-SPEC or IBM-PIN/IBM-PINO (CV bits 0 - 4 = B'0000' or B'0001'), CPINGEN off, EPINGENA off, EPINGEN off, CPINGENA off (CV bits 18 - 21 = B'0000')</td>
<td>VER-ONLY (&quot;V&quot;)</td>
<td>X'0195'</td>
<td>TR31 Export - Permit PINVER:NO-SPEC/IBM-PIN/IBM-PINO to V1</td>
</tr>
<tr>
<td>VARXOR-A</td>
<td></td>
<td>PINGEN, double length, NO-SPEC or IBM-PIN/IBM-PINO (CV bits 0 - 4 = B'0000' or B'0001')</td>
<td>ANY (&quot;N&quot;) (requires both commands)</td>
<td>X'0196'</td>
<td>TR31 Export - Permit PINGEN:NO-SPEC/IBM-PIN/IBM-PINO to V1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X'01B0'</td>
<td>TR31 Export - Permit PINGEN/PINVER to V0/V1/V2:N</td>
</tr>
<tr>
<td>VARXOR-A,</td>
<td>VARDRV-B, VARXOR-C</td>
<td>PINGEN, double length, NO-SPEC or IBM-PIN/IBM-PINO (CV bits 0 - 4 = B'0000' or B'0001'), EPINVER off (CV bit 22 = B'0')</td>
<td>GEN-ONLY (&quot;G&quot;)</td>
<td>X'0196'</td>
<td>TR31 Export - Permit PINGEN:NO-SPEC/IBM-PIN/IBM-PINO to V1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PINGEN, double length, NO-SPEC or IBM-PIN/IBM-PINO (CV bits 0 - 4 = B'0000' or B'0001'), EPINVER on (CV bit 22 = B'1')</td>
<td>GENVER (&quot;C&quot;)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Table 210. Export translation table for a TR-31 PIN encryption or PIN verification key (PINENC, PINVO, PINV3624, VISAPVV) (continued)

<table>
<thead>
<tr>
<th>Key usage keyword</th>
<th>Key block protection method keyword</th>
<th>CCA key type and required control vector attributes</th>
<th>Mode of use keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>VISAPVV (&quot;V2&quot;)</td>
<td>VARXOR-A</td>
<td>PINVER, double length, NO-SPEC or VISA-PVV (CV bits 0 - 4 = B'0000' or B'0010')</td>
<td>ANY (&quot;N&quot;) (requires both commands)</td>
<td>X'0197</td>
<td>TR31 Export - Permit PINVER:NO-SPEC/VISA-PVV to V2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X'01B0</td>
<td>TR31 Export - Permit PINGEN/PINVER to V0/V1/V2:N</td>
</tr>
<tr>
<td></td>
<td>VARXOR-A, VARDRV-B, VARXOR-C</td>
<td>PINVER, double length, NO-SPEC or VISA-PVV (CV bits 0 - 4 = B'0000' or B'0010'), CPINGEN off, EPINGEN off, EPINGEN off, CPINGEN off (CV bits 18 - 21 = B'0000')</td>
<td>VER-ONLY (&quot;V&quot;)</td>
<td>X'0197</td>
<td>TR31 Export - Permit PINVER:NO-SPEC/VISA-PVV to V2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X'01B0</td>
<td>TR31 Export - Permit PINGEN/PINVER to V0/V1/V2:N</td>
</tr>
<tr>
<td></td>
<td>VARXOR-A</td>
<td>PINGEN, double length, NO-SPEC or VISA-PVV (CV bits 0 - 4 = B'0000' or B'0010')</td>
<td>ANY (&quot;N&quot;) (requires both commands)</td>
<td>X'0198</td>
<td>TR31 Export - Permit PINGEN:NO-SPEC/VISA-PVV to V2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X'01B0</td>
<td>TR31 Export - Permit PINGEN/PINVER to V0/V1/V2:N</td>
</tr>
<tr>
<td></td>
<td>VARXOR-A, VARDRV-B, VARXOR-C</td>
<td>PINGEN, double length, NO-SPEC or VISA-PVV (CV bits 0 - 4 = B'0000' or B'0010'), EPINVER off (CV bit 22 = B'0')</td>
<td>GEN-ONLY (&quot;G&quot;)</td>
<td>X'0198</td>
<td>TR31 Export - Permit PINGEN:NO-SPEC/VISA-PVV to V2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:**

1. EMV/chip issuer master-key keys are used by the chip cards to perform cryptographic operations or, in some cases, to derive keys used to perform operations. In CCA, these keys are (a) diversified key-generating keys (key type DKYGENKY), allowing derivation of operational keys, or (b) operational keys. Note that in this context, the term *master key* has a different meaning than for CCA. These master keys, also called KMCs, are described by EMV as DES master keys for personalization session keys. They are used to derive the corresponding chip card master keys, and not typically used directly for cryptographic operations other than key derivation. In CCA, these keys are usually key generating keys with derivation level DKYL1 (CV bits 12 - 14 = B'001'), used to derive other key generating keys (the chip card master keys). For some cases, or for older EMV key derivation methods, the issuer master keys could be level DKYL0 (CV bits 12 - 14 = B'000').

2. This table defines the only supported translations for this TR-31 usage. Usage must be one of the following values:

   - "E0" Application cryptograms
   - "E1" Secure messaging for confidentiality
   - "E2" Secure messaging for integrity
   - "E3" Data authentication code
   - "E4" Dynamic numbers
“E5”  Card personalization  
3. EMV support in CCA is different than TR-31 support, and CCA key types do not match TR-31 types.
4. DKYGENKY keys are double length only.

Table 211. Export translation table for a TR-31 EMV/chip issuer master-key key (DKYGENKY, DATA)

<table>
<thead>
<tr>
<th>Key usage keyword</th>
<th>Key block protection method keyword</th>
<th>CCA key type and required control vector attributes</th>
<th>Mode of use keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMVACMK (&quot;E0&quot;)</td>
<td>VARXOR-A</td>
<td>DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMAC (CV bits 19 - 22 = B'0010')</td>
<td>ANY ('N')</td>
<td>X'0199'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0 +DMAC to E0</td>
</tr>
<tr>
<td></td>
<td>VARDRV-B, VARXOR-C</td>
<td>DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMV (CV bits 19 - 22 = B'0011')</td>
<td>ANY ('N')</td>
<td>X'019A'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0 +DMV to E0</td>
</tr>
<tr>
<td></td>
<td>VARXOR-B, VARXOR-C</td>
<td>DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DALL (CV bits 19 - 22 = B'1111')</td>
<td>ANY ('N')</td>
<td>X'019B'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0 +DALL to E0</td>
</tr>
<tr>
<td></td>
<td>VARXOR-A</td>
<td>DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DMAC (CV bits 19 - 22 = B'0010')</td>
<td>ANY ('N')</td>
<td>X'019C'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1 +DMAC to E0</td>
</tr>
<tr>
<td></td>
<td>VARDRV-B, VARXOR-C</td>
<td>DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DMV (CV bits 19 - 22 = B'1001')</td>
<td>ANY ('N')</td>
<td>X'019D'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1 +DMV to E0</td>
</tr>
<tr>
<td></td>
<td>VARXOR-A</td>
<td>DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DALL (CV bits 19 - 22 = B'1111')</td>
<td>ANY ('N')</td>
<td>X'019E'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1 +DALL to E0</td>
</tr>
<tr>
<td></td>
<td>VARXOR-B, VARXOR-C</td>
<td>DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DDATA (CV bits 19 - 22 = B'0001')</td>
<td>ANY ('N')</td>
<td>X'019F'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0 +DDATA to E1</td>
</tr>
<tr>
<td></td>
<td>VARXOR-A</td>
<td>DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMPIN (CV bits 19 - 22 = B'1001')</td>
<td>ANY ('N')</td>
<td>X'01A0'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0 +DMPIN to E1</td>
</tr>
<tr>
<td></td>
<td>VARXOR-B, VARXOR-C</td>
<td>DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DALL (CV bits 19 - 22 = B'1111')</td>
<td>ANY ('N')</td>
<td>X'01A1'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0 +DALL to E1</td>
</tr>
<tr>
<td></td>
<td>VARXOR-A</td>
<td>DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DDATA (CV bits 19 - 22 = B'0001')</td>
<td>ANY ('N')</td>
<td>X'01A2'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1 +DDATA to E1</td>
</tr>
<tr>
<td></td>
<td>VARXOR-B, VARXOR-C</td>
<td>DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DMPIN (CV bits 19 - 22 = B'1001')</td>
<td>ANY ('N')</td>
<td>X'01A3'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1 +DMPIN to E1</td>
</tr>
<tr>
<td></td>
<td>VARXOR-A</td>
<td>DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DALL (CV bits 19 - 22 = B'1111')</td>
<td>ANY ('N')</td>
<td>X'01A4'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1 +DALL to E1</td>
</tr>
<tr>
<td></td>
<td>VARXOR-B, VARXOR-C</td>
<td>DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DDATA (CV bits 19 - 22 = B'0001')</td>
<td>ANY ('N') derived</td>
<td>X'0199'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0 +DDATA to E1</td>
</tr>
<tr>
<td></td>
<td>VARXOR-A</td>
<td>DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMPIN (CV bits 19 - 22 = B'1001')</td>
<td>ANY ('N') derived</td>
<td>X'01A0'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0 +DMPIN to E1</td>
</tr>
<tr>
<td></td>
<td>VARXOR-B, VARXOR-C</td>
<td>DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DALL (CV bits 19 - 22 = B'1111')</td>
<td>ANY ('N') derived</td>
<td>X'01A1'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0 +DALL to E1</td>
</tr>
<tr>
<td></td>
<td>VARXOR-A</td>
<td>DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DDATA (CV bits 19 - 22 = B'0001')</td>
<td>ANY ('N') derived</td>
<td>X'01A2'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1 +DDATA to E1</td>
</tr>
<tr>
<td></td>
<td>VARXOR-B, VARXOR-C</td>
<td>DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DMPIN (CV bits 19 - 22 = B'1001')</td>
<td>ANY ('N') derived</td>
<td>X'01A3'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1 +DMPIN to E1</td>
</tr>
<tr>
<td></td>
<td>VARXOR-A</td>
<td>DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DALL (CV bits 19 - 22 = B'1111')</td>
<td>ANY ('N') derived</td>
<td>X'01A4'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1 +DALL to E1</td>
</tr>
</tbody>
</table>

Chapter 16. TR-31 symmetric key verbs  773
**Table 211. Export translation table for a TR-31 EMV/chip issuer master-key key (DKYGENKY, DATA) (continued)**

<table>
<thead>
<tr>
<th>Key usage keyword</th>
<th>Key block protection method keyword</th>
<th>CCA key type and required control vector attributes</th>
<th>Mode of use keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EMVSIMK (&quot;E2&quot;)</strong></td>
<td>VARXOR-A</td>
<td>DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMAC (CV bits 19 - 22 = B'0010')</td>
<td>ANY ('N&quot;)</td>
<td>X'01A5&quot;</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0 +DMAC to E2</td>
</tr>
<tr>
<td></td>
<td>VARDRV-B, VARXOR-C</td>
<td>DKYGENKY, double length, DKL0 (CV bits 12 - 14 = B'000'), DALL (CV bits 19 - 22 = B'1111')</td>
<td>ANY ('N&quot;)</td>
<td>X'01A6&quot;</td>
<td>TR31 Export - Permit DKYGENKY:DKYLO +DALL to E2</td>
</tr>
<tr>
<td></td>
<td>VARXOR-A</td>
<td>DKYGENKY, double length, DKL1 (CV bits 12 - 14 = B'001'), DMAC (CV bits 19 - 22 = B'0010')</td>
<td>ANY ('N&quot;)</td>
<td>X'01A7&quot;</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1 +DMAC to E2</td>
</tr>
<tr>
<td></td>
<td>VARDRV-B, VARXOR-C</td>
<td>DKYGENKY, double length, DKL1 (CV bits 12 - 14 = B'001'), DALL (CV bits 19 - 22 = B'1111')</td>
<td>ANY ('N&quot;)</td>
<td>X'01A8&quot;</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1 +DALL to E2</td>
</tr>
<tr>
<td><strong>EMVDAMK (&quot;E3&quot;)</strong></td>
<td>VARXOR-A</td>
<td>DATA, double length</td>
<td>ANY ('N&quot;)</td>
<td>X'01A9&quot;</td>
<td>TR31 Export - Permit DATA/MAC/CIPHER/ENCIPHER to E3</td>
</tr>
<tr>
<td></td>
<td>VARDRV-B, VARXOR-C</td>
<td>MAC (not MACVER), double length</td>
<td>ANY ('N&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARXOR-A</td>
<td>MAC (not MACVER), double length</td>
<td>ANY ('N&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARDRV-B, VARXOR-C</td>
<td>DERIVE ('X&quot;)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARXOR-A</td>
<td>CIPHER, double length</td>
<td>ANY ('N&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARDRV-B, VARXOR-C</td>
<td>DERIVE ('X&quot;)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARXOR-A</td>
<td>ENCIPHER, double length</td>
<td>ANY ('N&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARDRV-B, VARXOR-C</td>
<td>DERIVE ('X&quot;)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EMVDNMK (&quot;E4&quot;)</strong></td>
<td>VARXOR-A</td>
<td>DKYGENKY, double length, DKL0 (CV bits 12 - 14 = B'000'), DDATA (CV bits 19 - 22 = B'0001')</td>
<td>ANY ('N&quot;)</td>
<td>X'01AA&quot;</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0 +DDATA to E4</td>
</tr>
<tr>
<td></td>
<td>VARDRV-B, VARXOR-C</td>
<td>DKYGENKY, double length, DKL0 (CV bits 12 - 14 = B'000'), DALL (CV bits 19 - 22 = B'1111')</td>
<td>ANY ('N&quot;)</td>
<td>X'01AB&quot;</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0 +DALL to E4</td>
</tr>
<tr>
<td>Key usage keyword</td>
<td>Key block protection method keyword</td>
<td>CCA key type and required control vector attributes</td>
<td>Mode of use keyword</td>
<td>Offset</td>
<td>Command</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------</td>
<td>---------------------------------------------------</td>
<td>-------------------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td><strong>EMVCPMK</strong> (&quot;E5&quot;)</td>
<td><strong>VARXOR-A</strong></td>
<td>DKYGENKY, double length, DKL0 (CV bits 12 - 14 = B'0000'), DEXP (CV bits 19 - 22 = B'0101')</td>
<td>ANY (&quot;N&quot;)</td>
<td>X'01AC'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0 +DEXP to E5</td>
</tr>
<tr>
<td><strong>VARDRV-B, VARXOR-C</strong></td>
<td><strong>VARXOR-A</strong></td>
<td>DKYGENKY, double length, DKL0 (CV bits 12 - 14 = B'0000'), DMAC (CV bits 19 - 22 = B'0010')</td>
<td>ANY (&quot;N&quot;)</td>
<td>X'01AD'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0 +DMAC to E5</td>
</tr>
<tr>
<td><strong>VARDRV-B, VARXOR-C</strong></td>
<td><strong>VARXOR-A</strong></td>
<td>DKYGENKY, double length, DKL0 (CV bits 12 - 14 = B'0000'), DDATA (CV bits 19 - 22 = B'0001')</td>
<td>ANY (&quot;N&quot;)</td>
<td>X'01AE'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0 +DDATA to E5</td>
</tr>
<tr>
<td><strong>VARXOR-A</strong></td>
<td><strong>VARDRV-B, VARXOR-C</strong></td>
<td>DKYGENKY, double length, DKL0 (CV bits 12 - 14 = B'0000'), DALL (CV bits 19 - 22 = B'1111')</td>
<td>ANY (&quot;N&quot;)</td>
<td>X'01AF'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0 +DALL to E5</td>
</tr>
</tbody>
</table>

**key_version_number**

*Direction:* Input  
*Type:* String

A pointer to a string variable containing two numeric ASCII bytes that are copied into the key version number field of the output TR-31 key block. Use a value of "00" (X'3030') if no key version number is needed. This value is ignored if the key identified by the *source_key_identifier* parameter contains a partial key, that is, the KEY-PART bit (CV bit 44) is on in the control vector. When a partial key is passed, the verb sets the key version number field in the TR-31 key block to C0 (X'6330'). According to TR-31, this value indicates that the TR-31 key block contains a component of a key (key part).

**key_field_length**

*Direction:* Input  
*Type:* Integer

A pointer to an integer variable containing the length of the key field that is encrypted in the TR-31 block. The length must be a multiple of the DES cipher block size, which is eight. It must also be greater than or equal to the length of the cleartext key passed using the *source_key_identifier* parameter plus the length of the key length field (two bytes) that precedes this key in the TR-31 block. For example, if the source key is a double-length TDES key (its length is 16 bytes), then the key field length must be greater than or equal to (16 + 2) bytes, and must also be a multiple of 8. This means that the minimum *key_field_length* in this case would be 24.

TR-31 allows a variable number of padding bytes to follow the cleartext key, and the application designer can choose to pad with more than the minimum number of bytes needed to form a block that is a multiple of 8. This padding is generally done to hide the length of the cleartext key from those who cannot decipher that key. Most often, all keys (single, double, or triple length) are padded to the same length so that it is not possible to determine which length is carried in the TR-31 block by examining the encrypted block.
Note: This parameter is not expected to allow for ASCII encoding of the encrypted data stored in the key field according to the TR-31 specification. For example, when a value of 24 is passed here, following the minimum example above, the length of the final ASCII-encoded encrypted data in the key field in the output TR-31 key block is 48 bytes.

**source_key_identifier_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the length in bytes of the `source_key_identifier` variable. The value must be 64.

**source_key_identifier**

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing either the key label for the source key, or the key token containing the source key. The source key is the key that is to be exported. The key must be a CCA fixed-length DES internal or external key-token. If the source key is an external token, an identifier for the KEK that wraps the source key must be identified by the `unwrap_kek_identifier` parameter. TR-31 currently supports only DES and TDES keys. AES is not supported.

**unwrap_kek_identifier_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the length in bytes of the `unwrap_kek_identifier` variable. The value must be greater than or equal to 0. A null key-token can have a length of 1. Set this value to 64 for a key label or a KEK.

**unwrap_kek_identifier**

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing either the key label for the source key KEK, or the key token containing the source key KEK when the source key is an external CCA key token, and a NULL key token otherwise. The source key KEK can also be the wrapping key for the key that is to be exported if the `wrap_kek_identifier` is not specified. The source key KEK must be a CCA internal DES KEK token of type EXPORTER or OKEYXLAT.

**Note:** ECB-mode wrapped DES keys (CCA legacy wrap mode) cannot be used to wrap or unwrap TR-31 "B" or "C" key blocks that have or will have "E" exportability, because ECB mode does not comply with ANSI X9.24 Part 1.

**wrap_kek_identifier_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the length in bytes of the `wrap_kek_identifier` variable. Set this value to 64.

**wrap_kek_identifier**

- **Direction:** Input
Type: String

A pointer to a string variable containing an operational fixed-length DES key-token with a key type of EXPORTER or OKEYXLAT to use for wrapping the output TR-31 key block, a null key token, or a key label of such a key in DES key-storage. If the identified key token is null, then the key identified by the unwrap_kek_identifier parameter is also used for wrapping the output TR-31 key block.

**Note:** ECB-mode wrapped DES keys (CCA legacy wrap mode) cannot be used to wrap or unwrap TR-31 'B' or 'C' key blocks that have or will have "E" exportability, because ECB-mode does not comply with ANSI X9.24 Part 1. This parameter exists to allow for KEK separation. It is possible that KEKs are restricted as to what they can wrap, such that a KEK for wrapping CCA external keys might not be usable for wrapping TR-31 external keys, or the other way around.

### opt_blocks_length

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable that specifies the length in bytes of the opt_blocks variable. If no optional data is to be included in the TR-31 key block, set this value to zero.

### opt_blocks

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing optional blocks data that is to be included in the output TR-31 key block. The optional blocks data can be constructed using the TR31 Optional Data Build verb.

**Note:** The Padding Block, ID "PB" cannot be added by the user, and therefore is not accepted in the opt_blocks parameter. CCA adds a Padding Block of the appropriate size as needed when building the TR-31 key block in Key Export to TR31. The Padding Block for optional blocks serves no security purpose, unlike the padding in the encrypted key portion of the payload.

### tr31_key_block_length

- **Direction:** Input/Output
- **Type:** Integer

A pointer to an integer variable containing the length in bytes of the tr31_key_block variable. On input, specify the size of the application program buffer available for the output key-token. On return from the verb, this variable is updated to contain the actual length of that returned token. TR-31 key blocks are variable in length.

### tr31_key_block

- **Direction:** Output
- **Type:** String

A pointer to a string variable containing the output key block produced by the verb. The output key block contains the external form of the key created by the verb, wrapped according to the method specified.
**Key Export to TR31 (CSNBT31X)**

**Note:** The padding optional block in the output TR-31 key block can be present with zero data bytes. This situation can occur if the optional block portion of the header needs exactly four bytes of padding, the size of an optional block header without the data portion. The data portion is defined as optional by TR-31, which allows this.

**Restrictions**

The restrictions for CSNBT31X.

- The only proprietary values for the TR-31 header fields supported by this verb are those values defined and used by IBM CCA when carrying a control vector in an optional block in the header.
- AES is not currently supported for TR-31 key blocks.
- The export is prohibited if the CCA key does not have attributes XPORT-OK (CV bit 17 = B’1’) and T31XPTOK (CV bit 57 = B’1’).

**Required commands**

The required commands for CSNBT31X.

The Key Export to TR31 verb requires the following commands to be enabled in the active role:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>TR-31 key block protection method</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARXOR-A</td>
<td>X’014D’</td>
<td>TR31 Export - Permit version A TR-31 key blocks</td>
</tr>
<tr>
<td>VARDRV-B</td>
<td>X’014E’</td>
<td>TR31 Export - Permit version B TR-31 key blocks</td>
</tr>
<tr>
<td>VARXOR-C</td>
<td>X’014F’</td>
<td>TR31 Export - Permit version C TR-31 key blocks</td>
</tr>
<tr>
<td>INCL-CV</td>
<td>X’0158’</td>
<td>TR31 Export - Permit any CCA key if INCL-CV is specified</td>
</tr>
</tbody>
</table>

When providing the INCL-CV keyword:
- If this command is enabled in the active role, the key-type specific commands are not checked.
- If this command is not enabled in the active role, the key-type specific commands are required.

| ATTR-CV | N/A  | Note: No commands relating to specific CCA to TR-31 transitions are checked when this keyword is specified. Only the general access control commands related to export are checked. |

Be aware of the interaction of access-control point X’0158’ (TR31 Export - Permit any CCA key if INCL-CV is specified) with the INCL-CV keyword. Without the INCL-CV keyword, most export translations are guarded by key-type-specific access control points, to guard the source CCA system against attacks involving re-import of the exported key under ambiguous circumstances. When the control vector is exported along with the key as an optional block securely bound to the encrypted key, the source system is somewhat protected because the key on import is allowed to have only the form of the included control vector. No expansion of capability is allowed. If access-control point X’0158’ is enabled in the active role, the key-type-specific access control points are not checked when INCL-CV is provided. If ACP X’0158’ is not enabled, the type-specific access control points are still required. The following figure illustrates this concept:
Be aware of access-control point X'01B0' (TR31 Export - Permit PINGEN/PINVER to V0/V1/V2:N) for export of PINGEN or PINVER keys to wrapping method A, usage V0, V1, or V2, and mode N. TR-31 key blocks with legacy key usage A (key block protected using the Key Variant Binding Method 2005 Edition) use the same mode N for PINGEN as well as PINVER keys. For usage A keys only, enabling a PINGEN and PINVER access-control point while enabling offset X'01B0' (for keyword ANY, mode N) is NOT recommended. Failure to comply with this recommendation allows changing PINVER keys into PINGEN and the other way around.

In addition to the above commands, the verb requires these additional commands to be enabled in the active role depending on the TR-31 key usage rule-array keyword provided and additional information as shown in the table referenced in the rightmost column:

<table>
<thead>
<tr>
<th>TR-31 key usage keyword</th>
<th>Offset</th>
<th>Command</th>
<th>Specific key type and control vector attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;B0&quot;: TR-31 BDK base derivation keys</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BDK</td>
<td>X'0180'</td>
<td>TR31 Export - Permit KEYGENKY:UKPT to B0</td>
<td>See Table 204 on page 762</td>
</tr>
<tr>
<td>&quot;C0&quot;: TR-31 CVK card verification keys</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVK</td>
<td>X'0181'</td>
<td>TR31 Export - Permit MAC/MACVER:AMEX-CSC to C0:G/C/V</td>
<td>See Table 205 on page 764</td>
</tr>
<tr>
<td></td>
<td>X'0182'</td>
<td>TR31 Export - Permit MAC/MACVER:CVV-KEYA to C0:G/C/V</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0183'</td>
<td>TR31 Export - Permit MAC/MACVER:ANY-MAC to C0:G/C/V</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0184'</td>
<td>TR31 Export - Permit DATA to C0:G/C</td>
<td></td>
</tr>
<tr>
<td>&quot;D0&quot;: TR-31 data encryption keys</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENC</td>
<td>X'0185'</td>
<td>TR31 Export - Permit ENCIIPHER/DECIPHER/CIPHER to D0:E/D/B</td>
<td>See Table 206 on page 765</td>
</tr>
<tr>
<td></td>
<td>X'0186'</td>
<td>TR31 Export - Permit DATA to D0:B</td>
<td></td>
</tr>
<tr>
<td>&quot;E0&quot;, &quot;E1&quot;, &quot;E2&quot;, &quot;E3&quot;, &quot;E4&quot;, and &quot;E5&quot;: TR-31 EMC/chip issuer master-key keys</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMVACMK</td>
<td>X'0199'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DMAC to E0</td>
<td>See Table 207 on page 766</td>
</tr>
<tr>
<td></td>
<td>X'019A'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DMV to E0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'019B'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DALL to E0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'019C'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DMAC to E0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'019D'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DMV to E0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'019E'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DALL to E0</td>
<td></td>
</tr>
<tr>
<td>TR-31 key usage keyword</td>
<td>Offset</td>
<td>Command</td>
<td>Specific key type and control vector attributes</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------</td>
<td>---------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>EMVSCMK</td>
<td>X'019F'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DDATA to E1</td>
<td>See Table 207 on page 766</td>
</tr>
<tr>
<td></td>
<td>X'01A0'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DMPIN to E1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'01A1'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DALL to E1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'01A2'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DDATA to E1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'01A3'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DMPIN to E1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'01A4'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DALL to E1</td>
<td></td>
</tr>
<tr>
<td>EMVSIMK</td>
<td>X'01A5'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DMAC to E2</td>
<td>See Table 207 on page 766</td>
</tr>
<tr>
<td></td>
<td>X'01A6'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DALL to E2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'01A7'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DMAC to E2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'01A8'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DALL to E2</td>
<td></td>
</tr>
<tr>
<td>EMVDAMK</td>
<td>X'01A9'</td>
<td>TR31 Export - Permit DATA/MAC/CIPHER/ENCIPHER to E3</td>
<td>See Table 207 on page 766</td>
</tr>
<tr>
<td>EMVDNMK</td>
<td>X'01AA'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DDATA to E4</td>
<td>See Table 207 on page 766</td>
</tr>
<tr>
<td></td>
<td>X'01AB'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DALL to E4</td>
<td></td>
</tr>
<tr>
<td>EMVCPMK</td>
<td>X'01AC'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DEXP to E5</td>
<td>See Table 207 on page 766</td>
</tr>
<tr>
<td></td>
<td>X'01AD'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DMAC to E5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'01AE'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DDATA to E5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'01AF'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DALL to E5</td>
<td></td>
</tr>
</tbody>
</table>

"K0" and "K1": TR-31 key encryption or wrapping, or key block protection keys

| KEK                    | X'0187' | TR31 Export - Permit EXPORTER/OKEYXLAT to K0:E | See Table 208 on page 767 |
|                        | X'0188' | TR31 Export - Permit IMPORTER/IKEYXLAT to K0:D |

| KEK-WRAP               | X'0189' | TR31 Export - Permit EXPORTER/OKEYXLAT to K1:E |
|                        | X'018A' | TR31 Export - Permit IMPORTER/IKEYXLAT to K1:D |

"M0", "M1", and "M3": TR-31 ISO MAC algorithm keys

| ISOMAC0                | X'018F' | TR31 Export - Permit MAC/DATA/DATAM to M0:G/C | See Table 210 on page 770 |
|                        | X'018C' | TR31 Export - Permit MACVER/DATAMV to M0:V |

| ISOMAC1                | X'018D' | TR31 Export - Permit MAC/DATA/DATAM to M1:G/C |
|                        | X'018E' | TR31 Export - Permit MACVER/DATAMV to M1:V |

| ISOMAC3                | X'018F' | TR31 Export - Permit MAC/DATA/DATAM to M3:G/C |
|                        | X'0190' | TR31 Export - Permit MACVER/DATAMV to M3:V |

"P0", "V0", "V1": TR-31 PIN encryption or PIN verification keys
### JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBT31XJ.

See “Building Java applications using the CCA JNI” on page 28.

#### Format

```java
public native void CSNBT31XJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    byte[] key_version_number,
    hikmNativeNumber key_field_length,
    hikmNativeNumber source_key_identifier_length,
    byte[] source_key_identifier,
    hikmNativeNumber unwrap_kek_identifier_length,
    byte[] unwrap_kek_identifier,
    hikmNativeNumber wrap_kek_identifier_length,
    byte[] wrap_kek_identifier,
    hikmNativeNumber opt_blocks_length,
    byte[] opt_blocks,
    hikmNativeNumber tr31_key_block_length,
    byte[] tr31_key_block);
```

### TR31 Key Import (CSNBT31I)

Use the TR31 Key Import verb to convert a non-proprietary external key-block that is formatted under the rules of TR-31 into a proprietary CCA external or internal fixed-length DES key-token with its attributes in a control vector.

After being imported into a CCA key-token, the key and its attributes are ready to be used in a CCA system. The verb takes as input an external TR-31 key block and the internal DES IMPORTER or IKEYXLAT key-encrypting key of the key that was used to wrap the TR-31 key block.
The TR31 Key Import verb is analogous to the existing Key Import verb, except that TR31 Key Import accepts an external non-CCA DES key-token instead of an external CCA fixed-length DES key-token, and it translates the key to either an external or internal fixed-length DES key-token instead of only an internal fixed-length DES key-token. An import by TR31 Key Import to an external key-token requires a suitable internal fixed-length DES key-encrypting key. The purpose of both verbs is to import a DES key from another party.

An external-to-external translation would not normally be called an export or import operation. Instead, it would be called a key translation and would be handled by a verb such as Key Translate or Key Translate2. For practical reasons, the export of an external CCA DES key-token to an external TR-31 format is supported by the Key Export to TR31 verb, and the import of an external TR-31 key block to an external CCA DES key-token format is supported by the TR31 Key Import verb.

Note that the TR31 Key Import verb does not support the translation of an external key from encryption under one key-encrypting key to encryption under a different key-encrypting key. When converting an external TR-31 key block to an external fixed-length DES key-token, the key-encrypting key used to wrap the external TR-31 key block must be the same as the one used to wrap the external fixed-length DES key-token. Use the Key Translate or Key Translate2 verbs for switching external key wrapping keys: the normal function of those verbs.

Both CCA and TR-31 define key attributes that control key usage. In both cases, the usage information is securely bound to the key so that the attributes cannot be changed in unintended or uncontrolled ways. CCA maintains its DES key attributes in a control vector (CV), while a TR-31 key block uses fields: key usage, algorithm, mode of use, and exportability.

Each attribute in a CCA control vector falls under one of these categories:
1. There is a one-to-one correspondence between the CV attribute and the TR-31 attribute. For these attributes, conversion is straightforward.
2. There is not a one-to-one correspondence between the CV attribute and the TR-31 attribute, but the attribute can be automatically translated when performing this export operation.
3. There is not a one-to-one correspondence between the CV attribute and the TR-31 attribute, in which case a rule-array keyword has been defined to specify which attribute is to be used in the TR-31 key block.
4. Category (1), (2), or (3) applies, but there are some attributes that are lost completely on translation (for example, key-generating bits in key-encrypting keys).
5. None of the above categories applies, because the key type, its attributes, or both simply cannot be reasonably translated into a TR-31 key block.

The control vector is always checked for compatibility with the TR-31 attributes. It is an error if the specified control vector attributes are in any way incompatible with the attributes of the input key. In addition, access control points are defined that can be used to restrict the permitted attribute conversions.

The import operation produces the CCA external or internal fixed-length DES key-token as its output. It does not return any field values or optional block data from the TR-31 key block header. To obtain the header field values, use the TR31 Key Token Parse verb. To obtain optional block data from the header, use the TR31 Optional Data Read verb.
An optional control vector transport control rule-array keyword can be passed to the Key Export to TR31 verb. Such a keyword specifies that the verb is to copy the control vector from the CCA DES key into the TR-31 key block. A copy of the control vector is passed in an IBM-proprietary optional block. See “TR-31 optional block data” on page 967.

If the TR-31 key block contains an IBM-proprietary block, the TR31 Key Import verb verifies that the control vector is compatible with the attributes in the TR-31 key block. If any incompatibility is found, the verb rejects the import. If the control vector is valid for the key, the verb uses it for the control vector of the CCA DES key-token. Note that the import operation is always subject to the restrictions imposed by the relevant access control points, even if a control vector is received.

A control vector, if present, can be in the TR-31 key block in one of two ways, depending on the control vector transport control keyword specified in the rule array of the Key Export to TR31 verb when the key was exported. One keyword option is ATTR-CV, and the other is INCL-CV:

ATTR-CV
Causes a copy of the control vector to be included in the TR-31 key block. The TR-31 key usage and mode of use fields are set to IBM proprietary. See “TR-31 optional block data” on page 967. These proprietary values indicate that the usage and mode information is contained in the included control vector. In this case, if the TR31 Key Import verb successfully verifies that the included control vector does not conflict with the rule-array keywords specified, it uses it as the control vector for the imported CCA DES key-token.

INCL-CV
Causes a copy of the control vector to be included in the TR-31 key block. The TR-31 key usage and mode of use fields contain attributes from the set defined in the TR-31 standard. In this case, the TR31 Key Import verb verifies that the usage and mode information in those fields are compatible with the included control vector. The verb also verifies that no rule array keywords conflict with the control vector.

Note that the included CV could have more capability from a CCA perspective than the TR-31 usage and mode fields indicate. This difference is not an error, because the key block binding methods give the importer assurance that the key block optional blocks are as secure as any other attribute.

Note: This verb supports PCI-HSM 2016 compliant-tagged key tokens.

With PCI-HSM 2016, CCA introduces compliance-tagged key support for domains that enter compliance mode. Compliance-tagged TDES keys may be converted to or from TR-31 key blocks. The wrapping key encrypting key (KEK) must be compliance-tagged as well as the key to be exported or imported. A compliance-tagged KEK cannot be used to import or export a key that is not compliance-tagged.

Special notes
Additional information about CSNB3T31I.
1. Several import situations might require keywords. Keywords are ignored for INCL-CV scenarios unless they directly conflict with the included CV. For
example, the verb returns an error if the control vector indicates that a DKYGENKY key has a subtype of DKYL0 and the user specifies the DKYL1 keyword.

2. Be aware of the interaction of ACP X'0158' (TR31 Export - Permit Any CCA Key if INCL-CV Is Specified) with the INCL-CV keyword for the Key Export to TR31 verb. Without the INCL-CV keyword specified, most export translations are guarded by key-type specific ACPs. These ACPs are used to guard the source CCA system against attacks involving reimport of the exported key under ambiguous circumstances. When the control vector is exported in an optional block along with the key, it is securely bound to the encrypted key. This somewhat protects the source system because the key on import is allowed to have only the form of the included control vector. Expansion of capability is blocked. If ACP X'0158' is not enabled in the active role, the type-specific ACPs are still required. However, if ACP X'0158' is enabled, the key-type specific ACPs are not checked when INCL-CV is specified.

3. Be aware of ACP X'017C' (TR31 Import - Permit V0/V1/V2:N to PINGEN/PINVER) for import of PINGEN or PINVER keys to wrapping mode A, usage V0, V1, V2, and mode N.

The extra translation-specific ACPs are intended to enable control of situations where the CCA imported key type is ambiguous based on the TR-31 key attributes. This ambiguity is never the case when INCL-CV has been specified with the Key Export to TR31 verb, which ensures that the imported TR-31 key block has a valid CV to precisely control the resultant CCA key. Therefore, there are no translation-specific ACPs governing INCL-CV import translations.

Examples

Examples for CSNBT31I.

1. A full MAC key that is exported as TR-31 "C0" key block with an included control vector will be re-imported as a full MAC key.

2. A DKYGENKY key with key usage DALL key exported as a TR-31 "E0", "E1", or "E2" key block with an included control vector will be re-imported as a DKYGENKY key with key usage DALL, even though the "E0", "E1", and "E2" types are more restricted.

Format

The format of CSNBT31I.

```
CSNBT31I(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    tr31_key_block_length,
    tr31_key_block,
    unwrap_kek_identifier_length,
    unwrap_kek_identifier,
    wrap_kek_identifier_length,
    wrap_kek_identifier,
    output_key_identifier_length,
    output_key_identifier,
    num_opt_blocks,
    cv_source,
    protection_method)
```
Parameters

The parameter definitions for CSNBT31I.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see "Parameters common to all verbs" on page 22.

\textbf{rule_array_count}

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the \text{rule_array} variable. The value must be 1, 2, 3, or 4.

\textbf{rule_array}

- **Direction:** Input
- **Type:** String array

A pointer to a string variable containing an array of keywords. The keywords are 8 bytes in length and must be left-aligned and padded on the right with space characters. The \text{rule_array} keywords are shown in the following table:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token identifier (one required)</td>
<td></td>
</tr>
<tr>
<td>INTERNAL</td>
<td>Specifies to return the output key in an internal CCA key-token.</td>
</tr>
<tr>
<td>EXTERNAL</td>
<td>Specifies to return the output key in an external CCA key-token, wrapped by the transport key identified by the wrap_kek_identifier parameter.</td>
</tr>
</tbody>
</table>

**CCA output key usage subgroups** (One from one subgroup required based on TR-31 input key usage. Keywords for the subgroup are valid only for given TR-31 key usage.)

\textbf{Note:} None of the following keywords are allowed if the TR-31 key block provided as input has an optional block that contains a CCA control vector. See Table 279 on page 968. If the TR-31 key block header contains an optional block with a control vector in it, the control vector is used in place of keywords to produce the output CCA key-token. If the key usage and mode of use fields of the key block are not IBM-defined (see Table 279 on page 968), the control vector must not conflict with any TR-31 header fields.

**CV subtype extension for "C0" key usage** (one required). Only valid for TR-31 key block with key usage "C0" and no control vector in optional block.

- **CVK-CVV** Convert a TR-31 card verification key (CVK) to a double-length CCA DES MAC key that has a subtype extension of CVVKEY-A. This restricts the key to generating or verifying a Visa CVV or MasterCard CVC.

- **CVK-CSC** Convert a TR-31 CVK to a CCA DES MAC key that has a subtype extension of AMEX-CSC. This restricts the key to generating or verifying an American Express card security code, also known as a card identification number (CID).

**CV key type for "K0" key usage** (one required). Only valid for TR-31 key block with key usage "K0" and no control vector in optional block.

- **EXPORTER** For TR-31 key usage "K0" and mode of use "E" or "B", convert a TR-31 key encryption or wrapping key to a CCA EXPORTER key.

- **OKEYXLAT** For TR-31 key usage "K0" and mode of use "E" or "B", convert a TR-31 key encryption or wrapping key to a CCA OKEYXLAT key.

- **IMPORTER** For TR-31 key usage "K0" and mode of use "D" or "B", convert a TR-31 key encryption or wrapping key to a CCA IMPORTER key.

- **IKEYXLAT** For TR-31 key usage "K0" and mode of use "D" or "B", convert a TR-31 key encryption or wrapping key to a CCA IKEYXLAT key.
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CV key type for &quot;V0&quot;, &quot;V1&quot;, or &quot;V2&quot; key usage</strong> (one required). Only valid for TR-31 key block with key usage &quot;V0&quot;, &quot;V1&quot;, or &quot;V2&quot; and no control vector in optional block. When this keyword is specified, an optional CV key type modifier can be specified for key usage &quot;V0&quot; or &quot;V1&quot;.</td>
<td></td>
</tr>
<tr>
<td>PINGEN</td>
<td>Convert a TR-31 PIN verification key to a CCA PINGEN key.</td>
</tr>
<tr>
<td>PINVER</td>
<td>Convert a TR-31 PIN verification key to a CCA PINVER key.</td>
</tr>
<tr>
<td><strong>CV key usage for &quot;E0&quot; or &quot;E2&quot; key usage</strong> (one required) Only valid for TR-31 key block with key usage &quot;E0&quot; or &quot;E2&quot; and no control vector in optional block.</td>
<td></td>
</tr>
<tr>
<td>DMAC</td>
<td>Convert TR-31 EMV/chip issuer master key: application cryptograms or secure messaging for integrity to CCA DKYGENKY with key usage DMAC.</td>
</tr>
<tr>
<td>DMV</td>
<td>Convert TR-31 EMV/chip issuer master key: application cryptograms or secure messaging for integrity to CCA DKYGENKY with key usage DMV.</td>
</tr>
<tr>
<td><strong>CV key usage for &quot;E1&quot; key usage</strong> (one required) Only valid for TR-31 key block with key usage &quot;E1&quot; and no control vector in optional block.</td>
<td></td>
</tr>
<tr>
<td>DMPIN</td>
<td>Convert TR-31 EMV/chip issuer master key: secure messaging for confidentiality to CCA DKYGENKY with key usage DMPIN</td>
</tr>
<tr>
<td>DDATA</td>
<td>Convert TR-31 EMV/chip issuer master key: secure messaging for confidentiality to CCA DKYGENKY with key usage DDATA.</td>
</tr>
<tr>
<td><strong>CV key usage for &quot;E5&quot; key usage</strong> (one required) Only valid for TR-31 key block with key usage &quot;E5&quot; and no control vector in optional block.</td>
<td></td>
</tr>
<tr>
<td>DMAC</td>
<td>Convert TR-31 EMV/chip issuer master key: card personalization to CCA DKYGENKY with key usage DMAC.</td>
</tr>
<tr>
<td>DMV</td>
<td>Convert TR-31 EMV/chip issuer master key: card personalization to CCA DKYGENKY with key usage DMV.</td>
</tr>
<tr>
<td>DEXP</td>
<td>Convert TR-31 EMV/chip issuer master key: card personalization to CCA DKYGENKY with key usage DEXP.</td>
</tr>
<tr>
<td><strong>CV subtype for “E0”, “E1”, or “E2” key usage</strong> (one required). Only valid for TR-31 key block with key usage &quot;E0&quot;, &quot;E1&quot;, or &quot;E2&quot; and no control vector in optional block.</td>
<td></td>
</tr>
<tr>
<td>DKYL0</td>
<td>Convert TR-31 EMV/chip issuer master key: application cryptograms, secure message for confidentiality, or secure message for integrity to CCA DKYGENKY with subtype DKYL0.</td>
</tr>
<tr>
<td>DKYL1</td>
<td>Convert TR-31 EMV/chip issuer master key: application cryptograms, secure message for confidentiality, or secure message for integrity to CCA DKYGENKY with subtype DKYL1.</td>
</tr>
<tr>
<td><strong>CV key type modifier for &quot;V0&quot; or &quot;V1&quot; key usage</strong> (one required). Only valid for TR-31 key block with key usage &quot;V0&quot; or &quot;V1&quot; and no control vector in optional block.</td>
<td></td>
</tr>
<tr>
<td>NOOFFSET</td>
<td>Convert a TR-31 PIN verification key to a CCA PINGEN or PINVER key with the key type modifier NOOFFSET, so that the key cannot participate in a PIN offset process or PVV process.</td>
</tr>
<tr>
<td><strong>Key-wrapping method</strong> (one, optional)</td>
<td></td>
</tr>
<tr>
<td>USECONFIGN</td>
<td>This is the default. Specifies to wrap the key using the configuration setting for the default wrapping method. The default wrapping method configuration setting may be changed using the TKE. This keyword is ignored for AES keys.</td>
</tr>
<tr>
<td><strong>Note</strong>: Do not use this keyword if the default wrapping method is WRAP-ECB and a control vector is present in an optional block of the TR-31 key block with CV bit 56 = B1’ (ENH-ONLY). Use the WRAP-ENH keyword instead.</td>
<td></td>
</tr>
<tr>
<td>WRAP-ECB</td>
<td>Specifies to wrap the key using the legacy wrapping method.</td>
</tr>
<tr>
<td><strong>Note</strong>: Do not use this keyword if a control vector is present in an optional block of the TR-31 key block with CV bit 56 = B1’ (ENH-ONLY).</td>
<td></td>
</tr>
<tr>
<td>WRAP-ENH</td>
<td>Specifies to wrap the key using the enhanced wrapping method.</td>
</tr>
</tbody>
</table>

*Translation control* (optional). This keyword is valid only with key-wrapping method WRAP-ENH or with USECONFIGN when the default wrapping method is WRAP-ENH. This option cannot be used on a key with a control vector valued to binary zeros.
Keyword | Meaning
--- | ---
ENH-ONLY | Specifies to restrict the key from being wrapped with the legacy method once it has been wrapped with the enhanced method. Sets CV bit 56 = B'1' (ENH-ONLY).

Note: If a control vector is present in an optional block of the TR-31 key block with CV bit 56 = B'0', this keyword overrides that value in the CCA key-token. This keyword has no effect if the control vector in an optional block is all zeros.

Table 212 shows all valid translations for import of a TR-31 BDK base derivation key (usage "B0") to a CCA KEYGENKY key, along with any access control commands that must be enabled in the active role for that key type and control vector attributes. These keys are for translating derived unique key per transaction (DUKPT) base derivation keys.

Table 212. Import translation table for a TR-31 BDK base derivation key (usage “B0”)

<table>
<thead>
<tr>
<th>Key usage</th>
<th>Key block protection method keyword (version ID)</th>
<th>Mode of use</th>
<th>Rule-array keywords</th>
<th>CCA key type and control vector attributes</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;B0&quot;</td>
<td>&quot;A&quot;</td>
<td>&quot;N&quot;</td>
<td>N/A</td>
<td>KEYGENKY, double length, UKPT (CV bit 18 = B'1')</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>&quot;B0&quot;</td>
<td>&quot;B&quot; or &quot;C&quot;</td>
<td>&quot;X&quot;</td>
<td>N/A</td>
<td>KEYGENKY, double length, UKPT (CV bit 18 = B'1')</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Note:
1. These keys are the base keys from which derived unique key per transaction (DUKPT) initial keys are derived for individual devices such as PIN pads.
2. This table defines the only supported translations for this TR-31 usage. Usage must be the following value: "B0" BDK base derivation key.
3. There are no specific access-control commands for this translation because it is not ambiguous or in need of interpretation.

Table 213 shows all valid translations for import of a TR-31 CVK card verification key (usage "C0") to a CCA MAC or DATA key, along with any access control commands that must be enabled in the active role for that key type and control vector attributes. These keys are for computing or verifying (against supplied value) a card verification code with the CVV, CVC, CVC2, and CVV2 algorithm.

Table 213. Import translation table for a TR-31 CVK card verification key (usage "C0")

<table>
<thead>
<tr>
<th>Key usage</th>
<th>Key block protection method keyword (version ID)</th>
<th>Mode of use</th>
<th>Rule-array keywords</th>
<th>CCA key type and control vector attributes</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
</table>
Table 213. Import translation table for a TR-31 CVK card verification key (usage "C0") (continued)

<table>
<thead>
<tr>
<th>&quot;C0&quot;</th>
<th>&quot;A&quot;, &quot;B&quot;, or &quot;C&quot;</th>
<th>&quot;G&quot; or &quot;C&quot;</th>
<th>CVK-CSC</th>
<th>MAC, single or double length, AMEX-CSC (CV bits 0 - 3 = B'0100')</th>
<th>X'015B'</th>
<th>TR31 Import - Permit C0 to MAC/MACVER:AMEXCSC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVK-CVV</td>
<td>MAC, double length, CVVK-A (CV bits 0 - 3 = B'0010')</td>
<td>X'015A'</td>
<td>TR31 Import - Permit C0 to MAC/MACVER:CVVK-A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;V&quot;</td>
<td>CVK-CSC</td>
<td>MACVER, single or double length, AMEX-CSC (CV bits 0 - 3 = B'0100')</td>
<td>X'015B'</td>
<td>TR31 Import - Permit C0 to MAC/MACVER:AMEXCSC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVK-CVV</td>
<td>MACVER, double length, CVVK-A (CV bits 0 - 3 = B'0010')</td>
<td>X'015A'</td>
<td>TR31 Import - Permit C0 to MAC/MACVER:CVVK-A</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Security considerations:
1. There is an asymmetry in the translation from a CCA DATA key to a TR-31 key. The asymmetry results from CCA DATA keys having attributes of both data encryption keys and MAC keys, while TR-31 separates data encryption keys from MAC keys. A CCA DATA key can be exported to a TR-31 "C0" key, if one or both applicable MAC generate and MAC verify control vector bits are on. However, a TR-31 "C0" key cannot be imported to the lower-security CCA DATA key, it can be imported only to a CCA key type of MAC or MACVER. This restriction eliminates the ability to export a CCA MAC or MACVER key to a TR-31 key and re-importing it back as a CCA DATA key with the capability to Encipher, Decipher, or both.
2. The translation from TR-31 usage "C0" is controlled by rule array keywords when using the TR31_Key_Import verb. This makes it possible to convert an exported CCA CVVK-A key into an AMEX-CSC key or the other way around. To prevent such a conversion, do not enable offsets X'015A' (TR31 Import - Permit C0 to MAC/MACVER:CVVK-A) and X'015B' (TR31 Import - Permit C0 to MAC/MACVER:AMEXCSC) at the same time. However, if both CVVK-A and AMEX-CSC translation types are required, then offsets X'015A' and X'015B' must be enabled. In this case, control is up to the development, deployment, and execution of the applications themselves.

Note:
1. Card verification keys are used for computing or verifying (against supplied value) a card verification code with the CVV, CVC, CVC2, and CVV2 algorithms. In CCA, these keys correspond to keys used with two algorithms:
   - Visa CVV and MasterCard CVC codes are generated and verified using the CVV Generate and CVV Verify verbs. These verbs require a key type of DATA or MAC/MACVER with a subtype extension (CV bits 0 - 3) of ANY-MAC, single-length CVVK-A and single-length CVVK-B, and a double-length CVVK-A (see CVV Key Combine verb). The MAC generate and the MAC verify (CV bits 20 - 21) key usage values must be set appropriately.
   - American Express CSC codes are generated and verified using the Transaction Validation verb. This verb requires a key type of MAC or MACVER with a subtype extension of ANY-MAC or AMEX-CSC.
2. The translation from TR-31 usage "C0" to a CCA MAC/MACVER key with a subtype extension of ANY-MAC (CV bits 0 - 3 = B'0000') is not allowed.
3. This table defines the only supported translations for this TR-31 usage. Usage must be the following value:
   "C0"  CVK card verification key
4. CCA does not have an equivalent to the TR-31 "generate only" mode of use, so a translation from TR-31 mode "G" will result in a CCA MAC key with both MAC generate and MAC verify attributes (CV bits 20 - 21 = B'11'). Note that any key that can perform a generate operation can readily verify a MAC as well.
5. The CCA representation and the TR-31 representation of CVV keys are incompatible. CCA represents the CVVK-A and CVVK-B keys as two 8-byte (single length) keys, while TR-31 represents these keys as one 16-byte key. The CVV Generate and CVV Verify verbs have support added to accept one 16-byte CVV key, using left and right key parts as A and B. Current Visa standards require this.
6. Import and export of 8-byte CVVK-A and CVVK-B MAC/MACVER keys is allowed only using the proprietary TR-31 usage+mode values ("10" and "1", respectively) to indicate encapsulation of the IBM control vector in an optional block, because the 8-byte CVVK-A is meaningless and useless as a TR-31 "C0" usage key of any mode.

[Table 214 on page 789] shows all valid translations for import of a TR-31 data encryption key (usage "D0") to a CCA ENCIPHER, DECIPHER, CIPHER, or DATA key, along with any access control commands that must be enabled in
the active role for that key type and control vector attributes. These keys are used for the encryption and/or decryption of data.

### Table 214. Import translation table for a TR-31 data encryption key (usage "D0")

<table>
<thead>
<tr>
<th>Key usage</th>
<th>Key block protection method keyword (version ID)</th>
<th>Mode of use</th>
<th>Rule-array keywords</th>
<th>CCA key type and control vector attributes</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;D0&quot;</td>
<td>&quot;A&quot;, &quot;B&quot;, or &quot;C&quot;</td>
<td>&quot;E&quot;</td>
<td>N/A</td>
<td>ENCIPHER, single or double length</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;D&quot;</td>
<td>N/A</td>
<td>DECIPHER, single or double length</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;B&quot;</td>
<td>N/A</td>
<td>CIPHER, single or double length</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Security consideration:** There is asymmetry in the translation from a CCA DATA key to a TR-31 key. The asymmetry results from CCA DATA keys having attributes of both data encryption keys and MAC keys, while TR-31 separates data encryption keys from MAC keys. A CCA DATA key can be exported to a TR-31 "D0" key, if one or both applicable Encipher or Decipher control vector bits are on. However, a TR-31 "D0" key cannot be imported to the lower-security CCA DATA key; it can be imported only to a CCA key type of ENCIPHER, DECIPHER, or CIPHER. This restriction eliminates the ability to export a CCA DATA key to a TR-31 key, and re-importing it back as a CCA DATA key with the capability to MAC generate and MAC verify.

**Note:**
1. Data encryption keys are used for the encryption and decryption of data.
2. This table defines the only supported translations for this TR-31 usage. Usage must be the following value:
   - "D0" Data encryption
3. There are no specific access-control commands for this translation since it is not ambiguous or in need of interpretation.

---

### Table 215 shows all valid translations for import of a TR-31 key encryption or wrapping, or key block protection key (usages "K0", "K1") to a CCA EXPORT, OKEYXLAT, IMPORTER, or IKEYXLAT key, along with any access control commands that must be enabled in the active role for that key type and control vector attributes. These keys are used only to encrypt or decrypt other keys, or as a key used to derive keys that are used for that purpose.

### Table 215. Import translation table for a TR-31 key encryption or wrapping, or key block protection key (usages "K0", "K1")

<table>
<thead>
<tr>
<th>Key usage</th>
<th>Key block protection method keyword (version ID)</th>
<th>Mode of use</th>
<th>Rule-array keywords</th>
<th>CCA key type and control vector attributes</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;K0&quot;</td>
<td>&quot;A&quot;, &quot;B&quot;, or &quot;C&quot;</td>
<td>&quot;E&quot;</td>
<td>OKEYXLAT</td>
<td>OKEYXLAT, double length</td>
<td>X'015C'</td>
<td>TR31 Import - Permit K0:E to EXPORTER/OKEYXLAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EXPORTER</td>
<td>EXPORTER, double length, EXPORT on (CV bit 21 = B'1')</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IKEYXLAT</td>
<td>IKEYXLAT, double length</td>
<td>X'015D'</td>
<td>TR31 Import - Permit K0:D to IMPORTER/IKEYXLAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IMPORTER</td>
<td>IMPORTER, double length, IMPORT on (CV bit 21 = B'1')</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OKEYXLAT</td>
<td>OKEYXLAT, double length</td>
<td>X'015F'</td>
<td>TR31 Import - Permit K0:B to EXPORTER/OKEYXLAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EXPORTER</td>
<td>EXPORTER, double length, EXPORT on (CV bit 21 = B'1')</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IKEYXLAT</td>
<td>IKEYXLAT, double length</td>
<td>X'015F'</td>
<td>TR31 Import - Permit K0:B to IMPORTER/IKEYXLAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IMPORTER</td>
<td>IMPORTER, double length, IMPORT on (CV bit 21 = B'1')</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 215. Import translation table for a TR-31 key encryption or wrapping, or key block protection key (usages "K0", "K1") (continued)

<table>
<thead>
<tr>
<th>&quot;K1&quot;</th>
<th>&quot;B&quot; or &quot;C&quot;</th>
<th>&quot;E&quot;</th>
<th>OKEYXLAT</th>
<th>OKEYXLAT, double length</th>
<th>X'0160'</th>
<th>TR31 Import - Permit K1:E to EXPORTER/OKEYXLAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;D&quot;</td>
<td>IKEYXLAT</td>
<td>IKEYXLAT, double length</td>
<td>X'0161'</td>
<td>TR31 Import - Permit K1:D to IMPORTER/IKEYXLAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;B&quot;</td>
<td>OKEYXLAT</td>
<td>OKEYXLAT, double length</td>
<td>X'0162'</td>
<td>TR31 Import - Permit K1:B to EXPORTER/OKEYXLAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IMPORTER</td>
<td>IMPORTER, double length, IMPORT on (CV bit 21 = B'1')</td>
<td>X'0163'</td>
<td>TR31 Import - Permit K1:B to IMPORTER/IKEYXLAT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Security considerations:
1. The CCA OKEYXLAT, EXPORTER, IKEYXLAT, or IMPORTER KEK translation to a TR-31 "K0" key with mode "B" (both wrap and unwrap) is not allowed for security reasons. Even with access-control point control, this capability would give an immediate path to turn a CCA EXPORTER key into a CCA IMPORTER, and the other way around.
2. When a TR-31 key block does not have an included control vector as an optional block, the default control vector is used to construct the output key-token. Default CCA EXPORTER or IMPORTER keys have CV bits 18 - 20 on, which are used for key generation.

Note:
1. Key encryption or wrapping keys are used only to encrypt or decrypt other keys, or as a key used to derive keys that are used for that purpose.
2. This table defines the only supported translations for this TR-31 usage. Usage must be the following value:
   "K0" Key encryption or wrapping
   "K1" TR-31 key block protection key
3. Any attempt to import a TR-31 "K0" or "K1" key that has algorithm "D" (DEA) will result in an error because CCA does not support single-length KEKs.
4. CCA mode support is the same for version IDs "A", "B", and "C", because the distinction between TR-31 "K0" and "K1" does not exist in CCA keys. CCA does not distinguish between targeted protocols currently, and so there is no good way to represent the difference. Also note that most wrapping mechanisms now involve derivation or key variation steps.

Table 216 shows all valid translations for import of a TR-31 ISO MAC algorithm key (usages "M0", "M1", "M3") to a CCA MAC, MACVER, DATA, DATAM, or DATAMV key, along with any access control commands that must be enabled in the active role for that key type and control vector attributes. These keys are to compute or verify a code for message authentication.

Table 216. Import translation table for a TR-31 ISO MAC algorithm key (usages "M0", "M1", "M3")

<table>
<thead>
<tr>
<th>Key usage</th>
<th>Key block protection method keyword (version ID)</th>
<th>Mode of use</th>
<th>Rule-array keywords</th>
<th>CCA key type and control vector attributes</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
</table>
### Table 216. Import translation table for a TR-31 ISO MAC algorithm key (usages "M0", "M1", "M3") (continued)

<table>
<thead>
<tr>
<th>Usage</th>
<th>Field</th>
<th>Length</th>
<th>Access Control Attributes</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;M0&quot;</td>
<td>&quot;A&quot;, &quot;B&quot;, or &quot;C&quot;</td>
<td>MAC, double length, ANY-MAC (CV bits 0 - 3 = B'0000')</td>
<td>TR31 Import - Permit M0/M1/M3 to MAC/MACVER:ANY-MAC</td>
<td>&quot;V&quot;</td>
</tr>
<tr>
<td>&quot;M1&quot;</td>
<td>&quot;G&quot; or &quot;C&quot;</td>
<td>MACVER, double length, ANY-MAC (CV bits 0 - 3 = B'0000')</td>
<td></td>
<td>&quot;V&quot;</td>
</tr>
<tr>
<td>&quot;M3&quot;</td>
<td>&quot;G&quot; or &quot;C&quot;</td>
<td>MAC, single or double length, ANY-MAC (CV bits 0 - 3 = B'0000')</td>
<td></td>
<td>&quot;V&quot;</td>
</tr>
<tr>
<td>&quot;M0&quot;</td>
<td>&quot;A&quot;, &quot;B&quot;, or &quot;C&quot;</td>
<td>MAC, double length, ANY-MAC (CV bits 0 - 3 = B'0000')</td>
<td></td>
<td>&quot;V&quot;</td>
</tr>
<tr>
<td>&quot;M1&quot;</td>
<td>&quot;G&quot; or &quot;C&quot;</td>
<td>MACVER, single or double length, ANY-MAC (CV bits 0 - 3 = B'0000')</td>
<td></td>
<td>&quot;V&quot;</td>
</tr>
<tr>
<td>&quot;M3&quot;</td>
<td>&quot;G&quot; or &quot;C&quot;</td>
<td>MAC, single or double length, ANY-MAC (CV bits 0 - 3 = B'0000')</td>
<td></td>
<td>&quot;V&quot;</td>
</tr>
</tbody>
</table>

#### Security consideration:
There is asymmetry in the translation from a CCA DATA key to a TR-31 key. The asymmetry results from CCA DATA keys having attributes of both data encryption keys and MAC keys, while TR-31 separates data encryption keys from MAC keys. A CCA DATA key can be exported to a TR-31 "M0", "M1", or "M3" key, if one or both applicable MAC generate and MAC verify control vector bits are on. However, a TR-31 "M0", "M1", or "M3" key cannot be imported to the lower-security CCA DATA key, it can be imported only to a CCA key type of MAC or MACVER. This restriction eliminates the ability to export a CCA MAC or MACVER key to a TR-31 key, and re-importing it back as a CCA DATA key with the capability to Encipher, Decipher, or both.

#### Note:
1. MAC keys are used to compute or verify a code for message authentication.
2. This table defines the only supported translations for this TR-31 usage. Usage must be one of the following values:
   "M0" SO 16609 MAC algorithm 1, TDEA
   The ISO 16609 MAC algorithm 1 is based on ISO 9797. It is identical to "M1" except that it does not support 8-byte DES keys.
   "M1" SO 9797 MAC algorithm 1
   The ISO 9797 MAC algorithm 1 is identical to "M0" except that it also supports 8-byte DES keys.
   "M3" ISO 9797 MAC algorithm 3
   The X.9.19 style of Triple-DES MAC.
3. A CCA control vector has no bits defined to limit key usage by algorithm, such as CBC MAC (TR-31 usage "M0" and "M1") or X9.19 (TR-31 usage "M3"). When importing a TR-31 key block, the resulting CCA key token deviates from the restrictions of usages "M0", "M1", and "M3". Importing a TR-31 key block which allows MAC generation ("G" or "C") results in a control vector with the ANY-MAC attribute rather than for the restricted algorithm that is set in the TR-31 key block. The ANY-MAC attribute provides the same restrictions as what CCA currently uses for generating and verifying MACs.

---

**Table 217 on page 792** shows all valid translations for import of a TR-31 PIN encryption or PIN verification key (usages "P0", "V0", "V1", "V2") to a CCA OPINENC, IPINENC, PINGEN, or PINVER key, along with any access control commands that must be enabled in the active role for that key type and control vector attributes. These keys are used to protect PIN blocks and to generate or verify a PIN using a particular PIN-calculation method for that key type.
### Table 217. Import translation table for a TR-31 PIN encryption or PIN verification key (usages "P0", "V0", "V1", "V2")

<table>
<thead>
<tr>
<th>Key usage</th>
<th>Key block protection method keyword (version ID)</th>
<th>Mode of use</th>
<th>Rule-array keywords</th>
<th>CCA key type and control vector attributes</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;P0&quot;</td>
<td>&quot;A&quot;, &quot;B&quot;, or &quot;C&quot;</td>
<td>&quot;E&quot;</td>
<td>N/A</td>
<td>OPINENC, double length</td>
<td>X'0165'</td>
<td>TR31 Import - Permit P0:E to OPINENC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;D&quot;</td>
<td></td>
<td>IPINENC, double length</td>
<td>X'0166'</td>
<td>TR31 Import - Permit P0:D to IPINENC</td>
</tr>
<tr>
<td>&quot;V0&quot;</td>
<td>&quot;A&quot;</td>
<td>&quot;N&quot; (requires both commands)</td>
<td>PINGEN</td>
<td>PINGEN, double length, NO-SPEC (CV bits 0 - 3 = B'0000')</td>
<td>X'0167'</td>
<td>TR31 Import - Permit V0 to PINGEN:NO-SPEC</td>
</tr>
<tr>
<td></td>
<td>&quot;A&quot;, &quot;B&quot;, or &quot;C&quot;</td>
<td>&quot;G&quot; or &quot;C&quot;</td>
<td>NOOFFSET</td>
<td>NOOFFSET off (CV bit 37 = B'0')</td>
<td>X'0167'</td>
<td>TR31 Import - Permit V0 to PINGEN:NO-SPEC</td>
</tr>
<tr>
<td></td>
<td>&quot;A&quot;</td>
<td>&quot;N&quot; (requires both commands)</td>
<td>PINGEN, NOOFFSET</td>
<td>NOOFFSET on (CV bit 37 = B'1')</td>
<td>X'0167'</td>
<td>TR31 Import - Permit V0 to PINGEN:NO-SPEC</td>
</tr>
<tr>
<td></td>
<td>&quot;A&quot;, &quot;B&quot;, or &quot;C&quot;</td>
<td>&quot;G&quot; or &quot;C&quot;</td>
<td>NOOFFSET</td>
<td>NOOFFSET on (CV bit 37 = B'1')</td>
<td>X'0167'</td>
<td>TR31 Import - Permit V0 to PINGEN:NO-SPEC</td>
</tr>
<tr>
<td></td>
<td>&quot;A&quot;</td>
<td>&quot;N&quot; (requires both commands)</td>
<td>PINVER</td>
<td>PINVER, double length, NO-SPEC (CV bits 0 - 3 = B'0000')</td>
<td>X'0168'</td>
<td>TR31 Import - Permit V0 to PINVER:NO-SPEC</td>
</tr>
<tr>
<td></td>
<td>&quot;A&quot;, &quot;B&quot;, or &quot;C&quot;</td>
<td>&quot;V&quot;</td>
<td>NOOFFSET</td>
<td>NOOFFSET off (CV bit 37 = B'0')</td>
<td>X'0168'</td>
<td>TR31 Import - Permit V0 to PINVER:NO-SPEC</td>
</tr>
<tr>
<td></td>
<td>&quot;A&quot;</td>
<td>&quot;N&quot; (requires both commands)</td>
<td>PINVER, NOOFFSET</td>
<td>NOOFFSET on (CV bit 37 = B'1')</td>
<td>X'0168'</td>
<td>TR31 Import - Permit V0 to PINVER:NO-SPEC</td>
</tr>
<tr>
<td></td>
<td>&quot;A&quot;, &quot;B&quot;, or &quot;C&quot;</td>
<td>&quot;V&quot;</td>
<td>NOOFFSET</td>
<td>NOOFFSET on (CV bit 37 = B'1')</td>
<td>X'0168'</td>
<td>TR31 Import - Permit V0 to PINVER:NO-SPEC</td>
</tr>
</tbody>
</table>
Table 217. Import translation table for a TR-31 PIN encryption or PIN verification key (usages "P0", "V0", "V1", "V2") (continued)

<table>
<thead>
<tr>
<th>'V1'</th>
<th>&quot;A&quot;</th>
<th>&quot;N&quot; (requires both commands)</th>
<th>PINGEN</th>
<th>PINGEN, double length, IBM PIN/IBM-PINO (CV bits 0 - 3 = B'0001')</th>
<th>NOOFFSET off (CV bit 37 = B'0')</th>
<th>X'0169'</th>
<th>TR31 Import - Permit V1 to PINGEN:IBM-PIN/IBMPINO</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;A&quot;, &quot;B&quot;, or &quot;C&quot;</td>
<td>&quot;G&quot; or &quot;C&quot;</td>
<td></td>
<td></td>
<td>NOOFFSET off (CV bit 37 = B'0')</td>
<td>X'0169'</td>
<td>TR31 Import - Permit V0/V1/V2:N to PINGEN/PINVER</td>
<td></td>
</tr>
<tr>
<td>&quot;A&quot;</td>
<td>&quot;N&quot; (requires both commands)</td>
<td>PINGEN, NOOFFSET</td>
<td>NOOFFSET on (CV bit 37 = B'1')</td>
<td>X'0169'</td>
<td>TR31 Import - Permit V1 to PINGEN:IBM-PIN/IBMPINO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;A&quot;, &quot;B&quot;, or &quot;C&quot;</td>
<td>&quot;G&quot; or &quot;C&quot;</td>
<td></td>
<td>NOOFFSET on (CV bit 37 = B'1')</td>
<td>X'0169'</td>
<td>TR31 Import - Permit V0/V1/V2:N to PINGEN/PINVER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;A&quot;</td>
<td>&quot;N&quot; (requires both commands)</td>
<td>PINVER</td>
<td>PINVER, double length, IBM PIN/IBM-PINO (CV bits 0 - 3 = B'0001')</td>
<td>NOOFFSET off (CV bit 37 = B'0')</td>
<td>X'016A'</td>
<td>TR31 Import - Permit V1 to PINVER:IBM-PIN/IBMPINO</td>
<td></td>
</tr>
<tr>
<td>&quot;A&quot;, &quot;B&quot;, or &quot;C&quot;</td>
<td>&quot;V&quot;</td>
<td></td>
<td></td>
<td>NOOFFSET off (CV bit 37 = B'0')</td>
<td>X'016A'</td>
<td>TR31 Import - Permit V1 to PINVER:IBM-PIN/IBMPINOEC</td>
<td></td>
</tr>
<tr>
<td>&quot;A&quot;</td>
<td>&quot;N&quot; (requires both commands)</td>
<td>PINVER, NOOFFSET</td>
<td>NOOFFSET on (CV bit 37 = B'1')</td>
<td>X'016A'</td>
<td>TR31 Import - Permit V1 to PINVER:IBM-PIN/IBMPINOC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;A&quot;, &quot;B&quot;, or &quot;C&quot;</td>
<td>&quot;V&quot;</td>
<td></td>
<td></td>
<td>NOOFFSET on (CV bit 37 = B'1')</td>
<td>X'016A'</td>
<td>TR31 Import - Permit V0/V1/V2:N to PINGEN/PINVER</td>
<td></td>
</tr>
</tbody>
</table>
Table 217. Import translation table for a TR-31 PIN encryption or PIN verification key (usages "P0", "V0", "V1", "V2") (continued)

<table>
<thead>
<tr>
<th>&quot;V2&quot;</th>
<th>&quot;A&quot;</th>
<th>&quot;N&quot; (requires both commands)</th>
<th>PINGEN</th>
<th>PINGEN, double length, VISA-PVV (CV bits 0 - 3 = B'0010')</th>
<th>X'016B'</th>
<th>TR31 Import - Permit V2 to PINGEN:VISA-PVV</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>&quot;A&quot;, &quot;B&quot;, or &quot;C&quot;</td>
<td>&quot;G&quot; or &quot;C&quot;</td>
<td>PINGEN, double length, VISA-PVV (CV bits 0 - 3 = B'0010')</td>
<td>X'016B'</td>
<td>TR31 Import - Permit V2 to PINGEN:VISA-PVV</td>
</tr>
<tr>
<td>&quot;A&quot;</td>
<td>&quot;N&quot; (requires both commands)</td>
<td>PINVER</td>
<td>PINVER, double length, VISA-PVV (CV bits 0 - 3 = B'0010')</td>
<td>X'016C'</td>
<td>TR31 Import - Permit V2 to PINVER:VISA-PVV</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;A&quot;, &quot;B&quot;, or &quot;C&quot;</td>
<td>&quot;V&quot;</td>
<td>PINVER, double length, VISA-PVV (CV bits 0 - 3 = B'0010')</td>
<td>X'016C'</td>
<td>TR31 Import - Permit V2 to PINVER:VISA-PVV</td>
<td></td>
</tr>
</tbody>
</table>

Security note: TR-31 key blocks that are protected under legacy version ID "A" (using the Key Variant Binding Method 2005 Edition) use the same mode of use "N" for PINGEN and PINVER keys. For version ID "A" keys only, for a given PIN key usage, enabling both the PINGEN and PINVER access-control points at the same time while enabling offset X'017C' (for mode "N", no special restrictions) is NOT recommended. In other words, for a particular PIN verification key usage, you should not simultaneously enable the four commands shown below for that usage:

Table 218. Commands

<table>
<thead>
<tr>
<th>Key type, mode, or version</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;V0&quot;: For usage V0, a user with the following four commands enabled in the active role can change a PINVER key into a PINGEN key and the other way around. Avoid simultaneously enabling these four commands.</td>
<td>X'0167'</td>
<td>TR31 Import - Permit V0 to PINGEN:NO-SPEC</td>
</tr>
<tr>
<td>Key type PINGEN</td>
<td>X'0168'</td>
<td>TR31 Import - Permit V0 to PINGEN:NO-SPEC</td>
</tr>
<tr>
<td>Mode &quot;N&quot;</td>
<td>X'017C'</td>
<td>TR31 Import - Permit V0/V1/V2:N to PINGEN/PINVER</td>
</tr>
<tr>
<td>Version &quot;A&quot;</td>
<td>X'0150'</td>
<td>TR31 Import - Permit Version A TR-31 Key Blocks</td>
</tr>
<tr>
<td>&quot;V1&quot;: For usage V1, a user with the following four commands enabled in the active role can change a PINVER key into a PINGEN key and the other way around. Avoid simultaneously enabling these four commands.</td>
<td>X'0169'</td>
<td>TR31 Import - Permit V1 to PINGEN:IBM-PIN/IBMPINO</td>
</tr>
<tr>
<td>Key type PINGEN</td>
<td>X'016A'</td>
<td>TR31 Import - Permit V1 to PINGEN:IBM-PIN/IBMPINO</td>
</tr>
<tr>
<td>Mode &quot;N&quot;</td>
<td>X'017C'</td>
<td>TR31 Import - Permit V0/V1/V2:N to PINGEN/PINVER</td>
</tr>
<tr>
<td>Version &quot;A&quot;</td>
<td>X'0150'</td>
<td>TR31 Import - Permit Version A TR-31 Key Blocks</td>
</tr>
<tr>
<td>&quot;V2&quot;: For usage V2, a user with the following four commands enabled in the active role can change a PINVER key into a PINGEN key and the other way around. Avoid simultaneously enabling these four commands.</td>
<td>X'016B'</td>
<td>TR31 Import - Permit V2 to PINGEN:VISA-PVV</td>
</tr>
</tbody>
</table>

794  Common Cryptographic Architecture Application Programmer's Guide
Table 218. Commands (continued)

<table>
<thead>
<tr>
<th>Key type, mode, or version</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key type PINVER</td>
<td>X'016C'</td>
<td>TR31 Import - Permit V2 to PINVER:VISA-PVV</td>
</tr>
<tr>
<td>Mode &quot;N&quot;</td>
<td>X'017C'</td>
<td>TR31 Import - Permit V0/V1/V2:N to PINGEN/PINVER</td>
</tr>
<tr>
<td>Version &quot;A&quot;</td>
<td>X'0150'</td>
<td>TR31 Import - Permit Version A TR-31 Key Blocks</td>
</tr>
</tbody>
</table>

Failure to comply with this recommendation allows changing PINVER keys into PINGEN and the other way around.

Note:
1. PIN encryption keys are used to protect PIN blocks. PIN verification keys are used to generate or verify a PIN using a particular PIN-calculation method for that key type.
2. This table defines the only supported translations for this TR-31 usage. Usage must be one of the following values:
   - "P0"  PIN encryption
   - "V0"  PIN verification, KPV, other algorithm
     Usage "V0" does not have its own PIN-calculation method defined. The mapping to NO-SPEC is sub-optimal. Exporting to "N" mode restricts keys from being imported with the IBM-PIN/IBM-PINO or VISA-PVV attribute, while CCA NO-SPEC allows any method.
   - "V1"  PIN verification, IBM 3624
   - "V2"  PIN verification, Visa PVV
     The NOOFFSET keyword is not allowed for the Visa PVV algorithm because it does not support this attribute.
3. Mode must be one of the following values:
   - "E"  Encrypt/wrap only
     This mode restricts PIN encryption keys to encrypting a PIN block. May be used to create or reencipher an encrypted PIN block (for key-to-key translation).
   - "D"  Decrypt/unwrap only
     This mode restricts PIN encryption keys to decrypting a PIN block. Generally used in a PIN translation to decrypt the incoming PIN block.
   - "N"  No special restrictions (other than restrictions implied by the key usage)
     This mode is used by several vendors for a PIN generate or PIN verification key when the key block version ID is "A".
   - "G"  Generate only
     This mode is used for a PINGEN key that may not perform a PIN verification. The control vector will not have its EPINVER attribute on (CV bit 22 = B'0').
   - "V"  Verify only
     This mode is used for PIN verification only. If the TR-31 key block does not have a control vector included, the only usage bits set on in the control vector is the EPINVER bit (CV bits 18 - 22 = B'00001').
   - "C"  Both generate and verify (combined)
This mode indicates that the control vector will have the default PINGEN bits on (CV bits 18 -22 = B'11111').

4. Any attempt to import a TR-31 "P0" key that has mode "B" (both encrypt and decrypt) will result in an error because CCA does not support this combination of attributes.

5. If the TR-31 key block contains a control vector, and the control vector has NOOFFSET on, the NOOFFSET keyword is not necessary because the verb will automatically set NOOFFSET on in this case.

Table 219 shows all valid translations for import of a TR-31 EMV/chip issuer master-key key (usages "E0", "E1", "E2", "E3", "E4", "E5") to a CCA DKYGENKY, DATA, MAC, CIPHER, or ENCIPHER key, along with any access control commands that must be enabled in the active role for that key type and control vector attributes. These keys are used by the chip cards to perform cryptographic operations or, in some cases, to derive keys used to perform operations.

Table 219. Import translation table for a TR-31 EMV/chip issuer master-key key (usages "E0", "E1", "E2", "E3", "E4", "E5")

<table>
<thead>
<tr>
<th>Key usage</th>
<th>Key block protection method keyword (version ID)</th>
<th>Mode of use</th>
<th>Rule-array keywords</th>
<th>CCA key type and control vector attributes</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;E0&quot;</td>
<td>&quot;A&quot;</td>
<td>&quot;N&quot;</td>
<td>DKYL0, DMAC</td>
<td>DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMAC (CV bits 19 - 22 = B'0010')</td>
<td>X'016D'</td>
<td>TR31 Import - Permit E0 to DKYGENKY:DKYL0+DMAC</td>
</tr>
<tr>
<td></td>
<td>&quot;B&quot; or &quot;C&quot;</td>
<td>&quot;X&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;A&quot;</td>
<td>&quot;N&quot;</td>
<td>DKYL0, DMV</td>
<td>DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMV (CV bits 19 - 22 = B'0011')</td>
<td>X'016E'</td>
<td>TR31 Import - Permit E0 to DKYGENKY:DKYL0+DMV</td>
</tr>
<tr>
<td></td>
<td>&quot;B&quot; or &quot;C&quot;</td>
<td>&quot;X&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;A&quot;</td>
<td>&quot;N&quot;</td>
<td>DKYL1, DMAC</td>
<td>DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DMAC (CV bits 19 - 22 = B'0010')</td>
<td>X'016F'</td>
<td>TR31 Import - Permit E0 to DKYGENKY:DKYL1+DMAC</td>
</tr>
<tr>
<td></td>
<td>&quot;B&quot; or &quot;C&quot;</td>
<td>&quot;X&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;A&quot;</td>
<td>&quot;N&quot;</td>
<td>DKYL1, DMV</td>
<td>DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DMV (CV bits 19 - 22 = B'0011')</td>
<td>X'0170'</td>
<td>TR31 Import - Permit E0 to DKYGENKY:DKYL1+DMV</td>
</tr>
<tr>
<td>&quot;E1&quot;</td>
<td>&quot;A&quot;</td>
<td>&quot;N&quot;</td>
<td>&quot;E&quot;</td>
<td>&quot;D&quot;</td>
<td>&quot;B&quot;</td>
<td>&quot;B&quot; or &quot;C&quot;</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-------------</td>
</tr>
<tr>
<td>&quot;A&quot;</td>
<td>&quot;N&quot;</td>
<td>&quot;E&quot;</td>
<td>&quot;D&quot;</td>
<td>&quot;B&quot;</td>
<td>&quot;B&quot; or &quot;C&quot;</td>
<td>&quot;X&quot;</td>
</tr>
<tr>
<td>&quot;B&quot; or &quot;C&quot;</td>
<td>&quot;X&quot;</td>
<td>DKYL1, DMPIN</td>
<td>DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DMPIN (CV bits 19 - 22 = B'1001')</td>
<td>X'0174'</td>
<td>TR31 Import - Permit E1 to DKYGENKY:DKYL1+DDATA</td>
<td></td>
</tr>
<tr>
<td>&quot;B&quot; or &quot;C&quot;</td>
<td>&quot;X&quot;</td>
<td>DKYL1, DDATA</td>
<td>DKYGENKY, double length, DKYL1 (CV bits 12 - 14 = B'001'), DDATA (CV bits 19 - 22 = B'0001')</td>
<td>X'0175'</td>
<td>TR31 Import - Permit E2 to DKYGENKY:DKYL0+DMAC</td>
<td></td>
</tr>
<tr>
<td>&quot;A&quot;</td>
<td>&quot;N&quot;</td>
<td>&quot;E&quot;</td>
<td>&quot;D&quot;</td>
<td>&quot;B&quot;</td>
<td>&quot;B&quot; or &quot;C&quot;</td>
<td>&quot;X&quot;</td>
</tr>
<tr>
<td>&quot;B&quot; or &quot;C&quot;</td>
<td>&quot;X&quot;</td>
<td>DKYL0, DMAC</td>
<td>DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMAC (CV bits 19 - 22 = B'0001')</td>
<td>X'0177'</td>
<td>TR31 Import - Permit E3 to ENCIPHER</td>
<td></td>
</tr>
<tr>
<td>&quot;A&quot;</td>
<td>&quot;N&quot;</td>
<td>&quot;E&quot;</td>
<td>&quot;D&quot;</td>
<td>&quot;B&quot;</td>
<td>&quot;B&quot; or &quot;C&quot;</td>
<td>&quot;X&quot;</td>
</tr>
</tbody>
</table>
### Table 219. Import translation table for a TR-31 EMV/chip issuer master-key key (usages "E0", "E1", "E2", "E3", "E4", "E5") (continued)

<table>
<thead>
<tr>
<th>E5</th>
<th>A</th>
<th>C</th>
<th>V</th>
<th>E</th>
<th>D</th>
<th>B</th>
<th>N</th>
<th>DKYL0, DMAC</th>
<th>DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DMAC (CV bits 19 - 22 = B'0010')</th>
<th>X'0179'</th>
<th>TR31 Import - Permit E5 to DKYGENKY:DKYL0+DMAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;B&quot; or &quot;C&quot;</td>
<td>&quot;X&quot;</td>
<td>&quot;B&quot; or &quot;C&quot;</td>
<td>&quot;X&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E5</th>
<th>A</th>
<th>C</th>
<th>V</th>
<th>E</th>
<th>D</th>
<th>B</th>
<th>N</th>
<th>DKYL0, DDATA</th>
<th>DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DDATA (CV bits 19 - 22 = B'0001')</th>
<th>X'017A'</th>
<th>TR31 Import - Permit E5 to DKYGENKY:DKYL0+DDATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;B&quot; or &quot;C&quot;</td>
<td>&quot;X&quot;</td>
<td>&quot;B&quot; or &quot;C&quot;</td>
<td>&quot;X&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E5</th>
<th>A</th>
<th>C</th>
<th>V</th>
<th>E</th>
<th>D</th>
<th>B</th>
<th>N</th>
<th>DKYL0, DEXP</th>
<th>DKYGENKY, double length, DKYL0 (CV bits 12 - 14 = B'000'), DEXP (CV bits 19 - 22 = B'0101')</th>
<th>X'017B'</th>
<th>TR31 Import - Permit E5 to DKYGENKY:DKYL0+DEXP</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;B&quot; or &quot;C&quot;</td>
<td>&quot;X&quot;</td>
<td>&quot;B&quot; or &quot;C&quot;</td>
<td>&quot;X&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:**

1. EMV/chip issuer master-key keys are used by the chip cards to perform cryptographic operations or, in some cases, to derive keys used to perform operations. In CCA, these keys are (a) diversified key-generating keys (key type DKYGENKY), allowing derivation of operational keys, or (b) operational keys. Note that in this context, "master key" has a different meaning than for CCA. These master keys, also called KMCs, are described by EMV as DES master keys for personalization session keys. They are used to derive the corresponding chip card master keys, and not typically used directly for cryptographic operations other than key derivation. In CCA, these keys are usually key generating keys with derivation level DKYL1 (CV bits 12 - 14 = B'001'), used to derive other key generating keys (the chip card master keys). For some cases, or for older EMV key derivation methods, the issuer master keys could be level DKYL0 (CV bits 12 - 14 = B'000').

2. This table defines the only supported translations for this TR-31 usage. Usage must be one of the following values:
   - "E0" Application cryptograms
   - "E1" Secure messaging for confidentiality
   - "E2" Secure messaging for integrity
   - "E3" Data authentication code
   - "E4" Dynamic numbers
   - "E5" Card personalization

3. EMV support in CCA is different than TR-31 support, and CCA key types do not match TR-31 types.

4. DKYGENKY keys are double length only.

5. In CCA, a MAC key that can perform a MAC generate operation also can perform a MAC verify. For TR-31 mode "G" (generate only), the translation to a CCA key results in a key that can perform MAC generate and MAC verify.
**tr31_key_block_length**

**Direction:** Input  
**Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the `tr31_key_block` variable. The length field in the TR-31 block is a 4-digit decimal number, so the maximum acceptable length is 9992 bytes. For more information, see "TR31 Key Token Parse (CSNBT31P)" on page 804.

**tr31_key_block**

**Direction:** Input  
**Type:** String

A pointer to a string variable containing the TR-31 key block that is to be imported. The key block is protected with the key identified by the `unwrap_kek_identifier` parameter.

**unwrap_kek_identifier_length**

**Direction:** Input  
**Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the `unwrap_kek_identifier` variable. Set this value to 64.

**unwrap_kek_identifier**

**Direction:** Input  
**Type:** String

A pointer to a string variable containing the operational fixed-length DES key-token used to unwrap the key identified by the `tr31_key_block` parameter, or a key label of such a key in DES key-storage. The key must have a key type of IMPORTER or IKEYXLAT, and be authorized for import.

**Note:** DES keys wrapped in ECB mode (CCA legacy wrap mode) cannot be used to wrap or unwrap TR-31 "B" or "C" key blocks that have or will have "E" exportability, because ECB mode does not comply with ANSI X9.24 Part 1.

**wrap_kek_identifier_length**

**Direction:** Input  
**Type:** Integer

A pointer to an integer variable containing the length in bytes of the `wrap_kek_identifier` variable. The value must be greater than or equal to 0. A null key-token can have a length of 1. Set this value to 64 for a key label or a KEK.

**wrap_kek_identifier**

**Direction:** Input  
**Type:** String

A pointer to a string variable containing the operational fixed-length DES key-token used to wrap the key identified by the `output_key_identifier` parameter, a null key-token, or a key label of such a key in DES key-storage. If the parameter identifies a null key-token, then the `unwrap_kek_identifier` parameter is also used for wrapping the CCA output key token.
TR31 Key Import (CSNBT31I)

**output_key_identifier_length**

- **Direction:** Input/Output
- **Type:** Integer

A pointer to an integer specifying the length in bytes of the `output_key_identifier` variable. This is an input/output parameter.

**output_key_identifier**

- **Direction:** Input/Output
- **Type:** String

A pointer to a string variable containing the key token or the key label for the token that is to receive the imported key. The output key-token is a CCA internal or external key token containing the key received in the TR-31 token. If a key token is provided, it must be a null token (64 bytes of X'00'). If a key label is provided, the imported token is stored in the key storage file and identified by that label.

**num_opt_blocks**

- **Direction:** Output
- **Type:** Integer

A pointer to an integer variable where the verb stores the number of optional blocks that are present in the TR-31 key token.

**cv_source**

- **Direction:** Output
- **Type:** Integer

A pointer to an integer variable where the verb stores a value indicating how the control vector in the output key token was created. It can be one of the values in Table 220.

<table>
<thead>
<tr>
<th>CSNBT31I CV source</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No CV was present in an optional block, and the output CV was created by the verb based on input parameters and on the attributes in the TR-31 key block header.</td>
</tr>
<tr>
<td>1</td>
<td>A CV was obtained from an optional block in the TR-31 key block, and the key usage and mode of use were also specified in the TR-31 header. The verb verified compatibility of the header values with the CV and then used that CV in the output key token.</td>
</tr>
<tr>
<td>2</td>
<td>A CV was obtained from an optional block in the TR-31 key block, and the key usage and mode of use in the TR-31 header held the proprietary values indicating that key use and mode should be obtained from the included CV. The CV from the TR-31 token was used as the CV for the output key token.</td>
</tr>
</tbody>
</table>

Any values other than these three are reserved and are currently invalid.

**protection_method**

- **Direction:** Output
- **Type:** Integer

A pointer to an integer variable where the verb stores a value indicating what method was used to protect the input TR-31 key block. The TR-31 standard
allows two methods, and the application program might want to know which was used for security purposes. The variable can have one of the values in Table 221.

### Table 221. TR31 Key Import protection methods

<table>
<thead>
<tr>
<th>CSNBT31I protection method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The TR-31 key block was protected using the variant method as identified by a Key Block Version ID value of “A” (X’41’).</td>
</tr>
<tr>
<td>1</td>
<td>The TR-31 key block was protected using the derived key method as identified by a Key Block Version ID value of ”B” (X’42’).</td>
</tr>
<tr>
<td>2</td>
<td>The TR-31 key block was protected using the variant method as identified by a Key Block Version ID value of ”C” (X’43’). Functionally this method is the same as ”A”, but to maintain consistency a different value is returned here for ”C”.</td>
</tr>
</tbody>
</table>

Any values other than these three are reserved and are currently invalid.

### Restrictions

The restrictions for CSNBT31I.

None.

### Required commands

The required commands for CSNBT31I.

The TR31 Key Import verb requires the following commands to be enabled in the active role:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRAP-ENH or WRAP-ECB</td>
<td>X’0153’</td>
<td>TR31 Import - Permit Override of default wrapping method</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TR-31 key block version ID</th>
<th>Offset</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>”A” (X’41’)</td>
<td>X’0150’</td>
<td>TR31 Import - Permit version A TR-31 key blocks</td>
</tr>
<tr>
<td>”B” (X’42’)</td>
<td>X’0151’</td>
<td>TR31 Import - Permit version B TR-31 key blocks</td>
</tr>
<tr>
<td>”C” (X’43’)</td>
<td>X’0152’</td>
<td>TR31 Import - Permit version C TR-31 key blocks</td>
</tr>
</tbody>
</table>

Be aware of access-control point X’017C’ (TR31 Import - Permit V0/V1/V2:N to PINGEN/PINVER) for import of PINGEN or PINVER keys to wrapping method “A”, usage ”V0”, ”V1”, or ”V2”, and mode ”N”. TR-31 key blocks with legacy key usage ”A” (key block protected using the Key Variant Binding Method 2005 Edition) use the same mode ”N” for PINGEN as well as PINVER keys. For usage ”A” keys only, enabling a PINGEN and PINVER access-control point while enabling offset X’017C’ (for mode ”N”) is NOT recommended. Failure to comply with this recommendation allows changing PINVER keys into PINGEN, and the other way around.
In addition to the required commands, the verb needs these additional commands to be enabled in the active role depending on the rule-array keyword provided:

<table>
<thead>
<tr>
<th>Rule-array keyword</th>
<th>Offset</th>
<th>Command</th>
<th>Specific key usage, version ID, and mode values</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;C0&quot;: TR-31 CVK card verification keys</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVK-CSC</td>
<td>X'015B'</td>
<td>TR31 Import - Permit C0 to MAC/MACVER:AMEX-CSC</td>
<td>See Table 213 on page 787</td>
</tr>
<tr>
<td>CVK-CVV</td>
<td>X'015A'</td>
<td>TR31 Import - Permit C0 to MAC/MACVER:CVKEY-A</td>
<td></td>
</tr>
<tr>
<td>&quot;K0&quot; and &quot;K1&quot;: TR-31 key encryption or wrapping, or key block protection keys</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OKEYXLAT or EXPORTER</td>
<td>X'015C'</td>
<td>TR31 Import - Permit K0:E to EXPORTER/ OKEYXLAT</td>
<td>See Table 215 on page 789</td>
</tr>
<tr>
<td></td>
<td>X'015E'</td>
<td>TR31 Import - Permit K0:B to EXPORTER/OKEYXLAT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0160'</td>
<td>TR31 Import - Permit K1:E to EXPORTER/OKEYXLAT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0162'</td>
<td>TR31 Import - Permit K1:B to EXPORTER/OKEYXLAT</td>
<td></td>
</tr>
<tr>
<td>IKEYXLAT or IMPORTER</td>
<td>X'015D'</td>
<td>TR31 Import - Permit K0:D to IMPORTER/IKEYXLAT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'015F'</td>
<td>TR31 Import - Permit K0:B to IMPORTER/IKEYXLAT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0161'</td>
<td>TR31 Import - Permit K1:D to IMPORTER/IKEYXLAT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0163'</td>
<td>TR31 Import - Permit K1:B to IMPORTER/IKEYXLAT</td>
<td></td>
</tr>
<tr>
<td>&quot;M0&quot;, &quot;M1&quot;, and &quot;M3&quot;: TR-31 ISO MAC algorithm keys</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A</td>
<td>X'0164'</td>
<td>TR31 Import - Permit M0/M1/M3 to MAC/MACVER:ANY-MAC</td>
<td>See Table 216 on page 790</td>
</tr>
<tr>
<td></td>
<td>X'018C'</td>
<td>TR31 Export - Permit MACVER/DATAMV to M0:V</td>
<td></td>
</tr>
<tr>
<td>&quot;P0&quot;, &quot;V0&quot;, &quot;V1&quot;: TR-31 PIN encryption or PIN verification keys</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A</td>
<td>X'0165'</td>
<td>TR31 Import - Permit P0:E to OPINENC</td>
<td>See Table 217 on page 792</td>
</tr>
<tr>
<td></td>
<td>X'0166'</td>
<td>TR31 Import - Permit P0:D to IPINENC</td>
<td></td>
</tr>
<tr>
<td>PINGEN</td>
<td>X'0167'</td>
<td>TR31 Import - Permit V0 to PINGEN:NO-SPEC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0169'</td>
<td>TR31 Import - Permit V1 to PINGEN:IBM-PIN/IBMPINO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'016B'</td>
<td>TR31 Import - Permit V2 to PINGEN:VISA-PVV</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'017C'</td>
<td>TR31 Import - Permit V0/V1/V2:N to PINGEN/PINVER</td>
<td></td>
</tr>
<tr>
<td>PINVER</td>
<td>X'0168'</td>
<td>TR31 Import - Permit V0 to PINVER:NO-SPEC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'016A'</td>
<td>TR31 Import - Permit V1 to PINVER:IBM-PIN/IBMPINO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'016C'</td>
<td>TR31 Import - Permit V2 to PINVER:VISA-PVV</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'017C'</td>
<td>TR31 Import - Permit V0/V1/V2:N to PINGEN/PINVER</td>
<td></td>
</tr>
<tr>
<td>&quot;E0&quot;, &quot;E1&quot;, &quot;E2&quot;, &quot;E3&quot;, &quot;E4&quot;, and &quot;E5&quot;: TR-31 EMC/chip issuer master-key keys</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DKYL0</td>
<td>X'016D'</td>
<td>TR31 Import - Permit E0 to DKYGENKY:DKYL0+DMAC</td>
<td>See Table 219 on page 796</td>
</tr>
<tr>
<td></td>
<td>X'016E'</td>
<td>TR31 Import - Permit E0 to DKYGENKY:DKYL0+DMV</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0171'</td>
<td>TR31 Import - Permit E1 to DKYGENKY:DKYL0+DMPIN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0172'</td>
<td>TR31 Import - Permit E1 to DKYGENKY:DKYL0+DDATA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0175'</td>
<td>TR31 Import - Permit E2 to DKYGENKY:DKYL0+DMAC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0179'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DALL to E0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'017A'</td>
<td>TR31 Import - Permit E5 to DKYGENKY:DKYL0+ODATA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'017B'</td>
<td>TR31 Import - Permit E5 to DKYGENKY:DKYL0+DEXP</td>
<td></td>
</tr>
<tr>
<td>Rule-array keyword</td>
<td>Offset</td>
<td>Command</td>
<td>Specific key usage, version ID, and mode values</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------</td>
<td>----------------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>DKYL1</td>
<td>X'016F'</td>
<td>TR31 Import - Permit E0 to DKYGENKY:DKYL1+DMAC</td>
<td>See Table 219 on page 796</td>
</tr>
<tr>
<td></td>
<td>X'0170'</td>
<td>TR31 Import - Permit E0 to DKYGENKY:DKYL1+DMV</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0173'</td>
<td>TR31 Import - Permit E1 to DKYGENKY:DKYL1+DMPIN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0174'</td>
<td>TR31 Import - Permit E1 to DKYGENKY:DKYL1+DDATA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0176'</td>
<td>TR31 Import - Permit E2 to DKYGENKY:DKYL1+DMAC</td>
<td></td>
</tr>
<tr>
<td>X'016D'</td>
<td>DMAC</td>
<td>TR31 Import - Permit E0 to DKYGENKY:DKYL0+DMAC</td>
<td>See Table 219 on page 796</td>
</tr>
<tr>
<td></td>
<td>X'016F'</td>
<td>TR31 Import - Permit E0 to DKYGENKY:DKYL1+DMAC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0175'</td>
<td>TR31 Import - Permit E2 to DKYGENKY:DKYL0+DMAC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0176'</td>
<td>TR31 Import - Permit E2 to DKYGENKY:DKYL1+DMAC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0179'</td>
<td>TR31 Import - Permit E5 to DKYGENKY:DKYL0+DMAC</td>
<td></td>
</tr>
<tr>
<td>X'016E'</td>
<td>DMV</td>
<td>TR31 Import - Permit E0 to DKYGENKY:DKYL0+DMV</td>
<td>See Table 219 on page 796</td>
</tr>
<tr>
<td></td>
<td>X'0170'</td>
<td>TR31 Import - Permit E0 to DKYGENKY:DKYL1+DMV</td>
<td></td>
</tr>
<tr>
<td>X'0171'</td>
<td>DMPIN</td>
<td>TR31 Import - Permit E1 to DKYGENKY:DKYL0+DMPIN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0173'</td>
<td>TR31 Import - Permit E1 to DKYGENKY:DKYL1+DMPIN</td>
<td></td>
</tr>
<tr>
<td>X'0172'</td>
<td>DDATA</td>
<td>TR31 Import - Permit E1 to DKYGENKY:DKYL0+DDATA</td>
<td>See Table 219 on page 796</td>
</tr>
<tr>
<td></td>
<td>X'0174'</td>
<td>TR31 Import - Permit E1 to DKYGENKY:DKYL1+DDATA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'017A'</td>
<td>TR31 Import - Permit E5 to DKYGENKY:DKYL0+DDATA</td>
<td></td>
</tr>
<tr>
<td>X'017B'</td>
<td>DEXP</td>
<td>TR31 Import - Permit E5 to DKYGENKY:DKYL0+DEXP</td>
<td>See Table 219 on page 796</td>
</tr>
<tr>
<td>N/A</td>
<td>X'0177'</td>
<td>TR31 Import - Permit E3 to ENCIPHER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X'0178'</td>
<td>TR31 Import - Permit E4 to DKYGENKY:DKYL0+DDATA</td>
<td></td>
</tr>
</tbody>
</table>

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBT31IJ.

See "Building Java applications using the CCA JNI" on page 28.

**Format**

```java
public native void CSNBT31IJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber tr31_key_block_length,
    byte[] tr31_key_block,
    hikmNativeNumber unwrap_kek_identifier_length,
    byte[] unwrap_kek_identifier,
    hikmNativeNumber wrap_kek_identifier_length,
    byte[] wrap_kek_identifier,
    hikmNativeNumber output_key_identifier_length,
    byte[] output_key_identifier,
    hikmNativeNumber num_opt_blocks,
    hikmNativeNumber cv_source,
    hikmNativeNumber protection_method);
```
Use the TR31 Key Token Parse verb to disassemble the unencrypted header of an external TR-31 key block into separate pieces of information.

The part of the header that is optional, called optional blocks, is not disassembled. To obtain the contents of optional blocks, use the TR31 Optional Data Read verb. Neither verb performs any cryptographic services, and both disassemble a key block in application storage. The validity of the key block is verified as much as can be done without performing any cryptographic services.

The TR-31 header fields that are disassembled into separate pieces of information include a key block version ID, key block length, key usage, algorithm, mode of use, key version number, exportability, and number of optional blocks. Except for the two length values, which are returned as integers, the verb returns the field values as ASCII strings. This format is used in the TR-31 key block itself. For more information, see X9 TR-31 2010: Interoperable Secure Key Exchange Block Specification for Symmetric Algorithms.

The following table summarizes the key blocks fields returned by this verb:

<table>
<thead>
<tr>
<th>TR-31 field name</th>
<th>Verb parameter</th>
<th>Field or buffer string length in bytes</th>
<th>Description of TR-31 field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key block version ID</td>
<td>key_block_version</td>
<td>1</td>
<td>Identifies the method by which the key block is cryptographically protected and the content layout of the block.</td>
</tr>
<tr>
<td>Key block length</td>
<td>key_block_length</td>
<td>4 (integer)</td>
<td>Entire key block length after encoding (header, encrypted confidential data, and MAC).</td>
</tr>
<tr>
<td>Key usage</td>
<td>key_usage</td>
<td>2</td>
<td>Provides information about the intended function of the protected key/sensitive data, such as data encryption, PIN encryption, or key wrapping. Numeric values are reserved for proprietary use (that is, not defined by TR-31).</td>
</tr>
<tr>
<td>Algorithm</td>
<td>algorithm</td>
<td>1</td>
<td>The approved symmetric algorithm for which the protected key may be used. Numeric values are reserved for proprietary use.</td>
</tr>
<tr>
<td>Mode of use</td>
<td>mode</td>
<td>1</td>
<td>Defines the operation for which the protected key can perform. Numeric values are reserved for proprietary use.</td>
</tr>
<tr>
<td>Key version number</td>
<td>key_version_number</td>
<td>2</td>
<td>Version number to optionally indicate that the contents of the key block is a component (key part), or to prevent re-injection of an old key. This field is a tool for enforcement of local key change rules.</td>
</tr>
<tr>
<td>Exportability</td>
<td>exportability</td>
<td>1</td>
<td>Defines whether the protected key may be exported.</td>
</tr>
<tr>
<td>Number of optional blocks</td>
<td>num_opt_blocks</td>
<td>4 (integer)</td>
<td>Defines the number of optional blocks included in the key block. If this value is greater than zero, use the TR31 Optional Data Read verb to obtain the contents of the optional blocks.</td>
</tr>
</tbody>
</table>

This verb does not perform cryptographic services on any key value. You cannot use this verb to change a key or to change the control vector related to a key.
Format

The format of CSNBT31P.

```c
CSNBT31P(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    tr31_key_length,
    tr31_key,
    key_block_version,
    key_block_length,
    key_usage,
    algorithm,
    mode,
    key_version_number,
    exportability,
    num_opt_blocks)
```

Parameters

The parameters for CSNBT31P.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. The value must be 0.

**rule_array**

- **Direction:** Input
- **Type:** String array

No rule array keywords are defined for this verb.

**tr31_key_length**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of bytes of data in the `tr31_key` variable. Specify a length that is greater than or equal to the size of the key block. The verb determines the actual length of the key by parsing its contents.

**tr31_key**

- **Direction:** Input
- **Type:** String

A pointer to a string variable containing the TR-31 key block to be disassembled.

**key_block_version**
TR31 Key Token Parse (CSNBT31P)

Direction: Output
Type: String

A pointer to a string variable. The verb copies the one byte found in the key block version ID field of the input key block to this variable.

Note that if the verb finds a proprietary key block version ID, the verb treats it as an invalid value, because the verb is not capable of disassembling a key block that has a proprietary ID. This variable is not updated if a processing error occurs.

**key_block_length**

Direction: Output
Type: Integer

A pointer to an integer variable. The verb parses the 2-byte numeric ASCII key block length field from the input key block, converts the string value into an integer, and returns the integer in this variable. This value must be less than or equal to the **tr31_key_length** input variable.

**key_usage**

Direction: Output
Type: String

A pointer to a string variable. The verb copies the two bytes found in the key usage field of the input key block to this variable.

**algorithm**

Direction: Output
Type: String

A pointer to a string variable. The verb copies the one byte found in the algorithm field of the input key block to this variable. **The verb does not treat a proprietary algorithm value as an error.**

**mode**

Direction: Output
Type: String

A pointer to a one-byte string variable containing the TR-31 mode of use for the key contained in the block. The value is obtained from the TR-31 header. The mode of use describes what operations the key can perform, within the limitations specified with the key usage value. For example, a key with usage for data encryption can have a mode to indicate that it can be used only for encryption, decryption, or both.

This pointer must be non-NULL and point to application storage with at least the size given by the byte count noted. The storage is updated with the noted value on a successful return from this verb, and unchanged otherwise.

**key_version_number**

Direction: Output
Type: String

A pointer to a 2-byte string variable obtained from the TR-31 header, which can be used for one of three purposes, or can be unused.

- If both bytes are X’30’ ("0"), then key versioning is unused for this key. In this case, the second byte is not examined and can contain any value.
If the first byte is X'63' ("c"), then the block contains a component of a key which must be combined with other components in order to form the complete key. TR-31 does not define the method through which the components are combined. TR-31 specifies that local rules are used for that purpose.

In this case, the second byte is not examined and can contain any value.

If the first byte is anything other than the two values above, then the 2-byte key version value is an identifier of the version of the key that is carried in the block. This key version value can be used by an application, for example, to ensure that an old version of a key is not reentered into the system.

This pointer must be non-NULL and point to application storage with at least the size given by the byte count noted. The storage is updated with the noted value on a successful return from this verb, and unchanged otherwise.

**exportability**

- **Direction:** Output
- **Type:** String

A pointer to a one-byte string variable containing the key exportability value from the TR-31 header. This value indicates whether the key can be exported from this system, and if so specifies conditions under which export is permitted. The following three values are possible:

- If the value is X'4E' ("N"), then the key is not exportable.
- If the value is X'53' ("S"), then the key is exportable under any key-encrypting key.
- If the value is X'45' ("E"), then the key is exportable only under a trusted key-encrypting key. TR 31 defines such a trusted key as either one that is encrypted under the HSM master key or one that is itself contained in a TR-31 key block. CCA does not support KEKs that are wrapped in TR-31 key blocks.

This pointer must be non-NULL and point to application storage with at least the size given by the byte count noted. The storage is updated with the noted value on a successful return from this verb, and unchanged otherwise.

**num_opt_block**

- **Direction:** Output
- **Type:** Integer

A pointer to an integer value containing the number of optional blocks that are part of the TR-31 key block. Information about each optional block can be obtained using the TR31 Optional Data Read verb. In this verb, use the number of optional blocks acquired with this verb to obtain a list of the IDs and lengths for each optional block. Then, use those lists to read the data from each desired block.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBT31PJ.

See “Building Java applications using the CCA JNI” on page 28.
TR31 Key Token Parse (CSNBT31P)

Format

```java
public native void CSNBT31PJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber tr31_key_length,
    byte[] tr31_key,
    byte[] key_block_version,
    hikmNativeNumber key_block_length,
    byte[] key_usage,
    byte[] algorithm,
    byte[] mode,
    byte[] key_version_number,
    byte[] exportability,
    hikmNativeNumber num_opt_blocks);
```

TR31 Optional Data Build (CSNBT31O)

Use the TR31 Optional Data Build verb to build a properly formatted TR-31 optional block from the data provided.

The newly constructed optional block can optionally be appended to an existing structure of optional blocks, or it can be returned as a new optional blocks structure. After the last optional block has been constructed, the completed structure containing the optional blocks can be included in a TR-31 key block during an export operation by the Key Export to TR31 verb by using its opt_blocks parameter. For information about TR-31, including the format of a TR-31 optional block, see X9 TR-31 2010: Interoperable Secure Key Exchange Block Specification for Symmetric Algorithms.

The TR-31 key block has an unencrypted header that can contain optional blocks. The header is securely bound to the key block using the integrated MAC. An optional block has a 2-byte ASCII block ID value that determines the use of the block. The ID of each block in an optional blocks structure must be unique.

The verb builds a structure of optional blocks by adding one optional block with each call. This process is repeated until the entire set of optional blocks has been added. For each call, provide the components for a single optional block. This includes the optional block ID, the optional block length in bytes, and the optional block data. In addition, provide an optional blocks buffer large enough to add the optional block being built.

There are two valid scenarios for the optional blocks buffer provided on input, as determined by the value of the opt_blocks_length variable:

1. The optional blocks buffer is empty. In this case, the newly constructed optional block is copied into the buffer.
2. The optional blocks buffer contains one or more existing optional blocks. In this case, the newly constructed optional block is appended to the existing optional blocks. No duplicate IDs are allowed.

Upon successful completion, the opt_blocks_length variable is updated to the length of the returned optional blocks structure.

This verb does not perform cryptographic services on any key value. You cannot use this verb to change a key or to change the control vector related to a key.
Format

The format of CSNBT31O.

```
CSNBT31O(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    opt_blocks_bfr_length,
    opt_blocks_length,
    opt_blocks,
    num_opt_blocks,
    opt_block_id,
    opt_block_data_length,
    opt_block_data)
```

Parameters

The parameters for CSNBT31O.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see “Parameters common to all verbs” on page 22.

`rule_array_count`

**Direction:** Input  
**Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. The value must be 0, because no keywords are currently defined for this verb.

`rule_array`

**Direction:** Input  
**Type:** String array

No rule array keywords are defined for this verb.

`opt_blocks_bfr_length`

**Direction:** Input  
**Type:** Integer

A pointer to an integer variable containing the length in bytes of the buffer allocated for the `opt_blocks` variable. Set this length to at least the size of any optional blocks structure in the buffer plus the optional block being added.

`opt_blocks_length`

**Direction:** Input/Output  
**Type:** Integer

A pointer to an integer variable containing the length in bytes of the data in the `opt_blocks` variable. This length must be less than or equal to the value of the `opt_blocks_bfr_length` variable. On input, set this variable to the length of the optional blocks structure being updated. Set this value to zero if it is the first optional block in the structure. On successful completion, this variable is updated with the length of the updated variable.

`opt_blocks`
TR31 Optional Data Build (CSNBT31O)

Direction:     Input/Output  
Type:          String

A pointer to a string variable containing a buffer for the optional blocks structure that the verb updates. In the first call to the verb, the buffer will generally be empty. The verb appends one optional block to the buffer with each call.

num_opt_blocks

Direction:     Output  
Type:          Integer

The num_opt_blocks parameter is a pointer to an integer variable containing the number of optional blocks contained in the opt_blocks variable that is returned by the verb.

opt_block_id

Direction:     Input  
Type:          String

A pointer to a string variable containing a 2-byte value that identifies the use of the optional block. Each ID must be unique, that is, no duplicates are allowed.

Note that a value of "PB" is not allowed. The Key Export to TR31 verb adds a padding block of the appropriate size as needed when building the TR-31 key block. Unlike the padding within the encryption key portion of the TR-31 key block, the padding block for optional blocks serves no security purpose.

opt_block_data_length

Direction:     Input  
Type:          Integer

A pointer to an integer variable containing the length in bytes of the data passed in the opt_block_data variable. Note that it is valid for this length to be zero, since an optional block can have an ID and a length, but no data.

opt_block_data

Direction:     Input  
Type:          String

A pointer to a string variable containing the data for the optional block that is to be constructed.

Restrictions

The restrictions for CSNBT31O.

An optional block with an ID of "PB" (padding block) cannot be added by the user. The Key Export to TR31 verb adds a padding block of the appropriate size as needed when building the TR-31 key block. Unlike the padding within the encrypted key portion of the key block, the padding block for optional blocks serves no security purpose.
JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBT31OJ.

See “Building Java applications using the CCA JNI” on page 28.

Format

```java
public native void CSNBT31OJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
    hikmNativeNumber exit_data_length,
    byte[] exit_data,
    hikmNativeNumber rule_array_count,
    byte[] rule_array,
    hikmNativeNumber opt_blocks_bfr_length,
    hikmNativeNumber opt_blocks_length,
    byte[] opt_blocks,
    hikmNativeNumber num_opt_blocks,
    byte[] opt_block_id,
    hikmNativeNumber opt_block_data_length,
    byte[] opt_block_data);
```

TR31 Optional Data Read (CSNBT31R)

Use the TR31 Optional Data Read verb to either obtain information about all of the optional blocks in the header of an external TR-31 key block, or obtain the length and data of the specified optional block.

To disassemble the part of the header that is not optional, use the TR31 Key Token Parse verb. Neither verb performs any cryptographic services, and both disassemble a key block in application storage. The validity of the key block is verified as much as can be done without performing any cryptographic services.

A TR-31 key block contains an unencrypted header that can include one or more optional blocks. All parts of the header are securely bound to the key block using the integrated MAC.

Optional blocks in a key block must each be identified by a unique 2-byte ID. The value of an ID must either be defined by TR-31 or be a numeric value, otherwise the key block is invalid. Numeric IDs are reserved for proprietary use. For more information, see X9 TR-31 2010: Interoperable Secure Key Exchange Block Specification for Symmetric Algorithms.

In order to obtain the data of a particular optional block from the header of an external TR-31 key block, perform the following steps:

1. Use the `tr31_key` parameter to identify the TR-31 key block that this verb is to process.
2. Call the TR31 Key Token Parse verb to parse the TR-31 key block. See “TR31 Key Token Parse (CSNBT31P)” on page 804. Upon successful completion:
   - Set the value of the `tr31_key_length` variable to the value returned in the `tr31_key_length` variable.
   - Set the value of the `num_opt_blocks` variable to the value returned in the `num_opt_blocks` variable.
   - Allocate a string buffer in bytes for the `opt_blocks_id` and an integer buffer in bytes for the `opt_blocks_length` variables. These buffers must be at least two times the value of the `num_opt_blocks` variable.
3. Specify a rule-array keyword of INFO to obtain information about the optional blocks in the key block. The opt_block_id, opt_block_data_length, and opt_block_data parameters are ignored.

4. Call the TR31 Optional Data Read verb to read data from the TR-31 key block. Upon successful completion, the verb returns an array of optional block IDs in the opt_blocks_id variable, and an array of lengths for the optional block IDs in the opt_blocks_length variable. The IDs and lengths are returned in same order as the optional blocks appear in the header of the TR-31 key block.

5. Determine which ID of the unique IDs contained in the opt_blocks_id variable is to be obtained from the TR-31 key block. Set the opt_block_id variable to this 2-byte value. Set the value of the opt_block_data_length variable to the corresponding length from the opt_blocks_length variable.

   Note: The offset used to locate the ID in the opt_blocks_id variable has the same value as the offset for the corresponding length in the opt_blocks_length variable.

6. Allocate a buffer in bytes for the opt_block_data variable that is at least the value of the opt_block_data_length variable.

7. Specify a rule-array keyword of DATA to obtain the length and data of the specified optional block. The num_opt_blocks and the opt_blocks_id parameters are ignored.

8. Call the TR31 Optional Data Read verb. Upon successful completion, the verb returns the data of the specified optional block in the opt_block_data variable. The verb updates the opt_block_data_length variable to the number of bytes returned in the opt_block_data variable.

This verb does not perform cryptographic services on any key value. You cannot use this verb to change a key or to change the control vector related to a key.

Format

The format of CSNBT31R.

```
CSNBT31R(  return_code,  
          reason_code,  
          exit_data_length,  
          exit_data,  
          rule_array_count,  
          rule_array,  
          TR31_key_length,  
          TR31_key,  
          opt_block_id,  
          num_opt_blocks,  
          opt_block_ids,  
          opt_block_lengths,  
          opt_block_data_length,  
          opt_block_data)
```

Parameters

The parameters for CSNBT31R.

For the definitions of the return_code, reason_code, exit_data_length, and exit_data parameters, see “Parameters common to all verbs” on page 22.

`rule_array_count`

Direction: Input
TR31 Optional Data Read (CSNBT31R)

Type: Integer

A pointer to an integer variable containing the number of elements in the rule_array variable. The value must be 1.

rule_array

Direction: Input
Type: String array

A pointer to a string variable containing an array of keywords. The keywords are 8 bytes in length and must be left-aligned and padded on the right with space characters. The following rule_array keywords are defined for this verb:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO</td>
<td>Return information about the optional blocks in the TR-31 key block.</td>
</tr>
<tr>
<td>DATA</td>
<td>Return the data contained in a specified optional block in the TR-31 key block.</td>
</tr>
</tbody>
</table>

tr31_key_length

Direction: Input
Type: Integer

A pointer to an integer variable containing the number of bytes of data in the tr31_key variable. Specify a length that is greater than or equal to the size of the key block. The verb determines the actual length of the key by parsing its contents.

tr31_key

Direction: Input
Type: String

A pointer to a string variable containing the TR-31 key block to be disassembled.

opt_block_id

Direction: Input
Type: String

This parameter is used when operation keyword DATA is specified, otherwise it is ignored. For keyword DATA, this parameter is a pointer to a string variable that identifies the 2-byte ID of the optional block to obtain from the TR-31 key block.

num_opt_blocks

Direction: Input
Type: Integer

This parameter is used when operation keyword INFO is specified, otherwise it is ignored. For keyword INFO, this parameter is a pointer to an integer variable that specifies the number of 2-byte optional block IDs that are allocated for (1) the opt_block_ids variable and (2) the number of 2-byte integers that are allocated for the opt_block_lengths variable. This the value must specify the exact number of optional blocks that are in the header of the TR-31 key block. Use the TR31 Key Token Parse verb to determine the number of optional blocks IDs in a TR-31 key block before calling this verb.
TR31 Optional Data Read (CSNBT31R)

**opt_block_ids**

- **Direction:** Output
- **Type:** String array

This parameter is used when operation keyword INFO is specified, otherwise it is ignored. For keyword INFO, this parameter is a pointer to a string array of 2-byte values that lists the identifiers of each optional block contained in the header of the TR-31 key block. Each ID must be unique, that is, no duplicates are allowed. The IDs, along with the associated lengths listed in the opt_block_lengths variable, are returned in the order that the optional blocks appear in the header of the TR-31 key block. The size of the variable must be at least two times the value of the num_opts_blocks variable.

**opt_block_lengths**

- **Direction:** Output
- **Type:** String array

This parameter is used when operation keyword INFO is specified, otherwise it is ignored. For keyword INFO, this parameter is a pointer to an integer array of 2-byte values that are 16-bit unsigned integers corresponding to the associated length of the optional block identified in the opt_block_ids variable. The lengths, along with the associated IDs listed in the opt_block_ids variable, are returned in the order that the optional blocks appear in the header of the TR-31 key block. The size of the variable must be at least two times the value of the num_opts_blocks variable.

**opt_block_data_length**

- **Direction:** Input/Output
- **Type:** Integer

This parameter is used when operation keyword DATA is specified, otherwise it is ignored. For keyword DATA, this parameter is a pointer to an integer variable containing the length of the opt_block_data parameter. On input, this variable specifies the maximum permissible length of the result. On output, the verb updates the value to length of the returned optional block data.

**opt_block_data**

- **Direction:** Output
- **Type:** String

This parameter is used when operation keyword DATA is specified, otherwise it is ignored. For keyword DATA, this parameter is a pointer to a string variable. If the TR-31 key block is found to be valid and the TR-31 key block contains an optional block specified by the optional_block_ID variable, the optional block is copied into this variable if it is large enough. The opt_block_data_length variable is updated with the length of the data returned in the variable.

**JNI version**

This verb has a Java Native Interface (JNI) version, which is named CSNBT31RJ.

See “Building Java applications using the CCA JNI” on page 28.

**Format**

```java
public native void CSNBT31RJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
```

814  Common Cryptographic Architecture Application Programmer's Guide
TR31 Optional Data Read (CSNBT31R)

```
hkmNativeNumber  exit_data_length,  
byte[]            exit_data,       
hkmNativeNumber  rule_array_count,  
byte[]            rule_array,     
hkmNativeNumber  tr31_key_length,  
byte[]            tr31_key,        
byte[]            opt_block_id,    
hkmNativeNumber  num_opt_blocks, 
byte[]            opt_block_ids,   
byte[]            opt_block_lengths, 
hkmNativeNumber  opt_block_data_length,  
byte[]            opt_block_data); 
```
TR31 Optional Data Read (CSNB3T3R)
Chapter 17. Utility verbs

Use the described utility verb for code conversion.

- "Code Conversion (CSNBXEA)"

Code Conversion (CSNBXEA)

Use the Code Conversion utility to convert between ASCII data, EBCDIC data, or a custom 8-bit based conversion table.

Format

The format of CSNBXEA.

```
CSNBXEA(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    text_length,
    source_text,
    target_text,
    code_table_length,
    code_table)
```

Parameters

The parameters for CSNBXEA.

For the definitions of the `return_code`, `reason_code`, `exit_data_length`, and `exit_data` parameters, see "Parameters common to all verbs" on page 22.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

A pointer to an integer variable containing the number of elements in the `rule_array` variable. This value must be 1.

**rule_array**

- **Direction:** Input
- **Type:** String array

An array of 8-byte keywords providing the processing control information. The keywords must be left-aligned and padded on the right with space characters. The rule_array keywords are described in [Table 222](#).

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service requested</td>
<td>(One, required)</td>
</tr>
</tbody>
</table>

Table 222. Keywords for the CSNBXEA utility

© Copyright IBM Corp. 2007, 2018
Table 222. Keywords for the CSNBXEA utility (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOASCII</td>
<td>Converts the contents of <code>source_text</code> from EBCDIC character set to ASCII character set using the appropriate code table that is displayed in &quot;Usage notes&quot; on page 819. The result is placed in parameter <code>target_text</code>. The parameters <code>code_table</code> and <code>code_table_length</code> are ignored for this keyword.</td>
</tr>
<tr>
<td>TOEBCDIC</td>
<td>Converts the contents of <code>source_text</code> from ASCII character set to EBCDIC character set by using the code table that is displayed in &quot;Usage notes&quot; on page 819. The result is placed in parameter <code>target_text</code>. The parameters <code>code_table</code> and <code>code_table_length</code> are ignored for this keyword.</td>
</tr>
<tr>
<td>USETABLE</td>
<td>Converts the contents of <code>source_text</code> from the original character set to a custom character set by using the appropriate code table that is passed by the application in the <code>code_table</code> parameter. Each byte of the <code>source_text</code> is used to index the <code>code_table</code> to discover the resulting byte for the <code>target_text</code>. This keyword assumes that the conversion data is single-byte aligned. One byte of <code>source_text</code> is used to look up 1 byte of conversion value in the code table. Therefore the <code>code_table_length</code> must be at least 256 bytes. If the <code>code_table_length</code> is larger than 256 bytes, the extra is ignored.</td>
</tr>
</tbody>
</table>

**text_length**
- Direction: Input
- Type: Integer

A pointer to an integer variable that contains the length of the `source_text` parameter. The length must be a positive nonzero value.

**source_text**
- Direction: Input
- Type: String

This parameter contains the string to be converted.

**target_text**
- Direction: Output
- Type: String

This parameter contains the converted text that is returned by this verb. The size of this buffer must be at least as long as indicated by `text_length`.

**code_table_length**
- Direction: Input
- Type: Integer

The size in bytes of the buffer that is passed as the `code_table` parameter.

**code_table**
- Direction: Input
- Type: String

A data conversion table that is specified by the user for converting the contents of `source_text` to `target_text`. 
Restrictions
The restrictions for CSNBXEA.
None.

Required commands
The required commands for CSNBXEA.
None.

Usage notes
The usage notes for CSNBXEA. Also the conversion tables for EBCDIC to ASCII and for ASCII to EBCDIC are listed.

This service is built to provide exact correspondence to the functions provided by the ICSF verbs CSNBXEA and CSNBXAE. The default conversion tables for EBCDIC-to-ASCII and ASCII-to-EBCDIC (see Table 223 on page 820 and Table 224 on page 821) match those used for the ICSF verbs. The origin of these conversion tables was a comparison of the EBCDIC 1047 code page to a widely used mapping for Extended ASCII. Round-trip conversions (achievable with two calls to this service for ASCII-to-EBCDIC-to-ASCII or EBCDIC-to-ASCII-to-EBCDIC) provide the original data as output.

This service is structured differently than the other services. It runs in the caller's address space in the caller's key and mode. The adapter need not be active for you to run this service.
### Table 223. EBCDIC to ASCII conversion table

<table>
<thead>
<tr>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>20</td>
<td>81</td>
<td>40</td>
<td>20</td>
<td>60</td>
<td>2D</td>
<td>80</td>
<td>F8</td>
<td>A0</td>
<td>C8</td>
<td>C0</td>
<td>7B</td>
<td>E0</td>
<td>5C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>21</td>
<td>82</td>
<td>41</td>
<td>A6</td>
<td>61</td>
<td>2F</td>
<td>81</td>
<td>61</td>
<td>A1</td>
<td>7E</td>
<td>C1</td>
<td>41</td>
<td>E1</td>
<td>E7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>02</td>
<td>22</td>
<td>1C</td>
<td>42</td>
<td>E1</td>
<td>62</td>
<td>DF</td>
<td>82</td>
<td>62</td>
<td>A2</td>
<td>73</td>
<td>C2</td>
<td>42</td>
<td>E2</td>
<td>53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>03</td>
<td>23</td>
<td>84</td>
<td>43</td>
<td>80</td>
<td>63</td>
<td>DC</td>
<td>83</td>
<td>63</td>
<td>A3</td>
<td>74</td>
<td>C3</td>
<td>43</td>
<td>E3</td>
<td>54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>CF</td>
<td>24</td>
<td>86</td>
<td>44</td>
<td>EB</td>
<td>64</td>
<td>9A</td>
<td>84</td>
<td>64</td>
<td>A4</td>
<td>75</td>
<td>C4</td>
<td>44</td>
<td>E4</td>
<td>55</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>09</td>
<td>25</td>
<td>0A</td>
<td>45</td>
<td>90</td>
<td>65</td>
<td>DD</td>
<td>85</td>
<td>65</td>
<td>A5</td>
<td>76</td>
<td>C5</td>
<td>45</td>
<td>E5</td>
<td>56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>D3</td>
<td>26</td>
<td>17</td>
<td>46</td>
<td>9F</td>
<td>66</td>
<td>DE</td>
<td>86</td>
<td>66</td>
<td>A6</td>
<td>77</td>
<td>C6</td>
<td>46</td>
<td>E6</td>
<td>57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>7F</td>
<td>27</td>
<td>1B</td>
<td>47</td>
<td>E2</td>
<td>67</td>
<td>98</td>
<td>87</td>
<td>67</td>
<td>A7</td>
<td>78</td>
<td>C7</td>
<td>47</td>
<td>E7</td>
<td>58</td>
<td></td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>D4</td>
<td>28</td>
<td>89</td>
<td>48</td>
<td>AB</td>
<td>68</td>
<td>9D</td>
<td>88</td>
<td>68</td>
<td>A8</td>
<td>79</td>
<td>C8</td>
<td>48</td>
<td>E8</td>
<td>59</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>D5</td>
<td>29</td>
<td>91</td>
<td>49</td>
<td>8B</td>
<td>69</td>
<td>AC</td>
<td>89</td>
<td>69</td>
<td>A9</td>
<td>7A</td>
<td>C9</td>
<td>49</td>
<td>E9</td>
<td>5A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0A</td>
<td>C3</td>
<td>2A</td>
<td>92</td>
<td>4A</td>
<td>9B</td>
<td>6A</td>
<td>BA</td>
<td>8A</td>
<td>96</td>
<td>AA</td>
<td>EF</td>
<td>CA</td>
<td>CB</td>
<td>EA</td>
<td>A0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0B</td>
<td>0B</td>
<td>2B</td>
<td>95</td>
<td>4B</td>
<td>2E</td>
<td>6B</td>
<td>2C</td>
<td>8B</td>
<td>A4</td>
<td>AB</td>
<td>C0</td>
<td>CB</td>
<td>CA</td>
<td>EB</td>
<td>85</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0C</td>
<td>0C</td>
<td>2C</td>
<td>A2</td>
<td>4C</td>
<td>3C</td>
<td>6C</td>
<td>25</td>
<td>8C</td>
<td>F3</td>
<td>AC</td>
<td>DA</td>
<td>CC</td>
<td>BE</td>
<td>EC</td>
<td>8E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0D</td>
<td>0D</td>
<td>2D</td>
<td>05</td>
<td>4D</td>
<td>28</td>
<td>6D</td>
<td>5F</td>
<td>8D</td>
<td>AF</td>
<td>AD</td>
<td>5B</td>
<td>CD</td>
<td>E8</td>
<td>ED</td>
<td>E9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0E</td>
<td>0E</td>
<td>2E</td>
<td>06</td>
<td>4E</td>
<td>2B</td>
<td>6E</td>
<td>3E</td>
<td>8E</td>
<td>AE</td>
<td>AE</td>
<td>F2</td>
<td>CE</td>
<td>EC</td>
<td>EE</td>
<td>E4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0F</td>
<td>0F</td>
<td>2F</td>
<td>07</td>
<td>4F</td>
<td>7C</td>
<td>6F</td>
<td>3F</td>
<td>8F</td>
<td>C5</td>
<td>AF</td>
<td>F9</td>
<td>CF</td>
<td>ED</td>
<td>EF</td>
<td>D1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>30</td>
<td>E0</td>
<td>50</td>
<td>26</td>
<td>70</td>
<td>D7</td>
<td>90</td>
<td>8C</td>
<td>B0</td>
<td>B5</td>
<td>D0</td>
<td>7D</td>
<td>F0</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>31</td>
<td>EE</td>
<td>51</td>
<td>A9</td>
<td>71</td>
<td>88</td>
<td>91</td>
<td>6A</td>
<td>B1</td>
<td>B6</td>
<td>D1</td>
<td>4A</td>
<td>F1</td>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>32</td>
<td>16</td>
<td>52</td>
<td>AA</td>
<td>72</td>
<td>94</td>
<td>92</td>
<td>6B</td>
<td>B2</td>
<td>FD</td>
<td>D2</td>
<td>4B</td>
<td>F2</td>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>33</td>
<td>E5</td>
<td>53</td>
<td>9C</td>
<td>73</td>
<td>B0</td>
<td>93</td>
<td>6C</td>
<td>B3</td>
<td>B7</td>
<td>D3</td>
<td>4C</td>
<td>F3</td>
<td>33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>C7</td>
<td>34</td>
<td>D0</td>
<td>54</td>
<td>DB</td>
<td>74</td>
<td>B1</td>
<td>94</td>
<td>6D</td>
<td>B4</td>
<td>B8</td>
<td>D4</td>
<td>4D</td>
<td>F4</td>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>B4</td>
<td>35</td>
<td>1E</td>
<td>55</td>
<td>A5</td>
<td>75</td>
<td>B2</td>
<td>95</td>
<td>6E</td>
<td>B5</td>
<td>B9</td>
<td>D5</td>
<td>4E</td>
<td>F5</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>08</td>
<td>36</td>
<td>EA</td>
<td>56</td>
<td>99</td>
<td>76</td>
<td>FC</td>
<td>96</td>
<td>6F</td>
<td>B6</td>
<td>E6</td>
<td>D6</td>
<td>4F</td>
<td>F6</td>
<td>36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>C9</td>
<td>37</td>
<td>04</td>
<td>57</td>
<td>E3</td>
<td>77</td>
<td>D6</td>
<td>97</td>
<td>70</td>
<td>BB</td>
<td>D7</td>
<td>50</td>
<td>F7</td>
<td>37</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>38</td>
<td>8A</td>
<td>58</td>
<td>A8</td>
<td>78</td>
<td>FB</td>
<td>98</td>
<td>71</td>
<td>B8</td>
<td>BC</td>
<td>D8</td>
<td>51</td>
<td>F8</td>
<td>38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>39</td>
<td>F6</td>
<td>59</td>
<td>9E</td>
<td>79</td>
<td>60</td>
<td>99</td>
<td>72</td>
<td>B9</td>
<td>BD</td>
<td>D9</td>
<td>52</td>
<td>F9</td>
<td>39</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1A</td>
<td>CC</td>
<td>3A</td>
<td>C6</td>
<td>5A</td>
<td>21</td>
<td>7A</td>
<td>3A</td>
<td>9A</td>
<td>97</td>
<td>BA</td>
<td>8D</td>
<td>DA</td>
<td>A1</td>
<td>FA</td>
<td>B3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B</td>
<td>CD</td>
<td>3B</td>
<td>C2</td>
<td>5B</td>
<td>24</td>
<td>7B</td>
<td>23</td>
<td>9B</td>
<td>87</td>
<td>BB</td>
<td>D9</td>
<td>DB</td>
<td>AD</td>
<td>FB</td>
<td>F7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1C</td>
<td>83</td>
<td>3C</td>
<td>14</td>
<td>5C</td>
<td>2A</td>
<td>7C</td>
<td>40</td>
<td>9C</td>
<td>CE</td>
<td>BC</td>
<td>BF</td>
<td>DC</td>
<td>F5</td>
<td>FC</td>
<td>F0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1D</td>
<td>1D</td>
<td>3D</td>
<td>15</td>
<td>5D</td>
<td>29</td>
<td>7D</td>
<td>27</td>
<td>9D</td>
<td>93</td>
<td>BD</td>
<td>5D</td>
<td>DD</td>
<td>F4</td>
<td>FD</td>
<td>FA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E</td>
<td>D2</td>
<td>3E</td>
<td>C1</td>
<td>5E</td>
<td>3B</td>
<td>7E</td>
<td>3D</td>
<td>9E</td>
<td>F1</td>
<td>BE</td>
<td>D8</td>
<td>DE</td>
<td>A3</td>
<td>FE</td>
<td>A7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1F</td>
<td>1F</td>
<td>3F</td>
<td>1A</td>
<td>5F</td>
<td>5E</td>
<td>7F</td>
<td>22</td>
<td>9F</td>
<td>FE</td>
<td>BF</td>
<td>C4</td>
<td>DF</td>
<td>8F</td>
<td>FF</td>
<td>FF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 224. ASCII to EBCDIC conversion table

<table>
<thead>
<tr>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>20</td>
<td>40</td>
<td>40</td>
<td>7C</td>
<td>60</td>
<td>79</td>
<td>80</td>
<td>43</td>
<td>A0</td>
<td>EA</td>
<td>C0</td>
<td>AB</td>
<td>E0</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>21</td>
<td>5A</td>
<td>41</td>
<td>C1</td>
<td>61</td>
<td>81</td>
<td>81</td>
<td>20</td>
<td>A1</td>
<td>DA</td>
<td>C1</td>
<td>3E</td>
<td>E1</td>
<td>42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>02</td>
<td>22</td>
<td>7F</td>
<td>42</td>
<td>C2</td>
<td>62</td>
<td>82</td>
<td>82</td>
<td>21</td>
<td>A2</td>
<td>2C</td>
<td>C2</td>
<td>3B</td>
<td>E2</td>
<td>47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>03</td>
<td>23</td>
<td>7B</td>
<td>43</td>
<td>C3</td>
<td>63</td>
<td>83</td>
<td>83</td>
<td>1C</td>
<td>A3</td>
<td>DE</td>
<td>C3</td>
<td>0A</td>
<td>E3</td>
<td>57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>37</td>
<td>24</td>
<td>5B</td>
<td>44</td>
<td>C4</td>
<td>64</td>
<td>84</td>
<td>84</td>
<td>23</td>
<td>A4</td>
<td>8B</td>
<td>C4</td>
<td>BF</td>
<td>E4</td>
<td>EE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>2D</td>
<td>25</td>
<td>6C</td>
<td>45</td>
<td>C5</td>
<td>65</td>
<td>85</td>
<td>85</td>
<td>EB</td>
<td>A5</td>
<td>55</td>
<td>C5</td>
<td>8F</td>
<td>E5</td>
<td>33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>2E</td>
<td>26</td>
<td>50</td>
<td>46</td>
<td>C6</td>
<td>66</td>
<td>86</td>
<td>86</td>
<td>24</td>
<td>A6</td>
<td>41</td>
<td>C6</td>
<td>3A</td>
<td>E6</td>
<td>B6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>2F</td>
<td>27</td>
<td>7D</td>
<td>47</td>
<td>C7</td>
<td>67</td>
<td>87</td>
<td>87</td>
<td>9B</td>
<td>A7</td>
<td>FE</td>
<td>C7</td>
<td>14</td>
<td>E7</td>
<td>E1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>16</td>
<td>28</td>
<td>4D</td>
<td>48</td>
<td>C8</td>
<td>68</td>
<td>88</td>
<td>88</td>
<td>71</td>
<td>A8</td>
<td>58</td>
<td>C8</td>
<td>A0</td>
<td>E8</td>
<td>CD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>05</td>
<td>29</td>
<td>5D</td>
<td>49</td>
<td>C9</td>
<td>69</td>
<td>89</td>
<td>89</td>
<td>28</td>
<td>A9</td>
<td>51</td>
<td>C9</td>
<td>17</td>
<td>E9</td>
<td>ED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0A</td>
<td>25</td>
<td>2A</td>
<td>5C</td>
<td>4A</td>
<td>D1</td>
<td>6A</td>
<td>91</td>
<td>8A</td>
<td>38</td>
<td>AA</td>
<td>52</td>
<td>CA</td>
<td>CB</td>
<td>EA</td>
<td>36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0B</td>
<td>0B</td>
<td>2B</td>
<td>4E</td>
<td>4B</td>
<td>D2</td>
<td>6B</td>
<td>92</td>
<td>8B</td>
<td>49</td>
<td>AB</td>
<td>48</td>
<td>CB</td>
<td>CA</td>
<td>EB</td>
<td>44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0C</td>
<td>0C</td>
<td>2C</td>
<td>6B</td>
<td>4C</td>
<td>D3</td>
<td>6C</td>
<td>93</td>
<td>8C</td>
<td>90</td>
<td>AC</td>
<td>69</td>
<td>CC</td>
<td>1A</td>
<td>EC</td>
<td>CE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0D</td>
<td>0D</td>
<td>2D</td>
<td>60</td>
<td>4D</td>
<td>D4</td>
<td>6D</td>
<td>94</td>
<td>8D</td>
<td>BA</td>
<td>AD</td>
<td>DB</td>
<td>CD</td>
<td>1B</td>
<td>ED</td>
<td>CF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0E</td>
<td>0E</td>
<td>2E</td>
<td>4B</td>
<td>4E</td>
<td>D5</td>
<td>6E</td>
<td>95</td>
<td>8E</td>
<td>EC</td>
<td>AE</td>
<td>8E</td>
<td>CE</td>
<td>9C</td>
<td>EE</td>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0F</td>
<td>0F</td>
<td>2F</td>
<td>61</td>
<td>4F</td>
<td>D6</td>
<td>6F</td>
<td>96</td>
<td>8F</td>
<td>DF</td>
<td>AF</td>
<td>8D</td>
<td>CF</td>
<td>04</td>
<td>EF</td>
<td>AA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>30</td>
<td>50</td>
<td>50</td>
<td>D7</td>
<td>70</td>
<td>97</td>
<td>90</td>
<td>45</td>
<td>B0</td>
<td>73</td>
<td>D0</td>
<td>34</td>
<td>F0</td>
<td>FC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>31</td>
<td>F1</td>
<td>51</td>
<td>D8</td>
<td>71</td>
<td>98</td>
<td>91</td>
<td>29</td>
<td>B1</td>
<td>74</td>
<td>D1</td>
<td>EF</td>
<td>F1</td>
<td>9E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>32</td>
<td>F2</td>
<td>52</td>
<td>D9</td>
<td>72</td>
<td>99</td>
<td>92</td>
<td>2A</td>
<td>B2</td>
<td>75</td>
<td>D2</td>
<td>1E</td>
<td>F2</td>
<td>AE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>33</td>
<td>F3</td>
<td>53</td>
<td>E2</td>
<td>73</td>
<td>A2</td>
<td>93</td>
<td>9D</td>
<td>B3</td>
<td>FA</td>
<td>D3</td>
<td>06</td>
<td>F3</td>
<td>8C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>3C</td>
<td>34</td>
<td>F4</td>
<td>54</td>
<td>E3</td>
<td>74</td>
<td>A3</td>
<td>94</td>
<td>72</td>
<td>B4</td>
<td>15</td>
<td>D4</td>
<td>08</td>
<td>F4</td>
<td>DD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>3D</td>
<td>35</td>
<td>F5</td>
<td>55</td>
<td>E4</td>
<td>75</td>
<td>A4</td>
<td>95</td>
<td>2B</td>
<td>B5</td>
<td>05</td>
<td>D5</td>
<td>09</td>
<td>F5</td>
<td>DC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>36</td>
<td>3F</td>
<td>56</td>
<td>56</td>
<td>E5</td>
<td>76</td>
<td>A5</td>
<td>96</td>
<td>8A</td>
<td>B6</td>
<td>01</td>
<td>D6</td>
<td>77</td>
<td>F6</td>
<td>39</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>26</td>
<td>37</td>
<td>F7</td>
<td>57</td>
<td>E6</td>
<td>77</td>
<td>A6</td>
<td>97</td>
<td>9A</td>
<td>B7</td>
<td>07</td>
<td>D7</td>
<td>70</td>
<td>F7</td>
<td>FB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>38</td>
<td>38</td>
<td>F8</td>
<td>58</td>
<td>E7</td>
<td>78</td>
<td>A7</td>
<td>98</td>
<td>67</td>
<td>B8</td>
<td>04</td>
<td>D8</td>
<td>68</td>
<td>F8</td>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>39</td>
<td>39</td>
<td>F9</td>
<td>59</td>
<td>E8</td>
<td>79</td>
<td>A8</td>
<td>99</td>
<td>56</td>
<td>B9</td>
<td>05</td>
<td>D9</td>
<td>59</td>
<td>BB</td>
<td>F9</td>
<td>AF</td>
<td></td>
</tr>
<tr>
<td>1A</td>
<td>3F</td>
<td>3A</td>
<td>7A</td>
<td>5A</td>
<td>E9</td>
<td>7A</td>
<td>A9</td>
<td>9A</td>
<td>64</td>
<td>BA</td>
<td>6A</td>
<td>DA</td>
<td>AC</td>
<td>FA</td>
<td>FD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B</td>
<td>27</td>
<td>3B</td>
<td>5E</td>
<td>5B</td>
<td>AD</td>
<td>7B</td>
<td>C0</td>
<td>9B</td>
<td>4A</td>
<td>BB</td>
<td>B7</td>
<td>DB</td>
<td>54</td>
<td>FB</td>
<td>78</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1C</td>
<td>22</td>
<td>3C</td>
<td>4C</td>
<td>5C</td>
<td>E0</td>
<td>7C</td>
<td>4F</td>
<td>9C</td>
<td>53</td>
<td>BC</td>
<td>B8</td>
<td>DC</td>
<td>63</td>
<td>FC</td>
<td>76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1D</td>
<td>1D</td>
<td>3D</td>
<td>7E</td>
<td>5D</td>
<td>BD</td>
<td>7D</td>
<td>D0</td>
<td>9D</td>
<td>68</td>
<td>BD</td>
<td>B9</td>
<td>DD</td>
<td>65</td>
<td>FD</td>
<td>B2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E</td>
<td>35</td>
<td>3E</td>
<td>6E</td>
<td>5E</td>
<td>5F</td>
<td>7E</td>
<td>A1</td>
<td>9E</td>
<td>59</td>
<td>BE</td>
<td>CC</td>
<td>DE</td>
<td>66</td>
<td>FE</td>
<td>9F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1F</td>
<td>1F</td>
<td>3F</td>
<td>6F</td>
<td>5F</td>
<td>6D</td>
<td>7F</td>
<td>07</td>
<td>9F</td>
<td>46</td>
<td>BF</td>
<td>BC</td>
<td>DF</td>
<td>62</td>
<td>FF</td>
<td>FF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### JNI version

This verb has a Java Native Interface (JNI) version, which is named CSNBXEAJ.

See "Building Java applications using the CCA JNI" on page 28.

### Format

```java
public native void CSNBXEAJ(
    hikmNativeNumber return_code,
    hikmNativeNumber reason_code,
)
```

Chapter 17. Utility verbs 821
Code Conversion (CSNBXEA)

```c
hikmNativeNumber exit_data_length,
byte[] exit_data,
hikmNativeNumber rule_array_count,
byte[] rule_array,
hikmNativeNumber text_length,
byte[] source_text,
byte[] target_text,
hikmNativeNumber code_table_length,
byte[] code_table);
```
Part 3. Reference information

The provided reference information informs about key formats, cryptographic algorithms, as well as return and reason codes when you program with CCA.

You can find the following reference information:

- Chapter 18, “Return codes and reason codes,” on page 825
- Chapter 19, “Key token formats,” on page 847
- Chapter 20, “Key forms and types used in the Key Generate verb,” on page 985
- Chapter 21, “Control vectors and changing control vectors with the Control Vector Translate verb,” on page 989
- Chapter 22, “PIN formats and algorithms,” on page 1005
- Chapter 23, “Cryptographic algorithms and processes,” on page 1021
- Chapter 24, “Access control points and verbs,” on page 1047
- Chapter 25, “Access control data structures,” on page 1073
- Chapter 26, “Using verbs and applications in PCI-HSM 2016 compliance mode,” on page 1081
- Chapter 27, “Sample verb call routines,” on page 1093
- Chapter 28, “Initial system set-up tips,” on page 1103
- Chapter 29, “CCA installation instructions,” on page 1107
- Chapter 30, “Coexistence of CEX6C and previous CEX*C features,” on page 1117
- Chapter 31, “Utilities,” on page 1119
- Chapter 32, “Security API command and sub-command codes,” on page 1135
- Chapter 33, “openCryptoki support,” on page 1139
- Chapter 34, “List of abbreviations,” on page 1151
Chapter 18. Return codes and reason codes

Read the contained reference information that describes the return codes and reason codes reported at the conclusion of verb processing.

Reason code numbers narrow down the meaning of a return code. All reason code numbers are unique and associated with a single return code. Generally, you can base your application program design on the return codes.

Each verb supplies a return code and a reason code in the variables identified by the return_code and reason_code parameters. See “Parameters common to all verbs” on page 22.

Return codes

A return code provides a general indication of the results of verb processing.

A return code can have the values shown in Table 225.

Table 225. Return code values

<table>
<thead>
<tr>
<th>Hex value</th>
<th>Decimal value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>This return code indicates a normal completion of verb processing. To provide additional information, there are also nonzero reason codes associated with this return code.</td>
</tr>
<tr>
<td>04</td>
<td>04</td>
<td>This return code is a warning indicating the verb completed processing; however, an unusual event occurred. The event is most likely related to a problem created by the user or is a normal occurrence based on the data supplied to the verb.</td>
</tr>
<tr>
<td>08</td>
<td>08</td>
<td>This return code indicates the verb prematurely stopped processing. Generally, the application programmer needs to investigate the significance of the associated reason code to determine the origin of the problem. In some cases, due to transient conditions, retrying the verb might produce different results.</td>
</tr>
<tr>
<td>0C</td>
<td>12</td>
<td>This return code indicates the verb prematurely stopped processing. Either a coprocessor is not available or a processing error occurred. The reason is most likely related to a problem in the set up of the hardware or in the configuration of the software.</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td>This return code indicates the verb prematurely stopped processing. A processing error occurred. If these errors persist, a repair of the coprocessor hardware or a correction to the coprocessor software might be required.</td>
</tr>
</tbody>
</table>

Note: If an application receives a return code greater than 4, an error occurred. In the case of an error, assume any output variables other than the return code and reason code are not valid, unless otherwise indicated in the description of verb processing.

Reason codes

A reason code details the results of verb processing.
Every reason code is associated with a single return code. A nonzero reason code can be associated with a zero return code.

User Defined Extensions (UDX) return reason codes in the range of 20480 (X'5000') - 24575 (X'5FFF').

The remainder of this topic lists the reason codes that accompany each of the return codes. The return codes are shown in decimal form and the reason codes are shown in decimal and in hexadecimal (hex) form.

**Reason codes that accompany return code 0**

Reason codes that accompany return code 0.

These codes are listed in Table 226.

<table>
<thead>
<tr>
<th>Return code, decimal</th>
<th>Reason code, decimal (hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000 (000)</td>
<td>The verb completed processing successfully.</td>
</tr>
<tr>
<td>0</td>
<td>002 (002)</td>
<td>One or more bytes of a key do not have odd parity.</td>
</tr>
<tr>
<td>0</td>
<td>008 (008)</td>
<td>No value is present to be processed.</td>
</tr>
<tr>
<td>0</td>
<td>151 (097)</td>
<td>The key token supplies the MAC length or MACLEN4 is the default for key tokens that contain MAC or MACVER keys.</td>
</tr>
<tr>
<td>0</td>
<td>701 (2BD)</td>
<td>A new master-key value has duplicate thirds.</td>
</tr>
<tr>
<td>0</td>
<td>702 (2BE)</td>
<td>A provided master-key part does not have odd parity. See “Master Key Process (CSNBMKP)” on page 157 about parity requirements for master key parts.</td>
</tr>
<tr>
<td>0</td>
<td>2013 (7DD)</td>
<td>The Pending Change Buffer (PCB) is empty. This return code and reason code pair applies only to IBM Z.</td>
</tr>
<tr>
<td>0</td>
<td>2146 (862)</td>
<td>A weaker key was used to wrap a stronger key and the Warn when weak wrap - Transport keys command (offset X'032C') was enabled in the active role.</td>
</tr>
<tr>
<td>0</td>
<td>2173 (87D)</td>
<td>The specified payload format version for the output key token matches the payload format version of the input key token.</td>
</tr>
<tr>
<td>0</td>
<td>3010 (BC2)</td>
<td>This card is currently disabled. A card is placed in this state so that it can be moved from one piece of hardware to another, while keeping its secret keys and master keys intact. Normally, when a card has been moved a ‘tamper’ event is recorded and all secrets are erased. A TKE workstation is typically required to put a card in this state and to remove it from this state after the card is installed on the new hardware. This return code and reason code pair applies only to IBM Z.</td>
</tr>
<tr>
<td>0</td>
<td>10001 (2711)</td>
<td>A key encrypted under the old master key was used.</td>
</tr>
<tr>
<td>0</td>
<td>10002 (2712)</td>
<td>A fully qualified dataset name is longer than 64 bytes and the environment variable CSU:xxxLD is not defined (where xxx is either AES, DES, or PKA). The current directory has been abbreviated as a single dot (period).</td>
</tr>
<tr>
<td>0</td>
<td>10003 (2713)</td>
<td>A fully qualified dataset name is longer than 64 bytes and the environment variable CSU:xxxLD is defined (where xxx is either AES, DES, or PKA). Only the dataset name is returned. Use the CSU:xxxLD environment variable to determine the fully qualified dataset name.</td>
</tr>
</tbody>
</table>
Reason codes that accompany return code 4

Reason codes that accompany return code 4

These codes are listed in Table 227.

Table 227. Reason codes for return code 4

<table>
<thead>
<tr>
<th>Return code, decimal</th>
<th>Reason code, decimal (hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>001 (001)</td>
<td>The verification test failed.</td>
</tr>
<tr>
<td>4</td>
<td>013 (00D)</td>
<td>The key token has an initialization vector and the initialization_vector parameter value is nonzero. The verb uses the value in the key token.</td>
</tr>
<tr>
<td>4</td>
<td>016 (010)</td>
<td>The rule_array and the rule_array_count are too small to contain the complete result.</td>
</tr>
<tr>
<td>4</td>
<td>017 (011)</td>
<td>The requested ID is not present in any profile in the specified cryptographic hardware component.</td>
</tr>
<tr>
<td>4</td>
<td>019 (013)</td>
<td>The financial PIN in a PIN block is not verified.</td>
</tr>
<tr>
<td>4</td>
<td>158 (09E)</td>
<td>The verb did not process any key records.</td>
</tr>
<tr>
<td>4</td>
<td>166 (0A6)</td>
<td>The control-vector is not valid because of parity bits, anti-variant bits, inconsistent KEK bits or because bits 59 - 62 are not zero.</td>
</tr>
<tr>
<td>4</td>
<td>179 (0B3)</td>
<td>The control-vector keywords in the rule_array are ignored.</td>
</tr>
<tr>
<td>4</td>
<td>195 (C3)</td>
<td>The key or key-part rule keyword provided does not match the length of the key in the enciphered key token. The output is based on the length specified, and not on the actual key length.</td>
</tr>
<tr>
<td>4</td>
<td>283 (11B)</td>
<td>The coprocessor battery is low.</td>
</tr>
<tr>
<td>4</td>
<td>287 (11F)</td>
<td>The PIN-block format is not consistent.</td>
</tr>
<tr>
<td>4</td>
<td>429 (1AD)</td>
<td>The digital signature is not verified. The verb completed its processing normally.</td>
</tr>
<tr>
<td>4</td>
<td>877 (36D)</td>
<td>The sub-CA certificate loaded has a later expiration (validity:NotAfter field) than its issuer certificate.</td>
</tr>
<tr>
<td>4</td>
<td>937 (3A9)</td>
<td>The UTF8 name compare was done via memcmp.</td>
</tr>
<tr>
<td>4</td>
<td>945 (3B1)</td>
<td>The requested hash method differs from the hash method used when the certificate was loaded and activated.</td>
</tr>
<tr>
<td>4</td>
<td>963 (3C3)</td>
<td>The OA request requires currently unavailable resources. This is a transient condition. Please retry.</td>
</tr>
<tr>
<td>4</td>
<td>1024 (400)</td>
<td>Sufficient shares have been processed to create a new master key.</td>
</tr>
<tr>
<td>4</td>
<td>2039 (7F7)</td>
<td>At least one control vector bit cannot be parsed.</td>
</tr>
<tr>
<td>4</td>
<td>2042 (7FA)</td>
<td>The supplied passphrase is not valid.</td>
</tr>
<tr>
<td>4</td>
<td>2133 (855)</td>
<td>The verb_data value identifies one or more PIN decimalization tables to be deleted that are not stored on the coprocessor. All PIN tables that were requested to be deleted are removed.</td>
</tr>
<tr>
<td>4</td>
<td>2162 (872)</td>
<td>At least two of the key parts of a new operational or master key have identical parts and a warning has been requested by the setting of an appropriate access control point.</td>
</tr>
</tbody>
</table>

Reason codes that accompany return code 8

Reason codes that accompany return code 8.

The codes are listed in Table 228 on page 828.
Table 228. Reason codes for return code 8

<table>
<thead>
<tr>
<th>Return code, decimal</th>
<th>Reason code, decimal (hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>012 (00C)</td>
<td>The token-validation value in an external key token is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>022 (016)</td>
<td>The ID number in the request field is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>023 (017)</td>
<td>An access to the data area is outside the data-area boundary.</td>
</tr>
<tr>
<td>8</td>
<td>024 (018)</td>
<td>The master key verification pattern is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>025 (019)</td>
<td>The value that the text_length parameter specifies is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>026 (01A)</td>
<td>The value of the PIN is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>029 (01D)</td>
<td>The token-validation value in an internal key token is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>030 (01E)</td>
<td>No record with a matching key label is in key storage.</td>
</tr>
<tr>
<td>8</td>
<td>031 (01F)</td>
<td>The control vector does not specify a DATA key.</td>
</tr>
<tr>
<td>8</td>
<td>032 (020)</td>
<td>A key label format is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>033 (021)</td>
<td>A rule_array or other parameter specifies a keyword that is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>034 (022)</td>
<td>A rule_array keyword combination is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>035 (023)</td>
<td>A rule_array_count is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>036 (024)</td>
<td>The action command must be specified in the rule_array.</td>
</tr>
<tr>
<td>8</td>
<td>037 (025)</td>
<td>The object type must be specified in the rule_array.</td>
</tr>
<tr>
<td>8</td>
<td>039 (027)</td>
<td>A control vector violation occurred. Check all control vectors employed with the verb. For security reasons, no detail is provided.</td>
</tr>
<tr>
<td>8</td>
<td>040 (028)</td>
<td>The service code does not contain numerical character data.</td>
</tr>
<tr>
<td>8</td>
<td>041 (029)</td>
<td>The keyword supplied with the key_form parameter is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>042 (02A)</td>
<td>The expiration date is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>043 (02B)</td>
<td>The keyword supplied with the key_length or the key_token_length parameter is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>044 (02C)</td>
<td>A record with a matching key label already exists in key storage.</td>
</tr>
<tr>
<td>8</td>
<td>045 (02D)</td>
<td>The input character string cannot be found in the code table.</td>
</tr>
<tr>
<td>8</td>
<td>046 (02E)</td>
<td>The card-validation value (CVV) is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>047 (02F)</td>
<td>A source key token is unusable because it contains data that is not valid or is undefined.</td>
</tr>
<tr>
<td>8</td>
<td>048 (030)</td>
<td>One or more keys has a master key verification pattern that is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>049 (031)</td>
<td>A key-token-version-number found in a key token is not supported.</td>
</tr>
<tr>
<td>8</td>
<td>050 (032)</td>
<td>The key-serial-number specified in the rule_array is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>051 (033)</td>
<td>The value that the text_length parameter specifies is not a multiple of eight bytes.</td>
</tr>
<tr>
<td>8</td>
<td>054 (036)</td>
<td>The value that the pad_character parameter specifies is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>055 (037)</td>
<td>The initialization vector in the key token is enciphered.</td>
</tr>
<tr>
<td>8</td>
<td>056 (038)</td>
<td>The master key verification pattern in the OCV is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>058 (03A)</td>
<td>The parity of the operating key is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>059 (03B)</td>
<td>Control information (for example, the processing method or the pad character) in the key token conflicts with that in the rule_array.</td>
</tr>
<tr>
<td>8</td>
<td>060 (03C)</td>
<td>A cryptographic request with the FIRST or MIDDLE keywords and a text length less than eight bytes is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>061 (03D)</td>
<td>The keyword supplied with the key_type parameter is not valid.</td>
</tr>
</tbody>
</table>
Table 228. Reason codes for return code 8 (continued)

<table>
<thead>
<tr>
<th>Return code, decimal</th>
<th>Reason code, decimal (hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>062 (03E)</td>
<td>The source key is not present.</td>
</tr>
<tr>
<td>8</td>
<td>063 (03F)</td>
<td>A key token has an invalid token header (for example, not an internal token).</td>
</tr>
<tr>
<td>8</td>
<td>064 (040)</td>
<td>The RSA key is not permitted to perform the requested operation. Likely cause is key distribution usage is not enabled for the key.</td>
</tr>
<tr>
<td>8</td>
<td>065 (041)</td>
<td>The key token failed consistency checking.</td>
</tr>
<tr>
<td>8</td>
<td>066 (042)</td>
<td>The recovered encryption block failed validation checking.</td>
</tr>
<tr>
<td>8</td>
<td>067 (043)</td>
<td>RSA encryption failed.</td>
</tr>
<tr>
<td>8</td>
<td>068 (044)</td>
<td>RSA decryption failed.</td>
</tr>
<tr>
<td>8</td>
<td>070 (046)</td>
<td>An invalid block identifier (identifier tag) was found. Either a block ID (identifier tag) that was proprietary was found, a reserved block ID was used, a duplicate block ID was found, or the specified optional block in the TR-31 key block could not be found.</td>
</tr>
<tr>
<td>8</td>
<td>072 (048)</td>
<td>The value that the size parameter specifies is not valid (too small, too large, negative, or zero).</td>
</tr>
<tr>
<td>8</td>
<td>085 (055)</td>
<td>The date or the time value is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>090 (05A)</td>
<td>Access control checking failed. See the Required Commands descriptions for the failing verb.</td>
</tr>
<tr>
<td>8</td>
<td>091 (05B)</td>
<td>The time that was sent in your logon request was more than five minutes different from the clock in the secure module.</td>
</tr>
<tr>
<td>8</td>
<td>092 (05C)</td>
<td>The user profile is expired.</td>
</tr>
<tr>
<td>8</td>
<td>093 (05D)</td>
<td>The user profile has not yet reached its activation date.</td>
</tr>
<tr>
<td>8</td>
<td>094 (05E)</td>
<td>The authentication data (for example, passphrase) is expired.</td>
</tr>
<tr>
<td>8</td>
<td>095 (05F)</td>
<td>Access to the data is not authorized.</td>
</tr>
<tr>
<td>8</td>
<td>096 (060)</td>
<td>An error occurred reading or writing the secure clock.</td>
</tr>
<tr>
<td>8</td>
<td>100 (064)</td>
<td>The PIN length is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>101 (065)</td>
<td>The PIN check length is not valid. It must be in the range from 4 to the PIN length inclusive.</td>
</tr>
<tr>
<td>8</td>
<td>102 (066)</td>
<td>The value of the decimalization table is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>103 (067)</td>
<td>The value of the validation data is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>104 (068)</td>
<td>The value of the customer-selected PIN is not valid or the PIN length does not match the value supplied with the PIN_length parameter or defined by the PIN-block format specified in the PIN profile.</td>
</tr>
<tr>
<td>8</td>
<td>105 (069)</td>
<td>The value of the transaction_security parameter is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>106 (06A)</td>
<td>The PIN-block format keyword is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>107 (06B)</td>
<td>The format control keyword is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>108 (06C)</td>
<td>The value or the placement of the padding data is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>109 (06D)</td>
<td>The extraction method keyword is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>110 (06E)</td>
<td>The value of the PAN data is not numeric character data.</td>
</tr>
<tr>
<td>8</td>
<td>111 (06F)</td>
<td>The sequence number is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>112 (070)</td>
<td>The PIN offset is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>114 (072)</td>
<td>The PVV value is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>116 (074)</td>
<td>The clear PIN value is not valid. For example, digits other than 0 - 9 were found.</td>
</tr>
</tbody>
</table>
Table 228. Reason codes for return code 8 (continued)

<table>
<thead>
<tr>
<th>Return code, decimal</th>
<th>Reason code, decimal (hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>118 (76)</td>
<td>The issuer domestic code is invalid. This value must be five alphanumeric characters.</td>
</tr>
<tr>
<td>8</td>
<td>120 (078)</td>
<td>An origin or destination identifier is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>121 (079)</td>
<td>The value of the inbound_key or source_key parameter is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>122 (07A)</td>
<td>The value of the inbound_kek_count or outbound_count parameter is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>125 (07D)</td>
<td>A PKA92-encrypted key having the same Environment Identifier (EID) as the local node cannot be imported.</td>
</tr>
<tr>
<td>8</td>
<td>129 (081)</td>
<td>Required rule-array keyword not found.</td>
</tr>
<tr>
<td>8</td>
<td>153 (099)</td>
<td>The text length exceeds the system limits.</td>
</tr>
<tr>
<td>8</td>
<td>154 (09A)</td>
<td>The key token the key_identifier parameter specifies is not an internal key-token or a key label.</td>
</tr>
<tr>
<td>8</td>
<td>155 (09B)</td>
<td>The value that the generated_key_identifier parameter specifies is not valid or it is not consistent with the value that the key_form parameter specifies.</td>
</tr>
<tr>
<td>8</td>
<td>156 (09C)</td>
<td>A keyword is not valid with the specified parameters.</td>
</tr>
<tr>
<td>8</td>
<td>157 (09D)</td>
<td>The key-token type is not specified in the rule_array.</td>
</tr>
<tr>
<td>8</td>
<td>159 (09F)</td>
<td>The keyword supplied with the option parameter is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>160 (0A0)</td>
<td>The key type and the key length are not consistent.</td>
</tr>
<tr>
<td>8</td>
<td>161 (0A1)</td>
<td>The value that the dataset_name_length parameter specifies is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>162 (0A2)</td>
<td>The offset value is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>163 (0A3)</td>
<td>The value that the dataset_name parameter specifies is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>164 (0A4)</td>
<td>The starting address of the output area falls inside the input area.</td>
</tr>
<tr>
<td>8</td>
<td>165 (0A5)</td>
<td>The carry_over_character_count specified in the chaining vector is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>168 (0A8)</td>
<td>A hexadecimal MAC value contains characters that are not valid or the MAC, on a request or reply failed, because the user session key in the host and the adapter card do not match.</td>
</tr>
<tr>
<td>8</td>
<td>169 (0A9)</td>
<td>Specific to MDC Generate, indicates that the length of the text supplied is not correct, either not long enough for the algorithm parameters used or not the correct multiple (must be multiple of eight bytes).</td>
</tr>
<tr>
<td>8</td>
<td>170 (0AA)</td>
<td>Special authorization through the operating system is required to use this verb.</td>
</tr>
<tr>
<td>8</td>
<td>171 (0AB)</td>
<td>The control_array_count value is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>175 (0AF)</td>
<td>The key token cannot be parsed because no control vector is present.</td>
</tr>
<tr>
<td>8</td>
<td>180 (0B4)</td>
<td>A key token presented for parsing is null.</td>
</tr>
<tr>
<td>8</td>
<td>181 (0B5)</td>
<td>The key token is not valid. The first byte is not valid or an incorrect token type was presented.</td>
</tr>
<tr>
<td>8</td>
<td>183 (0B7)</td>
<td>The key type is not consistent with the key type of the control vector.</td>
</tr>
<tr>
<td>8</td>
<td>184 (0B8)</td>
<td>An input pointer is null.</td>
</tr>
<tr>
<td>8</td>
<td>185 (0B9)</td>
<td>A disk I/O error occurred: perhaps the file is in-use, does not exist, and so forth.</td>
</tr>
<tr>
<td>8</td>
<td>186 (0BA)</td>
<td>The key-type field in the control vector is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>187 (0BB)</td>
<td>The requested MAC length (MACLEN4, MACLEN6, MACLEN8) is not consistent with the control vector (key-A, key-B).</td>
</tr>
<tr>
<td>8</td>
<td>191 (0BF)</td>
<td>The requested MAC length (MACLEN6, MACLEN8) is not consistent with the control vector (MAC-LN-4).</td>
</tr>
</tbody>
</table>
### Table 228. Reason codes for return code 8 (continued)

<table>
<thead>
<tr>
<th>Return code, decimal</th>
<th>Reason code, decimal (hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>192 (0C0)</td>
<td>A key-storage record contains a record validation value that is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>194 (0C2)</td>
<td>A private-key section length is invalid.</td>
</tr>
<tr>
<td>8</td>
<td>199 (0C7)</td>
<td>A public exponent is invalid.</td>
</tr>
<tr>
<td>8</td>
<td>204 (0CC)</td>
<td>A memory allocation failed. This can occur in the host and in the coprocessor. Try closing other host tasks. If the problem persists, contact the IBM support center.</td>
</tr>
<tr>
<td>8</td>
<td>205 (0CD)</td>
<td>The X9.23 ciphering method is not consistent with the use of the CONTINUE keyword.</td>
</tr>
<tr>
<td>8</td>
<td>323 (143)</td>
<td>The ciphering method the Decipher verb used does not match the ciphering method the Encipher verb used.</td>
</tr>
<tr>
<td>8</td>
<td>335 (14F)</td>
<td>Either the specified cryptographic hardware component or the environment cannot implement this function.</td>
</tr>
<tr>
<td>8</td>
<td>340 (154)</td>
<td>One of the input control vectors has odd parity.</td>
</tr>
<tr>
<td>8</td>
<td>343 (157)</td>
<td>Either the data block or the buffer for the block is too small or a variable has caused an attempt to create an internal data structure that is too large.</td>
</tr>
<tr>
<td>8</td>
<td>345 (159)</td>
<td>Insufficient storage space exists for the data in the data block buffer.</td>
</tr>
<tr>
<td>8</td>
<td>374 (176)</td>
<td>Less data was supplied than expected or less data exists than was requested.</td>
</tr>
<tr>
<td>8</td>
<td>377 (179)</td>
<td>A key-storage error occurred.</td>
</tr>
<tr>
<td>8</td>
<td>382 (17E)</td>
<td>A time-limit violation occurred.</td>
</tr>
<tr>
<td>8</td>
<td>385 (181)</td>
<td>The cryptographic hardware component reported that the data passed as part of a command is not valid for that command.</td>
</tr>
<tr>
<td>8</td>
<td>387 (183)</td>
<td>The cryptographic hardware component reported that the user ID or role ID is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>393 (189)</td>
<td>The command was not processed because the profile cannot be used.</td>
</tr>
<tr>
<td>8</td>
<td>394 (18A)</td>
<td>The command was not processed because the expiration date was exceeded.</td>
</tr>
<tr>
<td>8</td>
<td>397 (18D)</td>
<td>The command was not processed because the active profile requires the user to be verified first.</td>
</tr>
<tr>
<td>8</td>
<td>398 (18E)</td>
<td>The command was not processed because the maximum PIN or password failure limit is exceeded.</td>
</tr>
<tr>
<td>8</td>
<td>407 (197)</td>
<td>There is a PIN-block consistency-check-error.</td>
</tr>
<tr>
<td>8</td>
<td>439 (1B7)</td>
<td>Key cannot be completed because all required key parts have not yet been accumulated, or key is already complete.</td>
</tr>
<tr>
<td>8</td>
<td>441 (1B9)</td>
<td>Key part cannot be added because key is complete. The key to be processed should be partial, but the key is not partial according to the control vector or other control bits of the key.</td>
</tr>
<tr>
<td>8</td>
<td>442 (1BA)</td>
<td>DES keys with replicated halves are not allowed.</td>
</tr>
<tr>
<td>8</td>
<td>605 (25D)</td>
<td>The number of output bytes is greater than the number that is permitted.</td>
</tr>
<tr>
<td>8</td>
<td>703 (2BF)</td>
<td>A new master-key value is one of the weak DES keys.</td>
</tr>
<tr>
<td>8</td>
<td>704 (2C0)</td>
<td>A new master key cannot have the same master key version number as the current master-key.</td>
</tr>
<tr>
<td>8</td>
<td>705 (2C1)</td>
<td>Both exporter keys specify the same key-encrypting key.</td>
</tr>
<tr>
<td>8</td>
<td>706 (2C2)</td>
<td>Pad count in deciphered data is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>707 (2C3)</td>
<td>The master-key registers are not in the state required for the requested function.</td>
</tr>
<tr>
<td>Return code, decimal</td>
<td>Reason code, decimal (hex)</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>8</td>
<td>714 (2CA)</td>
<td>A reserved parameter must be a null pointer or an expected value.</td>
</tr>
<tr>
<td>8</td>
<td>715 (2CB)</td>
<td>A parameter that must have a value of zero is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>718 (2CE)</td>
<td>The hash value of the data block in the decrypted RSA-OAEP block does not match the hash of the decrypted data block.</td>
</tr>
<tr>
<td>8</td>
<td>719 (2CF)</td>
<td>The block format (BT) field in the decrypted RSA-OAEP block does not have the correct value.</td>
</tr>
<tr>
<td>8</td>
<td>720 (2D0)</td>
<td>The initial byte (I) in the decrypted RSA-OAEP block does not have a valid value.</td>
</tr>
<tr>
<td>8</td>
<td>721 (2D1)</td>
<td>The V field in the decrypted RSA-OAEP does not have the correct value.</td>
</tr>
<tr>
<td>8</td>
<td>752 (2F0)</td>
<td>The key-storage file path is not usable.</td>
</tr>
<tr>
<td>8</td>
<td>753 (2F1)</td>
<td>Opening the key-storage file failed.</td>
</tr>
<tr>
<td>8</td>
<td>754 (2F2)</td>
<td>An internal call to the key_test command failed.</td>
</tr>
<tr>
<td>8</td>
<td>756 (2F4)</td>
<td>Creation of the key-storage file failed.</td>
</tr>
<tr>
<td>8</td>
<td>760 (2F8)</td>
<td>An RSA-key modulus length in bits or in bytes is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>761 (2F9)</td>
<td>An RSA-key exponent length is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>762 (2FA)</td>
<td>A length in the key value structure is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>763 (2FB)</td>
<td>The section identification number within a key token is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>770 (302)</td>
<td>The PKA key token has a field that is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>771 (303)</td>
<td>The user is not logged on.</td>
</tr>
<tr>
<td>8</td>
<td>772 (304)</td>
<td>The requested role does not exist.</td>
</tr>
<tr>
<td>8</td>
<td>773 (305)</td>
<td>The requested profile does not exist.</td>
</tr>
<tr>
<td>8</td>
<td>774 (306)</td>
<td>The profile already exists.</td>
</tr>
<tr>
<td>8</td>
<td>775 (307)</td>
<td>The supplied data is not replaceable.</td>
</tr>
<tr>
<td>8</td>
<td>776 (308)</td>
<td>The requested ID is already logged on.</td>
</tr>
<tr>
<td>8</td>
<td>777 (309)</td>
<td>The authentication data is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>778 (30A)</td>
<td>The checksum for the role is in error.</td>
</tr>
<tr>
<td>8</td>
<td>779 (30B)</td>
<td>The checksum for the profile is in error.</td>
</tr>
<tr>
<td>8</td>
<td>780 (30C)</td>
<td>There is an error in the profile data.</td>
</tr>
<tr>
<td>8</td>
<td>781 (30D)</td>
<td>There is an error in the role data.</td>
</tr>
<tr>
<td>8</td>
<td>782 (30E)</td>
<td>The function-control-vector header is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>783 (30F)</td>
<td>The command is not permitted by the function-control-vector value.</td>
</tr>
<tr>
<td>8</td>
<td>784 (310)</td>
<td>The operation you requested cannot be performed because the user profile is in use.</td>
</tr>
<tr>
<td>8</td>
<td>785 (311)</td>
<td>The operation you requested cannot be performed because the role is in use.</td>
</tr>
<tr>
<td>8</td>
<td>786 (312)</td>
<td>A profile load is being attempted for a profile that has the same authentication key as another profile in the domain that already exists.</td>
</tr>
<tr>
<td>8</td>
<td>787 (313)</td>
<td>A profile load is being attempted for a profile that has a key that is too weak for the compliance level of the domain where the profile is being loaded.</td>
</tr>
<tr>
<td>8</td>
<td>816 (330)</td>
<td>The public-key certificate length is invalid.</td>
</tr>
<tr>
<td>8</td>
<td>817 (331)</td>
<td>The public key does not match.</td>
</tr>
<tr>
<td>8</td>
<td>818 (332)</td>
<td>The signature of the input public-key certificate does not verify.</td>
</tr>
<tr>
<td>8</td>
<td>819 (333)</td>
<td>The public-key certificate type is invalid or not allowed.</td>
</tr>
<tr>
<td>Return code, decimal</td>
<td>Reason code, decimal (hex)</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>8</td>
<td>821 (335)</td>
<td>The subject name provided is either missing, malformed, or of invalid length.</td>
</tr>
<tr>
<td>8</td>
<td>822 (336)</td>
<td>The issuer name provided is either missing, malformed, or of invalid length.</td>
</tr>
<tr>
<td>8</td>
<td>823 (337)</td>
<td>The serial number provided is either unexpected, missing, malformed, or of invalid length.</td>
</tr>
<tr>
<td>8</td>
<td>825 (339)</td>
<td>The extension data provided is either unexpected, missing, malformed, or of invalid length.</td>
</tr>
<tr>
<td>8</td>
<td>826 (33A)</td>
<td>The expiration days value provided is either unexpected, missing, or out of range.</td>
</tr>
<tr>
<td>8</td>
<td>827 (33B)</td>
<td>The pathLenConstraint provided is either unexpected, missing, or out of range.</td>
</tr>
<tr>
<td>8</td>
<td>829 (33D)</td>
<td>Error in GSK/SSL/ASN.1 processing.</td>
</tr>
<tr>
<td>8</td>
<td>830 (33E)</td>
<td>Error in ASN.1 processing. No more data.</td>
</tr>
<tr>
<td>8</td>
<td>831 (33F)</td>
<td>Error in ASN.1 processing. Length value is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>833 (341)</td>
<td>Error in ASN.1 processing. Attribute value separator is missing.</td>
</tr>
<tr>
<td>8</td>
<td>834 (342)</td>
<td>Error in ASN.1 processing. Unknown attribute identifier.</td>
</tr>
<tr>
<td>8</td>
<td>835 (343)</td>
<td>Error in ASN.1 processing. Object identifier syntax error.</td>
</tr>
<tr>
<td>8</td>
<td>837 (345)</td>
<td>Error in ASN.1 processing. Interval is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>838 (346)</td>
<td>Error in ASN.1 processing. X.500 name syntax error.</td>
</tr>
<tr>
<td>8</td>
<td>839 (347)</td>
<td>Error in ASN.1 processing. Data type is not correct.</td>
</tr>
<tr>
<td>8</td>
<td>841 (349)</td>
<td>Error in ASN.1 processing. Character string cannot be converted.</td>
</tr>
<tr>
<td>8</td>
<td>842 (34A)</td>
<td>Error in ASN.1 processing. Indefinite-length encoding is not supported.</td>
</tr>
<tr>
<td>8</td>
<td>843 (34B)</td>
<td>Error in ASN.1 processing. Data element must be constructed.</td>
</tr>
<tr>
<td>8</td>
<td>845 (34D)</td>
<td>Error in ASN.1 processing. Data element must be an ASN.1 primitive.</td>
</tr>
<tr>
<td>8</td>
<td>846 (34E)</td>
<td>Error in ASN.1 processing. Indefinite-length encoding is not allowed.</td>
</tr>
<tr>
<td>8</td>
<td>847 (34F)</td>
<td>Error in ASN.1 processing. Data encoding is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>849 (351)</td>
<td>Error in ASN.1 processing. Data value overflow.</td>
</tr>
<tr>
<td>8</td>
<td>850 (352)</td>
<td>Error in ASN.1 processing. Unused bit count is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>851 (353)</td>
<td>Error in ASN.1 processing. Unused bit count is not valid for a segmented bit string.</td>
</tr>
<tr>
<td>8</td>
<td>853 (355)</td>
<td>Error in ASN.1 processing. Required data element is missing.</td>
</tr>
<tr>
<td>8</td>
<td>854 (356)</td>
<td>Error in ASN.1 processing. Excess data found at end of data element.</td>
</tr>
<tr>
<td>8</td>
<td>855 (357)</td>
<td>Error in ASN.1 processing. Parameter is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>857 (359)</td>
<td>Error in ASN.1 processing. Data value is not present.</td>
</tr>
<tr>
<td>8</td>
<td>858 (35A)</td>
<td>Error in ASN.1 processing. Selection is not within the valid range.</td>
</tr>
<tr>
<td>8</td>
<td>859 (35B)</td>
<td>Error in ASN.1 processing. No selection found.</td>
</tr>
<tr>
<td>8</td>
<td>861 (35D)</td>
<td>Error in ASN.1 processing. Syntax already set.</td>
</tr>
<tr>
<td>8</td>
<td>862 (35E)</td>
<td>Error in ASN.1 processing. Codeset is not allowed.</td>
</tr>
<tr>
<td>8</td>
<td>863 (35F)</td>
<td>Error in ASN.1 processing. Attribute value is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>865 (361)</td>
<td>Error in ASN.1 processing. Attribute value is missing.</td>
</tr>
<tr>
<td>8</td>
<td>866 (362)</td>
<td>Error in ASN.1 processing. Object identifier element count is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>867 (363)</td>
<td>Error in ASN.1 processing. Incorrect value for the first object identifier element.</td>
</tr>
<tr>
<td>8</td>
<td>869 (365)</td>
<td>Error in ASN.1 processing. Incorrect value for the second object identifier element.</td>
</tr>
</tbody>
</table>
Table 228. Reason codes for return code 8 (continued)

<table>
<thead>
<tr>
<th>Return code, decimal</th>
<th>Reason code, decimal (hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>870 (366)</td>
<td>Error in ASN.1 processing. Version is not supported.</td>
</tr>
<tr>
<td>8</td>
<td>873 (369)</td>
<td>The extension data provided conflicts with the rule array data provided.</td>
</tr>
<tr>
<td>8</td>
<td>874 (36A)</td>
<td>Error in Cert processing. Elliptic Curve is not supported.</td>
</tr>
<tr>
<td>8</td>
<td>875 (36B)</td>
<td>Error in Cert processing. Signature not supplied.</td>
</tr>
<tr>
<td>8</td>
<td>878 (36E)</td>
<td>Error in Cert processing. Cryptographic algorithm is not supported.</td>
</tr>
<tr>
<td>8</td>
<td>879 (36F)</td>
<td>Error in Cert processing. Incorrect Base64 encoding.</td>
</tr>
<tr>
<td>8</td>
<td>881 (371)</td>
<td>Error in Cert processing. Unrecognized file or message encoding.</td>
</tr>
<tr>
<td>8</td>
<td>882 (372)</td>
<td>The HSM internal clock has not been set.</td>
</tr>
<tr>
<td>8</td>
<td>883 (373)</td>
<td>Error in Cert processing. Key not supported by encryption or signature algorithm.</td>
</tr>
<tr>
<td>8</td>
<td>885 (375)</td>
<td>The X.509 certificate presented has an invalid, or missing KeyUsage extension.</td>
</tr>
<tr>
<td>8</td>
<td>886 (376)</td>
<td>Error in Cert processing. Certificate extension is not supported.</td>
</tr>
<tr>
<td>8</td>
<td>887 (377)</td>
<td>Error in Cert processing. Signature is not correct.</td>
</tr>
<tr>
<td>8</td>
<td>889 (379)</td>
<td>Error in Cert processing. Input/Output request failed.</td>
</tr>
<tr>
<td>8</td>
<td>890 (37A)</td>
<td>Error in Cert processing. Database is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>891 (37B)</td>
<td>Error in Cert processing. Handle is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>893 (37D)</td>
<td>Error in Cert processing. Certificate extension data has an incorrect critical indicator.</td>
</tr>
<tr>
<td>8</td>
<td>894 (37E)</td>
<td>Error in Cert processing. Required certificate extension is missing.</td>
</tr>
<tr>
<td>8</td>
<td>895 (37F)</td>
<td>Error in Cert processing. Certificate not valid for host.</td>
</tr>
<tr>
<td>8</td>
<td>897 (381)</td>
<td>Error in Cert processing. Subject name is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>898 (382)</td>
<td>Error in Cert processing. Certificate extension data is incorrect.</td>
</tr>
<tr>
<td>8</td>
<td>899 (383)</td>
<td>Error in Cert processing. Validation option is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>901 (385)</td>
<td>Error in Cert processing. Name constraints violated.</td>
</tr>
<tr>
<td>8</td>
<td>902 (386)</td>
<td>Error in Cert processing. Record not found.</td>
</tr>
<tr>
<td>8</td>
<td>903 (387)</td>
<td>Error in Cert processing. Certificate chain is not trusted.</td>
</tr>
<tr>
<td>8</td>
<td>905 (389)</td>
<td>Error in Cert processing. Required basic constraints certificate extension is missing.</td>
</tr>
<tr>
<td>8</td>
<td>906 (38A)</td>
<td>Error in Cert processing. An internal error has occurred.</td>
</tr>
<tr>
<td>8</td>
<td>907 (38B)</td>
<td>Error in Cert processing. Issuer certificate not found.</td>
</tr>
<tr>
<td>8</td>
<td>909 (38D)</td>
<td>Error in Cert processing. Name format is not supported.</td>
</tr>
<tr>
<td>8</td>
<td>910 (38E)</td>
<td>Error in Cert processing. Self-signed certificate not in database.</td>
</tr>
<tr>
<td>8</td>
<td>911 (38F)</td>
<td>Error in Cert processing. Certificate is expired.</td>
</tr>
<tr>
<td>8</td>
<td>913 (391)</td>
<td>Error in Cert processing. Certificate is not yet valid.</td>
</tr>
<tr>
<td>8</td>
<td>914 (392)</td>
<td>Error in Cert processing. Issuer name is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>915 (393)</td>
<td>Error in Cert processing. Certificate is revoked.</td>
</tr>
<tr>
<td>8</td>
<td>917 (395)</td>
<td>Error in Cert processing. Numeric value is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>918 (396)</td>
<td>Error in Cert processing. Variable argument security level is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>919 (397)</td>
<td>Error in Cert processing. Variable argument validate root is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>921 (399)</td>
<td>Error in Cert processing. Variable argument count is not valid.</td>
</tr>
<tr>
<td>Return code, decimal</td>
<td>Reason code, decimal (hex)</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>8</td>
<td>922 (39A)</td>
<td>Error in Cert processing. Extended key usage comparison failed.</td>
</tr>
<tr>
<td>8</td>
<td>923 (39B)</td>
<td>Error in Cert processing. Certificate does not have an extended key usage extension.</td>
</tr>
<tr>
<td>8</td>
<td>925 (39D)</td>
<td>Error in Cert processing. Extended key usage type is not supported for this operation.</td>
</tr>
<tr>
<td>8</td>
<td>926 (39E)</td>
<td>Error in Cert processing. Extended key usage input is not supplied.</td>
</tr>
<tr>
<td>8</td>
<td>927 (39F)</td>
<td>Error in Cert processing. Extended key usage input count is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>929 (3A1)</td>
<td>Error in Cert processing. Input certificate not supplied.</td>
</tr>
<tr>
<td>8</td>
<td>930 (3A2)</td>
<td>Error in Cert processing. Incorrect key usage.</td>
</tr>
<tr>
<td>8</td>
<td>931 (3A3)</td>
<td>Error in Cert processing. Cannot match CRL distribution points.</td>
</tr>
<tr>
<td>8</td>
<td>933 (3A5)</td>
<td>Error in Cert processing. Acceptable policy intersection cannot be found.</td>
</tr>
<tr>
<td>8</td>
<td>934 (3A6)</td>
<td>Error in Cert processing. Certification path is too long.</td>
</tr>
<tr>
<td>8</td>
<td>935 (3A7)</td>
<td>Error in Cert processing. Issuer is not a certification authority.</td>
</tr>
<tr>
<td>8</td>
<td>939 (3AB)</td>
<td>Certificate presented to load as a certification authority does not have a true value for CA in basic constraints certificate extension.</td>
</tr>
<tr>
<td>8</td>
<td>941 (3AD)</td>
<td>Certificate presented to use as an end entity has a true value for cA in basic constraints certificate extension.</td>
</tr>
<tr>
<td>8</td>
<td>942 (3AE)</td>
<td>Error in Cert processing. Label is not unique.</td>
</tr>
<tr>
<td>8</td>
<td>943 (3AF)</td>
<td>Error in Cert processing. Certificate is not unique.</td>
</tr>
<tr>
<td>8</td>
<td>946 (3B2)</td>
<td>The requested hash method differs from the hash method used when the certificate was loaded.</td>
</tr>
<tr>
<td>8</td>
<td>947 (3B3)</td>
<td>Error in Cert processing. Record label is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>949 (3B5)</td>
<td>Error in Cert processing. Multiple certificates exist for label.</td>
</tr>
<tr>
<td>8</td>
<td>950 (3B6)</td>
<td>Error in Cert processing. Record deleted.</td>
</tr>
<tr>
<td>8</td>
<td>954 (3BA)</td>
<td>Error in Cert processing. Subject name cannot be changed.</td>
</tr>
<tr>
<td>8</td>
<td>955 (3BB)</td>
<td>Error in Cert processing. Public key cannot be changed.</td>
</tr>
<tr>
<td>8</td>
<td>957 (3BD)</td>
<td>The Certificate presented is not in the proper state for the requested operation.</td>
</tr>
<tr>
<td>8</td>
<td>958 (3BE)</td>
<td>The Certificate hash presented does not match the stored hash.</td>
</tr>
<tr>
<td>8</td>
<td>959 (3BF)</td>
<td>Error in Cert processing. Database contains certificates signed by the certificate.</td>
</tr>
<tr>
<td>8</td>
<td>1025 (401)</td>
<td>The registered public key or retained private key name already exists.</td>
</tr>
<tr>
<td>8</td>
<td>1026 (402)</td>
<td>The key name (registered public key or retained private key) does not exist.</td>
</tr>
<tr>
<td>8</td>
<td>1027 (403)</td>
<td>Environment identifier data is already set.</td>
</tr>
<tr>
<td>8</td>
<td>1028 (404)</td>
<td>Master key share data is already set.</td>
</tr>
<tr>
<td>8</td>
<td>1029 (405)</td>
<td>There is an error in the Environment Identifier (EID) data.</td>
</tr>
<tr>
<td>8</td>
<td>1030 (406)</td>
<td>There is an error in using the master key share data.</td>
</tr>
<tr>
<td>8</td>
<td>1031 (407)</td>
<td>There is an error in using registered public key or retained private key data.</td>
</tr>
<tr>
<td>8</td>
<td>1032 (408)</td>
<td>There is an error in using registered public key hash data.</td>
</tr>
<tr>
<td>8</td>
<td>1033 (409)</td>
<td>The public key hash was not registered.</td>
</tr>
<tr>
<td>8</td>
<td>1034 (40A)</td>
<td>The public key was not registered.</td>
</tr>
<tr>
<td>8</td>
<td>1035 (40B)</td>
<td>The public key certificate signature was not verified.</td>
</tr>
<tr>
<td>8</td>
<td>1037 (40D)</td>
<td>There is a master key shares distribution error.</td>
</tr>
<tr>
<td>Return code, decimal</td>
<td>Reason code, decimal (hex)</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>8</td>
<td>1038 (40E)</td>
<td>The public key hash is not marked for cloning.</td>
</tr>
<tr>
<td>8</td>
<td>1039 (40F)</td>
<td>The registered public key hash does not match the registered hash.</td>
</tr>
<tr>
<td>8</td>
<td>1040 (410)</td>
<td>The master key share enciphering key failed encipher.</td>
</tr>
<tr>
<td>8</td>
<td>1041 (411)</td>
<td>The master key share enciphering key failed decipher.</td>
</tr>
<tr>
<td>8</td>
<td>1042 (412)</td>
<td>The master key share digital signature generate failed.</td>
</tr>
<tr>
<td>8</td>
<td>1043 (413)</td>
<td>The master key share digital signature verify failed.</td>
</tr>
<tr>
<td>8</td>
<td>1044 (414)</td>
<td>There is an error in reading VPD data from the adapter.</td>
</tr>
<tr>
<td>8</td>
<td>1045 (415)</td>
<td>Encrypting the cloning information failed.</td>
</tr>
<tr>
<td>8</td>
<td>1046 (416)</td>
<td>Decrypting the cloning information failed.</td>
</tr>
<tr>
<td>8</td>
<td>1047 (417)</td>
<td>There is an error loading the new master key from the master key shares.</td>
</tr>
<tr>
<td>8</td>
<td>1048 (418)</td>
<td>The clone information has one or more sections that are not valid.</td>
</tr>
<tr>
<td>8</td>
<td>1049 (419)</td>
<td>The master key share index is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>1050 (41A)</td>
<td>The public-key encrypted-key is rejected because the Environment Identifier (EID) with the key is the same as the EID for this node.</td>
</tr>
<tr>
<td>8</td>
<td>1051 (41B)</td>
<td>The private key is rejected because the key is not flagged for use in master-key cloning.</td>
</tr>
<tr>
<td>8</td>
<td>1052 (41C)</td>
<td>The token identifier of the trusted block's header section is in the range X'20' - X'FF'. Check the token identifier of the trusted block.</td>
</tr>
<tr>
<td>8</td>
<td>1053 (41D)</td>
<td>The active flag in the trusted block's trusted block section X'14' is not disabled. Use the Trusted Block Create verb to create an inactive/external trusted block.</td>
</tr>
<tr>
<td>8</td>
<td>1054 (41E)</td>
<td>The token identifier of the trusted block's header section is not X'1E' (external). Use the Trusted Block Create verb to create an inactive/external trusted block.</td>
</tr>
<tr>
<td>8</td>
<td>1055 (41F)</td>
<td>The active flag of the trusted block's trusted block section X'14' is not enabled. Use the Trusted Block Create verb to create an active/external trusted block.</td>
</tr>
<tr>
<td>8</td>
<td>1056 (420)</td>
<td>The token identifier of the trusted block's header section is not X'1F' (internal). Use the PKA Key Import verb to import the trusted block.</td>
</tr>
<tr>
<td>8</td>
<td>1057 (421)</td>
<td>The trusted block rule section X'12' rule ID does not match input parameter rule ID. Verify that the trusted block used has the rule section specified.</td>
</tr>
<tr>
<td>8</td>
<td>1058 (422)</td>
<td>The trusted block contains a value that is too small or too large.</td>
</tr>
<tr>
<td>8</td>
<td>1059 (423)</td>
<td>A trusted block parameter that must have a value of zero (or a grouping of bits set to zero) is invalid.</td>
</tr>
<tr>
<td>8</td>
<td>1060 (424)</td>
<td>The trusted block public key section failed consistency checking.</td>
</tr>
<tr>
<td>8</td>
<td>1061 (425)</td>
<td>The trusted block contains extraneous sections or subsections (TLVs). Check the trusted block for undefined sections or subsections.</td>
</tr>
<tr>
<td>8</td>
<td>1062 (426)</td>
<td>The trusted block contains missing sections or subsections (TLVs). Check the trusted block for required sections and subsections applicable to the verb invoked.</td>
</tr>
<tr>
<td>8</td>
<td>1063 (427)</td>
<td>The trusted block contains duplicate sections or subsections (TLVs). Check the trusted block's sections and subsections for duplicates. Multiple rule sections are allowed.</td>
</tr>
<tr>
<td>8</td>
<td>1064 (428)</td>
<td>The trusted block expiration date has expired (as compared to the IBM 4764 clock). Validate the expiration date in the trusted block's trusted information section's Activation and Expiration Date TLV object</td>
</tr>
<tr>
<td>Return code, decimal</td>
<td>Reason code, decimal (hex)</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>8</td>
<td>1065 (429)</td>
<td>The trusted block expiration date is at a date prior to the activation date. Validate the expiration date in the trusted block’s trusted information section’s Activation and Expiration Date TLV object.</td>
</tr>
<tr>
<td>8</td>
<td>1066 (42A)</td>
<td>The trusted block public key modulus length in bits is not consistent with the byte length. The bit length must be less than or equal to byte length * 8 and greater than (byte length - 1) * 8.</td>
</tr>
<tr>
<td>8</td>
<td>1067 (42B)</td>
<td>The trusted block public key modulus length in bits exceeds the maximum allowed bit length, as defined by the Function Control Vector.</td>
</tr>
<tr>
<td>8</td>
<td>1068 (42C)</td>
<td>One or more trusted block sections or TLV objects contained data that is invalid (an example would be invalid label data in label section X’13’).</td>
</tr>
<tr>
<td>8</td>
<td>1069 (42D)</td>
<td>Trusted block verification was attempted by a verb other than CSNDDSV, CSNDKTC, CSNPKI, CSNDRKX, or CSNDTBC.</td>
</tr>
<tr>
<td>8</td>
<td>1070 (42E)</td>
<td>The trusted block rule ID contained within a rule section has invalid characters.</td>
</tr>
<tr>
<td>8</td>
<td>1071 (42F)</td>
<td>The source key’s length or CV does not match what is expected by the rule section in the trusted block that was selected by the rule ID input parameter.</td>
</tr>
<tr>
<td>8</td>
<td>1072 (430)</td>
<td>The activation data is not valid. Validate the activation data in the trusted block’s trusted information section’s Activation and Expiration Date TLV object.</td>
</tr>
<tr>
<td>8</td>
<td>1073 (431)</td>
<td>The source-key label does not match the template in the export key DES token parameters TLV object of the selected trusted block rule section.</td>
</tr>
<tr>
<td>8</td>
<td>1074 (432)</td>
<td>The control-vector value specified in the common export key parameters TLV object in the selected rule section of the trusted block contains a control vector that is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>1075 (433)</td>
<td>The source-key label template in the export key DES token parameters TLV object in the selected rule section of the trusted block contains a label template that is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>1077 (435)</td>
<td>Key wrapping option input error.</td>
</tr>
<tr>
<td>8</td>
<td>1078 (436)</td>
<td>Key wrapping Security Relevant Data Item (SRDI) error.</td>
</tr>
<tr>
<td>8</td>
<td>1079 (437)</td>
<td>The format of the decrypted PIN block is not supported in this function.</td>
</tr>
<tr>
<td>8</td>
<td>1100 (44C)</td>
<td>There is a general hardware device driver execution error.</td>
</tr>
<tr>
<td>8</td>
<td>1101 (44D)</td>
<td>There is a hardware device driver parameter that is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>1102 (44E)</td>
<td>There is a hardware device driver non-valid buffer length.</td>
</tr>
<tr>
<td>8</td>
<td>1103 (44F)</td>
<td>The hardware device driver has too many opens. The device cannot open now.</td>
</tr>
<tr>
<td>8</td>
<td>1104 (450)</td>
<td>The hardware device driver is denied access.</td>
</tr>
<tr>
<td>8</td>
<td>1105 (451)</td>
<td>The hardware device driver device is busy and cannot perform the request now.</td>
</tr>
<tr>
<td>8</td>
<td>1106 (452)</td>
<td>The hardware device driver buffer is too small and the received data is truncated.</td>
</tr>
<tr>
<td>8</td>
<td>1107 (453)</td>
<td>The hardware device driver request is interrupted and the request is aborted.</td>
</tr>
<tr>
<td>8</td>
<td>1108 (454)</td>
<td>The hardware device driver detected a security tamper event.</td>
</tr>
<tr>
<td>8</td>
<td>1114 (45A)</td>
<td>The communications manager detected that the host-supplied buffer for the reply control block is too small.</td>
</tr>
<tr>
<td>8</td>
<td>1115 (45B)</td>
<td>The communications manager detected that the host-supplied buffer for the reply data block is too small.</td>
</tr>
<tr>
<td>8</td>
<td>1117 (45D)</td>
<td>Hardware device driver operation not permitted.</td>
</tr>
<tr>
<td>8</td>
<td>1118 (45E)</td>
<td>Hardware device driver received bad address.</td>
</tr>
<tr>
<td>8</td>
<td>1119 (45F)</td>
<td>Hardware device driver hardware error.</td>
</tr>
<tr>
<td>8</td>
<td>1121 (461)</td>
<td>Hardware device driver firmware error.</td>
</tr>
</tbody>
</table>
Table 228. Reason codes for return code 8 (continued)

<table>
<thead>
<tr>
<th>Return code, decimal</th>
<th>Reason code, decimal (hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1122 (462)</td>
<td>Hardware device driver temperature of out range.</td>
</tr>
<tr>
<td>8</td>
<td>1123 (463)</td>
<td>Hardware device driver received bad request.</td>
</tr>
<tr>
<td>8</td>
<td>1125 (465)</td>
<td>Hardware device driver host timeout.</td>
</tr>
<tr>
<td>8</td>
<td>2034 (7F2)</td>
<td>The environment variable that was used to set the default coprocessor is not valid, or does not exist for a coprocessor in the system.</td>
</tr>
<tr>
<td>8</td>
<td>2036 (7F4)</td>
<td>The contents of a chaining vector are not valid. Ensure the chaining vector was not modified by your application program.</td>
</tr>
<tr>
<td>8</td>
<td>2038 (7F6)</td>
<td>No RSA private key information is provided.</td>
</tr>
<tr>
<td>8</td>
<td>2041 (7F9)</td>
<td>A default card environment variable is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>2050 (802)</td>
<td>The current key serial number field in the PIN profile variable is not valid (not hexadecimal or too many one bits).</td>
</tr>
<tr>
<td>8</td>
<td>2051 (803)</td>
<td>There is a non-valid message length in the OAEP-decoded information.</td>
</tr>
<tr>
<td>8</td>
<td>2053 (805)</td>
<td>No message found in the OAEP-decoded data.</td>
</tr>
<tr>
<td>8</td>
<td>2054 (806)</td>
<td>There is a non-valid RSA Enciphered Key cryptogram: OAEP optional encoding parameters failed validation.</td>
</tr>
<tr>
<td>8</td>
<td>2055 (807)</td>
<td>Based on the hash method and size of the symmetric key specified, the RSA public key size is too small to format the symmetric key into a PKOAEP2 message.</td>
</tr>
<tr>
<td>8</td>
<td>2062 (80E)</td>
<td>The active role does not permit you to change the characteristic of a double-length key in the key_Part_Import parameter.</td>
</tr>
<tr>
<td>8</td>
<td>2065 (811)</td>
<td>The specified key token is not null.</td>
</tr>
<tr>
<td>8</td>
<td>2080 (820)</td>
<td>The group profile was not found.</td>
</tr>
<tr>
<td>8</td>
<td>2081 (821)</td>
<td>The group has duplicate elements.</td>
</tr>
<tr>
<td>8</td>
<td>2082 (822)</td>
<td>The group profile is not in the group.</td>
</tr>
<tr>
<td>8</td>
<td>2083 (823)</td>
<td>The group has the wrong user ID count.</td>
</tr>
<tr>
<td>8</td>
<td>2084 (824)</td>
<td>The group user ID failed.</td>
</tr>
<tr>
<td>8</td>
<td>2085 (825)</td>
<td>The profile is not in the specified group.</td>
</tr>
<tr>
<td>8</td>
<td>2086 (826)</td>
<td>The group role was not found.</td>
</tr>
<tr>
<td>8</td>
<td>2087 (827)</td>
<td>The group profile has not been activated.</td>
</tr>
<tr>
<td>8</td>
<td>2088 (828)</td>
<td>The expiration date of the group profile has been reached or exceeded.</td>
</tr>
<tr>
<td>8</td>
<td>2089 (829)</td>
<td>The verb contains multiple keywords or parameters that indicate the algorithm to be used, and at least one of these specifies a different algorithm from the others.</td>
</tr>
<tr>
<td>8</td>
<td>2090 (82A)</td>
<td>A required SRDI was not found.</td>
</tr>
<tr>
<td>8</td>
<td>2091 (82B)</td>
<td>A required CA SRDI was not found.</td>
</tr>
<tr>
<td>8</td>
<td>2093 (82D)</td>
<td>Specific toIBM Z - an AES key is encrypted under a DES master key, which is not acceptable for the requested operation.</td>
</tr>
<tr>
<td>8</td>
<td>2095 (82F)</td>
<td>The key_form is incompatible with the key_type.</td>
</tr>
<tr>
<td>8</td>
<td>2097 (831)</td>
<td>The key_length is incompatible with the key_type.</td>
</tr>
<tr>
<td>8</td>
<td>2098 (832)</td>
<td>Either a key bit length that was not valid was found in an AES key token (length not 128, 192, or 256 bits) or a version X'01' DES token had a token-marks field that was not valid.</td>
</tr>
<tr>
<td>8</td>
<td>2099 (833)</td>
<td>Invalid encrypted key length in the AES token, when an encrypted key is present.</td>
</tr>
<tr>
<td>Return code, decimal</td>
<td>Reason code, decimal (hex)</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>8</td>
<td>2106 (83A)</td>
<td>An input/output error occurred while accessing the logged on users table.</td>
</tr>
<tr>
<td>8</td>
<td>2110 (83E)</td>
<td>Invalid wrapping type.</td>
</tr>
<tr>
<td>8</td>
<td>2111 (83F)</td>
<td>Control vector enhanced bit (bit 56) conflicts with key wrapping keyword.</td>
</tr>
<tr>
<td>8</td>
<td>2113 (841)</td>
<td>A key token contains invalid payload.</td>
</tr>
<tr>
<td>8</td>
<td>2114 (842)</td>
<td>Clear-key bit length is out of range.</td>
</tr>
<tr>
<td>8</td>
<td>2115 (843)</td>
<td>Input key token cannot have a key present when importing the first key part; skeleton key token is required.</td>
</tr>
<tr>
<td>8</td>
<td>2118 (846)</td>
<td>One or more invalid values in the TR-31 key block header.</td>
</tr>
<tr>
<td>8</td>
<td>2119 (847)</td>
<td>The &quot;mode&quot; value in the TR-31 header is invalid or is not acceptable in the chosen operation.</td>
</tr>
<tr>
<td>8</td>
<td>2121 (849)</td>
<td>The &quot;algorithm&quot; value in the TR-31 header is invalid or is not acceptable in the chosen operation.</td>
</tr>
<tr>
<td>8</td>
<td>2122 (84A)</td>
<td>For import, the exportability byte in the TR-31 header contains a value that does not support import of the key into CCA. For export, the requested exportability does not match circumstances (for example, a 'B' Key Block Version ID key can be wrapped only by a KEK that is wrapped in CBC mode, the ECB mode KEK violates ANSI X9.24).</td>
</tr>
<tr>
<td>8</td>
<td>2123 (84B)</td>
<td>The length of the cleartext key in the TR-31 block is invalid (for example, the algorithm is 'D' for single-length DES, but the key length is not 64 bits).</td>
</tr>
<tr>
<td>8</td>
<td>2125 (84D)</td>
<td>The Key Block Version ID in the TR-31 header contains an invalid value.</td>
</tr>
<tr>
<td>8</td>
<td>2126 (84E)</td>
<td>The key-usage field in the TR-31 header contains a value that is not supported for import of the key into CCA.</td>
</tr>
<tr>
<td>8</td>
<td>2127 (84F)</td>
<td>The key-usage field in the TR-31 header contains a value that is not valid with the other parameters in the header.</td>
</tr>
<tr>
<td>8</td>
<td>2129 (851)</td>
<td>Either a parameter for building a TR-31 key block (a TR-31 key block or a component, such as a tag for an optional block) contains one or more ASCII characters that are not printable as described in TR-31, or a field contains ASCII characters that are not allowed for that field.</td>
</tr>
<tr>
<td>8</td>
<td>2130 (852)</td>
<td>The control vector carried in the optional blocks of the TR-31 key block is inconsistent with other attributes of the key.</td>
</tr>
<tr>
<td>8</td>
<td>2131 (853)</td>
<td>The TR-31 key-token failed the MAC validate step of the Key Block unwrap and verify steps (for either Key Block Version ID method). MAC validation failed for a parameter in a key block, such as a trusted block or a TR-31 key block. This might be the result of tampering, corruption, or using a validation key that is different from the one used to generate the MAC.</td>
</tr>
<tr>
<td>8</td>
<td>2134 (856)</td>
<td>No valid PIN decimalization tables are present.</td>
</tr>
<tr>
<td>8</td>
<td>2135 (857)</td>
<td>The PIN decimalization table provided as input is not allowed to be used because it does not match any of the active tables stored on the coprocessor.</td>
</tr>
<tr>
<td>8</td>
<td>2137 (859)</td>
<td>There is an error involving the PIN decimalization table input data. No PIN tables have been changed.</td>
</tr>
<tr>
<td>8</td>
<td>2138 (85A)</td>
<td>At least one of the PIN decimalization tables requested to be activated is empty or already in the active state (not in the loaded state). No PIN tables have been activated.</td>
</tr>
<tr>
<td>8</td>
<td>2139 (85B)</td>
<td>At least one PIN decimalization table provided as input to be activated does not match the corresponding table that is loaded on the coprocessor. No PIN tables have been changed from the loaded state to the active state.</td>
</tr>
<tr>
<td>8</td>
<td>2141 (84D)</td>
<td>The key verification pattern for the key-encrypting key is not valid.</td>
</tr>
<tr>
<td>Return code, decimal</td>
<td>Reason code, decimal (hex)</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>8 2142 (85E)</td>
<td></td>
<td>A key-usage field setting prevents operation.</td>
</tr>
<tr>
<td>8 2143 (85F)</td>
<td></td>
<td>A key-management field setting prevents operation.</td>
</tr>
<tr>
<td>8 2145 (861)</td>
<td></td>
<td>An attempt to wrap a stronger key with a weaker key was disallowed.</td>
</tr>
<tr>
<td>8 2147 (863)</td>
<td></td>
<td>The key type to be generated is not valid.</td>
</tr>
<tr>
<td>8 2149 (865)</td>
<td></td>
<td>The key to be generated is stronger than the input material.</td>
</tr>
<tr>
<td>8 2151 (867)</td>
<td></td>
<td>At least one PIN decimalization table identifier provided as input is out of range or is a duplicate. No PIN tables have been changed.</td>
</tr>
<tr>
<td>8 2153 (869)</td>
<td></td>
<td>The input token is incompatible with the service (that is, clear key when encrypted key was expected).</td>
</tr>
<tr>
<td>8 2154 (86A)</td>
<td></td>
<td>At least one key token does not have the required key type for the specified function.</td>
</tr>
<tr>
<td>8 2158 (86E)</td>
<td></td>
<td>There is a mismatch between ECC key tokens of curve types, key lengths, or both. Curve types and key lengths must match.</td>
</tr>
<tr>
<td>8 2159 (86F)</td>
<td></td>
<td>A key-encrypting key is invalid.</td>
</tr>
<tr>
<td>8 2161 (871)</td>
<td></td>
<td>A wrap type, either requested or default, is in conflict with one or more input tokens.</td>
</tr>
<tr>
<td>8 2163 (873)</td>
<td></td>
<td>At least two of the key parts of a new operational or master key have identical parts and an error has been requested by the setting of an appropriate access control point.</td>
</tr>
<tr>
<td>8 2165 (875)</td>
<td></td>
<td>An RSA key token contains a private section that is not valid with this command.</td>
</tr>
<tr>
<td>8 2167 (877)</td>
<td></td>
<td>Invalid hash type in certificate.</td>
</tr>
<tr>
<td>8 2169 (879)</td>
<td></td>
<td>Invalid signature type in certificate.</td>
</tr>
<tr>
<td>8 2170 (87A)</td>
<td></td>
<td>Translation of text using an outbound key that has an effective key strength weaker than the effective strength of the inbound key is not allowed.</td>
</tr>
<tr>
<td>8 2174 (87E)</td>
<td></td>
<td>The provided data was not hexadecimal digits.</td>
</tr>
<tr>
<td>8 2175 (87F)</td>
<td></td>
<td>A weak PIN was presented. The PIN change has been rejected.</td>
</tr>
<tr>
<td>8 2177 (881)</td>
<td></td>
<td>The PAN presented to the PAN change verb was the same as the PAN in the encrypted PIN block. The change has been rejected.</td>
</tr>
<tr>
<td>8 2178 (882)</td>
<td></td>
<td>The PAN provided is inconsistent with a PAN incorporated in another piece of data.</td>
</tr>
<tr>
<td>8 2181 (885)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 2183 (887)</td>
<td></td>
<td>There is an error in the weak PIN entry structure input header length. No entries have been changed.</td>
</tr>
<tr>
<td>8 2185 (889)</td>
<td></td>
<td>For at least one of the inputs, the weak PIN entry requested to be activated is not in the loaded state. No weak PIN entries have been activated.</td>
</tr>
<tr>
<td>8 2186 (88A)</td>
<td></td>
<td>For at least one of the inputs, the weak PIN entry requested to be activated did not match the weak PIN entry structure to be activated. No weak PIN entries have been activated.</td>
</tr>
<tr>
<td>8 2187 (88B)</td>
<td></td>
<td>One or more of the weak PIN entry ID numbers in the input verb data was invalid, out of range, or a duplicate. No weak PIN entries have been changed.</td>
</tr>
<tr>
<td>8 2189 (88D)</td>
<td></td>
<td>There is an error in the weak PIN entry structure input type. No entries have been changed.</td>
</tr>
<tr>
<td>8 2190 (88E)</td>
<td></td>
<td>There is an error in the weak PIN entry structure input header version. No entries have been changed.</td>
</tr>
<tr>
<td>8 2191 (88F)</td>
<td></td>
<td>There is an error in the weak PIN entry structure input header count. No entries have been changed.</td>
</tr>
<tr>
<td>Return code, decimal</td>
<td>Reason code, decimal (hex)</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>8 2193 (891)</td>
<td></td>
<td>The presented PIN is a duplicate of one already in the table. No entries have been changed.</td>
</tr>
<tr>
<td>8 2194 (892)</td>
<td></td>
<td>Invalid or out of range passphrase length.</td>
</tr>
<tr>
<td>8 2197 (895)</td>
<td></td>
<td>The presented PIN failed verification. No processing has been done.</td>
</tr>
<tr>
<td>8 2198 (896)</td>
<td></td>
<td>The presented CMAC failed verification. No processing has been done.</td>
</tr>
<tr>
<td>8 2199 (897)</td>
<td></td>
<td>A variable-length symmetric key-token (version X'05') contains invalid key-usage field data.</td>
</tr>
<tr>
<td>8 2201 (899)</td>
<td></td>
<td>A variable-length symmetric key-token (version X'05') contains invalid key-management field data.</td>
</tr>
<tr>
<td>8 2203 (89B)</td>
<td></td>
<td>RSA engine check-sum error.</td>
</tr>
<tr>
<td>8 2227 (8B3)</td>
<td></td>
<td>The triple-length key cannot be imported because the TR-31 key block does not include a CCA control vector.</td>
</tr>
<tr>
<td>8 2229 (8B5)</td>
<td></td>
<td>The type of the specified key is not valid because a diversified key-generating key must be used to derive this symmetric key type.</td>
</tr>
<tr>
<td>8 2231 (8B7)</td>
<td></td>
<td>There was a problem converting or formatting the PAN.</td>
</tr>
<tr>
<td>8 2232 (8B8)</td>
<td></td>
<td>There was a problem converting or formatting the cardholder name.</td>
</tr>
<tr>
<td>8 2233 (8B9)</td>
<td></td>
<td>There was a problem converting or formatting the track 1 data.</td>
</tr>
<tr>
<td>8 2235 (8BB)</td>
<td></td>
<td>There was a problem converting or formatting the track 2 data.</td>
</tr>
<tr>
<td>8 2237 (8BD)</td>
<td></td>
<td>Data presented for VFPE processing is not in VFPE enciphered.</td>
</tr>
<tr>
<td>8 2238 (8BE)</td>
<td></td>
<td>An incorrect PIN profile is specified.</td>
</tr>
<tr>
<td>8 2239 (8BF)</td>
<td></td>
<td>The check digit compliance indicator/keyword denotes compliant check digit but the input PAN does not have a compliant check digit.</td>
</tr>
<tr>
<td>8 2243 (8C3)</td>
<td></td>
<td>The key-derivation section is missing or the attributes in the key-derivation section do not match those in the output skeleton token.</td>
</tr>
<tr>
<td>8 2245 (8C5)</td>
<td></td>
<td>A randomly generated source key is required, but the pedigree of the source key indicates that the key is not randomly generated.</td>
</tr>
<tr>
<td>8 2246 (8C6)</td>
<td></td>
<td>A required tag-length-value (TLV) object is not present in the IBM Extended Associated Data (IEAD) section.</td>
</tr>
<tr>
<td>8 2247 (8C7)</td>
<td></td>
<td>Error in PSS signature salt length.</td>
</tr>
<tr>
<td>8 2254 (8CE)</td>
<td></td>
<td>The SECURE LOG SRDI that is stored on the coprocessor is full, no auditable actions are allowed.</td>
</tr>
<tr>
<td>8 2261 (8D5)</td>
<td></td>
<td>Cannot adjust time twice within a 24 hour period.</td>
</tr>
<tr>
<td>8 2262 (8D6)</td>
<td></td>
<td>Last adjustment time was &gt; 24 hours; however, the amount of time to be adjusted is &gt; 1 second.</td>
</tr>
<tr>
<td>8 2401 (961)</td>
<td></td>
<td>Tried to enter compliance mode or change compliance state but not in correct starting mode.</td>
</tr>
<tr>
<td>8 2402 (962)</td>
<td></td>
<td>Tried to use a compliance-tagged key but domain is not in active compliance mode.</td>
</tr>
<tr>
<td>8 2403 (963)</td>
<td></td>
<td>This verb is not allowed to use compliance-tagged key tokens.</td>
</tr>
<tr>
<td>8 2405 (965)</td>
<td></td>
<td>This service of this verb is not allowed to use with compliance-tagged tokens.</td>
</tr>
<tr>
<td>8 2406 (966)</td>
<td></td>
<td>This service has been passed 1 or more tokens that are compliance-tagged and 1 or more tokens that are not compliance-tagged. All must match.</td>
</tr>
</tbody>
</table>
### Table 228. Reason codes for return code 8 (continued)

<table>
<thead>
<tr>
<th>Return code, decimal</th>
<th>Reason code, decimal (hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2407 (967)</td>
<td>This service has been asked to generate or derive a comp-tagged key, or to check a key token for compliance but the requested/given strength is too weak for the configured compliance mode.</td>
</tr>
<tr>
<td>8</td>
<td>2409 (969)</td>
<td>This service has been asked to generate or derive a comp-tagged key, or to check a key token for compliance, but the requested/given key type or usage is non-compliant.</td>
</tr>
<tr>
<td>8</td>
<td>2410 (96A)</td>
<td>This service has been asked to use or create a Key-Encrypting-Key (KEK) that has had the NOCV flag set in token flags. Since the flags are not part of the CV it is possible the flag was added on the host side. These types of KEKs are non-compliant.</td>
</tr>
<tr>
<td>8</td>
<td>2411 (96B)</td>
<td>The requested service is only available when the domain is configured in migration mode, and the domain is not in this mode now.</td>
</tr>
<tr>
<td>8</td>
<td>2413 (96D)</td>
<td>The service has been asked to use a comp-tagged KEK to wrap or unwrap an external token but the external token key type is non-compliant.</td>
</tr>
<tr>
<td>8</td>
<td>2414 (96E)</td>
<td>The service for comp-tag migration checking or migration tagging has been given a token type that is not currently supported.</td>
</tr>
<tr>
<td>8</td>
<td>2415 (96F)</td>
<td>The service to enter imprint or compliance mode cannot complete because card cannot support compliance mode.</td>
</tr>
<tr>
<td>8</td>
<td>2417 (971)</td>
<td>A token passed in identifies a KDF for a compliance mode that is not the current compliance mode.</td>
</tr>
<tr>
<td>8</td>
<td>2418 (972)</td>
<td>A token passed into KTR2 already has a comp-tag in CV.</td>
</tr>
<tr>
<td>8</td>
<td>2419 (973)</td>
<td>Failed to retrieve the compliance mode flags.</td>
</tr>
<tr>
<td>8</td>
<td>2421 (975)</td>
<td>A CCA service was requested without the COMPMODE keyword for a domain that is in imprint mode or a compliance mode that requires the COMPMODE keyword.</td>
</tr>
<tr>
<td>8</td>
<td>2422 (976)</td>
<td>A CCA service was passed the COMPMODE keyword but the domain is not in imprint mode or a compliance mode.</td>
</tr>
<tr>
<td>8</td>
<td>2423 (977)</td>
<td>Cannot change default imprint mode role (INITADDM) while in imprint mode.</td>
</tr>
<tr>
<td>8</td>
<td>2425 (979)</td>
<td>INIT-AC inactive and activate COMPMODE keywords were different</td>
</tr>
<tr>
<td>8</td>
<td>2426 (97A)</td>
<td>Action is restricted because domain is in imprint mode. For example the default domain-scope profile cannot update DFLTxxxx or its own role. Also, the CSNBMKP and CSNDFPIM verbs cannot be used in imprint mode.</td>
</tr>
<tr>
<td>8</td>
<td>2427 (97B)</td>
<td>A compliance-tagged token was passed to the HSM while the HSM was in Migration Mode. Compliance-tag token services are not available in migration mode.</td>
</tr>
<tr>
<td>8</td>
<td>2429 (97D)</td>
<td>A CCA service was requested that, because of the compliance state of the domain, requires a signed command from a TKE. However, the request was not received in this format.</td>
</tr>
<tr>
<td>8</td>
<td>2430 (97E)</td>
<td>A CCA service has been passed a key part to be either the first key part or to be combined with previously passed key parts for either the Master Key or an operational key that is being built from parts. The key part passed may be long enough but is not valid because it matches (bit for bit) one of the known weak key patterns (such as all 0x00 bytes) for that key algorithm.</td>
</tr>
<tr>
<td>8</td>
<td>2431 (97F)</td>
<td>A CCA service has been passed an external token with the COMP-TAG marker or bit set. This is bit 58 in the Control Vector (CV) for DES tokens.</td>
</tr>
<tr>
<td>8</td>
<td>2433 (981)</td>
<td>The attempted operation must be performed as a dual-control operation when the target domain is in compliance or imprint mode.</td>
</tr>
<tr>
<td>8</td>
<td>2434 (982)</td>
<td>The ACP list associated with the role-to-load violates the PCI-HSM complimentary ACP restrictions.</td>
</tr>
<tr>
<td>8</td>
<td>2435 (983)</td>
<td>The ACP quorum rules are not satisfied.</td>
</tr>
<tr>
<td>Return code, decimal</td>
<td>Reason code, decimal (hex)</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>8</td>
<td>2437 (985)</td>
<td>Provided PIN block formats do not match in compliance mode.</td>
</tr>
<tr>
<td>8</td>
<td>2438 (986)</td>
<td>An attempt was made to update DFLTxxxx role with one or more ACPs from the complimentary ACP list while in imprint- or compliance-mode.</td>
</tr>
<tr>
<td>8</td>
<td>2439 (987)</td>
<td>An attempt to load DFLTxxxx from a card-scope domain when 'xxxx' is a domain-scope domain.</td>
</tr>
<tr>
<td>8</td>
<td>2849 (B21)</td>
<td>A verb data keyword specifies a keyword that is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>2850 (B22)</td>
<td>A verb data keyword combination is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>2851 (B23)</td>
<td>The verb data length value is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>2945 (B81)</td>
<td>A required verb data keyword is not found.</td>
</tr>
<tr>
<td>8</td>
<td>2946 (B82)</td>
<td>Initialization vector length is too small, or text length exceeds maximum.</td>
</tr>
<tr>
<td>8</td>
<td>2947 (B83)</td>
<td>The computed authentication tag does not match the data identified by the key_parms parameter.</td>
</tr>
<tr>
<td>8</td>
<td>3001 (BB9)</td>
<td>The RSA-OAEP block contains a PIN block and the verb did not request PINBLOCK processing.</td>
</tr>
<tr>
<td>8</td>
<td>3002 (BBA)</td>
<td>Specific to IBM Z - UDX already authorized.</td>
</tr>
<tr>
<td>8</td>
<td>3005 (BBD)</td>
<td>Specific to IBM Z - UDX not in UDX Authorization Table (UAT).</td>
</tr>
<tr>
<td>8</td>
<td>3006 (BBE)</td>
<td>Specific to IBM Z - UDX not authorized.</td>
</tr>
<tr>
<td>8</td>
<td>3007 (BBF)</td>
<td>Specific to IBM Z - Failed to obtain semaphore that guards the UAT.</td>
</tr>
<tr>
<td>8</td>
<td>3009 (BC1)</td>
<td>Specific to IBM Z - UDX Password hash mismatch.</td>
</tr>
<tr>
<td>8</td>
<td>3013 (BC5)</td>
<td>The longitudinal redundancy check (LRC) checksum in the AES key-token does not match the LRC checksum of the clear key.</td>
</tr>
<tr>
<td>8</td>
<td>3047 (BE7)</td>
<td>Use of clear key provided is not allowed. A secure key is required.</td>
</tr>
<tr>
<td>8</td>
<td>6000 (1770)</td>
<td>The specified device is already allocated.</td>
</tr>
<tr>
<td>8</td>
<td>6001 (1771)</td>
<td>No device is allocated.</td>
</tr>
<tr>
<td>8</td>
<td>6002 (1772)</td>
<td>The specified device does not exist.</td>
</tr>
<tr>
<td>8</td>
<td>6003 (1773)</td>
<td>The specified device is an improper type.</td>
</tr>
<tr>
<td>8</td>
<td>6013 (177D)</td>
<td>The length of the cryptographic resource name is not valid.</td>
</tr>
<tr>
<td>8</td>
<td>6014 (177E)</td>
<td>The cryptographic resource name is not valid or does not refer to a coprocessor that is available in the system.</td>
</tr>
<tr>
<td>8</td>
<td>6015 (177F)</td>
<td>An ECC curve type is invalid or its usage is inconsistent.</td>
</tr>
<tr>
<td>8</td>
<td>6017 (1781)</td>
<td>Curve size $p$ is invalid or its usage is inconsistent.</td>
</tr>
<tr>
<td>8</td>
<td>6018 (1782)</td>
<td>Error returned from CLiC module.</td>
</tr>
<tr>
<td>8</td>
<td>10028 (272C)</td>
<td>Specific to IBM Z - Invalid control vector in key token supplied.</td>
</tr>
<tr>
<td>8</td>
<td>10036 (2734)</td>
<td>Specific to IBM Z - Invalid control vectors (L-R) in key token supplied.</td>
</tr>
<tr>
<td>8</td>
<td>10044 (273C)</td>
<td>Specific to IBM Z - The key_type parameter and the CV key type for the supplied key token do not match.</td>
</tr>
<tr>
<td>8</td>
<td>10056 (2748)</td>
<td>Specific to IBM Z - The key_type parameter contains TOKEN, which is invalid for the requested operation.</td>
</tr>
<tr>
<td>8</td>
<td>10124 (278C)</td>
<td>Specific to IBM Z - The key id cannot be exported because of prohibit export restriction in the token supplied.</td>
</tr>
</tbody>
</table>
### Table 228. Reason codes for return code 8 (continued)

<table>
<thead>
<tr>
<th>Return code, decimal</th>
<th>Reason code, decimal (hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>10128 (2790)</td>
<td>Specific to IBM Z - The NOCV-KEK or CV-KEK <code>rule_array</code> keyword does not apply in this case. Check other keywords passed.</td>
</tr>
<tr>
<td>8</td>
<td>10129 (2791)</td>
<td>Specific to IBM Z - The NOCV-KEK importer key or transport key is not allowed in the Remote Key Export operation requested.</td>
</tr>
</tbody>
</table>

### Reason codes that accompany return code 12

Reason codes that accompany return code 12

The codes are listed in [Table 229](#)

### Table 229. Reason codes for return code 12

<table>
<thead>
<tr>
<th>Return code, decimal</th>
<th>Reason code, decimal (hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>097 (061)</td>
<td>File space in key storage is insufficient to complete the operation.</td>
</tr>
<tr>
<td>12</td>
<td>196 (0C4)</td>
<td>The device driver, the security server, or the directory server is not installed or is not active. File permissions are not valid for your application.</td>
</tr>
<tr>
<td>12</td>
<td>197 (0C5)</td>
<td>There is a key-storage file I/O error or the file is not found.</td>
</tr>
<tr>
<td>12</td>
<td>206 (0CE)</td>
<td>The key-storage file is not valid or the master-key verification failed. There is an unlikely, but possible, synchronization problem with the Master Key Process verb.</td>
</tr>
<tr>
<td>12</td>
<td>207 (0CF)</td>
<td>The verification method flags in the profile are not valid.</td>
</tr>
<tr>
<td>12</td>
<td>319 (13F)</td>
<td>Passed to the CVV Verify or CVV Generate verb, the Verb Unique data corresponds to a PÂN length of 19, but the overall length is wrong. This indicates that the host code is out of date.</td>
</tr>
<tr>
<td>12</td>
<td>324 (144)</td>
<td>There is insufficient memory available to process your request, either memory in the host computer or memory inside the coprocessor including the flash EPROM used to store keys, profiles, and other application data.</td>
</tr>
<tr>
<td>12</td>
<td>338 (152)</td>
<td>This cryptographic hardware device driver is not installed or is not responding, or the CCA code is not loaded in the coprocessor.</td>
</tr>
<tr>
<td>12</td>
<td>764 (2FC)</td>
<td>The master keys are not loaded and, therefore, a key cannot be recovered or enciphered.</td>
</tr>
<tr>
<td>12</td>
<td>768 (300)</td>
<td>One or more paths for key-storage directory operations are improperly specified.</td>
</tr>
<tr>
<td>12</td>
<td>769 (301)</td>
<td>An internal error has occurred with the parameters to a cryptographic algorithm.</td>
</tr>
<tr>
<td>12</td>
<td>2007 (7D7)</td>
<td>The change type in the Pending Change Buffer is not recognized.</td>
</tr>
<tr>
<td>12</td>
<td>2015 (7DF)</td>
<td>The domain stored in the domain mask does not match what was included as the domain in the CPRB.</td>
</tr>
<tr>
<td>12</td>
<td>2017 (7E1)</td>
<td>The operation is attempting to call 'SET' for a master key, but has passed an invalid Master Key Verification Pattern.</td>
</tr>
<tr>
<td>12</td>
<td>2021 (7E5)</td>
<td>The card is disabled in the TKE path.</td>
</tr>
<tr>
<td>12</td>
<td>2037 (7F5)</td>
<td>Invalid domain specified.</td>
</tr>
<tr>
<td>12</td>
<td>2043 (7FB)</td>
<td>In the course of TKE communication through the host library to an adapter, a particular requested OA certificate was not found. A small number of these errors are typical when communication with a TKE is initiated.</td>
</tr>
<tr>
<td>12</td>
<td>2045 (7FD)</td>
<td>The CCA software is unable to claim a semaphore. The system might be short of resources.</td>
</tr>
</tbody>
</table>
### Table 229. Reason codes for return code 12 (continued)

<table>
<thead>
<tr>
<th>Return code, decimal</th>
<th>Reason code, decimal (hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>2046 (7FE)</td>
<td>The CCA software is unable to list all the keys. The limit of 500,000 keys might have been reached.</td>
</tr>
<tr>
<td>12</td>
<td>2049 (801)</td>
<td>An error occurred while unlocking a semaphore in order to release the exclusive control of that semaphore.</td>
</tr>
<tr>
<td>12</td>
<td>2073 (819)</td>
<td>TKE command received when TKE disabled.</td>
</tr>
<tr>
<td>12</td>
<td>2074 (81A)</td>
<td>Invalid version found in Connectivity Programming Request/Reply Block (CPRB).</td>
</tr>
<tr>
<td>12</td>
<td>2101 (835)</td>
<td>Invalid AES flags in the function control vector (FCV).</td>
</tr>
<tr>
<td>12</td>
<td>2117 (845)</td>
<td>Thread specific CLiC objects are not in proper state.</td>
</tr>
<tr>
<td>12</td>
<td>2155 (86B)</td>
<td>The length of the fully qualified dataset name exceeds the maximum size that the verb can process.</td>
</tr>
<tr>
<td>12</td>
<td>2225 (881)</td>
<td>An internal outbound authentication manager error occurred, or the OA manager is disabled.</td>
</tr>
<tr>
<td>12</td>
<td>3046 (BE6)</td>
<td>The wrong usage was attempted in an operation with a retained key.</td>
</tr>
<tr>
<td>12</td>
<td>2242 (8C2)</td>
<td>The reply message block is too long for the host buffer.</td>
</tr>
</tbody>
</table>

### Reason codes that accompany return code 16

Reason codes that accompany return code 16.

These codes are listed in Table 230.

### Table 230. Reason codes for return code 16

<table>
<thead>
<tr>
<th>Return code, decimal</th>
<th>Reason code, decimal (hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>099 (063)</td>
<td>An unrecoverable error occurred in the security server; contact the IBM support center.</td>
</tr>
<tr>
<td>16</td>
<td>336 (150)</td>
<td>An error occurred in a cryptographic hardware or software component.</td>
</tr>
<tr>
<td>16</td>
<td>337 (151)</td>
<td>A device software error occurred.</td>
</tr>
<tr>
<td>16</td>
<td>339 (153)</td>
<td>A system error occurred in the interprocess communication routine.</td>
</tr>
<tr>
<td>16</td>
<td>444 (1BC)</td>
<td>The verb-unique-data has an invalid length.</td>
</tr>
<tr>
<td>16</td>
<td>556 (22C)</td>
<td>The request parameter block failed consistency checking.</td>
</tr>
<tr>
<td>16</td>
<td>708 (2C4)</td>
<td>The cryptographic engine is returning inconsistent data.</td>
</tr>
<tr>
<td>16</td>
<td>709 (2C5)</td>
<td>Cryptographic engine internal error. Could not access the master-key data.</td>
</tr>
<tr>
<td>16</td>
<td>710 (2C6)</td>
<td>An unrecoverable error occurred while attempting to update master-key data items.</td>
</tr>
<tr>
<td>16</td>
<td>712 (2C8)</td>
<td>An unexpected error occurred in the master-key manager.</td>
</tr>
<tr>
<td>16</td>
<td>800 (320)</td>
<td>A problem occurred in internal SHA operation processing.</td>
</tr>
<tr>
<td>16</td>
<td>2022 (7E6)</td>
<td>TKE-related internal file open error.</td>
</tr>
<tr>
<td>16</td>
<td>2047 (7FF)</td>
<td>Unable to transfer request data from host to coprocessor.</td>
</tr>
<tr>
<td>16</td>
<td>2057 (809)</td>
<td>Internal error: memory allocation failure.</td>
</tr>
<tr>
<td>16</td>
<td>2058 (80A)</td>
<td>Internal error: unexpected return code from OAEP routines.</td>
</tr>
<tr>
<td>16</td>
<td>2059 (80B)</td>
<td>Internal error: OAEP SHA-1 request failure.</td>
</tr>
<tr>
<td>16</td>
<td>2061 (80D)</td>
<td>Internal error in Symmetric Key Import, OAEP-decode: enciphered message too long.</td>
</tr>
<tr>
<td>Return code, decimal</td>
<td>Reason code, decimal (hex)</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>16</td>
<td>2063 (80F)</td>
<td>The reply message too long for the requestor’s command reply buffer.</td>
</tr>
<tr>
<td>16</td>
<td>2107 (83B)</td>
<td>Internal files failed verification check when loading from encrypted storage.</td>
</tr>
<tr>
<td>16</td>
<td>2150 (866)</td>
<td>An error occurred while attempting to open or save the DECTABLE SRDI that is stored on the coprocessor.</td>
</tr>
<tr>
<td>16</td>
<td>2195 (893)</td>
<td>An error occurred reading the weak PIN file stored on the coprocessor.</td>
</tr>
</tbody>
</table>
Chapter 19. Key token formats

The key token formats can be useful for debugging purposes.

This information unit provides the formats for:

- “AES internal fixed-length key token” on page 848
- “DES internal key token” on page 849
- “DES external key token” on page 850
- “External RXX DES key tokens” on page 851
- “DES null key token” on page 852
- “RSA public key token” on page 853
- “RSA private key token” on page 854
- “ECC key token” on page 874
- “PKA key tokens” on page 880
- “HMAC key token” on page 888
- “Variable-length symmetric key tokens” on page 889
- “TR-31 optional block data” on page 967
- “Trusted blocks” on page 968

AES internal fixed-length key token

The format for an AES internal fixed-length key token.

Table 231 on page 848 shows the format for an AES internal fixed-length key token.

CCA AES fixed-length key-token data structures are 64 bytes in length, and are made up of an internal key-token identifier and a token version number, reserved fields, a flag byte containing various flag bits, and a token-validation value.

Depending on the flag byte, the key token either contains an encrypted key, a clear key, or the key is absent. An encrypted key is encrypted under an AES master key identified by a master-key verification pattern (MKVP) in the key token. The key token contains a two-byte integer that specifies the length of the clear-key value in bits, valued to 0, 128, 192, or 256, and a two-byte integer that specifies the length of the encrypted-key value in bytes, valued to 0 or 32. An LRC checksum byte of the clear-key value is also in the key token.

All AES keys contained in fixed-length key tokens are DATA keys. If the flag byte indicates a control vector (CV) is present, it must be all binary zeros. An all-zero CV represents the CV value of an AES DATA key. If a key is present without a control vector in a key token, that is accepted and the key is interpreted as an AES DATA key. The AES internal fixed-length key-token, version X’04’, is the structure used to hold AES keys that are either encrypted with the AES master-key, or in cleartext format.

For an AES symmetric variable-length key token, see “AES CIPHER variable-length symmetric key token” on page 904.
### Key token formats

**Table 231. AES internal fixed-length key token format, version X'04’**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X'01' (flag indicating this is an internal key token)</td>
</tr>
<tr>
<td>1 - 3</td>
<td>Implementation-dependent bytes, must be X'000000'.</td>
</tr>
<tr>
<td>4</td>
<td>Key token version number, X'04'</td>
</tr>
<tr>
<td>5</td>
<td>Reserved (X'00')</td>
</tr>
<tr>
<td>6</td>
<td>Flag byte. See “AES internal fixed-length key-token flag byte.”</td>
</tr>
<tr>
<td>7</td>
<td>Longitudinal redundancy check (LRC) checksum of clear-key value (LRC is the XOR of each byte in the clear-key value).</td>
</tr>
<tr>
<td>8 - 15</td>
<td>Master key verification pattern (MKVP)</td>
</tr>
<tr>
<td></td>
<td>Contains the master-key verification pattern of the AES master-key used to encrypt the key contained in the token, or binary zeros if the token does not contain a key or the key is in the clear. The MKVP is calculated as the leftmost eight bytes of the SHA-256 hash of the string formed by pre-pending the byte X'01' to the cleartext master-key value.</td>
</tr>
<tr>
<td>16 - 47</td>
<td>Key value, if present. Contains either:</td>
</tr>
<tr>
<td></td>
<td>• A 256-bit encrypted-key value. The clear key value is padded on the right with binary zeros, and the entire 256-bit value is encrypted under the AES master-key using AES CBC mode with an initialization vector of binary zeros.</td>
</tr>
<tr>
<td></td>
<td>• A 128-bit, 192-bit, or 256-bit clear-key value left-aligned and padded on the right with binary zeros for the entire 256-bit field.</td>
</tr>
<tr>
<td>48 - 55</td>
<td>Control Vector (CV)</td>
</tr>
<tr>
<td></td>
<td>This value must be binary zeros for all AES key tokens that have a control vector present.</td>
</tr>
<tr>
<td>56 - 57</td>
<td>Clear-key bit length</td>
</tr>
<tr>
<td></td>
<td>An integer specifying the length in bits of the clear-key value. If no key is present in a completed token, this length is zero. In a skeleton token, this is the length of the key to be created in the token when used as input to the Key Generate verb.</td>
</tr>
<tr>
<td>58 - 59</td>
<td>Encrypted-key byte length</td>
</tr>
<tr>
<td></td>
<td>An integer specifying the length in bytes of the encrypted-key value. This value is zero if the token does not contain a key or the key is in the clear.</td>
</tr>
<tr>
<td>60 - 63</td>
<td>Token validation value (TVV).</td>
</tr>
</tbody>
</table>

### AES internal fixed-length key-token flag byte

The format for an AES internal fixed-length key token flag byte.

**Table 232. AES internal fixed-length key-token flag byte**

<table>
<thead>
<tr>
<th>Bits (MSB...LSB)†</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1xxx xxxx</td>
<td>Key is encrypted under the AES master-key (ignored if no key present).</td>
</tr>
<tr>
<td>0xxx xxxx</td>
<td>Key is in the clear (ignored if no key present).</td>
</tr>
<tr>
<td>x1xx xxxx</td>
<td>Control vector (CV) is present.</td>
</tr>
<tr>
<td>x0xx xxxx</td>
<td>Control vector (CV) is not present.</td>
</tr>
<tr>
<td>xx1x xxxx</td>
<td>No key and no MKVP present.</td>
</tr>
<tr>
<td>xx0x xxxx</td>
<td>Encrypted or clear key present, MKVP present if key is encrypted.</td>
</tr>
</tbody>
</table>

**Note:** All undefined bits are reserved and must be 0.
Token validation value

CCA uses the *token validation value (TVV)* to verify that a token is valid.

The TVV prevents a key token that is not valid or that is overlaid from being accepted by CCA. It provides a checksum to detect a corruption in the key token.

When an CCA verb generates a key token, it generates a TVV and stores the TVV in bytes 60-63 of the key token. When an application program passes a key token to a verb, CCA checks the TVV. To generate the TVV, CCA performs a twos complement ADD operation (ignoring carries and overflow) on the key token, operating on four bytes at a time, starting with bytes 0-3 and ending with bytes 56-59.

DES internal key token

The format for a DES internal key token. DES key tokens are almost always in a fixed-length token, except for a DESUSECV key type in a variable-length symmetric key token.

Table 233 shows the format for a DES internal key token.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X'01' (flag indicating this is an internal key token)</td>
</tr>
<tr>
<td>1 - 3</td>
<td>Implementation-dependent bytes (X'000000' for ICSF)</td>
</tr>
<tr>
<td>4</td>
<td>Key token version number (X'00' or X'01')</td>
</tr>
<tr>
<td>5</td>
<td>Reserved (X'00')</td>
</tr>
<tr>
<td>6</td>
<td>Flag byte 1</td>
</tr>
<tr>
<td></td>
<td>Bit</td>
</tr>
<tr>
<td>0</td>
<td>Encrypted key and master key verification pattern (MKVP) are present.</td>
</tr>
<tr>
<td>1</td>
<td>Control vector (CV) value in this token has been applied to the key.</td>
</tr>
<tr>
<td>2</td>
<td>Key is used for no control vector (NOCV) processing. Valid for transport keys only.</td>
</tr>
<tr>
<td>3</td>
<td>Key is an ANSI key-encrypting key (AKEK).</td>
</tr>
<tr>
<td>4</td>
<td>AKEK is a double-length key (16 bytes).</td>
</tr>
<tr>
<td></td>
<td>Note: When bit 3 is on and bit 4 is off, AKEK is a single-length key (eight bytes).</td>
</tr>
<tr>
<td>5</td>
<td>AKEK is partially notarized.</td>
</tr>
<tr>
<td>6</td>
<td>Key is an ANSI partial key.</td>
</tr>
<tr>
<td>7</td>
<td>Export prohibited.</td>
</tr>
<tr>
<td>7</td>
<td>Flag byte 2</td>
</tr>
<tr>
<td></td>
<td>Bit</td>
</tr>
<tr>
<td>0-2</td>
<td>B'000' The encrypted key, if present, is wrapped using the legacy (WRAP-ECB) method.</td>
</tr>
<tr>
<td></td>
<td>B'001' The encrypted key is wrapped using the enhanced WRAP-ENH) method.</td>
</tr>
<tr>
<td></td>
<td>All other bit combinations are reserved. Undefined bits must be zero.</td>
</tr>
<tr>
<td>8 - 15</td>
<td>Non-compliant-tagged tokens: Master Key Verification Pattern (MKVP). Compliant-tagged tokens: A 5-byte MKVP followed by 3-bytes internal compliance information.</td>
</tr>
<tr>
<td>16 - 23</td>
<td>A single-length key, the left half of a double-length key, or Part A of a triple-length key. The value is encrypted under the master key.</td>
</tr>
<tr>
<td>24 - 31</td>
<td>X'0000000000000000' if a single-length key, the right half of a double-length operational key, or Part B of a triple-length operational key. The right half of the double-length key or Part B of the triple-length key is encrypted under the master key.</td>
</tr>
</tbody>
</table>
### Key token formats

#### Table 233. DES internal key token format (continued)

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 - 39</td>
<td>The control vector (CV) for a single-length key or the left half of the control vector for a double-length key.</td>
</tr>
<tr>
<td>40 - 47</td>
<td>X'0000000000000000' if a single-length key or the right half of the control vector for a double-length operational key.</td>
</tr>
<tr>
<td>48 - 55</td>
<td>X'0000000000000000' if a single-length key or double-length key, or Part C of a triple-length operational key. Part C of a triple-length key is encrypted under the master key.</td>
</tr>
<tr>
<td>56 - 58</td>
<td>Reserved (X'000000')</td>
</tr>
<tr>
<td>59 bits 0 and 1</td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td>B'10'</td>
</tr>
<tr>
<td></td>
<td>B'00'</td>
</tr>
<tr>
<td></td>
<td>B'01'</td>
</tr>
<tr>
<td>59 bits 2 and 3</td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td>B'00'</td>
</tr>
<tr>
<td></td>
<td>B'01'</td>
</tr>
<tr>
<td></td>
<td>B'10'</td>
</tr>
<tr>
<td>59 bits 4 - 7</td>
<td>B'0000'</td>
</tr>
<tr>
<td>60 - 63</td>
<td>Token validation value (TVV).</td>
</tr>
</tbody>
</table>

**Note:** AKEKs are not supported by this version of CCA. Key tokens from other CCA systems, however, could have the AKEK flag bits set in a key token.

### DES external key token

The format for a DES external key token. DES key tokens are almost always in a fixed-length token, except for a DESUSECV key type in a variable-length symmetric key token.

Table 234 shows the format for a DES external key token.

#### Table 234. DES external key token format

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X'02' (flag indicating an external key token)</td>
</tr>
<tr>
<td>1</td>
<td>Reserved (X'00')</td>
</tr>
<tr>
<td>2 - 3</td>
<td>Implementation-dependent bytes (X'0000' for CCA)</td>
</tr>
<tr>
<td>4</td>
<td>Key token version number (X'00' or X'01')</td>
</tr>
<tr>
<td>5</td>
<td>Reserved (X'00')</td>
</tr>
<tr>
<td>6</td>
<td>Flag byte</td>
</tr>
<tr>
<td></td>
<td><strong>Bit</strong></td>
</tr>
<tr>
<td>0</td>
<td>Encrypted key is present.</td>
</tr>
<tr>
<td>1</td>
<td>Control vector (CV) value has been applied to the key.</td>
</tr>
<tr>
<td></td>
<td>Other bits are reserved and are binary zeros.</td>
</tr>
<tr>
<td>7</td>
<td>Reserved (X'00')</td>
</tr>
<tr>
<td>8 - 15</td>
<td>Reserved (X'0000000000000000')</td>
</tr>
<tr>
<td>16 - 23</td>
<td>Single-length key or left half of a double-length key, or Part A of a triple-length key. The value is encrypted under a transport key.</td>
</tr>
</tbody>
</table>
Table 234. DES external key token format (continued)

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 - 31</td>
<td>X’0000000000000000’ if a single-length key or right half of a double-length key, or Part B of a triple-length key. The right half of a double-length key or Part B of a triple-length key is encrypted under a transport (key-encrypting key) for export or import.</td>
</tr>
<tr>
<td>32 - 39</td>
<td>Control vector (CV) for single-length key or left half of CV for double-length key</td>
</tr>
<tr>
<td>40 - 47</td>
<td>X’0000000000000000’ if single-length key or right half of CV for double-length key</td>
</tr>
<tr>
<td>48 - 55</td>
<td>X’0000000000000000’ if a single-length key, double-length key, or Part C of a triple-length key.</td>
</tr>
<tr>
<td>56 - 58</td>
<td>Reserved (X’000000’)</td>
</tr>
<tr>
<td>59 bits 0 and 1</td>
<td>B’00’</td>
</tr>
<tr>
<td>59 bits 2 and 3</td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td>B’00’</td>
</tr>
<tr>
<td></td>
<td>B’01’</td>
</tr>
<tr>
<td></td>
<td>B’10’</td>
</tr>
<tr>
<td>59 bits 4 - 7</td>
<td>B’0000’</td>
</tr>
<tr>
<td>60 - 63</td>
<td>Token validation value (see “Token validation value” on page 849 for a description).</td>
</tr>
</tbody>
</table>

External RKX DES key tokens

The Remote Key Export (CSNDRKX) verb and DES key-storage verbs use a special RKX key token.

Table 235 defines an external fixed-length DES key-token called an RKX key-token. An RKX key-token is a special token used exclusively by the Remote Key Export (CSNDRKX) verb and DES key-storage verbs (for example, DES Key Record Write). No other verbs use or reference an RKX key-token or key-token record. For additional information about the usage of RKX key tokens, see “Remote key loading” on page 53. Verbs other than Remote Key Export and the DES key-storage do not support RKX key tokens or RKX key token records.

As can be seen in the table, RKX key tokens are 64 bytes in length, have a token identifier flag (X’02’), a token version number (X’10’), and room for encrypted keys, same as normal fixed-length DES key tokens. Unlike normal fixed-length DES key tokens, RKX key tokens do not have a control vector, flag bits, and a token-validation value. In addition, RKX key tokens have a confounder value, a MAC value, and room for a third encrypted key.

Table 235. External RKX DES key-token format, version X’10’

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1</td>
<td>X’02’ (a token identifier flag that indicates an external key-token)</td>
</tr>
<tr>
<td>01</td>
<td>3</td>
<td>Reserved, binary zero</td>
</tr>
<tr>
<td>04</td>
<td>1</td>
<td>Token version number (X’10’)</td>
</tr>
<tr>
<td>05</td>
<td>2</td>
<td>Reserved, binary zero</td>
</tr>
<tr>
<td>07</td>
<td>1</td>
<td>Key length in bytes, including confounder</td>
</tr>
<tr>
<td>08</td>
<td>8</td>
<td>Confounder</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>Key left</td>
</tr>
<tr>
<td>24</td>
<td>8</td>
<td>Key middle (binary zero if not used)</td>
</tr>
</tbody>
</table>
### Key token formats

#### Table 235. External RKX DES key-token format, version X'10' (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>8</td>
<td>Key right (binary zero if not used)</td>
</tr>
<tr>
<td>40</td>
<td>8</td>
<td>Rule ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The trusted block rule identifier used to create this key token. A subsequent call to Remote Key Export (CSNDRKX) can use this token with a trusted block rule that references the rule ID that was used to create this token. The trusted block rule can be compared with this rule ID for verification purposes. The Rule ID is an 8-byte string of ASCII characters, left-aligned and padded on the right with space characters. Acceptable characters are A...Z, a...z, 0...9, - (X'2D'), and _ (X'5F'). All other characters are reserved for future use.</td>
</tr>
<tr>
<td>48</td>
<td>8</td>
<td>Reserved, binary zero</td>
</tr>
<tr>
<td>56</td>
<td>8</td>
<td>MAC value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ISO 16609 CBC-mode Triple-DES MAC, computed over the 56 bytes starting at offset 0 and including the encrypted key value and the rule ID using the same MAC key that is used to protect the trusted block itself. This MAC value guarantees that the key and the rule ID cannot be modified without detection, providing integrity and binding the rule ID to the key itself. This MAC value must verify with the same trusted block used to create the key, thus binding the key structure to that specific trusted block.</td>
</tr>
</tbody>
</table>

**Note:**
1. A fixed, randomly derived variant is exclusive-ORed with the MAC key before it is used to encipher the generated or exported key and confounder.
2. The MAC key is located within a trusted block (internal format) and can be recovered by decipherment under a variant of the PKA master key.
3. The trusted block is originally created in external form by the Trusted Block Create verb, and then converted to internal form by the PKA Key Import verb prior to the Remote Key Export call.

### DES null key token

The format for a DES null key token. DES key tokens are almost always in a fixed-length token, except for a DESUSECV key type in a variable-length symmetric key token.

Table 236 shows the format for a DES null key token.

#### Table 236. DES null key token format

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X'00' (flag indicating this is a null key token).</td>
</tr>
<tr>
<td>1 - 15</td>
<td>Reserved (set to binary zeros).</td>
</tr>
<tr>
<td>16 - 23</td>
<td>Single-length encrypted key, left half of double-length encrypted key, or Part A of triple-length encrypted key.</td>
</tr>
<tr>
<td>24 - 31</td>
<td>X'0000000000000000' if a single-length encrypted key, the right half of double-length encrypted key, or Part B of triple-length encrypted key.</td>
</tr>
<tr>
<td>32 - 39</td>
<td>X'0000000000000000' if a single-length encrypted key or double-length encrypted key.</td>
</tr>
<tr>
<td>40 - 47</td>
<td>Reserved (set to binary zeros).</td>
</tr>
</tbody>
</table>
Table 236. DES null key token format (continued)

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>48 - 55</td>
<td>Part C of a triple-length encrypted key.</td>
</tr>
<tr>
<td>56 - 63</td>
<td>Reserved (set to binary zeros).</td>
</tr>
</tbody>
</table>

**RSA public key token**

The sections of an RSA public key token.

An RSA public key token contains the following sections.

- A required token header, starting with the token identifier X'1E'
- A required RSA public key section, starting with the section identifier X'04'

Table 237 presents the format of an RSA public key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390® format).

Table 237. RSA Public Key Token format

<table>
<thead>
<tr>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>(decimal)</td>
</tr>
</tbody>
</table>

**Token Header (Required)**

| 000  | 001 | Token identifier. X'1E' indicates an external token.         |
| 001  | 001 | Version, X'00'.                                              |
| 002  | 002 | Length of the key token structure.                          |
| 004  | 004 | Ignored. Should be 0.                                       |

**RSA Public Key Section (Required)**

| 000  | 001 | X'04', section identifier, RSA public key.                |
| 001  | 001 | X'00', version.                                           |
| 002  | 002 | Section length, 12 + xxx + yyy                           |
| 004  | 002 | Reserved field.                                          |
| 006  | 002 | RSA public key exponent field length in bytes, “xxx”.      |
| 008  | 002 | Public key modulus length in bytes.                       |

| 010  | 002 | RSA public modulus field length in bytes, yyy.              |
| **Note**: If the token contains an RSA private key section, this field length, yyy, should be 0. The RSA private key section contains the modulus. |

| 012  | xxx | Public exponent, e. (This field length is typically 1, 3, or 64 - 512 bytes). e must be odd and 1 ≤ e < n (e is frequently valued to 3, 5, 17, 257, or 65537. Otherwise e is of the same order of magnitude as the modulus). |

| 12 + xxx | yyy | Modulus, n. n = pq, where p and q are prime and 2^{512} ≤ n < 2^{606}. This field is absent when the modulus is contained in the private-key section. If present, the field length range is 64 - 512 bytes. |

**RSA private key token**

The contained subtopics describe the RSA private key tokens for both the external and internal format combined into one table for each token type.
Key token formats

First, the topics “RSA private external key token” and “RSA private internal key token” on page 855 describe the general structure of private external and internal key tokens. The subsequent information units then present the detailed formats of the available key token types.

RSA private external key token

Read the contained information about the basic structure of RSA private external key tokens.

An RSA private external key token contains the following sections:

- a required PKA token header starting with the token identifier X’1E’
- a required RSA private key section, one of those shown in Table 238, each starting with a certain section identifiers
- a required RSA public key section, starting with the section identifier X’04’
- an optional private key name section, starting with the section identifier X’10’.

Table 238 presents the basic record format of an RSA private external key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (big-endian format). All binary fields (exponents, modulus, and so on) in the private sections of tokens are right-aligned and padded with zeros to the left.

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Token Header</strong> (Required)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>001</td>
<td>Token identifier. X’1E’ indicates an external token. The private key is either in cleartext or enciphered with a transport key-encrypting key.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Version, X’00’.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the key token structure.</td>
</tr>
<tr>
<td>004</td>
<td>004</td>
<td>Ignored. Should be zero.</td>
</tr>
<tr>
<td><strong>RSA Private Key Section</strong> (Required)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>See the following sections:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• “RSA private key token, 1024-bit Modulus-Exponent” on page 857</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• “RSA private key token, 1024-bit Modulus-Exponent format with OPK section” on page 859</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• “RSA private key token, 4096-bit Modulus-Exponent” on page 861</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• “RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section” on page 863</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• “RSA private key, 4096-bit Chinese Remainder Theorem with OPK” on page 867</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• “RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section” on page 869</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RSA Public Key Section</strong> (Required)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>001</td>
<td>X’04’, section identifier, RSA public key.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X’00’, version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length, 12 + xxx.</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Reserved field.</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>RSA public key exponent field length in bytes, “xxx”.</td>
</tr>
<tr>
<td>008</td>
<td>002</td>
<td>Public key modulus length in bits.</td>
</tr>
</tbody>
</table>
### RSA private external key token basic record format (continued)

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X'10', section identifier, private key name.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X'00', version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length, X'0044' (68 decimal).</td>
</tr>
<tr>
<td>004</td>
<td>064</td>
<td>Private key name (in ASCII), left-aligned, padded with space characters (X'20'). An access control system can use the private key name to verify the calling application is entitled to use the key. When generating an RSA retained private key, the name supplied in this part of the skeleton key-token is subsequently used in the coprocessor to locate the retained key.</td>
</tr>
</tbody>
</table>

### RSA private internal key token

Read the contained information about the basic structure of RSA private internal key tokens.

An RSA private internal key token contains the following sections:

- A required PKA token header, starting with the token identifier X'1F'
- Basic record format of an RSA private internal key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, big-endian format). All binary fields (exponents, modulus, and so on) in the private sections of tokens are right-aligned and padded with zeros to the left.

Table 239 shows the format.

### RSA private internal key token basic record format

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token Header (Required)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>001</td>
<td>Token identifier. X'1F' indicates an internal token. The private key is enciphered with a PKA master key.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Version, X'00'.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the key token structure excluding the internal information section.</td>
</tr>
<tr>
<td>004</td>
<td>004</td>
<td>Ignored; should be zero.</td>
</tr>
</tbody>
</table>
### Key token formats

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RSA Private Key Section and Secured Subsection</strong> (Required)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See the following sections:

- “RSA private key token, 1024-bit Modulus-Exponent” on page 857.
- “RSA private key token, 1024-bit Modulus-Exponent format with OPK section” on page 859.
- “RSA private key token, 4096-bit Modulus-Exponent” on page 861.
- “RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section” on page 863.
- “RSA private key, 4096-bit Chinese Remainder Theorem with OPK” on page 867.
- “RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section” on page 869.
- “RSA private key token, 1024-bit Modulus-Exponent internal format for cryptographic coprocessor feature” on page 872.

### RSA Public Key Section (Required)

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X’04’, section identifier, RSA public key.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X’00’, version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length, 12 + xxx.</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Reserved field.</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>RSA public key exponent field length in bytes, xxx.</td>
</tr>
<tr>
<td>008</td>
<td>002</td>
<td>Public key modulus length in bits.</td>
</tr>
<tr>
<td>010</td>
<td>002</td>
<td>RSA public key modulus field length in bytes, which is zero for a private token.</td>
</tr>
<tr>
<td>012</td>
<td>xxx</td>
<td>Public key exponent (this is generally a 1, 3, or 64 - 256-byte quantity), e. e must be odd and 1 &lt; e &lt; n. (Frequently, the value of e is 2^16 + 1 (= 65,537). <strong>Note:</strong> You can import an RSA public key having an exponent valued to two (2). Such a public key can correctly validate an ISO 9796-1 digital signature. However, the current product implementation does not generate an RSA key with a public exponent valued to two (a Rabin key).</td>
</tr>
</tbody>
</table>

### Private Key Name (Optional)

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X’10’, section identifier, private key name.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X’00’, version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length, X’0044’ (68 decimal).</td>
</tr>
<tr>
<td>004</td>
<td>064</td>
<td>Private key name (in ASCII), left-aligned, padded with space characters (X’20’). An access control system can use the private key name to verify the calling application is entitled to use the key.</td>
</tr>
</tbody>
</table>

### Internal Information Section (Required)

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>004</td>
<td>Eye catcher ‘PKTN’.</td>
</tr>
<tr>
<td>004</td>
<td>004</td>
<td>PKA token type.</td>
</tr>
<tr>
<td><strong>Bit</strong></td>
<td><strong>Meaning When Set On</strong></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>RSA key.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Private key.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Public key.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Private key name section exists.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Private key unenciphered.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Blinding information present.</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Retained private key.</td>
<td></td>
</tr>
<tr>
<td>008</td>
<td>004</td>
<td>Address of token header.</td>
</tr>
<tr>
<td>012</td>
<td>002</td>
<td>Total length of total structure including this information section.</td>
</tr>
</tbody>
</table>
Table 239. RSA private internal key token basic record format (continued)

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>014</td>
<td>002</td>
<td>Count of number of sections.</td>
</tr>
<tr>
<td>016</td>
<td>016</td>
<td>PKA master key hash pattern.</td>
</tr>
<tr>
<td>032</td>
<td>001</td>
<td>Domain of retained key.</td>
</tr>
<tr>
<td>033</td>
<td>008</td>
<td>Serial number of processor holding retained key.</td>
</tr>
<tr>
<td>041</td>
<td>007</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

**RSA private key token, 1024-bit Modulus-Exponent**

This RSA private key token is supported starting with CEX3C. It is supported as the external X'02' and the internal X'06' token format.

Table 240 shows the external and internal format.

Table 240. RSA private key, 1024-bit Modulus-Exponent format section (X'02')

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X'02', section identifier, RSA private key, Modulus-Exponent format (RSA-PRIV) for the external format</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X'00', version.</td>
</tr>
</tbody>
</table>
| 002              | 002            | External format: Length of the RSA private key section X'016C' (364 decimal).  
External format: Length of the RSA private key section X'0198' (408 decimal) + rrr + iii + xxx. |
| 004              | 020            | External format: SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.  
Internal format: SHA-1 hash value of the private key subsection cleartext, offset 28 to and including the modulus at offset 236. |
| 024              | 004            | Reserved; set to binary zero.                                               |
| 028              | 001            | Key format and security:                                                   |
|                  |                | External format:                                                            |
|                  |                | X'00'  Unencrypted RSA private key subsection identifier.                   |
|                  |                | X'82'  Encrypted RSA private key subsection identifier.                     |
|                  |                | Internal format:                                                            |
|                  |                | X'02'  RSA private key                                                      |
| 029              | 001            | External format: Reserved, binary zero.                                     |
|                  |                | Internal format: Format of external key from which this token was derived: |
|                  |                | X'21'  External private key was specified in the clear.                    |
|                  |                | X'22'  External private key was encrypted.                                 |
|                  |                | X'23'  Private key was generated using regeneration data.                  |
|                  |                | X'24'  Private key was randomly generated.                                 |
| 030              | 020            | SHA-1 hash of the optional key-name section and any following optional sections. If there are no optional sections, this field is set to binary zeros. |
### Key token formats

**Table 240. RSA private key, 1024-bit Modulus-Exponent format section (X'02') (continued)**

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>050</td>
<td>001</td>
<td>Key use flag bits.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'11xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Only key unwrapping (KM-ONLY)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'10xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Both signature generation and key unwrapping (KEY-MGMT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'01xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Undefined</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'00xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Only signature generation (SIG-ONLY)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Translation control flag bits:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx xx1x'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Private key translation is allowed (XLA-TE-OK)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx xx0x'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Private key translation is not allowed (NO-XLA-TE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All other bits reserved, set to binary zero.</td>
</tr>
<tr>
<td>051</td>
<td>009</td>
<td>Reserved; set to binary zero.</td>
</tr>
</tbody>
</table>

**External format: 060 - 235**

| 060  | 024 | Reserved; set to binary zero. |
| 084  | 024 | Random number, confounder. |
| 108  | 128 | Private-key exponent, d. d = e^(-1) mod((p-1)(q-1)), and 1 < d < n where e is the public exponent. |

End of the optionally-encrypted subsection; the confounder field and the private-key exponent field are enciphered for key confidentiality when the key format and security flags (offset 28) indicate the private key is enciphered. They are enciphered under a double-length transport key using the ede2 algorithm.

| 236  | 128 | Modulus, n. n = pq where p and q are prime and 1 < n < 2^1024. |

**External format ends here.**

**Internal format: 060 - 235**

| 060  | 048 | Object Protection Key (OPK) encrypted under the Asymmetric Keys Master Key using the ede3 algorithm. |
| 108  | 128 | Private key exponent d, encrypted under the OPK using the ede5 algorithm. d = e^(-1) mod((p-1)(q-1)), and 1 < d < n where e is the public exponent. |
| 236  | 128 | Modulus, n. n = pq where p and q are prime and 1 < n < 2^1024. |

**Internal format only, until end of table**

| 364  | 016 | Asymmetric-Keys Master Key hash pattern. |
| 380  | 020 | SHA-1 hash value of the blinding information subsection cleartext, offset 400 to the end of the section. |
| 400  | 002 | Length of the random number r, in bytes: rrr |
| 402  | 002 | Length of the random number r^(-1), in bytes: iii |
| 404  | 002 | Length of the padding field, in bytes: xxx |
| 406  | 002 | Reserved; set to binary zeros. |
| 408  |     | Start of the encrypted blinding subsection |
| 408  | rrr | Random number r (used in blinding). |
### Table 240. RSA private key, 1024-bit Modulus-Exponent format section (X'02') (continued)

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>408 + rrr</td>
<td>iii</td>
<td>Random number r¹ (used in blinding).</td>
</tr>
<tr>
<td>408 + rrr +</td>
<td>xxx</td>
<td>X'00' padding of length xxx bytes such that the length from the start of the encrypted blinding subsection to the end of the padding field is a multiple of eight bytes.</td>
</tr>
</tbody>
</table>

End of the encrypted blinding subsection; all of the fields starting with the random number r and ending with the variable length pad field are encrypted under the OPK using TDES (CBC outer chaining) algorithm.

### RSA private key token, 1024-bit Modulus-Exponent format with OPK section

View a table describing the RSA private key token format with section identifier X'06'. For this key token, there is only an internal format available.

### Table 241. RSA private key, 1024-bit Modulus-Exponent format with OPK section (X'06')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Section identifier: X'06' RSA private key, 1024-bit maximum Modulus-Exponent format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This section type is created by the IBM Version 2 and later CCA Support Program. This section type provides compatibility and interchangeability with the CCF hardware in z/OS processors.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length in bytes (X'0198').</td>
</tr>
<tr>
<td>004</td>
<td>020</td>
<td>SHA-1 hash value of the private-key subsection cleartext, offset 28 up to and including the modulus that ends at offset 363.</td>
</tr>
<tr>
<td>024</td>
<td>004</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>028</td>
<td>001</td>
<td>Key format and security flag byte.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal key-token: X'02' Encrypted RSA private key with OPK subsection identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All other values are reserved and undefined.</td>
</tr>
<tr>
<td>029</td>
<td>001</td>
<td>Private key source flag byte: X'21' Imported from cleartext X'22' Imported from ciphertext X'23' Generated using regeneration data X'24' Randomly generated</td>
</tr>
<tr>
<td>030</td>
<td>020</td>
<td>SHA-1 hash of all optional sections that follow the public-key section, if any, else 20 bytes of X'00'.</td>
</tr>
</tbody>
</table>
### Key token formats

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>050</td>
<td>001</td>
<td>Key-usage and translation control flag byte.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key usage:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'11xx xxxx' Only key unwrapping (KM-ONLY)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'10xx xxxx' Both signature generation and key unwrapping (KEY-MGMT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'01xx xxxx' Undefined</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'00xx xxxx' Only signature generation (SIG-ONLY)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Translation control:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx xx1x' Private key translation is allowed (XLAT-OK)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx xx0x' Private key translation is not allowed (NO-XLAT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All other bits are reserved and must be zero.</td>
</tr>
<tr>
<td>051</td>
<td>001</td>
<td>Format restriction for digital-signature hash-formatting method:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0000 0000'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0000 0001'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0000 0010'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0000 0011'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0000 0100'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0000 0101'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0000 0110'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All other values are reserved and undefined.</td>
</tr>
<tr>
<td>052</td>
<td>002</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>054</td>
<td>006</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>060</td>
<td>048</td>
<td>Object Protection Key (OPK) data. 8-byte confounder, three 8-byte DES keys, and two 8-byte initialization vector values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>External key-token: Reserved, binary zero.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal key-token: The asymmetric master key encrypts the OPK data using the EDE3 algorithm. See “Triple-DES ciphering algorithms” on page 1030.</td>
</tr>
<tr>
<td>108</td>
<td>128</td>
<td>Private-key exponent, ( d ), ( d = e' \mod ((p-1)(q-1)) ), ( 1 &lt; d &lt; n ), and where ( e ) is the public exponent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The OPK encrypts the private key exponent using the EDE5 algorithm. See “Triple-DES ciphering algorithms” on page 1030.</td>
</tr>
<tr>
<td>236</td>
<td>128</td>
<td>Modulus, ( n = pq ), where ( p ) and ( q ) are prime and ( 2^{252} \leq n &lt; 2^{1024} ).</td>
</tr>
<tr>
<td>364</td>
<td>016</td>
<td>Asymmetric-keys master-key verification pattern.</td>
</tr>
<tr>
<td>380</td>
<td>020</td>
<td>SHA-1 hash value of the subsection cleartext, offset 400 to the section end. This hash value is checked after an enciphered private key is deciphered for use.</td>
</tr>
</tbody>
</table>
Table 241. RSA private key, 1024-bit Modulus-Exponent format with OPK section (X'06') (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>002</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>402</td>
<td>002</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>404</td>
<td>002</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>406</td>
<td>002</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

RSA private key token, 4096-bit Modulus-Exponent

This RSA private key token is supported on a CCA Crypto Express coprocessor (external and internal X'09' token).

Table 242. RSA Private Key Token, 4096-bit Modulus-Exponent

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Number of bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X'09', section identifier, RSA private key, modulus-exponent format (RSAMEVAR). This format is used for a clear or an encrypted RSA private-key in an external key-token up to a modulus size of 4096 bits.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the RSA private key section: 132+ddd+nnn+xxx.</td>
</tr>
<tr>
<td>004</td>
<td>020</td>
<td>SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.</td>
</tr>
<tr>
<td>024</td>
<td>002</td>
<td>External format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Length of the encrypted private key section 8+ddd+xxx.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Length in bytes of the optionally encrypted secure subsection, or X'0000' if the subsection is not encrypted.</td>
</tr>
<tr>
<td>026</td>
<td>002</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>028</td>
<td>001</td>
<td>Key format and security flags:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>External token:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00' Unencrypted RSA private-key subsection identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'82' Encrypted RSA private-key subsection identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All other values are reserved and undefined.</td>
</tr>
<tr>
<td>029</td>
<td>001</td>
<td>External format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reserved, set to binary zero.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Private key source flag:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00' Generation method unknown</td>
</tr>
<tr>
<td>030</td>
<td>020</td>
<td>SHA-1 hash of the optional key-name section. If there is no key-name section, then 20 bytes of X'00'.</td>
</tr>
</tbody>
</table>
### Key token formats

**Table 242. RSA Private Key Token, 4096-bit Modulus-Exponent (continued)**

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Number of bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>050</td>
<td>001</td>
<td>Key-usage and translation control flag byte.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key usage:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'11xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Only key unwrapping (KM-ONLY)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'10xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Both signature generation and key unwrapping (KEY-MGMT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'01xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Undefined</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'00xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Undefined</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Translation control:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx xx1x'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Private key translation is allowed (XLABTE-OK)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx xx0x'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Private key translation is not allowed (NO-XLABTE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All other bits are reserved and must be zero.</td>
</tr>
<tr>
<td>051</td>
<td>065</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>116</td>
<td>002</td>
<td>Private-key exponent field length, in bytes: ddd.</td>
</tr>
<tr>
<td>118</td>
<td>002</td>
<td>Private-key modulus field length, in bytes: nnn.</td>
</tr>
<tr>
<td>120</td>
<td>002</td>
<td>Length of padding field, in bytes: xxx.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Padding of X'00' bytes for a length of xxx bytes such that the length from the start of the confounder at offset 124 to the end of the padding field is a multiple of 8 bytes.</td>
</tr>
<tr>
<td>122</td>
<td>002</td>
<td>Reserved; set to binary zero.</td>
</tr>
</tbody>
</table>

Start of the (optionally) encrypted subsection; all of the fields starting with the confounder field and ending with the variable-length pad field are enciphered for key confidentiality when the key format and security flags (offset 28) indicate that the private key is enciphered.

| 124              | 008             | Confounder. |
|                  |                 | This is an eight-byte random number. |
|                  |                 | Data encrypted with two-part key-encrypting key. |
| 132              | ddd             | Private-key exponent, d: |
|                  |                 | d = e⁻¹ mod((p - 1)(q - 1)) |
|                  |                 | where 1 < d < n, and e is the public exponent. |
|                  |                 | The transport key encrypts the private key exponent using the EDE2 algorithm. |
| 132 + ddd        | xxx             | Pad of X'00' bytes. |

End of the optionally encrypted subsection.

| 132 + ddd + xxx  | nnn             | Private-key modulus. |
**RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section**

View the RSA private key token, 4096-bit Modulus-Exponent format with AES encrypted OPK section in external and internal format (X'30').

Table 243. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') external and internal form

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Section identifier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'30'  RSA private key, 4096-bit Modulus-Exponent format (RSA-AESM) with AES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>encrypted OPK.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length: 122 + nnn + ppp;</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Length of Associated Data section</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>Length of payload data: ppp</td>
</tr>
<tr>
<td>008</td>
<td>002</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Start of Associated Data</strong></td>
</tr>
<tr>
<td>010</td>
<td>001</td>
<td>Associated Data Version:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'  Version 2</td>
</tr>
<tr>
<td>011</td>
<td>001</td>
<td>Key format and security flag:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>External format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'  Unencrypted ME RSA private-key subsection identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'82'  Encrypted ME RSA private-key subsection identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'  Encrypted ME RSA private-key subsection identifier</td>
</tr>
<tr>
<td>012</td>
<td>001</td>
<td>Key source flag:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>External key-token:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal key-token:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'21'  Imported from cleartext</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'22'  Imported from ciphertext</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'23'  Generated using regeneration data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'24'  Randomly generated</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All other values are reserved and undefined.</td>
</tr>
<tr>
<td>013</td>
<td>001</td>
<td>Reserved, binary zeroes.</td>
</tr>
<tr>
<td>014</td>
<td>001</td>
<td>Hash type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'  Clear key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'  SHA-256</td>
</tr>
<tr>
<td>015</td>
<td>032</td>
<td>SHA-256 hash of all optional sections that follow the public key section,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>if any; else 32 bytes of X'00'.</td>
</tr>
<tr>
<td>047</td>
<td>003</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>
### Key token formats

**Table 243. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') external and internal form (continued)**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>050</td>
<td>001</td>
<td>Key-usage flag:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'11xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Only key unwrapping (KM-ONLY)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'10xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Both signature generation and key unwrapping (KEY-MGMT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'01xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Undefined</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'00xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Only signature generation (SIG-ONLY)</td>
</tr>
</tbody>
</table>

Translation control:

| B'xxxx xx1x' |
| Private key translation is allowed (XLA-TE-OK) |
| B'xxxx xx0x' |
| Private key translation is not allowed (NO-XLA-TE) |

| 051            | 001            | Format restriction for digital-signature hash-formatting method: |
|                |                | Value | Meaning  |
|                |                | B'0000 0000' | No format restriction |
|                |                | B'0000 0001' | ISO-9796 only |
|                |                | B'0000 0010' | PKCS-1.0 only |
|                |                | B'0000 0011' | PKCS-1.1 only |
|                |                | B'0000 0100' | PKCS-PSS only |
|                |                | B'0000 0101' | X9.31 only |
|                |                | B'0000 0110' | ZERO-PAD only |

All other values are reserved and undefined.

| 052            | 002            | Length of modulus: nnn bytes |
| 054            | 002            | Length of private exponent: ddd bytes |

**End of Associated Data**

| 056            | 048            | 16 byte confounder + 32-byte Object Protection Key. |
|                |                | OPK used as an AES key. |
|                |                | External key-token: Encrypted with an AES key-encrypting key. |
|                |                | Internal key-token: Encrypted with the APKA master key. |
Table 243. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X’30’) external and internal form (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
<td>016</td>
<td>Key verification pattern</td>
</tr>
<tr>
<td></td>
<td></td>
<td>External key-token:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Encrypted private key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key-encrypting key verification pattern</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clear private key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Binary zero</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Skeleton</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Binary zero</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal key-token:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Encrypted private key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>APKA master-key verification pattern</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Skeleton</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Binary zero</td>
</tr>
<tr>
<td>120</td>
<td>002</td>
<td>Reserved, binary zeros.</td>
</tr>
<tr>
<td>122</td>
<td>nnn</td>
<td>Modulus</td>
</tr>
<tr>
<td>122+nnn</td>
<td>ppp</td>
<td>Payload starts here and includes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When this section is unencrypted:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Clear private exponent d.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Length ppp bytes : ddd + 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When this section is encrypted:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Private exponent d within the AESKW-wrapped payload.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Length ppp bytes : ddd + AESKW format overhead</td>
</tr>
</tbody>
</table>

RSA private key, 2048-bit Chinese Remainder Theorem

View the RSA private key token, 2048-bit Chinese Remainder Theorem (X’05’). This is an asymmetric key token.

Table 244. RSA private key, 2048-bit Chinese Remainder Theorem format section (X’05’)

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Number of bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X’05’, section identifier, RSA private key, 2048-bit maximum CRT (replaced by RSA-CRT) format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This section type is no longer created by CCA.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X’00’).</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length in bytes (76 + ppp + qqq + rrr + sss + ttt + uuu + xxx + nnn).</td>
</tr>
<tr>
<td>004</td>
<td>020</td>
<td>SHA-1 hash value of the private-key subsection cleartext, offset 28 to the end of the modulus.</td>
</tr>
<tr>
<td>024</td>
<td>002</td>
<td>Length in bytes of the optionally encrypted secure subsection, or X’0000’ if the subsection is not encrypted.</td>
</tr>
<tr>
<td>026</td>
<td>002</td>
<td>Master-key verification pattern in an internal key-token, else X’0000’.</td>
</tr>
</tbody>
</table>
### Key Token Formats

**Table 244. RSA Private Key, 2048-bit Chinese Remainder Theorem Format Section (X’05’) (continued)**

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Number of bytes</th>
<th>Description</th>
</tr>
</thead>
</table>
| 028              | 001             | Key format and security flag byte.  
|                  |                 | External key-token:  
|                  |                 | X’40’ Unencrypted RSA private-key subsection identifier,  
|                  |                 | Chinese-Remainder Theorem format  
|                  |                 | Internal key-token:  
|                  |                 | X’42’ Encrypted RSA private-key subsection identifier,  
|                  |                 | Chinese-Remainder Theorem format  
|                  |                 | All other values are reserved and undefined. |
| 029              | 001             | Reserved, binary zero. |
| 030              | 020             | SHA-1 hash of the optional key-name section; if there is no name section, then 20 bytes of X’00’. |
| 050              | 001             | Key-usage flag byte.  
|                  |                 | Key usage:  
|                  |                 | B’11xx xxxx’  
|                  |                 | Only key unwrapping (KM-ONLY)  
|                  |                 | B’10xx xxxx’  
|                  |                 | Both signature generation and key unwrapping (KEY-MGMT)  
|                  |                 | B’01xx xxxx’  
|                  |                 | Undefined  
|                  |                 | B’00xx xxxx’  
|                  |                 | Undefined  
|                  |                 | All other bits are reserved and must be zero. |
| 051              | 001             | Reserved, binary zero. |
| 052              |                 | Start of the optionally encrypted subsection.  
|                  |                 | Private key encryption:  
|                  |                 | • External token: EDE2 process using the double-length transport key  
|                  |                 | • Internal token: EDE3 process using the asymmetric master key.  
|                  |                 | See "Triple-DES ciphering algorithms" on page 1030. |
| 052              | 008             | Random number, confounder. |
| 060              | 002             | Length of prime number, \(p\), in bytes: \(ppp\). |
| 062              | 002             | Length of prime number, \(q\), in bytes: \(qqq\). |
| 064              | 002             | Length of \(d_p\) in bytes: \(rrr\). |
| 066              | 002             | Length of \(d_q\) in bytes: \(sss\). |
| 068              | 002             | Length of \(A_p\) in bytes: \(ttt\). |
| 070              | 002             | Length of \(A_q\) in bytes: \(uuu\). |
| 072              | 002             | Length of modulus, \(n\), in bytes: \(nnn\). |
| 074              | 002             | Length of padding field, in bytes: \(xxx\). |
| 076              | \(ppp\)         | Prime number, \(p\). |
| 076+\(ppp\)      | \(qqq\)         | Prime number, \(q\). |
| 076+\(ppp+qqq\)  | \(rrr\)         | \(d_p = d \mod (p - 1)\). |
Table 244. RSA private key, 2048-bit Chinese Remainder Theorem format section (X’05’) (continued)

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Number of bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>076+ppp+qqq+rrr</td>
<td>sss</td>
<td>$d_i = d \text{ mod}(q - 1)$.</td>
</tr>
<tr>
<td>076+ppp+qqq+rrr</td>
<td>sss+ttt</td>
<td>$A_p = q^{e-1} \text{ mod}(n)$.</td>
</tr>
<tr>
<td>076+ppp+qqq+rrr</td>
<td>sss+ttt+uuu</td>
<td>$A_q \cdot (n + 1 - A_p)$.</td>
</tr>
<tr>
<td>076+ppp+qqq+rrr</td>
<td>sss+ttt+uuu+xxx</td>
<td>$X'00'$ padding of length $xxx$ bytes such that the length from the start of the random number above to the end of the padding field is a multiple of 8 bytes.</td>
</tr>
</tbody>
</table>

End of the optionally encrypted subsection; all of the fields starting with the confounder field and ending with the variable-length pad field are enciphered for key confidentiality when the key-format-and-security flag byte (offset 28) indicates that the private key is enciphered.

RSA private key, 4096-bit Chinese Remainder Theorem with OPK

This RSA private key token with up to 4096-bit modulus is supported on the System z9® or System z10™, or later machines with the Nov. 2007 or later version of the licensed internal code installed on the CCA Crypto Express coprocessor.

Table 245. RSA private key token, 4096-bit Chinese Remainder Theorem with OPK section (X’08’)

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Number of bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X'08’, section identifier, RSA private key, CRT format (RSA-CRT) with OPK</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X'00’, version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the RSA private-key section, 132 + sss+ttt+uuu+xxx+nnn.</td>
</tr>
<tr>
<td>004</td>
<td>020</td>
<td>SHA-1 hash value of the private-key subsection cleartext, offset 28 to the end of the modulus.</td>
</tr>
<tr>
<td>024</td>
<td>004</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>028</td>
<td>001</td>
<td>Key format and security flag:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>External key token:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’40’ Unencrypted RSA private-key subsection identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’42’ Encrypted RSA private-key subsection identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal key token:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’08’ Encrypted RSA private-key subsection identifier, Chinese Remainder form.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All other values are reserved and undefined.</td>
</tr>
</tbody>
</table>
## Key token formats

Table 245. RSA private key token, 4096-bit Chinese Remainder Theorem with OPK section (X'08') (continued)

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Number of bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>029</td>
<td>001</td>
<td>Key source flag byte:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>External key tokens: Reserved, binary zero.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal key tokens:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'21' External private key was specified in the clear.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'22' External private key was encrypted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'23' Private key was generated using regeneration data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'24' Private key was randomly generated.</td>
</tr>
<tr>
<td>030</td>
<td>020</td>
<td>SHA-1 hash of the optional key-name section and any following sections. If there are no optional sections, then 20 bytes of X'00'.</td>
</tr>
<tr>
<td>050</td>
<td>001</td>
<td>Key use and translation control flag byte.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key usage:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'11xx xxxx' Only key unwrapping (KM-ONLY)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'10xx xxxx' Both signature generation and key unwrapping (KEY-MGMT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'01xx xxxx' Undefined</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'00x xxxx' Undefined</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Translation control:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx xx1x' Private key translation is allowed (XLA-TE-OK)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx xx0x' Private key translation is not allowed (NO-XLA-TE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All other bits reserved, set to binary zero.</td>
</tr>
<tr>
<td>051</td>
<td>003</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>054</td>
<td>002</td>
<td>Length of prime number p in bytes: ppp.</td>
</tr>
<tr>
<td>056</td>
<td>002</td>
<td>Length of prime number q in bytes: qqq.</td>
</tr>
<tr>
<td>058</td>
<td>002</td>
<td>Length of d_p in bytes: rrr.</td>
</tr>
<tr>
<td>060</td>
<td>002</td>
<td>Length of d_q in bytes: sss.</td>
</tr>
<tr>
<td>062</td>
<td>002</td>
<td>Length of U in bytes: uuu.</td>
</tr>
<tr>
<td>064</td>
<td>002</td>
<td>Length of modulus n in bytes: nnn.</td>
</tr>
<tr>
<td>066</td>
<td>002</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>068</td>
<td>002</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>070</td>
<td>002</td>
<td>Length of padding field in bytes: xxx.</td>
</tr>
<tr>
<td>072</td>
<td>004</td>
<td>Reserved, set to binary zero.</td>
</tr>
<tr>
<td>076</td>
<td>016</td>
<td>External key token: Reserved, binary zero.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal key token: Asymmetric master-key verification pattern.</td>
</tr>
</tbody>
</table>
### RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section

View the format of an RSA private key token, 4096-bit Chinese Remainder Theorem, with AES encrypted OPK (RSA-AESC) format (X'31'), in internal and external format.

#### Table 246. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Section identifier: X'31' RSA private key, 4096-bit Chinese-Remainder Theorem format with AES encrypted OPK (RSA-AESC)</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X'00')</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length: 134 + nnn + xxx</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Length in bytes of Associated Data section</td>
</tr>
</tbody>
</table>
### Key token formats

**Table 246. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') (continued)**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>006</td>
<td>002</td>
<td>Length in bytes of payload data: xxx</td>
</tr>
<tr>
<td>008</td>
<td>002</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

Start of Associated Data section

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>001</td>
<td>Associated Data version:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03' Version 3</td>
</tr>
<tr>
<td>011</td>
<td>001</td>
<td>Key format and security flag:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>External key-token:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'40' Unencrypted RSA private-key subsection identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'42' Encrypted RSA private-key subsection identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal key-token:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'08' Encrypted RSA private-key subsection identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All other values are reserved and undefined.</td>
</tr>
<tr>
<td>012</td>
<td>001</td>
<td>Key source flag:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>External key-token: Reserved, binary zero.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal key-token:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'21' Imported from cleartext</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'22' Imported from ciphertext</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'23' Generated using regeneration data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'24' Randomly generated</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All other values are reserved and undefined.</td>
</tr>
<tr>
<td>013</td>
<td>001</td>
<td>Reserved, binary zeroes.</td>
</tr>
<tr>
<td>014</td>
<td>001</td>
<td>Hash type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00' Clear key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02' SHA-256</td>
</tr>
<tr>
<td>015</td>
<td>032</td>
<td>SHA-256 hash of all optional sections that follow the public key section, if any. Otherwise 32 bytes of X'00'.</td>
</tr>
<tr>
<td>047</td>
<td>003</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>
### Key token formats

#### Table 246. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>050</td>
<td>001</td>
<td>Key-usage flag:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'11xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Only key unwrapping (KM-ONLY)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'10xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Both signature generation and key unwrapping (KEY-MGMT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'01xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Undefined</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'00xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Only signature generation (SIG-ONLY)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Translation control flag:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx xx1x'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Private key translation is allowed (XLA-TE-OK)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx xx0x'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Private key translation is not allowed (NO-XLA-TE)</td>
</tr>
<tr>
<td>051</td>
<td>001</td>
<td>Format restriction for digital-signature hash-formatting method:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0000 0000'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0000 0001'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0000 0010'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0000 0011'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0000 0100'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0000 0101'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0000 0110'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All other values are reserved and undefined.</td>
</tr>
<tr>
<td>052</td>
<td>002</td>
<td>Length of the prime number, p, in bytes: ppp.</td>
</tr>
<tr>
<td>054</td>
<td>002</td>
<td>Length of the prime number, q, in bytes: qqq</td>
</tr>
<tr>
<td>056</td>
<td>002</td>
<td>Length of dp : rrr.</td>
</tr>
<tr>
<td>058</td>
<td>002</td>
<td>Length of dq : sss.</td>
</tr>
<tr>
<td>060</td>
<td>002</td>
<td>Length of U: uuu.</td>
</tr>
<tr>
<td>062</td>
<td>002</td>
<td>Length of modulus n: nnn.</td>
</tr>
<tr>
<td>064</td>
<td>004</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

#### End of Associated Data

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>068</td>
<td>048</td>
<td>Object Protection Key (OPK) data: 16-byte confounder followed by 32-byte AES key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>External key-token: Encrypted with an AES key-encrypting key (AES KEK)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal key-token: Encrypted with the APKA master key.</td>
</tr>
</tbody>
</table>
### Key token formats

Table 246. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X’31’) (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>116</td>
<td>016</td>
<td>Key verification pattern</td>
</tr>
<tr>
<td></td>
<td></td>
<td>External key-token:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Encrypted private key</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key-encrypting key verification pattern</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Clear private key</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Binary zero</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Skeleton</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Binary zero</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal key-token:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Encrypted private key</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>APKA master-key verification pattern</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Skeleton</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Binary zero</td>
</tr>
<tr>
<td>132</td>
<td>002</td>
<td>Reserved, binary zeros</td>
</tr>
<tr>
<td>134</td>
<td>nnn</td>
<td>Modulus, n, n=pq, where p and q are prime.</td>
</tr>
<tr>
<td>134+nnn</td>
<td>xxx</td>
<td>Payload starts here and includes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When this section is unencrypted:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Clear prime number p</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Clear prime number q</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Clear d_p</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Clear d_q</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Clear U</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Length xxx bytes: ppp + qqq + rrr + sss +uuu + 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When this section is encrypted:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>It contains key within the AESKW-wrapped payload. The encrypted-key section is a blob that will never be revealed in the clear, and therefore is not defined here.</td>
</tr>
</tbody>
</table>

### RSA private key token, 1024-bit Modulus-Exponent internal format for cryptographic coprocessor feature

The format of the RSA private key token, 1024-bit Modulus-Exponent internal format for cryptographic coprocessor feature.

Table 247 shows the format of the RSA private key token, 1024-bit Modulus-Exponent internal format for cryptographic coprocessor feature.

Table 247. RSA private internal key token, 1024-bit Modulus-Exponent format for cryptographic coprocessor feature

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X’02’, section identifier, RSA private key.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X’00’, version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the RSA private key section X’016C’ (364 decimal).</td>
</tr>
</tbody>
</table>
Table 247. RSA private internal key token, 1024-bit Modulus-Exponent format for cryptographic coprocessor feature (continued)

<table>
<thead>
<tr>
<th>Offset (decimal)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>004</td>
<td>020</td>
<td>SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.</td>
</tr>
<tr>
<td>024</td>
<td>004</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>028</td>
<td>001</td>
<td>Key format and security: X'02' RSA private key.</td>
</tr>
<tr>
<td>029</td>
<td>001</td>
<td>Format of external key from which this token was derived:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'21' External private key was specified in the clear.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'22' External private key was encrypted.</td>
</tr>
<tr>
<td>030</td>
<td>020</td>
<td>SHA-1 hash of the key token structure contents that follow the public key section. If no sections follow, this field is set to binary zeros.</td>
</tr>
<tr>
<td>050</td>
<td>001</td>
<td>Key use flag bits.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'11xx xxxx' Only key unwrapping (KM-ONLY)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'10xx xxxx' Both signature generation and key unwrapping (KEY-MGMT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'01xx xxxx' Undefined</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'00xx xxxx' Only signature generation (SIG-ONLY)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All other bits reserved, set to binary zero.</td>
</tr>
<tr>
<td>051</td>
<td>009</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>060</td>
<td>048</td>
<td>Object Protection Key (OPK) encrypted under a PKA master key—can be under the Signature Master Key (SMK) or Key Management Master Key (KMMK) depending on key use.</td>
</tr>
<tr>
<td>108</td>
<td>128</td>
<td>Secret key exponent ( d ), encrypted under the OPK. ( d = e^{-1} \mod((p-1)(q-1)) )</td>
</tr>
<tr>
<td>236</td>
<td>128</td>
<td>Modulus, ( n = pq ) where ( p ) and ( q ) are prime and ( 1 &lt; n &lt; 2^{1024} ).</td>
</tr>
</tbody>
</table>

**RSA variable Modulus-Exponent token**


Table 248 describes the fields in the new variable length Modulus-Exponent token. Currently, only the external form of the token will be used. There are no blinding values for the token. The latest level hardware makes this unnecessary.

Table 248. RSA variable Modulus-Exponent token format

<table>
<thead>
<tr>
<th>Number</th>
<th>If External Key</th>
<th>New version '09' field</th>
<th>If Internal Key</th>
<th>Length in bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'09'</td>
<td>sectionId</td>
<td>'09'</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>'00'</td>
<td>version</td>
<td>'00'</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>132 + dLength + nLength + padLength</td>
<td>sectionLength</td>
<td>132 + dLength + nLength + padLength</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Hash over fields 7 - end of section (clear values)</td>
<td>sha1Hash</td>
<td>Hash over fields 7 - end of section</td>
<td>20</td>
</tr>
</tbody>
</table>
### Key token formats

#### Table 248. RSA variable Modulus-Exponent token format (continued)

<table>
<thead>
<tr>
<th>Number</th>
<th>If External Key</th>
<th>New version '09 field</th>
<th>If Internal Key</th>
<th>Length in bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>8 + dLength + padLength</td>
<td>encrypted sectionLength</td>
<td>8 + dLength + padLength</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>This is actually a reserved field, not a pad '0000'</td>
<td>pad</td>
<td>'0000'</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>'82' encrypted external key or '00' clear external key</td>
<td>keyFormat</td>
<td>'02' encrypted operational key</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>'00'</td>
<td>pedigree</td>
<td>'21', '22', '23', or '24' as '06' token</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>Hash over sections which follow the public key section, or '00'</td>
<td>sha1KeyNameHash</td>
<td>Hash over sections which follow the public key section, or '00'</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>'02' indicates that the key is translatable</td>
<td>keyUsageFlag</td>
<td>same as in '06'</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>'00'</td>
<td>reserved1</td>
<td>'00'</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>Binary zeroes</td>
<td>OPK</td>
<td>8 byte confounder + 40-byte (5-part) DES key, encrypted with the PKA master key</td>
<td>48</td>
</tr>
<tr>
<td>13</td>
<td>Binary zeroes</td>
<td>mkHash Pattern</td>
<td>16 byte MKVP</td>
<td>16</td>
</tr>
<tr>
<td>14</td>
<td>Length of private exponent</td>
<td>dLength</td>
<td>Length of private exponent</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>Length of modulus</td>
<td>nLength</td>
<td>Length of modulus</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>Length required to pad dLength to a multiple of 8</td>
<td>padLength</td>
<td>Length required to pad dLength to a multiple of 8</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>'0000'</td>
<td>reserved2</td>
<td>'0000'</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>Random value - encrypted data (with PKA MK) begins here</td>
<td>confounder</td>
<td>encrypted data (with 5-part OPK) begins here</td>
<td>8</td>
</tr>
<tr>
<td>19</td>
<td>&lt;d follows, then pad, then n&gt;</td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

### ECC key token

The format of ECC public and private key tokens.

Table 253 on page 878 and Table 251 on page 875 show the format of ECC public and private key tokens.

CCA allows a choice between two types of elliptic curves when generating an ECC key. One is Brainpool, and the other is Prime. Table 249 and Table 250 on page 875 show the size and name of each supported elliptic curve, along with its object identifier (OID) in dot notation.

#### Table 249. Supported Prime elliptic curves by size, name, and object identifier

<table>
<thead>
<tr>
<th>Size of prime ( p ) in bits (key length)</th>
<th>OID in dot notation</th>
<th>ANSI X9.62 ECDSA prime curve ID</th>
<th>NIST-recommended elliptic curve ID</th>
<th>SEC 2 recommended elliptic curve domain parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>192</td>
<td>1.2.840.10045.3.1.1</td>
<td>prime192v1</td>
<td>P-192</td>
<td>secp192r1</td>
</tr>
<tr>
<td>224</td>
<td>1.3.132.0.33</td>
<td>N/A</td>
<td>P-224</td>
<td>secp224r1</td>
</tr>
<tr>
<td>256</td>
<td>1.2.840.10045.3.1.7</td>
<td>prime256v1</td>
<td>P-256</td>
<td>secp256r1</td>
</tr>
<tr>
<td>384</td>
<td>1.3.132.0.34</td>
<td>N/A</td>
<td>P-384</td>
<td>secp384r1</td>
</tr>
</tbody>
</table>
### Table 249. Supported Prime elliptic curves by size, name, and object identifier (continued)

<table>
<thead>
<tr>
<th>Size of prime $p$ in bits (key length)</th>
<th>OID in dot notation</th>
<th>ANSI X9.62 ECDSA prime curve ID</th>
<th>NIST-recommended elliptic curve ID</th>
<th>SEC 2 recommended elliptic curve domain parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>521</td>
<td>1.3.132.0.35</td>
<td>N/A</td>
<td>P-521</td>
<td>secp521r1</td>
</tr>
</tbody>
</table>

### Table 250. Supported Brainpool elliptic curves by size, name, and object identifier

<table>
<thead>
<tr>
<th>Size of prime $p$ in bits (key length)</th>
<th>OID in dot notation</th>
<th>Brainpool elliptic curve ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>160</td>
<td>1.3.36.3.2.8.1.1.1</td>
<td>brainpoolP160r1</td>
</tr>
<tr>
<td>192</td>
<td>1.3.36.3.2.8.1.1.3</td>
<td>brainpoolP192r1</td>
</tr>
<tr>
<td>224</td>
<td>1.3.36.3.2.8.1.1.5</td>
<td>brainpoolP224r1</td>
</tr>
<tr>
<td>256</td>
<td>1.3.36.3.2.8.1.1.7</td>
<td>brainpoolP256r1</td>
</tr>
<tr>
<td>320</td>
<td>1.3.36.3.2.8.1.1.9</td>
<td>brainpoolP320r1</td>
</tr>
<tr>
<td>384</td>
<td>1.3.36.3.2.8.1.1.11</td>
<td>brainpoolP384r1</td>
</tr>
<tr>
<td>512</td>
<td>1.3.36.3.2.8.1.1.13</td>
<td>brainpoolP512r1</td>
</tr>
</tbody>
</table>

### Table 251. ECC private-key section (X'20')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Section identifier: X'20' ECC private key (ECC-PAIR)</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number:</td>
</tr>
<tr>
<td></td>
<td>Value X'00'</td>
<td>Legacy section version, no pedigree field, no IBM Extended Associated Data (IEAD), no support for ECC key-derivation information section (X'23').</td>
</tr>
<tr>
<td></td>
<td>Meaning</td>
<td>Note: Use of this section version is provided for backward compatibility. Version 1 has additional security features and should be used instead. Latest section version. Includes pedigree field, IEAD, SHA-256 hash of all optional sections to IEAD, supports ECC key-derivation information section (X'23').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of section in bytes.</td>
</tr>
<tr>
<td>004</td>
<td>001</td>
<td>Wrapping method: X'00' Section is unencrypted (clear) X'01' AESKW (ANS X9.102)</td>
</tr>
<tr>
<td>005</td>
<td>001</td>
<td>Algorithm used to hash associated data section: X'00' None (no key present or key is clear key) X'02' SHA-256</td>
</tr>
<tr>
<td></td>
<td>Note:</td>
<td>The message digest is calculated on the associated data section, offset 76 to offset 92 + kl + lead + uad. The message digest becomes part of the payload prior to its encryption.</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>Offset (bytes)</td>
<td>Length (bytes)</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 008           | 001            | Key-usage flag.  
Management of symmetric keys and generation of digital signatures:  
B'11xx xxxx'  
Only key establishment (KM-ONLY)  
B'10xx xxxx'  
Both signature generation and key establishment (KEY-MGMT)  
B'01xx xxxx'  
Undefined  
B'00xx xxxx'  
Only signature generation (SIG-ONLY)  
Translation control:  
B'xxxx xx1x'  
Private key translation is allowed (XLA-TE-OK)  
B'xxxx xx0x'  
Private key translation is not allowed (NO-XLA-TE) |
| 009           | 001            | Curve type:  
X'00'  
Prime  
X'01'  
Brainpool |
| 010           | 001            | Key format and security flag:  
X'08'  
Encrypted internal ECC private key  
X'40'  
Unencrypted external ECC private key  
X'42'  
Encrypted external ECC private key |
| 011           | 001            | Section version X'00' (see offset 001):  
Reserved, binary zero  
Section version X'01' (see offset 001):  
Pedigree/Key source flag byte  
External key-token:  
Value  
X'00'  
None/Clear  
X'21'  
Imported from cleartext  
X'22'  
Imported from ciphertext  
X'24'  
Randomly generated |
|               |                | Internal key-token:  
Value  
X'00'  
None/Clear  
X'21'  
Imported from cleartext  
X'22'  
Imported from ciphertext  
X'24'  
Randomly generated |
| 012           | 002            | Length of prime $p$ in bits. See [Table 249 on page 874](#) and [Table 250 on page 875](#)  
X'00A0'  
160 (Brainpool)  
X'00C0'  
192 (Brainpool, Prime)  
X'00E0'  
224 (Brainpool, Prime)  
X'0100'  
256 (Brainpool, Prime)  
X'0140'  
320 (Brainpool)  
X'0180'  
384 (Brainpool, Prime)  
X'0200'  
512 (Brainpool)  
X'0209'  
521 (Prime) |
| 014           | 002            | Length in bytes of IBM associated data. |
### Table 251. ECC private-key section (X'20') (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 016            | 008            | Key verification pattern.  
|                |                | External key-token  
|                |                | • For an encrypted private key, KEK verification pattern (KVP)  
|                |                | • For a clear private key, binary zeros  
|                |                | • For a skeleton, binary zeros  
|                |                | Internal key-token  
|                |                | • For encrypted private key, master-key verification pattern (MKVP)  
|                |                | • For a skeleton, binary zeros  
| 024            | 048            | Object Protection Key (OPK).  
|                |                | External key-token: Reserved, binary zero.  
|                |                | Internal key token:  
|                |                | OPK data consists of an 8-byte integrity check value (ICV) and length indicators, an 8-byte confounder, and a 256-bit AES key used with the AESKW algorithm to encrypt the ECC private key contained in an AESKW formatted section.  
|                |                | **Note:** The OPK is encrypted by the APKA master key using AESKW (ANS X9.102). The OPK has no associated data.  
| 072            | 002            | Length in bytes of associated data.  
| 074            | 002            | Length in bytes of formatted section, bb.  

#### Associated data section

**Start of IBM associated data**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 076            | 001            | Associated data section version number.  
|                |                | Includes IBM associated data and user-definable associated data.  
|                |                | **Value** | **Meaning** |
|                |                | X'00' | Legacy associated data section version. Only defined for section version number X'00' (see offset 001).  
|                |                | X'01' | Latest associated data section version. Only defined for section version number X'01' (see offset 001).  
| 077            | 001            | Length in bytes of the key label: kl (0 - 64).  
| 078            | 002            | Length in bytes of the IBM associated data (AD), including key label and IBM extended associated data.  
|                |                | **AD data section version number (see offset 076)**  
|                |                | **Length of AD**  
|                |                | X'00' | ≥ 16  
|                |                | X'01' | ≥ 52  
| 080            | 002            | Length in bytes of the IBM extended associated data: iead.  
|                |                | **AD data section version number (see offset 076)**  
|                |                | **Length of AD**  
|                |                | X'00' | iead = 0  
|                |                | X'01' | iead = 36  
| 082            | 001            | Length in bytes of the user-definable associated data: uad (0 - 100).  
| 083            | 001            | Curve type (see offset 009).  

---

Chapter 19. Key token formats 877
## Key token formats

### Table 251. ECC private-key section (X'20') (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>084</td>
<td>002</td>
<td>Length of $p$ in bits (see offset 012).</td>
</tr>
<tr>
<td>086</td>
<td>001</td>
<td>Key-usage flag (see offset 008).</td>
</tr>
<tr>
<td>087</td>
<td>001</td>
<td>Key format and security flag (see offset 010).</td>
</tr>
<tr>
<td>088</td>
<td>001</td>
<td>AD section version number X'00' (see offset 001): Reserved, binary zero</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AD section version number X'01' (see offset 001): Pedigree/Key source flag byte (see offset 011)</td>
</tr>
<tr>
<td>089</td>
<td>003</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>092</td>
<td>$kl$</td>
<td>Optional key label.</td>
</tr>
<tr>
<td>$092 + kl$</td>
<td>$i ead$</td>
<td>Optional IBM extended associated data. For AD section version number X'01' (see offset 076):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Consists of a single section hash tag-length-value (TLV) object with TLV tag identifier X'60'. Refer to Table 252.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: A section hash TLV object cannot be present in section version number X'00', and will always be present in section version number of X'01'. When present, it contains the SHA-256 hash digest of all the optional sections that follow the public key section, if any. Otherwise, it contains binary zeros.</td>
</tr>
<tr>
<td>End of IBM associated data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$092 + kl + iead$</td>
<td>$uad$</td>
<td>Optional user-definable associated data.</td>
</tr>
<tr>
<td>End of associated data section</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$092 + kl + iead + uad$</td>
<td>$bb$</td>
<td>Formatted section (payload), which includes private key $d$:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Clear-key section contains $d$.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Encrypted-key section contains $d$ within the AESKW-wrapped payload.</td>
</tr>
</tbody>
</table>

### Table 252. ECC section hash TLV object (X'60') of Version 1 ECC private-key section (X'20')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Tag identifier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'60' ECC section hash TLV object</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>TLV object version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>TLV object length in bytes (X'0024').</td>
</tr>
<tr>
<td>004</td>
<td>032</td>
<td>SHA-256 hash of all the optional sections that follow the public-key section, if any. Otherwise binary zeros.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: A section hash TLV object will always be present in a PKA key token that has an ECC private key section (X'20') with a section version number of X'01'.</td>
</tr>
</tbody>
</table>

### Table 253. ECC public key section (X'21')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Section identifier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'21' ECC public key (ECC-PUBL)</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length in bytes.</td>
</tr>
<tr>
<td>004</td>
<td>004</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>
### Table 253. ECC public key section (X'21') (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>008</td>
<td>001</td>
<td>Curve type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00' Prime</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01' Brainpool</td>
</tr>
<tr>
<td>009</td>
<td>001</td>
<td>Reserved, binary zero</td>
</tr>
<tr>
<td>010</td>
<td>002</td>
<td>Length of prime ( p ) in bits. See Table 249 on page 874 and Table 250 on page 875</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00A0' 160 (Brainpool)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00C0' 192 (Brainpool, Prime)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00E0' 224 (Brainpool, Prime)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0100' 256 (Brainpool, Prime)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0140' 320 (Brainpool)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0180' 384 (Brainpool, Prime)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0200' 512 (Brainpool)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0209' 521 (Prime)</td>
</tr>
<tr>
<td>012</td>
<td>002</td>
<td>Length of public key ( q ) in bytes. Value includes key material length plus a one-byte flag to indicate if the key material is compressed.</td>
</tr>
<tr>
<td>014</td>
<td>cc</td>
<td>Public key ( q ).</td>
</tr>
</tbody>
</table>

### Table 254. ECC key-derivation information section (X'23')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Section identifier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'23' ECC key-derivation information</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X'00')</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length in bytes (8)</td>
</tr>
<tr>
<td>004</td>
<td>001</td>
<td>Algorithm of key to be derived:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value Algorithm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01' DES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02' AES</td>
</tr>
<tr>
<td>005</td>
<td>001</td>
<td>Type of key to be derived:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value Meaning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01' DATA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02' EXPORTER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03' IMPORTER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'04' CIPHER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'05' DECIPHER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'06' ENCIPHER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'07' CIPHERXI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'08' CIPHERXL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'09' CIPHERXO</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>Key-bit length:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value Meaning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0040' (64) DES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0000' (128) AES, DES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00C0' (192) AES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0100' (256) AES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: Bit length of DES keys includes parity bits.</td>
</tr>
</tbody>
</table>
PKA key tokens

PKA key tokens contain RSA or ECC private or public keys.

PKA tokens are variable length because they contain either RSA or ECC key values, which are variable in length. Consequently, length parameters precede all PKA token parameters. The maximum allowed size is 3500 bytes. PKA key tokens consist of a token header, any required sections, and any optional sections. Optional sections depend on the token type. PKA key tokens can be public or private, and private key tokens can be internal or external. Therefore, there are three basic types of tokens, each of which can contain either RSA or ECC information:

- A public key token
- A private external key token
- A private internal key token

Public key tokens contain only the public key. Private key tokens contain the public and private key pair. Table 255 summarizes the sections in each type of token.

Table 255. Summary of PKA key token sections

<table>
<thead>
<tr>
<th>Section</th>
<th>Public external key token</th>
<th>Private external key token</th>
<th>Private internal key token</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RSA or ECC private key info</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RSA or ECC public key info</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Key name (optional)</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Internal information</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

As with DES key tokens, the first byte of a PKA key token contains the token identifier which indicates the type of token.

A first byte of X'1E' indicates an external token with a cleartext public key and optionally a private key that is either in cleartext or enciphered by a transport key-encrypting key. An external key token is in importable key form. It can be sent on the link.

A first byte of X'1F' indicates an internal token with a cleartext public key and a private key that is enciphered by the PKA master key and ready for internal use. An internal key token is in operational key form. A PKA private key token must be in operational form for the coprocessor to use it. (PKA public key tokens are used directly in the external form.)

Formats for public and private external and internal RSA and ECC key tokens begin in "RSA public key token" on page 853.

PKA null key token

The format for a PKA null key token.

Table 256 on page 881 shows the format for a PKA null key token.
Table 256. PKA null key token format

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X'00' Token identifier (indicates that this is a null key token).</td>
</tr>
<tr>
<td>1</td>
<td>Version, X'00'.</td>
</tr>
<tr>
<td>2 - 3</td>
<td>X'0008' Length of the key token structure.</td>
</tr>
<tr>
<td>4 - 7</td>
<td>Ignored (should be zero).</td>
</tr>
</tbody>
</table>

PKA key token sections

A PKA key token is either for an ECC key or an RSA key. In either case, it is the concatenation of an ordered set of sections.

PKA key tokens can be built with the PKA Key Token Build verb. Either RSA or ECC key tokens can be built.

An RSA key-token is a concatenation of these ordered sections:

- A token header:
  - An external header (first byte X'1E')
  - An internal header (first byte X'1F')

- An optional RSA private-key section in one of the token types shown in Table 257

Table 257. Optional RSA private key sections

<table>
<thead>
<tr>
<th>Private key section identifier (token type)</th>
<th>Modulus bit length and key format</th>
<th>RSA key not protected by an object protection key (OPK)</th>
<th>RSA key protected by OPK that is wrapped by DES KEK or PKA MK</th>
<th>RSA key protected by OPK that is wrapped by an AES key</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'02' (RSA-PRIV)</td>
<td>512 - 1024 M-E (Modulus-Exponent)</td>
<td>Private key in the clear or wrapped by DES KEK</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>X'05' (input only, replaced by X'08')</td>
<td>512 - 2048 CRT (Chinese-Remainder Theorem)</td>
<td>Private key in the clear</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>X'08' (RSA-CRT)</td>
<td>512 - 4096 CRT</td>
<td>N/A</td>
<td>N/A</td>
<td>Private key in the clear or wrapped by OPK, with OPK data wrapped by DES KEK</td>
</tr>
<tr>
<td>X'09' (RSAMEVAR)</td>
<td>512 - 4096 M-E</td>
<td>Private key in the clear or wrapped by DES KEK</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
## Key token formats

Table 257. Optional RSA private key sections (continued)

<table>
<thead>
<tr>
<th>Private key section identifier (token type)</th>
<th>Modulus bit length and key format</th>
<th>RSA key not protected by an object protection key (OPK)</th>
<th>RSA key protected by OPK that is wrapped by DES KEK or PKA MK</th>
<th>RSA key protected by OPK that is wrapped by an AES key</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'30' (RSA-AESM)</td>
<td>512 - 4096 M-E</td>
<td>N/A</td>
<td>N/A</td>
<td>Private key in the clear or wrapped by OPK, with OPK data wrapped by AES KEK</td>
</tr>
<tr>
<td>X'31' (RSA-AESC)</td>
<td>512 - 4096 CRT</td>
<td>N/A</td>
<td>N/A</td>
<td>Private key in the clear or wrapped by OPK, with OPK data wrapped by AES KEK</td>
</tr>
</tbody>
</table>

Internal RSA key tokens (first byte of PKA key-token header X'1F')

| X'05' (input only, replaced by X'08')      | 512 - 2048 CRT                   | Private key wrapped by PKA MK                         | N/A                                                           | N/A |
| X'06' (old RSA-PRIV)                       | 512 - 1024 M-E                   | N/A                                                   | Private key wrapped by OPK, with OPK data wrapped by PKA MK   | N/A |
| X'08' (RSA-CRT)                            | 512 - 4096 CRT                   | N/A                                                   | Private key wrapped by OPK, with OPK data wrapped by PKA MK   | N/A |
| X'30' (RSA-AESM)                           | 512 - 4096 M-E                   | N/A                                                   | N/A                                                           | Private key in the clear or wrapped by OPK, with OPK data wrapped by APKA MK |
| X'31' (RSA-AESC)                           | 512 - 4096 CRT                   | N/A                                                   | N/A                                                           | Private key in the clear or wrapped by OPK, with OPK data wrapped by APKA MK |

- a required RSA public-key section (section identifier X'04')
- an optional PKA private-key name section (section identifier X'10')
- for internal key-tokens with private keys in X'02' or X'05' sections, a required RSA private-key blinding section (section identifier X'FF'), otherwise not allowed
Table 258. RSA private-key blinding information

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Section identifier: X'FF' RSA private-key blinding information. Used with internal key tokens created by the CCA Support Program, Version 1 (having section identifiers X'02' or X'05').</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length in bytes (34+rrr+iii+xxx).</td>
</tr>
<tr>
<td>004</td>
<td>020</td>
<td>SHA-1 hash value of the internal information subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.</td>
</tr>
<tr>
<td>024</td>
<td>002</td>
<td>Length in bytes of the encrypted secure subsection.</td>
</tr>
<tr>
<td>026</td>
<td>002</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

Start of the encrypted secure information subsection. An internal token with section identifiers X'02' or X'05' uses the asymmetric master key and the EDE3 algorithm. See “Triple-DES ciphering algorithms” on page 1030.

- an optional PKA public-key certificate section (section identifier X'40' with subsidiary sections).

An ECC key-token is a concatenation of these ordered sections:
- a required PKA key-token header:
  - an external header (first byte X'1E')
  - an internal header (first byte X'1F')
  - an optional ECC private-key section in the token type shown in Table 259.

Table 259. Optional ECC private key sections

<table>
<thead>
<tr>
<th>Private key section identifier (token type)</th>
<th>ECC curve type: length of prime p in bits</th>
<th>ECC key protected by OPK that is wrapped by an AES key</th>
</tr>
</thead>
<tbody>
<tr>
<td>External ECC key tokens (first byte of PKA key-token header X'1E')</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'20' (ECC-PAIR)</td>
<td>Brainpool: 160, 192, 224, 256, 320, 384, or 512 Prime: 192, 224, 256, 384, or 521</td>
<td>Private key in the clear or wrapped by OPK, with OPK and its data wrapped by AES KEK</td>
</tr>
<tr>
<td>Internal ECC key tokens (first byte of PKA key-token header X'1F')</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 259. Optional ECC private key sections (continued)

<table>
<thead>
<tr>
<th>Private key section identifier (token type)</th>
<th>ECC curve type: length of prime p in bits</th>
<th>ECC key protected by OPK that is wrapped by an AES key</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'20' (ECC-PAIR)</td>
<td>Brainpool: 160, 192, 224, 256, 320, 384, or 512</td>
<td>Private key wrapped by OPK, with OPK and its data wrapped by APKA MK</td>
</tr>
<tr>
<td></td>
<td>Prime: 192, 224, 256, 384, or 521</td>
<td></td>
</tr>
</tbody>
</table>

- a required ECC public-key section (section identifier X'21')
- an optional PKA private-key name section (section identifier X'10')
- an optional PKA public-key certificate section (section identifier X'40')
- an optional ECC key-derivation information section (section identifier X'23').

Integrity of PKA private key sections containing an encrypted RSA key

With the exception of RSA key tokens containing an AES encrypted OPK (sections X'30' and X'31'), if an RSA key-token contains information for an encrypted private-key, then the integrity of the information within the token can be verified by computing and comparing the SHA-1 message digest values that are found at offsets 4 and 30 within the private-key section.

The SHA-1 message digest at offset 4 requires access to the cleartext values of the private-key components. The cryptographic engine verifies this hash quantity whenever it retrieves the secret key for productive use.

A second SHA-1 message digest, located at offset 30 (excluding sections X'30' and X'31'), is computed on optional, designated key-token information following the public-key section. The value of this SHA-1 message digest is included in the computation of the message digest at offset 4. As with the offset 4 value, the message digest at offset 30 is validated whenever a private key is recovered from the token for productive use.

CCA provides PKA private key sections X'30' and X'31'. These sections can contain an AESKW-wrapped (ANS X9.102) RSA private key (wrapped by the OPK) and an AES encrypted OPK. When the RSA key with an AES encrypted OPK is wrapped, a message digest is calculated over the associated data section contained in the private-key section. The calculated message digest becomes part of the payload before it is wrapped. There is no way for a user to retrieve this value to validate it. The same is true for ECC private keys, which are AESKW-wrapped.

In addition to the hash checks, various token-format and content checks are performed to validate the key values.

The optional PKA private-key name section can be used by access-monitor systems to ensure that the application program is entitled to employ the particular private key.

Number representation in PKA key tokens

View tables that present information about number representations in the available formats of PKA key tokens.
Key token formats

- All length fields are in binary.
- All binary fields (exponents, lengths, and so forth) are stored with the high-order byte first. Thus the least significant bits are to the right and preceded with zero-bits to the width of a field.
- In variable-length binary fields that have an associated field-length value, leading bytes that would otherwise contain X'00' can be dropped and the field shortened to contain only the significant bits.

Table 260. PKA key token header

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Token identifier (a flag that indicates token type)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00' PKA null key-token</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'1E' PKA external key-token; the optional private-key is either in cleartext or enciphered by a transport key-encrypting key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'1F' PKA internal key-token; the private key is enciphered by a master key</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Token version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length in bytes of the token structure (big endian).</td>
</tr>
<tr>
<td>004</td>
<td>004</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

PKA public-key certificate section

A PKA public-key certificate section can be optionally included in a PKA key token. The section is composed of a series of subsections and optional tag-length-value (TLV) objects to form a self-defining data structure. One or more TLV objects can be included in the variable portion of a higher-level TLV object. The section and subsections must occur in the following order:

- A required PKA public-key certificate (section identifier X'40')
- A required public-key subsection:
  - For an ECC key, an ECC public-key subsection (subsection identifier X'22')
  - For an RSA key, an RSA public-key subsection (subsection identifier X'41')
- An optional PKA certificate information subsection (subsection identifier X'42') which includes one, two, or three TLV objects:
  1. PKA user-data TLV object (tag identifier X'50')
  2. PKA private key ID object (tag identifier X'51')
  3. PKA serial number TLV object (tag identifier X'52')
- A required PKA signature subsection (subsection identifier X'45'), followed by any number of optional PKA signature subsections

The PKA public-key certification section is described followed by descriptions of the related subsections and TLV objects that can be concatenated to the section.

Table 261. PKA public-key certificate section (X'40')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Section identifier: X'40' PKA public-key certificate</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X'00').</td>
</tr>
</tbody>
</table>
| 002            | 002           | Section length in bytes. Includes:
  - Section header
  - Public key subsection
  - Information subsection (optional)
  - Signature subsections |
### Table 262. ECC public-key subsection (X'22') of PKA public-key certificate section (X'40')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Subsection identifier: X'22' ecc public-key</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Subsection version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Subsection length in bytes (14 + xxx)</td>
</tr>
<tr>
<td>004</td>
<td>004</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>008</td>
<td>001</td>
<td>Curve type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td>009</td>
<td>001</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>010</td>
<td>002</td>
<td>Length of prime $p$ in bits. Refer to <a href="#">Table 250 on page 875</a> and <a href="#">Table 249 on page 874</a>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00A0'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00C0'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00E0'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0100'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0140'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0180'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0200'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0209'</td>
</tr>
<tr>
<td>012</td>
<td>002</td>
<td>Length of public key $q$ in bytes. Value includes length key material plus one (to include a one-byte flag that indicates if the key material is compressed).</td>
</tr>
<tr>
<td>014</td>
<td>xxx</td>
<td>Public key $q$.</td>
</tr>
</tbody>
</table>

### Table 263. RSA public-key subsection (X'41') of PKA public-key certificate section (X'40')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Subsection identifier: X'41' RSA public-key</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Subsection version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Subsection length in bytes (12 + xxx + yyy).</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>RSA public-key exponent field length in bytes, xxx.</td>
</tr>
<tr>
<td>008</td>
<td>002</td>
<td>Public-key modulus length in bits.</td>
</tr>
<tr>
<td>010</td>
<td>002</td>
<td>RSA public-key modulus field length in bytes, yyy.</td>
</tr>
<tr>
<td>012</td>
<td>xxx</td>
<td>Public-key exponent, $e$ (this field length is typically 1, 3, or 64 - 512 bytes). $e$ must be odd, and $1 \leq e &lt; n$.</td>
</tr>
<tr>
<td>012+xxx</td>
<td>yyy</td>
<td>Modulus, $n, n = pq$, where $p$ and $q$ are prime and $2^{512} \leq n &lt; 2^{4096}$. This field is absent when the modulus is contained in the private-key section. If present, the field length is 64 - 512 bytes.</td>
</tr>
</tbody>
</table>

### Table 264. PKA certificate-information subsection (X'42') of PKA public-key certificate section (X'40')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Subsection identifier: X'42' PKA certificate information</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Subsection version number (X'00').</td>
</tr>
</tbody>
</table>
Table 264. PKA certificate-information subsection (X'42') of PKA public-key certificate section (X'40') (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>002</td>
<td>002</td>
<td>Subsection length in bytes (4+(iii)) where (iii) is:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>length of TLV object X'50' +</td>
</tr>
<tr>
<td></td>
<td></td>
<td>length of TLV object X'51' +</td>
</tr>
<tr>
<td></td>
<td></td>
<td>length of TLV object X'52'.</td>
</tr>
<tr>
<td>004</td>
<td>(iii)</td>
<td>The information field that contains any of the optional TLV objects:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'50'</td>
<td>User data</td>
</tr>
<tr>
<td>X'51'</td>
<td>Private key EID</td>
</tr>
<tr>
<td>X'52'</td>
<td>Serial number</td>
</tr>
</tbody>
</table>

Table 265. PKA user-data TLV object (X'50') of PKA certificate-information subsection (X'42')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Tag identifier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'50' PKA user-data TLV object</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>TLV object version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>TLV object length in bytes (4+(uuu); 0≤(uuu)≤64).</td>
</tr>
<tr>
<td>004</td>
<td>(uuu)</td>
<td>User-provided data.</td>
</tr>
</tbody>
</table>

Table 266. PKA private-key EID TLV object (X'51') of PKA certificate-information subsection (X'42')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Tag identifier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'51' PKA private-key EID TLV object</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>TLV object version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>TLV object length in bytes (X'0014').</td>
</tr>
<tr>
<td>004</td>
<td>016</td>
<td>EID string of the CCA node that generated the public and private key. This TLV must be provided in a skeleton key-token with usage of the PKA Key Generate verb. The verb fills in the EID string prior to certifying the public key. The EID value is encoded using the ASCII character set.</td>
</tr>
</tbody>
</table>

Table 267. PKA serial number TLV object (X'52') of PKA certificate-information subsection (X'42')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Tag identifier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'52' PKA serial number TLV object</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>TLV object version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>TLV object length in bytes (X'000C').</td>
</tr>
<tr>
<td>004</td>
<td>008</td>
<td>Serial number of the coprocessor that generated the public and private key. This TLV must be provided in a skeleton key-token with usage of the PKA Key Generate verb. The verb fills in the serial number prior to certifying the public key.</td>
</tr>
</tbody>
</table>

Table 268. PKA signature subsection (X'45') of PKA public-key certificate section (X'40')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Subsection identifier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'45' PKA signature</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Subsection version number (X'00').</td>
</tr>
</tbody>
</table>
### Key token formats

**Table 268. PKA signature subsection (X'45') of PKA public-key certificate section (X'40') (continued)**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>002</td>
<td>002</td>
<td>Subsection length in bytes (70+sss).</td>
</tr>
<tr>
<td>004</td>
<td>001</td>
<td>Hash algorithm identifier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For RSA public-key (X'04') with PKA public-key certificate (X'40'):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For ECC public-key (X'21') with PKA public-key certificate (X'40'):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'04'</td>
</tr>
<tr>
<td>005</td>
<td>001</td>
<td>Signature formatting identifier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For RSA public-key (X'04') with PKA public-key certificate (X'40'):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For ECC public-key (X'21') with PKA public-key certificate (X'40'):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'05'</td>
</tr>
<tr>
<td>006</td>
<td>064</td>
<td>Signature-key identifier; the key label of the key used to generate the signature.</td>
</tr>
<tr>
<td>070</td>
<td>sss</td>
<td>The signature field:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The signature is calculated on data that begins with the signature section identifier (X'40') through the byte immediately preceding this signature field.</td>
</tr>
</tbody>
</table>

**Note:** More than one signature subsection can be included in a signature section. This accommodates the possibility of a self-signature as well as a device-key signature.

### HMAC key token

The two formats of the HMAC key token.

HMAC key tokens have two formats, "HMAC MAC variable-length symmetric key token" on page 920 and "HMAC symmetric null key token."

#### HMAC symmetric null key token

**Table 269. HMAC symmetric null key token format**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X'00' Token identifier, which indicates that this is a null key token.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X'00' Version</td>
</tr>
<tr>
<td>2 - 3</td>
<td>2</td>
<td>X'0008' Length of the key token structure.</td>
</tr>
<tr>
<td>4 - 7</td>
<td>4</td>
<td>Ignored (zero).</td>
</tr>
</tbody>
</table>
Variable-length symmetric key tokens

CCA supports a variable-length symmetric key-token. This key token has a version number of X'05' (offset 4). Use the Key Token Build2 (CSNBKTB2) verb to build skeleton variable-length symmetric key tokens used as input by the Key Generate2 (CSNBKGN2) or Key Part Import2 (CSNBKPI2) verbs, which return these key tokens with encrypted keys in the key-token payload.

Table 270 shows the general format of the token version number X'05' key token.

Table 271 on page 904 shows the format of the CIPHER variable-length symmetric key-token that can be used with the AES algorithm. An AES CIPHER key-token is used by the Symmetric_Algorithm_Decipher (CSNBSAD) and Symmetric_Algorithm_Encipher (CSNBSAE) verbs to decipher or encipher data with the AES algorithm.

Table 272 on page 912 shows the format of the MAC variable-length symmetric key-token that can be used with the AES algorithm.

Table 273 on page 920 shows the format of the MAC variable-length symmetric key-token that can be used with the HMAC algorithm. An HMAC MAC key-token is used by the HMAC_Generate (CSNBHMG) and HMAC_Verify (CSNBHMV) verbs to generate or verify keyed hash message authentication codes.

Table 274 on page 928 shows the format of the EXPORTER and IMPORTER variable-length symmetric key tokens that can be used with the AES algorithm. An EXPORTER operational key-token is used by the Symmetric_Key_Export (CSNDSYX) verb to export an internal AES or HMAC variable-length symmetric key-token into an external variable-length symmetric key-token, either into an AESKW or PKOAEP2 wrapped payload. An IMPORTER operational key-token is used by the Symmetric_Key_Import2 (CSNDSYI2) verb to import an external AES or HMAC variable-length symmetric key-token, containing either an AESKW or PKOAEP2 wrapped payload, into an internal variable-length symmetric key-token.

Table 275 on page 938 shows the format of the PINPROT, PINCALC, and PINPR variable-length symmetric key tokens that can be used with the AES algorithm.

Table 276 on page 947 shows the format of the DESUSECV variable-length symmetric key tokens that can be used with the AES algorithm.

“AES DKYGENKY variable-length symmetric key token” on page 951 shows the format of the DKYGENKY variable-length symmetric key tokens that can be used with the AES algorithm.

Table 278 on page 960 shows the format of the SECMSG variable-length symmetric key tokens that can be used with the AES algorithm.

General format of a variable-length symmetric key-token

View a table showing the general format of a variable-length symmetric key-token.

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Chapter 19. Key token formats  889
### Key token formats

**Table 270. General format of a variable-length symmetric key-token, version X’05’ (continued)**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>01</td>
<td>Token identifier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’00’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’01’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’02’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>001</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>002</td>
<td>02</td>
<td>Length in bytes of the overall token structure. For a null key-token, the value is 8. Otherwise, the length is calculated as:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[46 + (2 \times kuf) + (2 \times kmf) + kl + iead + uad + tlv + \left(\frac{pl + 7}{8}\right)]</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>kuf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kmf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kl</td>
</tr>
<tr>
<td></td>
<td></td>
<td>iead</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uad</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tlv</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pl</td>
</tr>
<tr>
<td>004</td>
<td>01</td>
<td>Token version number (identifies the format of this key token):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’05’</td>
</tr>
<tr>
<td>005</td>
<td>03</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

**End of header**

**Wrapping information section (all data related to wrapping the key)**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>008</td>
<td>01</td>
<td>Key material state:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’00’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’01’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’02’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’03’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
</tbody>
</table>
Table 270. General format of a variable-length symmetric key-token, version X'05' (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>009</td>
<td>01</td>
<td>Key verification pattern (KVP) type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
</tbody>
</table>

All unused values are reserved and undefined.

| 010            | 16             | KVP of the key used to wrap the payload (value depends on value of key material state, that is, the value at offset 8): |
|                |                | **Value at offset 8**  |
|                |                | **Value of KVP** |
|                |                | X'00'      | The key-material state is no key present. The field should be filled with binary zeros. The key token is external or internal. |
|                |                | X'01'      | The key-material state is key is clear. The field should be filled with binary zeros. The key token is external or internal. |
|                |                | X'02'      | The key material state is the key is wrapped with a transport key. The value of the KVP depends on the value of the encrypted section key-wrapping method: |
|                |                |            | • When the key-wrapping method is AESKW (value at offset 26 is X'02'), the field contains the KVP of the key-encrypting key used to wrap the key. The 8-byte KEK KVP is left-aligned in the field and padded on the right low-order bytes with binary zeros. |
|                |                |            | • When the key-wrapping method is PKOAEP2 (value at offset 26 is X'03'), the value should be filled with binary zeros. The encoded message, which contains the key, is wrapped with an RSA public-key. |
|                |                |            | The key token is external. |
|                |                | X'03'      | The key-material state is the key is wrapped with the AES master-key. The field contains the MKVP of the AES master-key used to wrap the key. The 8-byte MKVP is left-aligned in the field and padded on the right low-order bytes with binary zeros. The key token is internal. |
Key token formats

Table 270. General format of a variable-length symmetric key-token, version X’05’ (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>026</td>
<td>01</td>
<td>Encrypted section key-wrapping method (indicates the key-wrapping method used to protect the data in the encrypted section):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td>X’00’</td>
<td></td>
<td>No key-wrapping method (no key present or key is clear). The key token is external or internal.</td>
</tr>
<tr>
<td>X’02’</td>
<td></td>
<td>AESKW (ANS X9.102). The key token is external with a key wrapped by an AES key-encrypting key, or the key token is internal with a key wrapped by the AES master-key.</td>
</tr>
<tr>
<td>X’03’</td>
<td></td>
<td>PKOAEP2. Message $M$, which contains the key, is encoded using the RSAES-OAEP scheme of the RSA PKCS #1 v2.1 standard. The encoded message ($EM$) is produced using the given hash algorithm by encoding message $M$ using the Bellare and Rogaway Optimal Asymmetric Encryption Padding (OAEP) method for encoding messages. For PKOAEP2, $M$ is defined as follows: $M = [32$ bytes: $hAD] \</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
</tbody>
</table>
Table 270. General format of a variable-length symmetric key-token, version X'05' (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>027</td>
<td>01</td>
<td>Hash algorithm used for wrapping key or encoding message. Meaning depends on whether the encrypted section key-wrapping method (value at offset 26) is no key-wrapping method, AESKW, or PKOAEP2:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No key-wrapping method (value at offset 26 is X'00')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hash algorithm used for wrapping key when encrypted section key-wrapping method is no key-wrapping method:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong>  <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td>X'00'</td>
<td>No hash (no key present or key is clear).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AESKW key-wrapping method (value at offset 26 is X'02')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hash algorithm used for wrapping key when encrypted section key-wrapping method is AESKW. The value indicates the algorithm used to calculate the message digest of the associated data. The message digest is included in the wrapped payload and is calculated starting at offset 30 for the length in bytes of all the associated data for the key token (length value at offset 32).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong>  <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td>X'01'</td>
<td>SHA-224 (not used)</td>
</tr>
<tr>
<td></td>
<td>X'02'</td>
<td>SHA-256</td>
</tr>
<tr>
<td></td>
<td>X'04'</td>
<td>SHA-384 (defined for future use)</td>
</tr>
<tr>
<td></td>
<td>X'08'</td>
<td>SHA-512 (defined for future use)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PKOAEP2 key-wrapping method (value at offset 26 is X'03')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hash algorithm used for encoding message when encrypted section key-wrapping method is PKOAEP2. The value indicates the given hash algorithm used for encoding message M using the RSAES-OAEP scheme of the RSA PKCS #1 v2.1 standard.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong>  <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td>X'01'</td>
<td>SHA-1</td>
</tr>
<tr>
<td></td>
<td>X'02'</td>
<td>SHA-256</td>
</tr>
<tr>
<td></td>
<td>X'04'</td>
<td>SHA-384</td>
</tr>
<tr>
<td></td>
<td>X'08'</td>
<td>SHA-512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined. The key token is external.</td>
</tr>
</tbody>
</table>
## Key token formats

Table 270. General format of a variable-length symmetric key-token, version X'05' (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>028</td>
<td>01</td>
<td>Payload format version (identifies format of the payload):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value at offset 26</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value at offset 26</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td>029</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

End of wrapping information section

Clear key, AESKW, or PKOAEP2 components: (1) associated data sections and (2) optional clear key payload, wrapped AESKW formatted payload, or wrapped PKOAEP2 encoded payload (no payload if no key present)

Associated data sections: (1) required associated data section and (2) optional associated data sections

Required associated data section
### Table 270. General format of a variable-length symmetric key-token, version X'05' (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>030</td>
<td>01</td>
<td>Associated data section version:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong>  <strong>Meaning</strong></td>
</tr>
<tr>
<td>031</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>032</td>
<td>02</td>
<td>Length in bytes of all the associated data for the key token: ≥ 16.</td>
</tr>
<tr>
<td>034</td>
<td>01</td>
<td>Length in bytes of the optional key label (kl): 0 or 64.</td>
</tr>
<tr>
<td>035</td>
<td>01</td>
<td>Length in bytes of the optional IBM extended associated data (iead): 0 - 255.</td>
</tr>
<tr>
<td>036</td>
<td>01</td>
<td>Length in bytes of the optional user-definable associated data (uad): 0 - 255.</td>
</tr>
<tr>
<td>037</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>038</td>
<td>02</td>
<td>Length in bits of the clear or wrapped payload (pl): ≥ 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For no key-wrapping method (no key present or key is clear), pl is the length in bits of the key. For no key present, pl is 0. For key is clear (AES CIPHER and HMAC MAC only), pl can be 128, 192, or 256 for an AES key, or 80 - 2048 for an HMAC key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For PKOAEP2 encoded payloads, pl is the length in bits of the modulus size of the RSA key used to wrap the payload. This can be 512 - 4096.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For an AESKW formatted payload, pl is based on the key size of the algorithm type (DES, AES, or HMAC) and the payload format version:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DES algorithm (value at offset 41 is X'01')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A DES key can have a length of 8, 16, or 24 bytes (64, 128, 192 bits). A DES key in an AESKW formatted payload is always wrapped with a V1 payload and has a fixed length payload of 576 bits.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AES algorithm (value at offset 41 is X'02'). An AES key can have a length of 16, 24, or 32 bytes (128, 192, or 256 bits). A V0 payload is only valid for HMAC MAC and AES CIPHER, EXPORTER, and EXPORTER keys. A V1 payload is not valid for an HMAC MAC key. The following table shows the payload length for a given AES key size and payload format:</td>
</tr>
</tbody>
</table>
|                |                | \[
<table>
<thead>
<tr>
<th>AES key size</th>
<th>Bit length of V0 payload (value at offset 28 is X'00')</th>
<th>Bit length of V1 payload (value at offset 28 is X'01')</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bytes (128 bits)</td>
<td>512</td>
<td>640</td>
</tr>
<tr>
<td>24 bytes (192 bits)</td>
<td>576</td>
<td>640</td>
</tr>
<tr>
<td>32 bytes (256 bits)</td>
<td>640</td>
<td>640</td>
</tr>
<tr>
<td></td>
<td>HMAC algorithm (value at offset 41 is X'03'). An HMAC key can have a length of 80 - 2048 bits. An HMAC key in an AESKW formatted payload is always wrapped with a V0 payload.</td>
<td></td>
</tr>
<tr>
<td>040</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>
### Table 270. General format of a variable-length symmetric key-token, version X'05' (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>041</td>
<td>01</td>
<td>Algorithm type (algorithm for which the key can be used):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>042</td>
<td>02</td>
<td>Key type (general class of the key):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0001'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0002'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0003'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0004'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0005'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0006'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0007'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0008'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0009'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00A'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>044</td>
<td>01</td>
<td>Key usage fields count (kuf): 0 - 255. Key-usage field information defines restrictions on the use of the key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Each key type can have a variable number of key usage fields from none to a maximum of 255. Each key-usage field is 2 bytes in length. The value in this field indicates how many 2-byte key usage fields follow.</td>
</tr>
<tr>
<td>045, for kuf &gt; 0</td>
<td>01</td>
<td>Optional key-usage field 1, high-order byte.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defined based on algorithm type (value at offset 41) and key type (value at offset 42).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
</tbody>
</table>
### Table 270. General format of a variable-length symmetric key-token, version X'05' (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 046, for kuf > 0 | 01             | Optional key-usage field 1, low-order byte (user-defined extension control):  
  **Value**  
  B'xxxx 1xxx'  
  Key can only be used in UDXs (UDX-ONLY).  
  B'xxxx 0xxx'  
  Key can be used in UDXs and CCA.  
  B'xxxx x1uu'  
  UDX-defined bit reserved for UDXs (UDX-100).  
  B'xxxx xu1u'  
  UDX-defined bit reserved for UDXs (UDX-010).  
  B'xxxx xu1u'  
  UDX-defined bit reserved for UDXs (UDX-001).  
  **Note:** This byte is common for all key types except for DES DESUSECV in which case this byte is reserved and must zero.  
  All unused bits are reserved and must be zero. |
| 047, for kuf > 1 | 01             | Optional key-usage field 2, high-order byte. |
| 048, for kuf > 1 | 01             | Optional key-usage field 2, low-order byte. |
|                 |                |            |
| 043 + (2 * kuf), for kuf > 0 | 01         | Optional key-usage field kuf, high-order byte. |
| 044 + (2 * kuf), for kuf > 0 | 01         | Optional key-usage field kuf, low-order byte. |
| 045 + (2 * kuf) | 01             | Key management fields count (kmf): 0 - 255. Key-management field information describes how the data is to be managed or helps with management of the key material.  
  Each key type can have a variable number of key management fields from none to a maximum of 255. Each key-management field is 2 bytes in length. The value in this field indicates how many 2-byte key management fields follow. |
### Key token formats

Table 270. General format of a variable-length symmetric key-token, version X’05’ (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>046 + (2 * kmf), for ( kmf &gt; 0 )</td>
<td>01</td>
<td>Optional key-management field 1, high-order byte (export control):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Symmetric-key export control:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> Meaning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’1xxx xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’0xxx xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Unauthenticated asymmetric-key export control:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> Meaning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’x1xx xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’x0xx xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Authenticated asymmetric-key export control:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> Meaning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xx1x xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xx0x xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RAW-key export control:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> Meaning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xxx1 xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xxx0 xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> This byte is common for all key types. Except for DES DESUSECV, this byte is reserved and must zero.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
</tbody>
</table>
Table 270. General format of a variable-length symmetric key-token, version X'05' (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>047 + (2 * (kuf)), for (kmf &gt; 0)</td>
<td>01</td>
<td>Optional key-management field 1, low-order byte (export control by algorithm):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DES-key export control:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'1xxx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0xxx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AES-key export control:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'x1xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'x0xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA-key export control:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx 1xxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx 0xxx'</td>
</tr>
<tr>
<td>Note: This byte is common for all key types except for DES DESUSECV in which case this byte is undefined.</td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>048 + (2 * (kuf)), for (kmf &gt; 1)</th>
<th>01</th>
<th>Optional key-management field 2, high-order byte (key completeness):</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'11xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'10xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'01xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'00xx xxxx'</td>
</tr>
<tr>
<td>Note: This byte is common for all key types except for DES DESUSECV in which case this byte is undefined.</td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
</tbody>
</table>
### Key token formats

#### Table 270. General format of a variable-length symmetric key-token, version X’05’ (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>049 + (2 * (kuf)) for (kmf &gt; 1)</td>
<td>01</td>
<td>Optional key-management field 2, low-order byte (security history). Used to reflect the overall history of the key, not just the history at the most recent import or other operation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xxx1 xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xxxx 1xxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xxxx x1xx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xxxx xx1x’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xxxx xxx1’</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> This byte is common for all key types except for DES DESUSECV in which case this byte is undefined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
<tr>
<td>050 + (2 * (kuf)) for (kmf &gt; 2)</td>
<td>01</td>
<td>Optional key-management field 3, high-order byte (pedigree original). Used to indicate how the key was originally created and how it got into the current system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’00’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’01’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’02’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’03’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’04’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’05’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’06’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’07’</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> This byte is common for all key types except for DES DESUSECV in which case this byte is undefined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and must be zero.</td>
</tr>
<tr>
<td>Offset (bytes)</td>
<td>Length (bytes)</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>051 + (2 * (kuf)) for (kmf &gt; 2)</td>
<td>01</td>
<td>Optional key-management field 3, low-order byte (pedigree current). Used to indicate how the key was originally created and how it got into the current system.</td>
</tr>
<tr>
<td>044 + (2 * (kuf)) + (2 * (kmf)), for (kmf &gt; 0)</td>
<td>01</td>
<td>Optional key-usage field (kmf), high-order byte.</td>
</tr>
<tr>
<td>045 + (2 * (kuf)) + (2 * (kmf)), for (kmf &gt; 0)</td>
<td>01</td>
<td>Optional key-usage field (kmf), low-order byte.</td>
</tr>
<tr>
<td>046 + (2 * (kuf)) + (2 * (kmf))</td>
<td>(kl)</td>
<td>Optional key label.</td>
</tr>
<tr>
<td>046 + (2 * (kuf)) + (2 * (kmf)) + (kl)</td>
<td>(ied)</td>
<td>Optional IBM extended associated data.</td>
</tr>
</tbody>
</table>
## Key token formats

**Table 270. General format of a variable-length symmetric key-token, version X'05' (continued)**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>046 + (2 * kuf) + (2 * kmf) + kl + iead + uad</td>
<td>uad</td>
<td>Optional user-defined associated data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>End of required associated data section</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional associated data sections (defined for future use)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional tag-length-value (TLV) fields.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The length of tlvn is in bytes and is calculated as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tlvn = size of TLVn tag + size of tlvn length field + size of the TLV n value in bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for n &gt; 0, where n = number of TLV fields.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The summation of TLV lengths is in bytes and is calculated as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tlvs = \sum_{i=1}^{n} tlv_i</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for n &gt; 0, where n = number of TLV fields. For n = 0, tlvs = 0.</td>
</tr>
<tr>
<td>046 + (2 * kuf) + (2 * kmf) + kl + iead + uad, for tlv1 &gt; 0</td>
<td>01</td>
<td>Optional tag-length-value 1 (TLV1) tag.</td>
</tr>
<tr>
<td>047 + (2 * kuf) + (2 * kmf) + kl + iead + uad, for tlv1 &gt; 0</td>
<td>02</td>
<td>Optional TLV1 length: tlo1.</td>
</tr>
<tr>
<td>049 + (2 * kuf) + (2 * kmf) + kl + iead + uad, for tlv1 &gt; 0</td>
<td>tlo1 - 3</td>
<td>Optional TLV1 value.</td>
</tr>
<tr>
<td>046 + (2 * kuf) + (2 * kmf) + kl + iead + uad + tlo1, for n &gt; 1</td>
<td>01</td>
<td>Optional tag-length-value 2 (TLV2) tag.</td>
</tr>
<tr>
<td>047 + (2 * kuf) + (2 * kmf) + kl + iead + uad + tlo1, for n &gt; 1</td>
<td>02</td>
<td>Optional TLV2 length: tlo2.</td>
</tr>
<tr>
<td>049 + (2 * kuf) + (2 * kmf) + kl + iead + uad + tlo1, for n &gt; 1</td>
<td>tlo2 - 3</td>
<td>Optional TLV2 value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>046 + (2 * kuf) + (2 * kmf) + kl + iead + uad + tlvs - tlv n, for n &gt; 0</td>
<td>01</td>
<td>Optional TLVn tag.</td>
</tr>
<tr>
<td>047 + (2 * kuf) + (2 * kmf) + kl + iead + uad + tlvs - tlv n, for n &gt; 0</td>
<td>02</td>
<td>Optional TLVn length: tlo n</td>
</tr>
<tr>
<td>049 + (2 * kuf) + (2 * kmf) + kl + iead + uad + tlvs - tlv n, for n &gt; 0</td>
<td>tlo n - 3</td>
<td>Optional TLVn value.</td>
</tr>
</tbody>
</table>
## Key token formats

### Table 270. General format of a variable-length symmetric key-token, version X'05’ (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>End of TLV fields</td>
<td></td>
<td></td>
</tr>
<tr>
<td>End of optional associated data sections</td>
<td></td>
<td></td>
</tr>
<tr>
<td>End of associated data sections</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional clear key payload, wrapped AESKW formatted payload, or wrapped PKOAEP2 encoded payload (no payload if no key present)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>046 + (2 * kmf) + (2 * kuf) + kl + uad + uad</td>
<td>(pl + 7) / 8</td>
<td>Contents of payload (pl is in bits) depending on the encrypted section key-wrapping method (value at offset 26):</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value at offset 26</td>
<td>Encrypted section key-wrapping method</td>
</tr>
<tr>
<td></td>
<td>X'00'</td>
<td>No key-wrapping method. Only applies when key is clear, that is, when key material state (value at offset 8) is X'01'.</td>
</tr>
<tr>
<td></td>
<td>X'02'</td>
<td>AESKW</td>
</tr>
</tbody>
</table>
| | X'03' | PKOAEP2 | An encrypted PKOAEP2 payload which the Segment 2 code creates using the RSAES-OAEP scheme of the PKCS #1 v2.1 standard. The message $M$ is encoded for a given hash algorithm using the Bellare and Rogaway Optimal Asymmetric Encryption Padding (OAEP) method for encoding messages. For PKOAEP2, $M$ is defined as follows:  

\[
M = [32 \text{ bytes: } hAD] \ || [2 \text{ bytes: bit length of the clear key}] \ || [\text{clear key}]
\]

where $hAD$ is the message digest of the associated data, and is calculated using the SHA-256 algorithm starting at offset 30 for the length in bytes of all the associated data for the key token (length value at offset 32). The encoded message is wrapped with an RSA public-key according to the standard. The key token is external. |

End of optional clear key payload, wrapped AESKW formatted payload, or wrapped PKOAEP2 encoded payload

End of clear key, AESKW, or PKOAEP2 components

**Note:** All numbers are in big endian format.
AES CIPHER variable-length symmetric key token

View a table showing the format of the CIPHER variable-length symmetric key-token that, beginning with Release 4.2, can be used with the AES algorithm. An AES CIPHER key-token is used by the Symmetric_Algorithm_Decipher (CSNBSAD) and Symmetric_Algorithm_Encipher (CSNBSAE) verbs to decipher or encipher data with the AES algorithm.

Table 271. AES CIPHER variable-length symmetric key-token, version X'05'

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>01</td>
<td>Token identifier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value   Meaning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'   Internal key-token (encrypted key is wrapped with the master key, the key is clear, or there is no payload).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'   External key-token (encrypted payload is wrapped with a transport key, the payload is clear, or there is no payload). A transport key can be a key-encrypting key or an RSA public-key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>001</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>002</td>
<td>02</td>
<td>Length in bytes of the overall token structure:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * kuf) + (2 * kmf) + kl + iead + uad + ((pl + 7) / 8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key token Minimum token length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Skeleton</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 3) + 0 + 0 + 0 = 56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clear V0 payload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 3) + 0 + 0 + 0 + ((128 + 7) / 8) = 72</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Encrypted V0 payload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 3) + 0 + 0 + 0 + ((512 + 7) / 8) = 120</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Encrypted V1 payload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 3) + 0 + 0 + 0 + ((640 + 7) / 8) = 136</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key token Maximum token length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>External*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 3) + 64 + 0 + 255 + ((4096 + 7) / 8) = 887</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 3) + 64 + 0 + 255 + ((640 + 7) / 8) = 455</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*This assumes a PKOAEP2 key-wrapping method using a 4096-bit RSA transport key.</td>
</tr>
<tr>
<td>004</td>
<td>01</td>
<td>Token version number (identifies the format of this key token):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value   Meaning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'05'   Version 5 format of the key token (variable-length symmetric key-token)</td>
</tr>
<tr>
<td>005</td>
<td>03</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

End of header

Wrapping information section (all data related to wrapping the key)
### Table 271. AES CIPHER variable-length symmetric key-token, version X'05' (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>008</td>
<td>01</td>
<td>Key material state:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>009</td>
<td>01</td>
<td>Key verification pattern (KVP) type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>010</td>
<td>16</td>
<td>KVP (value depends on value of key material state, that is, the value at offset 8):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value at offset 8</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Key token formats

Table 271. AES CIPHER variable-length symmetric key-token, version X'05' (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>026</td>
<td>01</td>
<td>Encrypted section key-wrapping method (how data in the encrypted section is protected):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
</tbody>
</table>

All unused values are reserved and undefined.
Table 271. AES CIPHER variable-length symmetric key-token, version X'05' (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>027</td>
<td>01</td>
<td>Hash algorithm used for wrapping key or encoding message. Meaning depends on whether the encrypted section key-wrapping method (value at offset 26) is no key-wrapping method, AESKW, or PKOAEP2:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- No key-wrapping method (value at offset 26 is X'00')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Hash algorithm used for wrapping key when encrypted section key-wrapping method is no key-wrapping method:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AESKW key-wrapping method (value at offset 26 is X'02')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hash algorithm used for wrapping key when encrypted section key-wrapping method is AESKW. The value indicates the algorithm used to calculate the message digest of the associated data. The message digest is included in the wrapped payload and is calculated starting at offset 30 for the length in bytes of all the associated data for the key token (length value at offset 32).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PKOAEP2 key-wrapping method (value at offset 26 is X'03')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hash algorithm used for encoding message when encrypted section key-wrapping method is PKOAEP2. The value indicates the given hash algorithm used for encoding message M using the RSAES-OAEP scheme of the RSA PKCS #1 v2.1 standard.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'04'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'08'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined. The key token is external.</td>
</tr>
</tbody>
</table>
### Key token formats

Table 271. AES CIPHER variable-length symmetric key-token, version X'05' (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>028</td>
<td>01</td>
<td>Payload format version (identifies format of the payload). Release 4.4 or later, otherwise undefined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'00'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>V0 payload (V0PYLD). The payload format depends on the encrypted section key-wrapping method (value at offset 26):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value at offset 26</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'00'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>There is no key-wrapping method. When no key is present, there is no payload. When the key is clear, the payload is unformatted. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'02'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The key-wrapping method is AESKW and the payload is variable length. The payload is formatted with the minimum size possible to contain the key material. The payload length varies for a given algorithm and key type. The key length can be inferred by the size of the payload. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'03'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The key-wrapping method is PKOAEP2 and the payload length is equal to the modulus size in bits of the RSA transport key used to wrap the encoded message. The key token is external. When the external key is exported, the internal target key will have the same V0 payload format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'01'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>V1 payload (Release 4.4 or later). The payload format depends on the encrypted section key-wrapping method (value at offset 26):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value at offset 26</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'00'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>There is no key-wrapping method. When no key is present, there is no payload. When the key is clear, the payload is unformatted. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'02'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The key-wrapping method is AESKW and the payload is fixed length based on the maximum possible key size of the algorithm for the key. The key is padded with random data to the size of the largest key for that algorithm. This helps to deter attacks on keys known to be weaker. The key length cannot be inferred by the size of the payload. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'03'</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The key-wrapping method is PKOAEP2 and the payload length is equal to the modulus size in bits of the RSA transport key used to wrap the encoded message. The key token is external. When the external key is exported, the internal target key will have the same V1 payload format.</td>
</tr>
</tbody>
</table>

All unused values are reserved and undefined.

| 029 | 01 | Reserved, binary zero. |

End of wrapping information section

Clear key, AESKW, or PKOAEP2 components: (1) associated data section and (2) optional clear key payload, wrapped AESKW formatted payload, or wrapped PKOAEP2 encoded payload (no payload if no key present)

Associated data section

<table>
<thead>
<tr>
<th>030</th>
<th>01</th>
<th>Associated data section version:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>031</th>
<th>01</th>
<th>Reserved, binary zero.</th>
</tr>
</thead>
<tbody>
<tr>
<td>032</td>
<td>02</td>
<td>Length in bytes of all the associated data for the key token: 26 - 345.</td>
</tr>
<tr>
<td>034</td>
<td>01</td>
<td>Length in bytes of the optional key label (kl): 0 or 64.</td>
</tr>
<tr>
<td>035</td>
<td>01</td>
<td>Length in bytes of the optional IBM extended associated data (iead): 0.</td>
</tr>
<tr>
<td>036</td>
<td>01</td>
<td>Length in bytes of the optional user-definable associated data (uad): 0 - 255.</td>
</tr>
</tbody>
</table>
### Table 271. AES CIPHER variable-length symmetric key-token, version X'05' (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>037</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>038</td>
<td>02</td>
<td>Length in bits of the clear or wrapped payload (pl): 0, 128, 192, 256, 512 - 4096.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- For no key-wrapping method (no key present or key is clear), pl is the length in bits of the key. For no key present, pl is 0. For key is clear, pl can be 128, 192, or 256.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- For PKOAEP2 encoded payloads, pl is the length in bits of the modulus size of the RSA key used to wrap the payload. This can be 512 - 4096.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- For an AESKW formatted payload, pl is based on the key size of the algorithm type and the payload format version:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AES algorithm (value at offset 41 is X'02') An AES key can have a length of 16, 24, or 32 bytes (128, 192, or 256 bits). The following table shows the payload length for a given AES key size and payload format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>AES key size</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>16 bytes (128 bits)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24 bytes (192 bits)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32 bytes (256 bits)</td>
</tr>
<tr>
<td>040</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>041</td>
<td>01</td>
<td>Algorithm type (algorithm for which the key can be used):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>042</td>
<td>02</td>
<td>Key type (general class of the key):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0001'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>044</td>
<td>01</td>
<td>Key usage fields count (kuf): 2. Key-usage field information defines restrictions on the use of the key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Each key-usage field is 2 bytes in length. The value in this field indicates how many 2-byte key usage fields follow.</td>
</tr>
</tbody>
</table>
### Table 271. AES CIPHER variable-length symmetric key-token, version X'05' (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>045</td>
<td>01</td>
<td>Key-usage field 1, high-order byte (encryption and translation control):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'1xxx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0xxx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'x1xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'x0xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xx1x xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xx0x xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'00xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'000x xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
<tr>
<td>046</td>
<td>01</td>
<td>Key-usage field 1, low-order byte (user-defined extension control). Refer to Table 270 on page 889.</td>
</tr>
<tr>
<td>047</td>
<td>01</td>
<td>Key-usage field 2, high-order byte (encryption mode):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'04'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'05'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'FF'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>048</td>
<td>01</td>
<td>Key-usage field 2, low-order byte (reserved). All bits are reserved and must be zero.</td>
</tr>
<tr>
<td>049</td>
<td>01</td>
<td>Key management fields count (kmf): 3. Key-management field information describes how the data is to be managed or helps with management of the key material. Each key-management field is 2 bytes in length. The value in this field indicates how many 2-byte key management fields follow.</td>
</tr>
<tr>
<td>050</td>
<td>01</td>
<td>Key-management field 1, high-order byte (symmetric-key export control). Refer to Table 270 on page 889.</td>
</tr>
<tr>
<td>051</td>
<td>01</td>
<td>Key-management field 1, low-order byte (export control by algorithm). Refer to Table 270 on page 889.</td>
</tr>
<tr>
<td>052</td>
<td>01</td>
<td>Key-management field 2, high-order byte (key completeness). Refer to Table 270 on page 889.</td>
</tr>
<tr>
<td>053</td>
<td>01</td>
<td>Key-management field 2, low-order byte (security history). Refer to Table 270 on page 889.</td>
</tr>
<tr>
<td>054</td>
<td>01</td>
<td>Key-management field 3, high-order byte (pedigree original). Refer to Table 270 on page 889.</td>
</tr>
<tr>
<td>055</td>
<td>01</td>
<td>Key-management field 3, low-order byte (pedigree current). Refer to Table 270 on page 889.</td>
</tr>
<tr>
<td>056</td>
<td>kl</td>
<td>Optional key label.</td>
</tr>
<tr>
<td>056 + kl</td>
<td>iead</td>
<td>Optional IBM extended associated data.</td>
</tr>
</tbody>
</table>
## AES MAC variable-length symmetric key token

View a table showing the format of an AES MAC variable-length symmetric key token.

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>056 + kl + iead</td>
<td>uad</td>
<td>Optional user-defined associated data.</td>
</tr>
</tbody>
</table>

End of associated data section

Optional clear key payload, wrapped AESKW formatted payload, or wrapped PKOAEP2 encoded payload (no payload if no key present)

<table>
<thead>
<tr>
<th>056 + kl + iead + uad</th>
<th>(pl + 7) / 8</th>
<th>Contents of payload (pl is in bits) depending on the encrypted section key-wrapping method (value at offset 26):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value at offset 26</td>
<td>Encrypted section key-wrapping method</td>
<td>Meaning</td>
</tr>
<tr>
<td>X'00'</td>
<td>No key-wrapping method. Only applies when key is clear, that is, when key material state (value at offset 8) is X'01'.</td>
<td>Only the key material will be in the payload. The key token is external or internal.</td>
</tr>
<tr>
<td>X'02'</td>
<td>AESKW</td>
<td>An encrypted payload which the Segment 2 code creates by wrapping the unencrypted AESKW formatted payload. The payload is made up of the integrity check value, pad length, length of hash options and hash, hash options, hash of the associated data, key material, and padding. The key token is internal.</td>
</tr>
<tr>
<td>X'03'</td>
<td>PKOAEP2</td>
<td>An encrypted PKOAEP2 encoded payload created using the RSAES-OAEP scheme of the PKCS #1 v2.1 standard. The message $M$ is encoded for a given hash algorithm using the Bellare and Rogaway Optimal Asymmetric Encryption Padding (OAEP) method for encoding messages. For PKAOEP2, $M$ is defined as follows: $M = [32 \text{ bytes: } hAD] \</td>
</tr>
</tbody>
</table>

End of optional clear key payload, wrapped AESKW formatted payload, or wrapped PKOAEP2 encoded payload

End of clear key, AESKW, or PKOAEP2 components

Note: All numbers are in big endian format.
### Key token formats

Table 272. AES MAC variable-length symmetric key-token, version X'05'.

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>01</td>
<td>Token identifier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>001</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>002</td>
<td>02</td>
<td>Length in bytes of the overall token structure:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * kuf) + (2 * kmf) + kl + iead + uad + ((pl + 7) / 8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Key token</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Minimum token length without DK enabled</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Skeleton</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 3) + 0 + 0 + 0 + 0 = 56</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Encrypted V1 payload</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 3) + 0 + 0 + 0 + ((640 + 7) / 8) = 136</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Key token</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Minimum token length with DK enabled</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Skeleton</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 3) + (2 * 3) + 0 + 0 + 0 + 0 = 58</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Encrypted V1 payload</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 3) + (2 * 3) + 0 + 0 + 0 + ((640 + 7) / 8) = 138</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Key token</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Maximum token length without DK enabled</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>External</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 3) + 64 + 0 + 255 + ((4096 + 7) / 8) = 887</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Internal</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 3) + 64 + 0 + 255 + ((640 + 7) / 8) = 455</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Key token</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Maximum token length with DK enabled</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>External</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 3) + (2 * 3) + 64 + 0 + 255 + ((4096 + 7) / 8) = 889</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Internal</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 3) + (2 * 3) + 64 + 0 + 255 + ((640 + 7) / 8) = 457</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*This assumes a PKOAEP2 key-wrapping method using a 4096-bit RSA transport</td>
</tr>
<tr>
<td></td>
<td></td>
<td>key.</td>
</tr>
<tr>
<td>004</td>
<td>01</td>
<td>Token version number (identifies the format of this key token):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'05'</td>
</tr>
<tr>
<td>005</td>
<td>03</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

End of header

Wrapping information section (all data related to wrapping the key)
Table 272. AES MAC variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>008</td>
<td>01</td>
<td>Key material state:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
</tbody>
</table>

| 009            | 01             | Key verification pattern (KVP) type: |
|                |                | **Value** | **Meaning** |
|                |                | X'00'   | No KVP (no key present or key is wrapped with an RSA public-key). The key token is external or internal. |
|                |                | X'01'   | AESMK (8 leftmost bytes of SHA-256 hash: X'01 || clear AES MK). The key token is internal. |
|                |                | X'02'   | KEK (8 leftmost bytes of SHA-256 hash: X'01 || clear KEK). The key token is external. |
|                |                | All unused values are reserved and undefined. |

| 010            | 16             | KVP (value depends on value of key material state, that is, the value at offset 8): |
|                |                | **Value at offset 8** |
|                |                | **Value of KVP** |
|                |                | X'00' | The key-material state is no key present. The field should be filled with binary zeros. The key token is external or internal. |
|                |                | X'02' | The key material state is the key is wrapped with a transport key. The value of the KVP depends on the value of the encrypted section key-wrapping method: |
|                |                | • When the key-wrapping method is AESKW (value at offset 26 is X'02'), the field contains the KVP of the key-encrypting key used to wrap the key. The 8-byte KEK KVP is left-aligned in the field and padded on the right low-order bytes with binary zeros. |
|                |                | • When the key-wrapping method is PKOAEP2 (value at offset 26 is X'03'), the value should be filled with binary zeros. The encoded message, which contains the key, is wrapped with an RSA public-key. |
|                |                | The key token is external. |
|                |                | X'03' | The key-material state is the key is wrapped with the AES master-key. The field contains the MKVP of the AES master-key used to wrap the key. The 8-byte MKVP is left-aligned in the field and padded on the right low-order bytes with binary zeros. The key token is internal. |
|                |                | All unused values are reserved and undefined. |
### Key token formats

#### Table 272. AES MAC variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>026</td>
<td>01</td>
<td>Encrypted section key-wrapping method (how data in the encrypted section is protected):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
</tbody>
</table>
|                |                | X'03'     | PKOAEP2. Message $M$, which contains the key, is encoded using the RSAES-OAEP scheme of the RSA PKCS #1 v2.1 standard. The encoded message ($EM$) is produced using the given hash algorithm by encoding message $M$ using the Bellare and Rogaway Optimal Asymmetric Encryption Padding (OAEP) method for encoding messages. For PKAOEP2, $M$ is defined as follows:  
  
  $$M = [32 \text{ bytes: } hAD] \ || \ [2 \text{ bytes: bit length of the clear key}] \ || \ [\text{ clear key}]$$  
  
  where $hAD$ is the message digest of the associated data, and is calculated using the SHA-256 algorithm on the data starting at offset 30 for the length in bytes of all the associated data for the key token (length value at offset 32).  
  
  $EM$ is wrapped with an RSA public-key. The key token is external. |

All unused values are reserved and undefined.
Table 272. AES MAC variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>027</td>
<td>01</td>
<td>Hash algorithm used for wrapping key or encoding message. Meaning depends on whether the encrypted section key-wrapping method (value at offset 26) is no key-wrapping method, AESKW, or PKOAEP2:</td>
</tr>
</tbody>
</table>

  - No key-wrapping method (value at offset 26 is X'00')
  - Hash algorithm used for wrapping key when encrypted section key-wrapping method is no key-wrapping method:

    | Value | Meaning                      |
    |-------|------------------------------|
    | X'00' | No hash (no key present)    |

    All unused values are reserved and undefined. The key token is external or internal.

  - AESKW key-wrapping method (value at offset 26 is X'02')

    Hash algorithm used for wrapping key when encrypted section key-wrapping method is AESKW. The value indicates the algorithm used to calculate the message digest of the associated data. The message digest is included in the wrapped payload and is calculated starting at offset 30 for the length in bytes of all the associated data for the key token (length value at offset 32).

    | Value | Meaning |
    |-------|---------|
    | X'02' | SHA-256 |

    All unused values are reserved and undefined. The key token is external or internal.

  - PKOAEP2 key-wrapping method (value at offset 26 is X'03')

    Hash algorithm used for encoding message when encrypted section key-wrapping method is PKOAEP2. The value indicates the given hash algorithm used for encoding message \( M \) using the RSAES-OAEP scheme of the RSA PKCS #1 v2.1 standard.

    | Value | Meaning |
    |-------|---------|
    | X'01' | SHA-1   |
    | X'02' | SHA-256 |
    | X'04' | SHA-384 |
    | X'08' | SHA-512 |

    All unused values are reserved and undefined. The key token is external.
### Key token formats

Table 272. AES MAC variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>028</td>
<td>01</td>
<td>Payload format version (identifies format of the payload):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
</tbody>
</table>

All unused values are reserved and undefined.

| 029           | 01             | Reserved, binary zero. |

End of wrapping information section

AESKW or PKOAEP2 components: (1) associated data section and (2) optional wrapped AESKW formatted payload or wrapped PKOAEP2 encoded payload (no payload if no key present)

### Associated data section

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>030</td>
<td>01</td>
<td>Associated data section version:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td>031</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>032</td>
<td>02</td>
<td>Length in bytes of all the associated data for the key token: 26 - 347.</td>
</tr>
<tr>
<td>034</td>
<td>01</td>
<td>Length in bytes of the optional key label (kl): 0 or 64.</td>
</tr>
<tr>
<td>035</td>
<td>01</td>
<td>Length in bytes of the optional IBM extended associated data (iextd): 0.</td>
</tr>
<tr>
<td>036</td>
<td>01</td>
<td>Length in bytes of the optional user-definable associated data (uadx): 0 - 255.</td>
</tr>
<tr>
<td>037</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>
Table 272. AES MAC variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>038</td>
<td>02</td>
<td>Length in bits of the wrapped payload (pl): 0, 512 - 4096.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For no key-wrapping method (no key present), pl is 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For PKOAEP2 encoded payloads, pl is the length in bits of the modulus size of the RSA key used to wrap the payload. This can be 512 - 4096.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For an AESKW formatted payload, pl is based on the key size of the algorithm type and the payload format version:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AES algorithm (value at offset 41 is X'02')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An AES key can have a length of 16, 24, or 32 bytes (128, 192, or 256 bits). The following table shows the payload length for a given AES key size and payload format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit length of V0 payload (value at offset 28 is X'00')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AES key size</td>
</tr>
<tr>
<td>16 bytes (128 bits)</td>
<td>Not applicable</td>
<td>640</td>
</tr>
<tr>
<td>24 bytes (192 bits)</td>
<td>Not applicable</td>
<td>640</td>
</tr>
<tr>
<td>32 bytes (256 bits)</td>
<td>Not applicable</td>
<td>640</td>
</tr>
<tr>
<td>040</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>041</td>
<td>01</td>
<td>Algorithm type (algorithm for which the key can be used):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>042</td>
<td>02</td>
<td>Key type (general class of the key):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0002'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>044</td>
<td>01</td>
<td>Key usage fields count (kuf): 2 - 3. Key-usage field information defines restrictions on the use of the key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Count is based on whether the key is DK enabled or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DK enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kuf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Each key-usage field is 2 bytes in length. The value in this field indicates how many 2-byte key usage fields follow. Key-usage field information defines restrictions on the use of the key.</td>
</tr>
</tbody>
</table>
Key token formats

Table 272. AES MAC variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>045</td>
<td>01</td>
<td><strong>Key-usage field 1, high-order byte</strong> (MAC operation).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'11xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For $kuf &gt; 2$, if DK enabled (value at offset 50 = X'01') then this value is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>undefined or not used. Otherwise, key can be used for generate; key can</td>
</tr>
<tr>
<td></td>
<td></td>
<td>be used for verify (GENERATE).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'10xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key can be used for generate; key cannot be used for verify (GENONLY).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'01xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key cannot be used for generate; key can be used for verify (VERIFY).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'00xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Undefined or not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
<tr>
<td>046</td>
<td>01</td>
<td><strong>Key-usage field 1, low-order byte</strong> (user-defined extension control).</td>
</tr>
<tr>
<td>047</td>
<td>01</td>
<td><strong>Key-usage field 2, high-order byte</strong> (MAC mode).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01' CMAC mode (CMAC). NIST SP 800-38B.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>048</td>
<td>01</td>
<td><strong>Key-usage field 2, low-order byte</strong> (reserved).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All bits are reserved and must be zero.</td>
</tr>
<tr>
<td>049, for $kuf &gt; 2$</td>
<td>01</td>
<td><strong>Key-usage field 3, high-order byte</strong>. The meaning is determined by the field format identifier (value at offset 50). Currently the only field format identifier is DK enabled:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DK enabled (value at offset 50 is X'01')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Common control:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01' PIN_OP (DKPINOP)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03' PIN_AD1 (DKPINAD1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'04' PIN_AD2 (DKPINAD2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>050, for $kuf &gt; 2$</td>
<td>01</td>
<td><strong>Key-usage field 3, low-order byte</strong> (field format identifier). Identifies the format of key-usage field 3, high-order byte (value at offset 49):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01' DK enabled (set when DKPINOP, DKPINAD1, or DKPINAD2 keyword is used)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>045 + (2 * $kuf$)</td>
<td>01</td>
<td>Key management fields count ($kmf$): 3. Key-management field information describes how the data is to be managed or helps with management of the key material. Each key-management field is 2 bytes in length. The value in this field indicates how many 2-byte key management fields follow.</td>
</tr>
<tr>
<td>046 + (2 * $kuf$)</td>
<td>01</td>
<td>Key-management field 1, high-order byte (symmetric-key export control).</td>
</tr>
<tr>
<td>047 + (2 * $kuf$)</td>
<td>01</td>
<td>Key-management field 1, low-order byte (export control by algorithm).</td>
</tr>
<tr>
<td>048 + (2 * $kuf$)</td>
<td>01</td>
<td>Key-management field 2, high-order byte (key completeness).</td>
</tr>
</tbody>
</table>
Table 272. AES MAC variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>049 + (2 * kuf)</td>
<td>01</td>
<td>Key-management field 2, low-order byte (security history).</td>
</tr>
<tr>
<td>050 + (2 * kuf)</td>
<td>01</td>
<td>Key-management field 3, high-order byte (pedigree original).</td>
</tr>
<tr>
<td>051 + (2 * kuf)</td>
<td>01</td>
<td>Key-management field 3, low-order byte (pedigree current).</td>
</tr>
<tr>
<td>052 + (2 * kuf)</td>
<td>klf</td>
<td>Optional key label.</td>
</tr>
<tr>
<td>052 + (2 * kuf) + kl</td>
<td>iead</td>
<td>Optional IBM extended associated data (unused).</td>
</tr>
<tr>
<td>052 + (2 * kuf) + kl + iead</td>
<td>uad</td>
<td>Optional user-defined associated data.</td>
</tr>
</tbody>
</table>

End of associated data section

Optional wrapped AESKW formatted payload or wrapped PKOAEP2 encoded payload (no payload if no key present)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
</table>

Contents of payload (pl is in bits) depending on the encrypted section key-wrapping method (value at offset 26):

<table>
<thead>
<tr>
<th>Value at offset 26</th>
<th>Encrypted section key-wrapping method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'02'</td>
<td>AESKW</td>
<td>An encrypted payload which the Segment 2 code creates by wrapping the unencrypted AESKW formatted payload. The payload is made up of the integrity check value, pad length, length of hash options and hash, hash options, hash of the associated data, key material, and padding. The key token is internal.</td>
</tr>
</tbody>
</table>
| X'03'              | PKOAEP2                               | An encrypted PKOAEP2 encoded payload created using the RSAES-OAEP scheme of the PKCS #1 v2.1 standard. The message \( M \) is encoded for a given hash algorithm using the Bellare and Rogaway Optimal Asymmetric Encryption Padding (OAEP) method for encoding messages. For PKOAEP2, \( M \) is defined as follows: \[
M = [32 \text{ bytes: } hAD] \ || \ [2 \text{ bytes: bit length of the clear key}] \ || \ [\text{clear key}]
\]
where \( hAD \) is the message digest of the associated data, and is calculated using the SHA-256 algorithm starting at offset 30 for the length in bytes of all the associated data for the key token (length value at offset 32). The encoded message is wrapped with an RSA public-key according to the standard. The key token is external. |

End of optional wrapped AESKW formatted payload or wrapped PKOAEP2 encoded payload

End of AESKW or PKOAEP2 components

**Note:** All numbers are in big endian format.
## HMAC MAC variable-length symmetric key token

View a table showing the format of the HMAC variable-length symmetric key-token.

Table 273 shows the format of the HMAC MAC variable-length symmetric key-token. An HMAC token is used by the HMAC Generate(CSNBHMG) and HMAC Verify(CSNBHMV) verbs to generate and verify keyed hash Message Authentication Codes.

### Table 273. HMAC MAC variable-length symmetric key-token, version X'05' (CCA 4.1.0 or later)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>01</td>
<td>Token identifier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>001</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>002</td>
<td>02</td>
<td>Length in bytes of the overall token structure:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * kuf) + (2 * kmf) + kl + ivad + uad + ((pl + 7) / 8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Key token</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Minimum token length (Release 4.1)</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Skeleton</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 2) + 0 + 0 + 0 + 0 = 54</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Clear V0 payload</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 2) + 0 + 0 + 0 + ((80 + 7) / 8) = 64</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Encrypted V0 payload</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 2) + 0 + 0 + 0 + ((448 + 7) / 8) = 110</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Key token</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Minimum token length (Release 4.2 or later)</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Skeleton</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 3) + 0 + 0 + 0 + 0 = 56</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Clear V0 payload</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 3) + 0 + 0 + 0 + ((80 + 7) / 8) = 66</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Encrypted V0 payload</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 3) + 0 + 0 + 0 + ((448 + 7) / 8) = 112</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Key token</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Maximum token length (Release 4.1)</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>External</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 2) + 64 + 0 + 255 + ((4096 + 7) / 8) = 885</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Input</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 2) + 64 + 0 + 255 + ((2432 + 7) / 8) = 677</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Key token</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Maximum token length (Release 4.2 or later)</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>External</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 3) + 64 + 0 + 255 + ((4096 + 7) / 8) = 887</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Internal</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 3) + 64 + 0 + 255 + ((2432 + 7) / 8) = 679</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>This assumes a PKOAEP2 key-wrapping method using a 4096-bit RSA transport key.</em></td>
</tr>
</tbody>
</table>
### Table 273. HMAC MAC variable-length symmetric key-token, version X'05' (CCA 4.1.0 or later) (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>004</td>
<td>01</td>
<td>Token version number (identifies the format of this key token):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'05'</td>
</tr>
<tr>
<td>005</td>
<td>03</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

**End of header**

**Wrapping information section (all data related to wrapping the key)**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>008</td>
<td>01</td>
<td>Key material state:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
</tbody>
</table>

All unused values are reserved and undefined.

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>009</td>
<td>01</td>
<td>Key verification pattern (KVP) type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
</tbody>
</table>

All unused values are reserved and undefined.

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>16</td>
<td>KVP (value depends on value of key material state, that is, the value at offset 8):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value at offset 8</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value of KVP</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X' 00 '</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• When the key-wrapping method is AESKW (value at offset 26 is X'02'), the field contains the KVP of the key-encrypting key used to wrap the key. The 8-byte KEK KVP is left-aligned in the field and padded on the right low-order bytes with binary zeros.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• When the key-wrapping method is PKOAEP2 (value at offset 26 is X'03'), the value should be filled with binary zeros. The encoded message, which contains the key, is wrapped with an RSA public-key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The key token is external.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
</tbody>
</table>
### Key token formats

**Table 273. HMAC MAC variable-length symmetric key-token, version X'05' (CCA 4.1.0 or later) (continued)**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>026</td>
<td>01</td>
<td>Encrypted section key-wrapping method (how data in the encrypted section is protected):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td>X'00'</td>
<td></td>
<td>No key-wrapping method (no key present or key is clear). The key token is external or internal.</td>
</tr>
<tr>
<td>X'02'</td>
<td></td>
<td>AESKW (ANS X9.102). The key token is external with a key wrapped by an AES key-encrypting key, or the key token is internal with a key wrapped by the AES master-key.</td>
</tr>
<tr>
<td>X'03'</td>
<td></td>
<td>PKOAE P2. Message M, which contains the key, is encoded using the RSAES-OAEP scheme of the RSA PKCS #1 v2.1 standard. The encoded message (EM) is produced using the given hash algorithm by encoding message M using the Bellare and Rogaway Optimal Asymmetric Encryption Padding (OAEP) method for encoding messages. For PKOAE P2, M is defined as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M = [32 bytes: hAD]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>where hAD is the message digest of the associated data, and is calculated using the SHA-256 algorithm on the data starting at offset 30 for the length in bytes of all the associated data for the key token (length value at offset 32).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EM is wrapped with an RSA public-key. The key token is external.</td>
</tr>
</tbody>
</table>

All unused values are reserved and undefined.
### Table 273. HMAC MAC variable-length symmetric key-token, version X'05' (CCA 4.1.0 or later) (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 027            | 01             | Hash algorithm used for wrapping key or encoding message. Meaning depends on whether the encrypted section key-wrapping method (value at offset 26) is no key-wrapping method, AESKW, or PKOAEP2:  
  - No key-wrapping method (value at offset 26 is X'00')  
  - Hash algorithm used for wrapping key when encrypted section key-wrapping method is no key-wrapping method:  
    - Value X'00' Meaning  
      - No hash (no key present or key is clear).  
      - All unused values are reserved and undefined. The key token is external or internal.  
    - AESKW key-wrapping method (value at offset 26 is X'02')  
      - Hash algorithm used for wrapping key when encrypted section key-wrapping method is AESKW. The value indicates the algorithm used to calculate the message digest of the associated data. The message digest is included in the wrapped payload and is calculated starting at offset 30 for the length in bytes of all the associated data for the key token (length value at offset 32).  
      - Value X'02' Meaning  
        - SHA-256  
        - All unused values are reserved and undefined. The key token is external or internal.  
    - PKOAEP2 key-wrapping method (value at offset 26 is X'03')  
      - Hash algorithm used for encoding message when encrypted section key-wrapping method is PKOAEP2. The value indicates the given hash algorithm used for encoding message M using the RSAES-OAEP scheme of the RSA PKCS #1 v2.1 standard.  
      - Value X'01' Meaning  
        - SHA-1  
      - Value X'02' Meaning  
        - SHA-256  
      - Value X'04' Meaning  
        - SHA-384  
      - Value X'08' Meaning  
        - SHA-512  
      - All unused values are reserved and undefined. The key token is external. |
Table 273. HMAC MAC variable-length symmetric key-token, version X'05' (CCA 4.1.0 or later) (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>028</td>
<td>01</td>
<td>Payload format version (identifies format of the payload). Release 4.4 or later, otherwise undefined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value at offset 26</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>029</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

End of wrapping information section

Clear key, AESKW, or PKOAEP2 components: (1) associated data section and (2) optional clear key payload, wrapped AESKW formatted payload, or wrapped PKOAEP2 encoded payload (no payload if no key present)

Associated data section

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>030</td>
<td>01</td>
<td>Associated data section version:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td>031</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>032</td>
<td>02</td>
<td>Length in bytes of all the associated data for the key token: 24 - 343 (Release 4.1); 26 - 345 (Release 4.2 or later).</td>
</tr>
<tr>
<td>034</td>
<td>01</td>
<td>Length in bytes of the optional key label (kl): 0 or 64.</td>
</tr>
<tr>
<td>035</td>
<td>01</td>
<td>Length in bytes of the optional IBM extended associated data (iemd): 0.</td>
</tr>
<tr>
<td>036</td>
<td>01</td>
<td>Length in bytes of the optional user-definable associated data (uad): 0 - 255.</td>
</tr>
<tr>
<td>037</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>038</td>
<td>02</td>
<td>Length in <em>bits</em> of the clear or wrapped payload (<em>pl</em>): 0, 80 - 4096.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For no key-wrapping method (no key present or key is clear), <em>pl</em> is the length in bits of the key. For no key present, <em>pl</em> is 0. For key is clear, <em>pl</em> can be 128, 192, or 256.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For PKOAEP2 encoded payloads, <em>pl</em> is the length in bits of the modulus size of the RSA key used to wrap the payload. This can be 512 - 4096.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For an AESKW formatted payload, <em>pl</em> is based on the key size of the algorithm type and the payload format version:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HMAC algorithm (value at offset 41 is X'03')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An HMAC key can have a length of 80 - 2048 bits. An HMAC key in an AESKW formatted payload is always wrapped with a V0 payload.</td>
</tr>
<tr>
<td>040</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>
### Table 273. HMAC MAC variable-length symmetric key-token, version X'05' (CCA 4.1.0 or later) (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>041</td>
<td>01</td>
<td><strong>Algorithm type</strong> (algorithm for which the key can be used):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>042</td>
<td>02</td>
<td><strong>Key type</strong> (general class of the key):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0002'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>044</td>
<td>01</td>
<td><strong>Key usage fields count (kuf)</strong>: 2. Key-usage field information defines restrictions on the use of the key. Refer to Figure 10 on page 288.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Each key-usage field is 2 bytes in length. The value in this field indicates how many 2-byte key usage fields follow.</td>
</tr>
<tr>
<td>045</td>
<td>01</td>
<td><strong>Key-usage field 1, high-order byte</strong> (MAC operation):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'11xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'10xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'01xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'00xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
<tr>
<td>046</td>
<td>01</td>
<td><strong>Key-usage field 1, low-order byte</strong> (user-defined extension control)</td>
</tr>
<tr>
<td>047</td>
<td>01</td>
<td><strong>Key-usage field 2, high-order byte</strong> (hash method):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'1xxx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0xxx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'x1xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'x0xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xx1x xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xx0x xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxx1 xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxx0 xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx 1xxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx 0xxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
</tbody>
</table>
Key token formats

Table 273. HMAC MAC variable-length symmetric key-token, version X'05' (CCA 4.1.0 or later) (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>048</td>
<td>01</td>
<td>Key-usage field 2, low-order byte (reserved). All bits are reserved and must be zero.</td>
</tr>
<tr>
<td>049</td>
<td>01</td>
<td>Key management fields count (kmf): 3. Key-management field information describes how the data is to be managed or helps with management of the key material. Each key-management field is 2 bytes in length. The value in this field indicates how many 2-byte key management fields follow.</td>
</tr>
<tr>
<td>050</td>
<td>01</td>
<td>Key-management field 1, high-order byte (symmetric-key export control).</td>
</tr>
<tr>
<td>051</td>
<td>01</td>
<td>Key-management field 1, low-order byte (export control by algorithm).</td>
</tr>
<tr>
<td>052</td>
<td>01</td>
<td>Key-management field 2, high-order byte (key completeness).</td>
</tr>
<tr>
<td>053</td>
<td>01</td>
<td>Key-management field 2, low-order byte (security history).</td>
</tr>
<tr>
<td>054, for kuf &gt; 2</td>
<td>01</td>
<td>Key-management field 3, high-order byte (pedigree original). Release 4.2 or later.</td>
</tr>
<tr>
<td>055, for kuf &gt; 2</td>
<td>01</td>
<td>Key-management field 3, low-order byte (pedigree current). Release 4.2 or later.</td>
</tr>
<tr>
<td>050 + (2 * kmf)</td>
<td>kl</td>
<td>Optional key label.</td>
</tr>
<tr>
<td>050 + (2 * kmf) + kl</td>
<td>iead</td>
<td>Optional IBM extended associated data (unused).</td>
</tr>
<tr>
<td>050 + (2 * kmf) + kl + iead</td>
<td>uad</td>
<td>Optional user-defined associated data.</td>
</tr>
</tbody>
</table>

End of associated data section

Optional clear key payload, wrapped AESKW formatted payload, or wrapped PKOAEP2 encoded payload (no payload if no key present)
Table 273. HMAC MAC variable-length symmetric key-token, version X’05’ (CCA 4.1.0 or later) (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>050+ (2 * kmf) + kl + iead + uad</td>
<td>(pl + 7) / 8</td>
<td>Contents of payload (pl is in bits) depending on the encrypted section key-wrapping method (value at offset 26):</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value at offset 26</th>
<th>Encrypted section key-wrapping method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X’00’</td>
<td>No key-wrapping method. Only applies when key is clear, that is, when key material state (value at offset 8) is X’01’.</td>
<td>Only the key material will be in the payload. The key token is external or internal.</td>
</tr>
<tr>
<td>X’02’</td>
<td>AESKW</td>
<td>An encrypted payload which the Segment 2 code creates by wrapping the unencrypted AESKW formatted payload. The payload is made up of the integrity check value, pad length, length of hash options and hash, hash options, hash of the associated data, key material, and padding. The key token is internal.</td>
</tr>
<tr>
<td>X’03’</td>
<td>PKOAEP2</td>
<td>An encrypted PKOAEP2 encoded payload created using the RSAES-OAEP scheme of the PKCS #1 v2.1 standard. The message M is encoded for a given hash algorithm using the Bellare and Rogaway Optimal Asymmetric Encryption Padding (OAEP) method for encoding messages. For PKOAEP2, M is defined as follows: M = [32 bytes: hAD]</td>
</tr>
</tbody>
</table>

End of optional clear key payload, wrapped AESKW formatted payload, or wrapped PKOAEP2 encoded payload

End of clear key, AESKW, or PKOAEP2 components

Note: All numbers are in big endian format.

AES EXPORTER and IMPORTER variable-length symmetric key token

View a table showing the format of the AES EXPORTER and IMPORTER variable-length symmetric key-token.

Table 274 on page 928 shows the format of the EXPORTER and IMPORTER variable-length symmetric key tokens that can be used with the AES algorithm. An EXPORTER operational key-token is used by the Symmetric Key Export (CSNDSYX) verb to export an internal AES or HMAC variable-length symmetric key-token into an external variable-length symmetric key-token, either into an
**Key token formats**

AESKW or PKOAEP2 wrapped payload. An IMPORTER operational key-token is used by the Symmetric Key Import2 (CSNDSYI2) verb to import an external AES or HMAC variable-length symmetric key-token, containing either an AESWK or PKOAEP2 wrapped payload, into an internal variable-length symmetric key-token.

Table 274. AES EXPORTER and IMPORTER variable-length symmetric key-token, version X'05'.

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>01</td>
<td>Token identifier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>001</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>002</td>
<td>02</td>
<td>Length in bytes of the overall token structure:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * kuf) + (2 * kmf) + kl + lead + uad + ((pl + 7) / 8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Key token</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Skeleton</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 4) + (2 * 3) + 0 + 0 + 0 + 0 = 60</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Encrypted V0 payload</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 4) + (2 * 3) + 0 + 0 + 0 + ((512 + 7) / 8) = 124</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Encrypted V1 payload</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 4) + (2 * 3) + 0 + 0 + 0 + ((640 + 7) / 8) = 140</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Key token</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Minimum token length</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 4) + (2 * 3) + 64 + 0 + 255 + ((4096 + 7) / 8) = 891</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Maximum token length</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 4) + (2 * 3) + 64 + 0 + 255 + ((640 + 7) / 8) = 459</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*This assumes a PKOAEP2 key-wrapping method using a 4096-bit RSA transport key.</td>
</tr>
<tr>
<td>004</td>
<td>01</td>
<td>Token version number (identifies the format of this key token):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'05'</td>
</tr>
<tr>
<td>005</td>
<td>03</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>End of header</td>
</tr>
</tbody>
</table>

Wrapping information section (all data related to wrapping the key)
Table 274. AES EXPORTER and IMPORTER variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>008</td>
<td>01</td>
<td>Key material state:</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td>X'00'</td>
<td>No key is present. This is called a skeleton key-token. The key token is external or internal.</td>
<td></td>
</tr>
<tr>
<td>X'02'</td>
<td>Key is wrapped with a transport key. When the encrypted section key-wrapping method is AESKW (value at offset 26 is X'02'), the transport key is an AES key-encrypting key. When it is PKOAEP2 (value at offset 26 is X'03'), the transport key is an RSA public-key. The key token is external.</td>
<td></td>
</tr>
<tr>
<td>X'03'</td>
<td>Key is wrapped with the AES master-key. The encrypted section key-wrapping method is AESKW. The key token is internal.</td>
<td></td>
</tr>
</tbody>
</table>

All unused values are reserved and undefined.

| 009            | 01             | Key verification pattern (KVP) type: |
| Value          | Meaning        |             |
| X'00'          | No KVP (no key present or key is wrapped with an RSA public-key). The key token is external or internal. |
| X'01'          | AESMK (8 leftmost bytes of SHA-256 hash: X'01  || clear AES MK). The key token is internal. |
| X'02'          | KEK (8 leftmost bytes of SHA-256 hash: X'01  || clear KEK). The key token is external. |

All unused values are reserved and undefined.

| 010            | 16             | KVP (value depends on value of key material state, that is, the value at offset 8): |
| Value at offset 8 |
| Value of KVP   |
| X'00'          | The key-material state is no key present. The field should be filled with binary zeros. The key token is external or internal. |
| X'02'          | The key material state is the key is wrapped with a transport key. The value of the KVP depends on the value of the encrypted section key-wrapping method: |
|                 | • When the key-wrapping method is AESKW (value at offset 26 is X'02'), the field contains the KVP of the key-encrypting key used to wrap the key. The 8-byte KEK KVP is left-aligned in the field and padded on the right low-order bytes with binary zeros. |
|                 | • When the key-wrapping method is PKOAEP2 (value at offset 26 is X'03'), the value should be filled with binary zeros. The encoded message, which contains the key, is wrapped with an RSA public-key. |
|                 | The key token is external. |
| X'03'          | The key-material state is the key is wrapped with the AES master-key. The field contains the MKVP of the AES master-key used to wrap the key. The 8-byte MKVP is left-aligned in the field and padded on the right low-order bytes with binary zeros. The key token is internal. |
### Key token formats

**Table 274. AES EXPORTER and IMPORTER variable-length symmetric key-token, version X'05' (continued).**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>026</td>
<td>01</td>
<td>Encrypted section key-wrapping method (how data in the encrypted section is protected):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td>All unused values are reserved and undefined.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 274. AES EXPORTER and IMPORTER variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>027</td>
<td>01</td>
<td>Hash algorithm used for wrapping key or encoding message. Meaning depends on whether the encrypted section key-wrapping method (value at offset 26) is no key-wrapping method, AESKW, or PKOAEP2:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No key-wrapping method (value at offset 26 is X'00')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hash algorithm used for wrapping key when encrypted section key-wrapping method is no key-wrapping method:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AESKW key-wrapping method (value at offset 26 is X'02')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hash algorithm used for wrapping key when encrypted section key-wrapping method is AESKW. The value indicates the algorithm used to calculate the message digest of the associated data. The message digest is included in the wrapped payload and is calculated starting at offset 30 for the length in bytes of all the associated data for the key token (length value at offset 32).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PKOAEP2 key-wrapping method (value at offset 26 is X'03')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hash algorithm used for encoding message when encrypted section key-wrapping method is PKOAEP2. The value indicates the given hash algorithm used for encoding message $M$ using the RSAES-OAEP scheme of the RSA PKCS #1 v2.1 standard.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'04'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'08'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined. The key token is external.</td>
</tr>
</tbody>
</table>
### Table 274. AES EXPORTER and IMPORTER variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>028</td>
<td>01</td>
<td>Payload format version (identifies format of the payload). Release 4.4 or later, otherwise undefined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00' V0 payload (V0PYLD). The payload format depends on the encrypted section key-wrapping method (value at offset 26):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value at offset 26</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00' There is no key-wrapping method. When no key is present, there is no payload. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02' The key-wrapping method is AESKW and the payload is variable length. The payload is formatted with the minimum size possible to contain the key material. The payload length varies for a given algorithm and key type. The key length can be inferred by the size of the payload. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03' The key-wrapping method is PKOAEP2 and the payload length is equal to the modulus size in bits of the RSA transport key used to wrap the encoded message. The key token is external. When the external key is exported, the internal target key will have the same V0 payload format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01' V1 payload (Release 4.4 or later). The payload format depends on the encrypted section key-wrapping method (value at offset 26):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value at offset 26</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00' There is no key-wrapping method. When no key is present, there is no payload. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02' The key-wrapping method is AESKW and the payload is fixed length based on the maximum possible key size of the algorithm for the key. The key is padded with random data to the size of the largest key for that algorithm. This helps to deter attacks on keys known to be weaker. The key length cannot be inferred by the size of the payload. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03' The key-wrapping method is PKOAEP2 and the payload length is equal to the modulus size in bits of the RSA transport key used to wrap the encoded message. The key token is external. When the external key is exported, the internal target key will have the same V1 payload format.</td>
</tr>
</tbody>
</table>

All unused values are reserved and undefined.

029 01 Reserved, binary zero.

### End of wrapping information section

AESKW or PKOAEP2 components: (1) associated data section and (2) optional wrapped AESKW formatted payload or wrapped PKOAEP2 encoded payload (no payload if no key present)

### Associated data section

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Associated data section version:</th>
</tr>
</thead>
<tbody>
<tr>
<td>030</td>
<td>01</td>
<td><strong>Value</strong> <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01' Version 1 format of associated data</td>
</tr>
<tr>
<td>031</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>032</td>
<td>02</td>
<td>Length in bytes of all the associated data for the key token: 30 - 349.</td>
</tr>
<tr>
<td>034</td>
<td>01</td>
<td>Length in bytes of the optional key label (k): 0 or 64.</td>
</tr>
<tr>
<td>035</td>
<td>01</td>
<td>Length in bytes of the optional IBM extended associated data (iead): 0.</td>
</tr>
<tr>
<td>036</td>
<td>01</td>
<td>Length in bytes of the optional user-definable associated data (uad): 0 - 255.</td>
</tr>
</tbody>
</table>
Table 274. AES EXPORTER and IMPORTER variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>037</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>038</td>
<td>02</td>
<td>Length in bits of the wrapped payload (pl): 0, 512 - 4096.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- For no key-wrapping method (no key present), pl is 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- For PKOAE2 encoded payloads, pl is the length in bits of the modulus size of the RSA key used to wrap the payload. This can be 512 - 4096.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- For an AESKW formatted payload, pl is based on the key size of the algorithm type and the payload format version:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- AES algorithm (value at offset 41 is X'02')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- An AES key can have a length of 16, 24, or 32 bytes (128, 192, or 256 bits). The following table shows the payload length for a given AES key size and payload format:</td>
</tr>
<tr>
<td>040</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>041</td>
<td>01</td>
<td>Algorithm type (algorithm for which the key can be used):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>042</td>
<td>02</td>
<td>Key type (general class of the key):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0003'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0004'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>044</td>
<td>01</td>
<td>Key usage fields count (kuf): 4. Key-usage field information defines restrictions on the use of the key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For key type EXPORTER, see AES EXPORTER Key Token Build2 keywords Figure 7 on page 277</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For key type IMPORTER, see AES IMPORTER Key Token Build2 keywords Figure 8 on page 281</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Each key-usage field is 2 bytes in length. The value in this field indicates how many 2-byte key usage fields follow.</td>
</tr>
</tbody>
</table>
### Key token formats

**Table 274. AES EXPORTER and IMPORTER variable-length symmetric key-token, version X'05’ (continued).**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>045 (1 of 2)</td>
<td>01</td>
<td>Key-usage field 1, high-order byte (KEK control). The meaning is determined by the key type (value at offset 42). The key type can be EXPORTER or IMPORTER.</td>
</tr>
</tbody>
</table>

**EXPORTER (value at offset 42 is X'0003')**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>B'1xxx xxxx'</td>
<td>Key can be used to export a key (EXPORT).</td>
</tr>
<tr>
<td>B'0xxx xxxx'</td>
<td>Key cannot be used to export a key.</td>
</tr>
<tr>
<td>B'x1xx xxxx'</td>
<td>Key can be used to translate a key (TRANSLAT).</td>
</tr>
<tr>
<td>B'x0xx xxxx'</td>
<td>Key cannot be used to translate a key.</td>
</tr>
<tr>
<td>B'xx1x xxxx'</td>
<td>Key can be used by KGN2 for generating an OPEX key pair (GEN-OPEX).</td>
</tr>
<tr>
<td>B'xx0x xxxx'</td>
<td>Key cannot be used by KGN2 for generating an OPEX key pair.</td>
</tr>
<tr>
<td>B'xxx1 xxxx'</td>
<td>Key can be used by KGN2 for generating an IMEX key pair (GEN-IMEX).</td>
</tr>
<tr>
<td>B'xxx0 xxxx'</td>
<td>Key cannot be used by KGN2 for generating an IMEX key pair.</td>
</tr>
<tr>
<td>B'xxxx 1xxx'</td>
<td>Key can be used by KGN2 for generating an EXEX key pair (GEN-EXEX).</td>
</tr>
<tr>
<td>B'xxxx 0xxx'</td>
<td>Key cannot be used by KGN2 for generating an EXEX key pair.</td>
</tr>
<tr>
<td>B'xxxx x1xx'</td>
<td>Key can be used by PKG for generating an ECC public-private key pair (GEN-PUB).</td>
</tr>
<tr>
<td>B'xxxx x0xx'</td>
<td>Key cannot be used by PKG for generating an ECC public-private key pair (GEN-PUB).</td>
</tr>
</tbody>
</table>

**Note:** At least one defined bit must be B’1’.

All unused bits are reserved and must be zero.
Table 274. AES EXPORTER and IMPORTER variable-length symmetric key-token, version X’05’ (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 045 (2 of 2)   | 01             | IMPORTER (value at offset 42 is X’0004’)

Value  Meaning  
B’1xxx xxxx’  Key can be used to import a key (IMPORT).  
B’0xxx xxxx’  Key cannot be used to import a key.  
B’x1xx xxxx’  Key can be used to translate a key (TRANSLAT).  
B’x0xx xxxx’  Key cannot be used to translate a key.  
B’xx1x xxxx’  Key can be used by KGN2 for generating an OPIM key pair (GEN-OPIM).  
B’xx0x xxxx’  Key cannot be used by KGN2 for generating an OPIM key pair.  
B’xxx1 xxxx’  Key can be used by KGN2 for generating an IMEX key pair (GEN-IMEX).  
B’xxx0 xxxx’  Key cannot be used by KGN2 for generating an IMEX key pair.  
B’xxxx 1xxx’  Key can be used by KGN2 for generating an IMIM key pair (GEN-IMIM).  
B’xxxx 0xxx’  Key cannot be used by KGN2 for generating an IMIM key pair.  
B’xxxx x1xx’  Key can be used by PKG for generating an ECC public-private key pair (GEN-PUB).  
B’xxxx x0xx’  Key cannot be used by PKG for generating an ECC public-private key pair (GEN-PUB).  

Note: At least one defined bit must be B’1’.  
All unused bits are reserved and must be zero.

| 046 | 01 | Key-usage field 1, low-order byte (user-defined extension control). |
| 047 | 01 | Key-usage field 2, high-order byte (TR-31 wrap control):  
Value  Meaning  
B’1xxx xxxx’  Key can wrap or unwrap a TR-31 key (WR-TR31). Defined for future use.  
B’0xxx xxxx’  Key cannot wrap or unwrap a TR-31 key. Defined for future use.  
All unused bits are reserved and must be zero.  

| 048 | 01 | Key-usage field 2, low-order byte (raw key wrap control):  
Value  Meaning  
B’xxxx xxxI’  Key can wrap or unwrap a raw key (KEK-RAW). Defined for future use.  
B’0xxx xxxx’  Key cannot wrap or unwrap a raw key. Defined for future use.  
All unused bits are reserved and must be zero.  

Chapter 19. Key token formats  935
## Key token formats

Table 274. AES EXPORTER and IMPORTER variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>049</td>
<td>01</td>
<td><strong>Key-usage field 3, high-order byte</strong> (algorithm wrap control):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> Meanings</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’1xxx xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’0xxx xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’x1xx xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’x0xx xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xx1x xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xx0x xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xxx1 xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xxx0 xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xxxx 1xxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xxxx 0xxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> At least one defined bit must be B’1’.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
<tr>
<td>050</td>
<td>01</td>
<td><strong>Key-usage field 3, low-order byte</strong> (reserved).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All bits are reserved and must be zero.</td>
</tr>
<tr>
<td>Offset (bytes)</td>
<td>Length (bytes)</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>051</td>
<td>01</td>
<td><strong>Key-usage field 4, high-order byte</strong> (class wrap control).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’1xxx xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key can wrap or unwrap data class keys (WR-DATA).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’0xxx xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key cannot wrap or unwrap data class keys.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’x1xx xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key can wrap or unwrap KEK class keys (WR-KEK).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’x0xx xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key cannot wrap or unwrap KEK class keys.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xx1x xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key can wrap or unwrap PIN class keys (WR-PIN).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xx0x xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key cannot wrap or unwrap PIN class keys.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xxx1 xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key can wrap or unwrap derivation class keys (WRDERIVE).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xxx0 xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key cannot wrap or unwrap derivation class keys.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xxx1 xxxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key can wrap or unwrap card class keys (WR-CARD).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xxxx 0xxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key cannot wrap or unwrap card class keys.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xxxx 1xxx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key can wrap or unwrap cryptovariable class keys (WR-CVAR). Undefined in releases before Release 4.4.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B’xxxx x0xx’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key cannot wrap or unwrap cryptovariable class keys. Undefined in releases before Release 4.4.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> At least one defined bit must be B’1’.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>052</td>
<td>01</td>
<td><strong>Key-usage field 4, low-order byte</strong> (reserved).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All bits are reserved and must be zero.</td>
</tr>
<tr>
<td>053</td>
<td>01</td>
<td>Key management fields count (kmf): 3. Key-management field information describes how the data is to be managed or helps with management of the key material.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For key type EXPORTER, see AES EXPORTER Key Token Build2 keywords [Figure 7 on page 277].</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For key type IMPORTER, see AES IMPORTER Key Token Build2 keywords [Figure 8 on page 281].</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Each key-management field is 2 bytes in length. The value in this field indicates how many 2-byte key management fields follow.</td>
</tr>
<tr>
<td>054</td>
<td>01</td>
<td>Key-management field 1, high-order byte (symmetric-key export control).</td>
</tr>
<tr>
<td>055</td>
<td>01</td>
<td>Key-management field 1, low-order byte (export control by algorithm).</td>
</tr>
<tr>
<td>056</td>
<td>01</td>
<td>Key-management field 2, high-order byte (key completeness).</td>
</tr>
<tr>
<td>057</td>
<td>01</td>
<td>Key-management field 2, low-order byte (security history).</td>
</tr>
<tr>
<td>058</td>
<td>01</td>
<td>Key-management field 3, high-order byte (pedigree original).</td>
</tr>
<tr>
<td>059</td>
<td>01</td>
<td>Key-management field 3, low-order byte (pedigree current).</td>
</tr>
<tr>
<td>060</td>
<td>kl</td>
<td>Optional key label.</td>
</tr>
</tbody>
</table>
**Table 274. AES EXPORTER and IMPORTER variable-length symmetric key-token, version X'05' (continued).**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>060 + kl idead</td>
<td></td>
<td>Optional IBM extended associated data (unused).</td>
</tr>
<tr>
<td>060 + kl + idead uad</td>
<td></td>
<td>Optional user-defined associated data.</td>
</tr>
</tbody>
</table>

End of associated data section

Optional wrapped AESKW formatted payload or wrapped PKOAEP2 encoded payload (no payload if no key present)

<table>
<thead>
<tr>
<th>060 + kl + idead uad</th>
<th>(pl + 7) / 8</th>
<th>Contents of payload (pl is in bits) depending on the encrypted section key-wrapping method (value at offset 26):</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Value at offset 26</td>
<td>Encrypted section key-wrapping method</td>
</tr>
<tr>
<td></td>
<td>X'02'</td>
<td>AESKW</td>
</tr>
</tbody>
</table>
|                      | X'03' | PKOAEP2 | An encrypted PKOAEP2 encoded payload created using the RSAES-OAEP scheme of the PKCS #1 v2.1 standard. The message M is encoded for a given hash algorithm using the Bellare and Rogaway Optimal Asymmetric Encryption Padding (OAEP) method for encoding messages. For PKAOEP2, M is defined as follows: $M = [32 \text{ bytes: } hAD] \ || [2 \text{ bytes: bit length of the clear key}] \ || [\text{clear key}]$  
  where $hAD$ is the message digest of the associated data, and is calculated using the SHA-256 algorithm starting at offset 30 for the length in bytes of all the associated data for the key token (length value at offset 32). The encoded message is wrapped with an RSA public-key according to the standard. The key token is external. |

End of optional wrapped AESKW formatted payload or wrapped PKAOEP2 encoded payload

End of AESKW or PKAOEP2 components

**Note:** All numbers are in big endian format.

---

**AES PINPROT, PINCALC, and PINPRW variable-length symmetric key token**

View a table showing the format of the PINPROT, PINCALC, and PINPRW variable-length symmetric key-tokens.

**Table 275. AES CIPHER variable-length symmetric key-token, version X'05'.**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 275. AES CIPHER variable-length symmetric key-token, version X’05’ (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>01</td>
<td>Token identifier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’01’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’02’</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>002</td>
<td>02</td>
<td>Length in bytes of the overall token structure:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * kuf) + (2 * kmf) + kl +lead + uad + ((pl + 7) / 8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key token</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Minimum token length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Skeleton</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 3) + (2 * 3) + 0 + 0 + 0 + 0 = 58</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Encrypted V1 payload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 3) + (2 * 3) + 0 + 0 + 0 + ((640 + 7) / 8) = 138</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key token</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum token length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>External*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 3) + (2 * 3) + 64 + 0 + 255 + ((4096 + 7) / 8) = 889</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 3) + (2 * 3) + 64 + 0 + 255 + ((640 + 7) / 8) = 457</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*This assumes a PKOAEP2 key-wrapping method using a 4096-bit RSA transport key.</td>
</tr>
<tr>
<td>004</td>
<td>01</td>
<td>Token version number (identifies the format of this key token):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’05’</td>
</tr>
<tr>
<td>005</td>
<td>03</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>End of header</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wrapping information section (all data related to wrapping the key)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>008</td>
<td>01</td>
<td>Key material state:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’00’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’02’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’03’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
</tbody>
</table>
### Key token formats

Table 275. AES CIPHER variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>009</td>
<td>01</td>
<td>Key verification pattern (KVP) type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>16</td>
<td>KVP (value depends on value of key material state, that is, the value at offset 8):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value at offset 8</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value of KVP</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td>026</td>
<td>01</td>
<td>Encrypted section key-wrapping method (how data in the encrypted section is protected):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 275. AES CIPHER variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>027</td>
<td>01</td>
<td>Hash algorithm used for wrapping key or encoding message. Meaning depends on whether the encrypted section key-wrapping method (value at offset 26) is no key-wrapping method, AESKW, or PKOAEP2:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No key-wrapping method (value at offset 26 is X'00')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hash algorithm used for wrapping key when encrypted section key-wrapping method is no key-wrapping method:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AESKW key-wrapping method (value at offset 26 is X'02')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hash algorithm used for wrapping key when encrypted section key-wrapping method is AESKW. The value indicates the algorithm used to calculate the message digest of the associated data. The message digest is included in the wrapped payload and is calculated starting at offset 30 for the length in bytes of all the associated data for the key token (length value at offset 32).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PKOAEP2 key-wrapping method (value at offset 26 is X'03')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hash algorithm used for encoding message when encrypted section key-wrapping method is PKOAEP2. The value indicates the given hash algorithm used for encoding message M using the RSAES-OAEP scheme of the RSA PKCS #1 v2.1 standard.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'04'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'08'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined. The key token is external.</td>
</tr>
</tbody>
</table>
### Key token formats

Table 275. AES CIPHER variable-length symmetric key-token, version X’05’ (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>028</td>
<td>01</td>
<td>Payload format version (identifies format of the payload):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’01’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V1 payload (V0PYLD). The payload format depends on the encrypted section</td>
</tr>
<tr>
<td></td>
<td></td>
<td>key-wrapping method (value at offset 26):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value at offset 26</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’00’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>There is no key-wrapping method. When no key is present, there is no</td>
</tr>
<tr>
<td></td>
<td></td>
<td>payload. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’02’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The key-wrapping method is AESKW and the payload is fixed length based</td>
</tr>
<tr>
<td></td>
<td></td>
<td>on the maximum possible key size of the algorithm for the key. The key is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>padded with random data to the size of the largest key for that algorithm.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This helps to deter attacks on keys known to be weaker. The key length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cannot be inferred by the size of the payload. The key token is external or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’03’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The key-wrapping method is PKOAEP2 and the payload length is equal to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the modulus size in bits of the RSA transport key used to wrap the encoded</td>
</tr>
<tr>
<td></td>
<td></td>
<td>message. The key token is external. When the external key is exported, the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>internal target key will have the same V1 payload format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>029</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

End of wrapping information section

AESKW or PKOAEP2 components: (1) associated data section and (2) optional wrapped AESKW formatted payload or wrapped PKOAEP2 encoded payload (no payload if no key present)

Associated data section

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>030</td>
<td>01</td>
<td>Associated data section version:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’01’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Version 1 format of associated data</td>
</tr>
<tr>
<td>031</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>032</td>
<td>02</td>
<td>Length in bytes of all the associated data for the key token: 16 - 347.</td>
</tr>
<tr>
<td>034</td>
<td>01</td>
<td>Length in bytes of the optional key label (kl): 0 or 64.</td>
</tr>
<tr>
<td>035</td>
<td>01</td>
<td>Length in bytes of the optional IBM extended associated data (iead): 0.</td>
</tr>
<tr>
<td>036</td>
<td>01</td>
<td>Length in bytes of the optional user-definable associated data (uad): 0 - 255.</td>
</tr>
<tr>
<td>037</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>
### Table 275. AES CIPHER variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 038           | 02             | Length in bits of the wrapped payload ($pl$): 0, 512 - 4096.  
  - For no key-wrapping method (no key present), $pl$ is 0.  
  - For PKOAEP2 encoded payloads, $pl$ is the length in bits of the modulus size of the RSA key used to wrap the payload. This can be 512 - 4096.  
  - For an AESKW formatted payload, $pl$ is based on the key size of the algorithm type and the payload format version:  
    - AES algorithm (value at offset 41 is X'02')  
      - An AES key can have a length of 16, 24, or 32 bytes (128, 192, or 256 bits). The following table shows the payload length for a given AES key size and payload format:

<table>
<thead>
<tr>
<th>AES key size</th>
<th>Bit length of V0 payload (value at offset 28 is X'00')</th>
<th>Bit length of V1 payload (value at offset 28 is X'01')</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bytes (128 bits)</td>
<td>Not applicable</td>
<td>640</td>
</tr>
<tr>
<td>24 bytes (192 bits)</td>
<td>Not applicable</td>
<td>640</td>
</tr>
<tr>
<td>32 bytes (256 bits)</td>
<td>Not applicable</td>
<td>640</td>
</tr>
<tr>
<td>040</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>
| 041           | 01             | Algorithm type (algorithm for which the key can be used):  
  **Value** | **Meaning** |
  X'02'   | AES |
  All unused values are reserved and undefined. |
| 042           | 02             | Key type (general class of the key):  
  **Value** | **Meaning** |
  X'0005' | PINPROT |
  X'0006' | PINCALC |
  X'0007' | PINPRW |
  All unused values are reserved and undefined. |
| 044           | 01             | Key usage fields count ($kuf$): 3. Key-usage field information defines restrictions on the use of the key.  
  For key type PINPROT, see AES PINPROT Key Token Build2 keywords [Figure 12 on page 293].  
  For key type PINCALC, see AES PINCALC Key Token Build2 keywords [Figure 11 on page 291].  
  For key type PINPRW, see AES PINPRW Key Token Build2 keywords [Figure 13 on page 296].  
  Each key-usage field is 2 bytes in length. The value in this field indicates how many 2-byte key usage fields follow. |
### Key token formats

Table 275. AES CIPHER variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>045</td>
<td>01</td>
<td>Key-usage field 1, high-order byte. The meaning is determined by the key type (value at offset 42). The key type can be PINPROT, PINCALC, or PINPRW:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>PINPROT (value at offset 42 is X'0005')</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Encryption operation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong>   <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'11xx xxxx'  Undefined or not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'10xx xxxx'  Key can be used for encryption; key cannot be used for decryption (ENCRYPT).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'01xx xxxx'  Key cannot be used for encryption; key can be used for decryption (DECRYPT).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'00xx xxxx'  Undefined or not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>PINCALC (value at offset 42 is X'0006')</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAC operation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong>   <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'1xxx xxxx'  Key can only be used for generate (GENONLY).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0xxx xxxx'  Undefined or not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>PINPRW (value at offset 42 is X'0007')</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAC operation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong>   <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'11xx xxxx'  Undefined or not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'10xx xxxx'  Key can be used for generate; key cannot be used for verify (GENONLY).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'01xx xxxx'  Key cannot be used for generate; key can be used for verify (VERIFY).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'00xx xxxx'  Undefined or not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
<tr>
<td>046</td>
<td>01</td>
<td>Key-usage field 1, low-order byte (user-defined extension control).</td>
</tr>
<tr>
<td>Offset (bytes)</td>
<td>Length (bytes)</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>047</td>
<td>01</td>
<td>Key-usage field 2, high-order byte. The meaning is determined by the key type (value at offset 42). The key type can be PINPROT, PINCALC, or PINPRW:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PINPROT (value at offset 42 is X'0005')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Encryption mode:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PINCALC (value at offset 42 is X'0006')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Encryption mode:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PINPRW (value at offset 42 is X'0007')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAC mode:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>048</td>
<td>01</td>
<td>Key-usage field 2, low-order byte:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>049</td>
<td>01</td>
<td><strong>Key-usage field 3, high-order byte.</strong> The meaning is determined by the field format identifier (value at offset 50). Currently the only field format identifier is DK enabled:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DK enabled (value at offset 50 is X'01')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Common control by key type, based on key type PINPROT, PINCALC, or PINPRW:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
</tbody>
</table>

| 050          | 01            | Key-usage field 3, low-order byte (field format identifier). Identifies the format of key-usage field 3, high-order byte (value at offset 49): |
|              |               | **Value** | **Meaning** |
|              |               | X'01'    | DK enabled (set when DKPINOP, DKPINOPP, or DKPINADI keyword is used) |
|              |               | All unused values are reserved and undefined. |
### Key Token Formats

Table 275. AES CIPHER variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>051</td>
<td>01</td>
<td>Key management fields count (kmf): 3. Key-management field information describes how the data is to be managed or helps with management of the key material. For key type PINPROT, see AES PINPROT Key Token Build2 keywords [Figure 12 on page 293]. For key type PINCALC, see AES PINCALC Key Token Build2 keywords [Figure 11 on page 291]. For key type PINPRW, see AES PINPRW Key Token Build2 keywords [Figure 13 on page 296]. Each key-management field is 2 bytes in length. The value in this field indicates how many 2-byte key management fields follow.</td>
</tr>
<tr>
<td>052</td>
<td>01</td>
<td>Key-management field 1, high-order byte (symmetric-key export control).</td>
</tr>
<tr>
<td>053</td>
<td>01</td>
<td>Key-management field 1, low-order byte (export control by algorithm).</td>
</tr>
<tr>
<td>054</td>
<td>01</td>
<td>Key-management field 2, high-order byte (key completeness).</td>
</tr>
<tr>
<td>055</td>
<td>01</td>
<td>Key-management field 2, low-order byte (security history).</td>
</tr>
<tr>
<td>056</td>
<td>01</td>
<td>Key-management field 3, high-order byte (pedigree original).</td>
</tr>
<tr>
<td>057</td>
<td>01</td>
<td>Key-management field 3, low-order byte (pedigree current).</td>
</tr>
<tr>
<td>058</td>
<td>kl</td>
<td>Optional key label.</td>
</tr>
<tr>
<td>058 + kl</td>
<td>iead</td>
<td>Optional IBM extended associated data (unused).</td>
</tr>
<tr>
<td>058 + kl + iead</td>
<td>uad</td>
<td>Optional user-defined associated data.</td>
</tr>
</tbody>
</table>

End of associated data section

Optional wrapped AESKW formatted payload or wrapped PKOAEP2 encoded payload (no payload if no key present)
Table 275. AES CIPHER variable-length symmetric key-token, version X’05’ (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>058 + kl + iead + uad</td>
<td>(pl + 7) / 8</td>
<td>Contents of payload (pl is in bits) depending on the encrypted section key-wrapping method (value at offset 26):</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value at offset 26</th>
<th>Encrypted section key-wrapping method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X’02’</td>
<td>AESKW</td>
<td>An encrypted payload which the Segment 2 code creates by wrapping the unencrypted AESKW formatted payload. The payload is made up of the integrity check value, pad length, length of hash options and hash, hash options, hash of the associated data, key material, and padding. The key token is internal.</td>
</tr>
<tr>
<td>X’03’</td>
<td>PKOAEP2</td>
<td>An encrypted PKOAEP2 encoded payload created using the RSAES-OAEP scheme of the PKCS #1 v2.1 standard. The message $M$ is encoded for a given hash algorithm using the Bellare and Rogaway Optimal Asymmetric Encryption Padding (OAEP) method for encoding messages. For PKOAEP2, $M$ is defined as follows:</td>
</tr>
</tbody>
</table>

$$M = [32 \text{ bytes: } hAD] \ || \ [2 \text{ bytes: bit length of the clear key}] \ || \ [\text{clear key}]$$

where $hAD$ is the message digest of the associated data, and is calculated using the SHA-256 algorithm starting at offset 30 for the length in bytes of all the associated data for the key token (length value at offset 32). The encoded message is wrapped with an RSA public-key according to the standard. The key token is external. |

End of optional wrapped AESKW formatted payload or wrapped PKOAEP2 encoded payload

End of AESKW or PKOAEP2 components

Note: All numbers are in big endian format.

AES DESUSECV variable-length symmetric key token

View a table showing the format of the DESUSECV variable-length symmetric key-token.

Table 276. AES DESUSECV variable-length symmetric key-token, version X’05’

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>01</td>
<td>Token identifier:</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td>X’02’</td>
<td>External key-token (encrypted payload is wrapped with a transport key or there is no payload). A transport key can be a key-encrypting key or an RSA public-key.</td>
<td></td>
</tr>
</tbody>
</table>

All unused values are reserved and undefined.

001 01 Reserved, binary zero.
## Key token formats

Table 276. AES DESUSECV variable-length symmetric key-token, version X'05' (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 002            | 02             | Length in bytes of the overall token structure:  
|                |                | 46 + (2 * kuf) + (2 * kmf) + kl + icad + uad + ((pl + 7) / 8)  
|                |                | Key token Minimum and maximum token length  
|                |                | External 46 + (2 * 1) + (2 * 1) + 0 + 11 + 0 + ((576 + 7) / 8) = 133  
| 004            | 01             | Token version number (identifies the format of this key token):  
|                |                | **Value** **Meaning**  
|                |                | X'05' Version 5 format of the key token (variable-length symmetric key-token)  
| 005            | 03             | Reserved, binary zero.  

End of header

Wrapping information section (all data related to wrapping the key)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 008            | 01             | Key material state:  
|                |                | **Value** **Meaning**  
|                |                | X'02' Key is wrapped with a transport key. The transport key is an AES key-encrypting key. The key token is external.  
|                |                | All unused values are reserved and undefined.  
| 009            | 01             | Key verification pattern (KVP) type:  
|                |                | **Value** **Meaning**  
|                |                | X'02' KEK (8 leftmost bytes of SHA-256 hash: X'01 || clear KEK). The key token is external.  
|                |                | All unused values are reserved and undefined.  
| 010            | 16             | KVP:  
|                |                | The field contains the KVP of the key-encrypting key used to wrap the key. The 8-byte KEK KVP is left-aligned in the field and padded on the right low-order bytes with binary zeros. The key token is external.  
| 026            | 01             | Encrypted section key-wrapping method (how data in the encrypted section is protected):  
|                |                | **Value** **Meaning**  
|                |                | X'02' AESKW (ANS X9.102). The key token is external with a key wrapped by an AES key-encrypting key.  
|                |                | All unused values are reserved and undefined.  
| 027            | 01             | Hash algorithm used for wrapping key. The value indicates the algorithm used to calculate the message digest of the associated data. The message digest is included in the wrapped payload and is calculated starting at offset 30 for the length in bytes of all the associated data for the key token (length value at offset 32).  
|                |                | **Value** **Meaning**  
|                |                | X'02' SHA-256  
|                |                | All unused values are reserved and undefined. The key token is external.
Table 276. AES DESUSECV variable-length symmetric key-token, version X’05’ (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>028</td>
<td>01</td>
<td>Payload format version (identifies format of the payload):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> X’01’ <strong>Meaning</strong> V1 payload (V1PYLD). The payload is fixed length based on the maximum possible key size of the algorithm for the key. The key is padded with random data to the size of the largest key for that algorithm. This helps to deter attacks on keys known to be weaker. The key length cannot be inferred by the size of the payload. The key token is external or internal. All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>029</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>End of wrapping information section</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>AESKW components:</strong> (1) associated data section and (2) optional wrapped AESKW payload</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Associated data section</strong></td>
</tr>
<tr>
<td>030</td>
<td>01</td>
<td>Associated data section version:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> X’01’ <strong>Meaning</strong> Version 1 format of associated data</td>
</tr>
<tr>
<td>031</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>032</td>
<td>02</td>
<td>Length in bytes of all the associated data for the key token: 31.</td>
</tr>
<tr>
<td>034</td>
<td>01</td>
<td>Length in bytes of the optional key label (kd): 0.</td>
</tr>
<tr>
<td>035</td>
<td>01</td>
<td>Length in bytes of the optional IBM extended associated data (iead): 11.</td>
</tr>
<tr>
<td>036</td>
<td>01</td>
<td>Length in bytes of the optional user-definable associated data (uad): 0.</td>
</tr>
<tr>
<td>037</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>038</td>
<td>02</td>
<td>Length in bits of the wrapped payload (pl): 576.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For an AESKW formatted payload, pl is based on the key size of the algorithm type and the payload format version:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DES algorithm (value at offset 41 is X’01’)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A DES key can have a length of 8, 16, or 24 bytes (64, 128, 192 bits). A DES key in an AESKW formatted payload is always wrapped with a V1 payload and has a fixed length payload of 576 bits.</td>
</tr>
<tr>
<td>040</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>041</td>
<td>01</td>
<td>Algorithm type (algorithm for which the key can be used):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> X’01’ <strong>Meaning</strong> DES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>042</td>
<td>02</td>
<td>Key type (general class of the key):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> X’0008’ <strong>Meaning</strong> DESUSECV</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>044</td>
<td>01</td>
<td>Key usage fields count (kuf): 1. Key-usage field information defines restrictions on the use of the key. Each key-usage field is 2 bytes in length. The value in this field indicates how many 2-byte key usage fields follow.</td>
</tr>
</tbody>
</table>
### Key token formats

Table 276. AES DESUSECV variable-length symmetric key-token, version X'05' (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>045</td>
<td>01</td>
<td>Key-usage field 1, high-order byte (reserved). All bits are reserved and must be zero.</td>
</tr>
<tr>
<td>046</td>
<td>01</td>
<td>Key-usage field 1, low-order byte (reserved). All bits are reserved and must be zero.</td>
</tr>
<tr>
<td>047</td>
<td>01</td>
<td>Key management fields count (kmf): 1. Key-management field information describes how the data is to be managed or helps with management of the key material. Each key-management field is 2 bytes in length. The value in this field indicates how many 2-byte key management fields follow.</td>
</tr>
<tr>
<td>048</td>
<td>01</td>
<td>Key-management field 1, high-order byte (reserved). All bits are reserved and must be zero.</td>
</tr>
<tr>
<td>049</td>
<td>01</td>
<td>Key-management field 1, low-order byte (reserved). All bits are reserved and must be zero.</td>
</tr>
<tr>
<td>050</td>
<td>iead</td>
<td>IBM extended associated data:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Item</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Structure version identifier</td>
<td>X'00'</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Flag byte 1</td>
<td>Copy of flag byte 1 taken from offset 6 of the internal fixed-length DES source key token exported by the CSNDSYX verb into this external enciphered key.</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Flag byte 2</td>
<td>Copy of flag byte 2 taken from offset 7 of the internal fixed-length DES source key token exported by the CSNDSYX verb into this external enciphered key.</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>Masked control vector</td>
<td>Copy of 8 bytes of control vector taken from offset 32 of the internal fixed-length DES source key token exported by the CSNDSYX verb into this external enciphered key. The key form bits (CV bits 40 - 42) are copied into the wrapped payload before being masked here to zero to conceal the length of the DES key.</td>
</tr>
</tbody>
</table>

End of associated data section

Optional wrapped AESKW formatted payload
## AES DES USECV variable-length symmetric key-token, version X'05' (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>061</td>
<td>72</td>
<td>Contents of payload: An encrypted payload which the Segment 2 code creates by wrapping the unencrypted AESKW formatted payload. The payload is made up of the integrity check value, pad length, length of hash options and hash, hash options, hash of the associated data, key material, and padding. The key token is internal. A DES DES USECV payload contains key material that is formatted. The key material is formatted as follows:</td>
</tr>
</tbody>
</table>

**Table**

### AES DKYGENKY variable-length symmetric key token

View a table showing the format of the DKYGENKY variable-length symmetric key-token.

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>01</td>
<td>Token identifier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'  Internal key-token (encrypted key is wrapped with the master key or there is no payload).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'  External key-token (encrypted payload is wrapped with a transport key or there is no payload). A transport key can be a key-encrypting key or an RSA public-key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>001</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>
### Key token formats

Table 277. AES DKYGENKY variable-length symmetric key-token, version X’05’ (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>002</td>
<td>02</td>
<td>Length in bytes of the overall token structure:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * kuf) + (2 * kmf) + kl + icad + uad + ((pl + 7) / 8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Key token</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Minimum token length</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Skeleton</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 3) + 0 + 0 + 0 = 56</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Encrypted V1 payload</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 2) + (2 * 3) + 0 + 0 + 0 + ((640 + 7) / 8) = 136</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Key token</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Maximum token length</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>External</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 6) + (2 * 3) + 64 + 0 + 255 + ((4096 + 7) / 8) = 895</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Internal</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46 + (2 * 6) + (2 * 3) + 64 + 0 + 255 + ((640 + 7) / 8) = 463</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>This assumes a PKOAEP2 key-wrapping method using a 4096-bit RSA transport key.</em></td>
</tr>
<tr>
<td>004</td>
<td>01</td>
<td>Token version number (identifies the format of this key token):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’05’ Version 5 format of the key token (variable-length symmetric key-token)</td>
</tr>
<tr>
<td>005</td>
<td>03</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

End of header

Wrapping information section (all data related to wrapping the key)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>008</td>
<td>01</td>
<td>Key material state:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’00’ No key is present. This is called a skeleton key-token. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’02’ Key is wrapped with a transport key. When the encrypted section key-wrapping method is AESKW (value at offset 26 is X’02’), the transport key is an AES key-encrypting key. When it is PKOAEP2 (value at offset 26 is X’03’), the transport key is an RSA public-key. The key token is external.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’03’ Key is wrapped with the AES master-key. The encrypted section key-wrapping method is AESKW. The key token is internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>009</td>
<td>01</td>
<td>Key verification pattern (KVP) type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’00’ No KVP (no key present or key is wrapped with an RSA public-key). The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’01’ AESMK (8 leftmost bytes of SHA-256 hash: X’01 ⊕ Y clear AES MK). The key token is internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’02’ KEK (8 leftmost bytes of SHA-256 hash: X’01 ⊕ Y clear KEK). The key token is external.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
</tbody>
</table>
Table 277. AES DKYGENKY variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>16</td>
<td>KVP (value depends on value of key material state, that is, the value at offset 8):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value at offset 8</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value of KVP</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00' The key-material state is no key present. The field should be filled with binary zeros. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02' The key material state is the key is wrapped with a transport key. The value of the KVP depends on the value of the encrypted section key-wrapping method:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• When the key-wrapping method is AESKW (value at offset 26 is X'02'), the field contains the KVP of the key-encrypting key used to wrap the key. The 8-byte KEK KVP is left-aligned in the field and padded on the right low-order bytes with binary zeros.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• When the key-wrapping method is PKOAEP2 (value at offset 26 is X'03'), the value should be filled with binary zeros. The encoded message, which contains the key, is wrapped with an RSA public-key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The key token is external.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03' The key-material state is the key is wrapped with the AES master-key. The field contains the MKVP of the AES master-key used to wrap the key. The 8-byte MKVP is left-aligned in the field and padded on the right low-order bytes with binary zeros. The key token is internal.</td>
</tr>
<tr>
<td>026</td>
<td>01</td>
<td>Encrypted section key-wrapping method (how data in the encrypted section is protected):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00' No key-wrapping method (no key present or key is clear). The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02' AESKW (ANS X9.102). The key token is external with a key wrapped by an AES key-encrypting key, or the key token is internal with a key wrapped by the AES master-key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03' PKOAEP2. Message M, which contains the key, is encoded using the RSAES-OAEP scheme of the RSA PKCS #1 v2.1 standard. The encoded message (EM) is produced using the given hash algorithm by encoding message M using the Bellare and Rogaway Optimal Asymmetric Encryption Padding (OAEP) method for encoding messages. For PKOAEP2, M is defined as follows:</td>
</tr>
</tbody>
</table>
|               |                | \[ M = [32 bytes: hAD] □□\[2 bytes: bit length of the clear key] □□[clear key] \]
|               |                | where hAD is the message digest of the associated data, and is calculated using the SHA-256 algorithm on the data starting at offset 30 for the length in bytes of all the associated data for the key token (length value at offset 32). |
|               |                | EM is wrapped with an RSA public-key. The key token is external. |

All unused values are reserved and undefined.
### Key token formats

Table 277. AES DKYGENKY variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>027</td>
<td>01</td>
<td>Hash algorithm used for wrapping key or encoding message. Meaning depends on whether the encrypted section key-wrapping method (value at offset 26) is no key-wrapping method, AESKW, or PKOAEP2:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No key-wrapping method (value at offset 26 is X'00')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hash algorithm used for wrapping key when encrypted section key-wrapping method is no key-wrapping method:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td>X'00'</td>
<td>No hash (no key present)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td>X'02'</td>
<td>SHA-256</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AESKW key-wrapping method (value at offset 26 is X'02')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hash algorithm used for wrapping key when encrypted section key-wrapping method is AESKW. The value indicates the algorithm used to calculate the message digest of the associated data. The message digest is included in the wrapped payload and is calculated starting at offset 30 for the length in bytes of all the associated data for the key token (length value at offset 32).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td>X'02'</td>
<td>SHA-256</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined. The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PKOAEP2 key-wrapping method (value at offset 26 is X'03')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hash algorithm used for encoding message when encrypted section key-wrapping method is PKOAEP2. The value indicates the given hash algorithm used for encoding message M using the RSAES-OAEP scheme of the RSA PKCS #1 v2.1 standard.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td>X'01'</td>
<td>SHA-1</td>
</tr>
<tr>
<td></td>
<td>X'02'</td>
<td>SHA-256</td>
</tr>
<tr>
<td></td>
<td>X'04'</td>
<td>SHA-384</td>
</tr>
<tr>
<td></td>
<td>X'08'</td>
<td>SHA-512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined. The key token is external.</td>
</tr>
</tbody>
</table>
### Table 277. AES DKYGENKY variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 028            | 01             | Payload format version (identifies format of the payload):  
|                |                | **Value** | **Meaning** |  
|                |                | X'01'     | V1 payload (V1PYLD). The payload format depends on the encrypted section key-wrapping method (value at offset 26):  
|                |                |           | **Value at offset 26** |  
|                |                |           | **Meaning** |  
|                |                | X'00'     | There is no key-wrapping method. When no key is present, there is no payload. The key token is external or internal. |  
|                |                | X'02'     | The key-wrapping method is AESKW and the payload is fixed length based on the maximum possible key size of the algorithm for the key. The key is padded with random data to the size of the largest key for that algorithm. This helps to deter attacks on keys known to be weaker. The key length cannot be inferred by the size of the payload. The key token is external or internal. |  
|                |                | X'03'     | The key-wrapping method is PKOAEP2 and the payload length is equal to the modulus size in bits of the RSA transport key used to wrap the encoded message. The key token is external. When the external key is exported, the internal target key will have the same V1 payload format. |  
|                |                |           | All unused values are reserved and undefined. |  
| 029            | 01             | Reserved, binary zero. |  

End of wrapping information section

AESKW or PKOAEP2 components: (1) associated data section and (2) optional wrapped AESKW payload or wrapped PKOAEP2 payload (no payload if no key present)

### Associated data section

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 030            | 01             | Associated data section version:  
|                |                | **Value** | **Meaning** |  
|                |                | X'01'     | Version 1 format of associated data |  
| 031            | 01             | Reserved, binary zero. |  
| 032            | 02             | Length in bytes of all the associated data for the key token: 26 - 353. |  
| 034            | 01             | Length in bytes of the optional key label (kl): 0 or 64. |  
| 035            | 01             | Length in bytes of the optional IBM extended associated data (iead): 0. |  
| 036            | 01             | Length in bytes of the optional user-definable associated data (uad): 0 - 255. |  
| 037            | 01             | Reserved, binary zero. |  

Chapter 19. Key token formats 955
### Key token formats

Table 277. AES DKEYGENKY variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>038</td>
<td>02</td>
<td>Length in <em>bits</em> of the wrapped payload (<em>pl</em>): 0, 512 - 4096.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For no key-wrapping method (no key present), <em>pl</em> is 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For PKOAEF2 encoded payloads, <em>pl</em> is the length in bits of the modulus size of the RSA key used to wrap the payload. This can be 512 - 4096.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For an AESKW formatted payload, <em>pl</em> is based on the key size of the algorithm type and the payload format version:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• AES algorithm (value at offset 41 is X'02')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An AES key can have a length of 16, 24, or 32 bytes (128, 192, or 256 bits). The following table shows the payload length for a given AES key size and payload format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><img src="#" alt="Bit length of V0 payload (value at offset 28 is X'00')" /></td>
</tr>
<tr>
<td>AES key size</td>
<td></td>
<td>16 bytes (128 bits)</td>
</tr>
<tr>
<td>040</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>041</td>
<td>01</td>
<td>Algorithm type (algorithm for which the key can be used):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>042</td>
<td>02</td>
<td>Key type (general class of the key):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0009'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>044</td>
<td>01</td>
<td>Key usage fields count (<em>kuf</em>): 2, 4 - 6. Key-usage field information defines restrictions on the use of the key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Count is based on type of key to diversify (value at offset 45):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value at offset 45</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'04'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'05'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'06'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'07'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Each key-usage field is 2 bytes in length. The value in this field indicates how many 2-byte key usage fields follow.</td>
</tr>
</tbody>
</table>
Table 277. AES DKYGENKY variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>045</td>
<td>01</td>
<td>Key-usage field 1, high-order byte (type of key to diversify). Defines the type of diversified key that this diversifying key can generate.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'04'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'05'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'06'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'07'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>046</td>
<td>01</td>
<td>Key-usage field 1, low-order byte (user-defined extension control).</td>
</tr>
<tr>
<td>047</td>
<td>01</td>
<td>Key-usage field 2, high-order byte (related generated key-usage field level of control):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'1xxx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0xxx xxxx'</td>
</tr>
<tr>
<td>048</td>
<td>01</td>
<td>Key-usage field 2, low-order byte (key-derivation sequence level):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
</tbody>
</table>
### Key token formats

Table 277. AES DKYGENKY variable-length symmetric key-token, version X’05’ (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>049, for kuf &gt; 3</td>
<td>02</td>
<td>Optional key-usage field 3 (related generated key usage fields). Controls the key usage field 1 values of the diversified key. Meaning depends on type of key to diversify (value at offset 45):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value at offset 45</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’01’ Same as key-usage field 1 of AES CIPHER key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’02’ Same as key-usage field 1 of AES MAC key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’03’ Same as key-usage field 1 of AES EXPORTER key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’04’ Same as key-usage field 1 of AES IMPORTER key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’05’ Same as key-usage field 1 of AES PINPROT key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’06’ Same as key-usage field 1 of AES PINCALC key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’07’ Same as key-usage field 1 of AES PINPRW key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
<tr>
<td>051, for kuf &gt; 3</td>
<td>02</td>
<td>Optional key-usage field 4 (related generated key usage fields). Controls the key usage field 2 values of the diversified key. Meaning depends on type of key to diversify (value at offset 45):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value at offset 45</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’01’ Same as key-usage field 2 of AES CIPHER key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’02’ Same as key-usage field 2 of AES MAC key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’03’ Same as key-usage field 2 of AES EXPORTER key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’04’ Same as key-usage field 2 of AES IMPORTER key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’05’ Same as key-usage field 2 of AES PINPROT key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’06’ Same as key-usage field 2 of AES PINCALC key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’07’ Same as key-usage field 2 of AES PINPRW key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
<tr>
<td>053, for kuf &gt; 4</td>
<td>02</td>
<td>Optional key-usage field 5 (related generated key usage fields). Controls the key usage field 3 values of the diversified key. Meaning depends on type of key to diversify (value at offset 45):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value at offset 45</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’02’ Same as key-usage field 3 of AES MAC key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’03’ Same as key-usage field 3 of AES EXPORTER key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’04’ Same as key-usage field 3 of AES IMPORTER key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’05’ Same as key-usage field 3 of AES PINPROT key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’06’ Same as key-usage field 3 of AES PINCALC key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’07’ Same as key-usage field 3 of AES PINPRW key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
<tr>
<td>055, for kuf &gt; 5</td>
<td>02</td>
<td>Optional key-usage field 6 (related generated key usage fields). Controls the key usage field 4 values of the diversified key. Meaning depends on type of key to diversify (value at offset 45):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value at offset 45</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’03’ Same as key-usage field 4 of AES EXPORTER key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’04’ Same as key-usage field 4 of AES IMPORTER key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
</tbody>
</table>
Table 277. AES DKYGENKY variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>045 + (2 * kuf)</td>
<td>01</td>
<td>Key management fields count (kmf): 3. Key-management field information describes how the data is to be managed or helps with management of the key material. Each key-management field is 2 bytes in length. The value in this field indicates how many 2-byte key management fields follow.</td>
</tr>
<tr>
<td>046 + (2 * kuf)</td>
<td>01</td>
<td>Key-management field 1, high-order byte (symmetric-key export control).</td>
</tr>
<tr>
<td>047 + (2 * kuf)</td>
<td>01</td>
<td>Key-management field 1, low-order byte (export control by algorithm).</td>
</tr>
<tr>
<td>048 + (2 * kuf)</td>
<td>01</td>
<td>Key-management field 2, high-order byte (key completeness).</td>
</tr>
<tr>
<td>049 + (2 * kuf)</td>
<td>01</td>
<td>Key-management field 2, low-order byte (security history).</td>
</tr>
<tr>
<td>050 + (2 * kuf)</td>
<td>01</td>
<td>Key-management field 3, high-order byte (pedigree original).</td>
</tr>
<tr>
<td>051 + (2 * kuf)</td>
<td>01</td>
<td>Key-management field 3, low-order byte (pedigree current).</td>
</tr>
<tr>
<td>052 + (2 * kuf)</td>
<td>kl</td>
<td>Optional key label.</td>
</tr>
<tr>
<td>052 + (2 * kuf) +</td>
<td>iead</td>
<td>Optional IBM extended associated data (not used).</td>
</tr>
<tr>
<td>kl</td>
<td></td>
<td>End of associated data section</td>
</tr>
<tr>
<td>052 + (2 * kuf) +</td>
<td>iead</td>
<td>Optional user-defined associated data.</td>
</tr>
</tbody>
</table>

Optional wrapped AESKW formatted payload or wrapped PKOAE2 encoded payload (no payload if no key present)
### Key token formats

Table 277. AES DKYGENKY variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>052 + (2 * $kuf$) + $kl + iead + uad$</td>
<td>$(pl + 7) / 8$</td>
<td>Contents of payload ($pl$ is in bits) depending on the encrypted section key-wrapping method (value at offset 26):</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value at offset 26</th>
<th>Encrypted section key-wrapping method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'02'</td>
<td>AESKW</td>
<td>An encrypted payload which the Segment 2 code creates by wrapping the unencrypted AESKW formatted payload. The payload is made up of the integrity check value, pad length, length of hash options and hash, hash options, hash of the associated data, key material, and padding. The key token is internal.</td>
</tr>
<tr>
<td>X'03'</td>
<td>PKOAEP2</td>
<td>An encrypted PKOAEP2 encoded payload created using the RSAES-OAEP scheme of the PKCS #1 v2.1 standard. The message $M$ is encoded for a given hash algorithm using the Bellare and Rogaway Optimal Asymmetric Encryption Padding (OAEP) method for encoding messages. For PKOAEP2, $M$ is defined as follows: $M = [32$ bytes: $hAD] \bowtie [2$ bytes: bit length of the clear key] $\bowtie$ [clear key] where $hAD$ is the message digest of the associated data, and is calculated using the SHA-256 algorithm starting at offset 30 for the length in bytes of all the associated data for the key token (length value at offset 32). The encoded message is wrapped with an RSA public-key according to the standard. The key token is external.</td>
</tr>
</tbody>
</table>

End of optional wrapped AESKW formatted payload or wrapped PKOAEP2 encoded payload

End of AESKW or PKOAEP2 components

**Note:** All numbers are in big endian format.

---

### AES SECMSG variable-length symmetric key token

View a table showing the format of the SECMSG variable-length symmetric key-token.

Table 278. AES SECMSG variable-length symmetric key-token, version X'05'.

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 278. AES SECMSG variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>01</td>
<td>Token identifier (note that it is intentional that an external key-token is not defined):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>002</td>
<td>02</td>
<td>Length in bytes of the overall token structure:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Key token</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Minimum token length</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Skeleton</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$46 + (2 \times 2) + (2 \times 3) + 0 + 0 + 0 = 56$</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Encrypted V1 payload</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$46 + (2 \times 2) + (2 \times 3) + 0 + 0 + 0 + ((640 + 7) \div 8) = 136$</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Key token</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Maximum token length</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Internal</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$46 + (2 \times 2) + (2 \times 3) + 64 + 0 + 255 + ((640 + 7) \div 8) = 455$</td>
</tr>
<tr>
<td>004</td>
<td>01</td>
<td>Token version number (identifies the format of this key token):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'05'</td>
</tr>
<tr>
<td>005</td>
<td>03</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

End of header

Wrapping information section (all data related to wrapping the key)

<table>
<thead>
<tr>
<th>008</th>
<th>01</th>
<th>Key material state:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'03'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>009</th>
<th>01</th>
<th>Key verification pattern (KVP) type:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Key token formats

**Table 278. AES SECMSG variable-length symmetric key-token, version X’05’ (continued).**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>16</td>
<td>KVP (value depends on value of key material state, that is, the value at offset 8):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value at offset 8</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value of KVP</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’00’ The key-material state is no key present. The field should be filled with binary zeros. The key token is internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’03’ The key-material state is the key is wrapped with the AES master-key. The field contains the MKVP of the AES master-key used to wrap the key. The 8-byte MKVP is left-aligned in the field and padded on the right low-order bytes with binary zeros. The key token is internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>026</td>
<td>01</td>
<td>Encrypted section key-wrapping method (how data in the encrypted section is protected):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’00’ No key-wrapping method (no key present). The key token is external or internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’02’ AESKW (ANS X9.102). The key token is internal with a key wrapped by the AES master-key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>027</td>
<td>01</td>
<td>Hash algorithm used for wrapping key or encoding message. Meaning depends on whether the encrypted section key-wrapping method (value at offset 26) is no key-wrapping method or AESKW:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’00’ No hash (no key present).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AESKW key-wrapping method (value at offset 26 is X’02’):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’02’ SHA-256</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined. The key token is internal.</td>
</tr>
</tbody>
</table>
Table 278. AES SECMSG variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>028</td>
<td>01</td>
<td>Payload format version (identifies format of the payload).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>029</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

End of wrapping information section

AESKW components: (1) associated data section and (2) optional wrapped AESKW formatted payload (no payload if no key present)

Associated data section

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>030</td>
<td>01</td>
<td>Associated data section version:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td>031</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>032</td>
<td>02</td>
<td>Length in bytes of all the associated data for the key token: 26 - 345.</td>
</tr>
<tr>
<td>034</td>
<td>01</td>
<td>Length in bytes of the optional key label (kl ) : 0 or 64 .</td>
</tr>
<tr>
<td>035</td>
<td>01</td>
<td>Length in bytes of the optional IBM extended associated data (iaed ) : 0 .</td>
</tr>
<tr>
<td>036</td>
<td>01</td>
<td>Length in bytes of the optional user-definable associated data: 0 - 255 (uad )</td>
</tr>
<tr>
<td>037</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>
### Key token formats

Table 278. AES SECMSG variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>038</td>
<td>02</td>
<td>Length in bits of the wrapped payload (( pl )): 0, 640. For no key-wrapping method (no key present), ( pl ) is 0. For an AESKW formatted payload, ( pl ) is based on the key size of the algorithm type and the payload format version:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• AES algorithm (value at offset 41 is X'02')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• An AES key can have a length of 16, 24, or 32 bytes (128, 192, or 256 bits). The payload length for a given AES key size and payload format are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AES key size</td>
</tr>
<tr>
<td>040</td>
<td>01</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>041</td>
<td>01</td>
<td>Algorithm type (algorithm for which the key can be used):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>042</td>
<td>02</td>
<td>Key type (general class of the key):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'000A'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>044</td>
<td>01</td>
<td>Key usage fields count (( kuf )): 2. Key-usage field information defines restrictions on the use of the key. Each key-usage field is 2 bytes in length. The value in this field indicates how many 2-byte key usage fields follow.</td>
</tr>
<tr>
<td>045</td>
<td>01</td>
<td>Key-usage field 1, high-order byte (secure message encryption enablement):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
<tr>
<td>046</td>
<td>01</td>
<td>Key-usage field 1, low-order byte (user-defined extension control).</td>
</tr>
<tr>
<td>047</td>
<td>01</td>
<td>Key-usage field 2, high-order byte (verb restriction):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused values are reserved and undefined.</td>
</tr>
</tbody>
</table>
### Table 278. AES SECMMSG variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>048</td>
<td>01</td>
<td>Key-usage field 2, low-order byte (reserved). All bits are reserved and must be zero.</td>
</tr>
<tr>
<td>049</td>
<td>01</td>
<td>Key management fields count (<em>kmf</em>): 3. Key-management field information describes how the data is to be managed or helps with management of the key material. Each key-management field is 2 bytes in length. The value in this field indicates how many 2-byte key management fields follow.</td>
</tr>
</tbody>
</table>
| 050            | 01             | Key-management field 1, high-order byte (symmetric-key export control). Note that an AES SECMMSG key cannot be exported because it has no external key-token defined. Symmetric-key export control:  
  | Value | Meaning |  
  |       | B'1xxx xxxx' | Undefined.  
  |       | B'0xxx xxxx' | Prohibit export using symmetric key (NOEX-SYM).  
  Unauthenticated asymmetric-key export control:  
  | Value | Meaning |  
  |       | B'x1xx xxxx' | Undefined.  
  |       | B'x0xx xxxx' | Prohibit export using unauthenticated asymmetric key (NOEXUASY).  
  Authenticated asymmetric-key export control:  
  | Value | Meaning |  
  |       | B'xx1x xxxx' | Undefined.  
  |       | B'xx0x xxxx' | Prohibit export using authenticated asymmetric key (NOEXAASY).  
  Raw-key export control:  
  | Value | Meaning |  
  |       | B'xxx1 xxxx' | Undefined.  
  |       | B'xxx0 xxxx' | Prohibit export using raw key (NOEX-RAW). Defined for future use. Currently ignored.  
All unused bits are reserved and must be zero.
### Key token formats

Table 278. AES SECMSG variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>051</td>
<td>01</td>
<td>Key-management field 1, low-order byte (export control by algorithm). Note that an AES SECMSG key can only be internal; it cannot be external. DES-key export control: Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'1xxx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0xxx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AES-key export control: Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'x1xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'x0xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA-key export control: Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx 1xxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx 0xxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All unused bits are reserved and must be zero.</td>
</tr>
<tr>
<td>052</td>
<td>01</td>
<td>Key-management field 2, high-order byte (key completeness).</td>
</tr>
<tr>
<td>053</td>
<td>01</td>
<td>Key-management field 2, low-order byte (security history).</td>
</tr>
<tr>
<td>054</td>
<td>01</td>
<td>Key-management field 2, low-order byte (security history).</td>
</tr>
<tr>
<td>055</td>
<td>01</td>
<td>Key-management field 3, low-order byte (pedigree current).</td>
</tr>
<tr>
<td>056</td>
<td>01</td>
<td>Optional key label.</td>
</tr>
<tr>
<td>056 + kl</td>
<td>iead</td>
<td>Optional IBM extended associated data (unused).</td>
</tr>
<tr>
<td>056 + kl + iead</td>
<td>uad</td>
<td>Optional user-defined associated data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>End of associated data section</td>
</tr>
</tbody>
</table>

Optional wrapped AESKW formatted payload, or wrapped PKOAEP2 encoded payload (no payload if no key present)
Table 278. AES SECMSG variable-length symmetric key-token, version X'05' (continued).

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>056 + kl + iead + uad</td>
<td>(pl + 7) / 8</td>
<td>Contents of payload (pl is in bits) depending on the encrypted section key-wrapping method (value at offset 26):</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value at offset 26</th>
<th>Encrypted section key-wrapping method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'02'</td>
<td>AESKW</td>
<td>An encrypted payload which the Segment 2 code creates by wrapping the unencrypted AESKW formatted payload. The payload is made up of the integrity check value, pad length, length of hash options and hash, hash options, hash of the associated data, key material, and padding. The key token is internal.</td>
</tr>
</tbody>
</table>

End of optional wrapped AESKW formatted payload

End of AESKW components

Note: All numbers are in big endian format.

TR-31 optional block data

A TR-31 key block can contain an optional block with IBM-defined data.

See X9 TR-31 2010: Interoperable Secure Key Exchange Block Specification for Symmetric Algorithms for the definition of a TR-31 key block. As defined by X9 TR-31, a TR-31 key block can contain one or more optional blocks. A TR-31 key block contains at least one optional block when byte number 12 - 13 is a value other than ASCII string "00".

The data of an IBM-defined optional block contains ASCII string 10 (X'3130') in the first two bytes, and contains ASCII string IBMC (X'49424D43') beginning at offset 4 of the data. CCA treats an optional block with these characteristics as a proprietary container for a CCA control vector. See Table 279 on page 968. An optional block with different characteristics is ignored by CCA.

If a TR-31 key block contains an optional block as defined by Table 279 on page 968, the data contains a copy of the 8-byte or 16-byte DES control vector that was in the CCA key-token of the key being exported. The copied control vector is in hex-ASCII format.

The control vector is only copied from the CCA key-token when the user of the Key Export to TR31 verb specifies a control vector transport control keyword (INCL-CV or ATTR-CV):

1. If the optional block contains a control vector as the result of specifying the INCL-CV keyword during export, the key usage and mode of use fields indicate the key attributes, and these attributes are verified during export to be compatible with the ones in the included control vector.
2. If the optional block contains a control vector as the result of specifying the ATTR-CV keyword during export, the key usage field (byte number 5 - 6 of the TR-31 key block) is set to the proprietary value 10 (X'3130'), and the mode of use field (byte number 8) is set to the proprietary value 1 (X'31'). These proprietary values indicate that the key attributes are specified in the included control vector.

See Chapter 5, “TR-31 symmetric key management,” on page 79 for additional information on how CCA uses an IBM-defined optional block in a TR-31 key block.

Table 279. IBM optional block data in a TR-31 key block

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>02</td>
<td>Proprietary ID of TR-31 optional block (alphanumeric-ASCII): X'3130' IBM proprietary optional block (ASCII string 10)</td>
</tr>
<tr>
<td>02</td>
<td>02</td>
<td>Length of optional block (hex-ASCII): For TLV valued to 01: X'3143' &quot;1C&quot; for 8-byte (single-length) control vector X'3243' &quot;2C&quot; for 16-byte (double-length) control vector</td>
</tr>
<tr>
<td>04</td>
<td>04</td>
<td>Magic value (alphanumeric-ASCII): X'49424D43' A constant value (&quot;IBMC&quot;) used to reduce ambiguity and the chance for false interpretation of the proprietary optional blocks of non-IBM vendors. An optional block that uses the same proprietary ID but does not include this magic value will be ignored.</td>
</tr>
<tr>
<td>08</td>
<td>02</td>
<td>Tag-length-value (TLV) ID (numeric-ASCII): X'3031' IBM CCA control vector (01)</td>
</tr>
<tr>
<td>10</td>
<td>02</td>
<td>Length of TLV (hex-ASCII): For TLV valued to 01: X'3134' &quot;14&quot; (decimal 20) TLV of 8-byte control vector X'3234' 24 (decimal 36) TLV of 16-byte control vector</td>
</tr>
<tr>
<td>12</td>
<td>16 or 32</td>
<td>Control vector (hex-ASCII)</td>
</tr>
</tbody>
</table>

Trusted blocks

A key token is a data structure that contains information about a key and usually contains a key or keys.

A trusted block is an extension of CCA key tokens using new section identifiers. Trusted blocks are an integral part of a remote key-loading process. See “Remote key loading” on page 53.

In general, a key that is available to an application program or held in key storage is multiply-enciphered by some other key. When a key is enciphered by the CCA node’s master key, the key is designated an internal key and is held in an internal key-token structure. Therefore, an internal key token or internal trusted block is used to hold a key and its related information for use at a specific CCA node.

An external key token or external trusted block is used to communicate a key between nodes, or to hold a key in a form not enciphered by a CCA master key. DES keys
and PKA private-keys contained in an external key-token or external trusted block are multiply-enciphered by a transport key. In a CCA node, a transport key is a double-length DES key encrypting key (KEK).

Trusted blocks contain various items, some of which are optional, and some of which can be present in different forms. Tokens are composed of concatenated sections that, unlike CCA PKA key tokens, occur in no prescribed order.

As with other CCA key-tokens, both internal and external forms are defined:

- An external trusted block contains a randomly generated confounder and a triple-length MAC key enciphered under a DES IMP-PKA transport key. The MAC key is used to calculate an ISO 16609 CBC mode TDES MAC of the trusted block contents. An external trusted block is created by the Trusted Block Create verb. This verb can:
  1. Create an inactive external trusted block
  2. Change an external trusted block from inactive to active

- An internal trusted block contains a confounder and triple-length MAC key enciphered under a variant of the PKA master key. The MAC key is used to calculate a TDES MAC of the trusted block contents. A PKA master-key verification pattern is also included to enable determination that the proper master key is available to process the key. The Remote Key Export verb only operates on trusted blocks that are internal. An internal trusted block must be imported from an external trusted block that is active using the PKA Key Import verb.

Note: Trusted blocks do not contain a private key section.

**Trusted block organization**

A trusted block is a concatenation of a header followed by an unordered set of sections.

Some elements are required, while others are optional. The data structures of these sections are summarized in [Table 280](#).

**Table 280. Trusted block sections and their use**

<table>
<thead>
<tr>
<th>Section</th>
<th>Reference</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Table 281 on page 971</td>
<td>Trusted block token header</td>
</tr>
<tr>
<td>X'11'</td>
<td>Table 282 on page 972</td>
<td>Trusted block public key</td>
</tr>
<tr>
<td>X'12'</td>
<td>Table 283 on page 973</td>
<td>Trusted block rule</td>
</tr>
<tr>
<td>X'13'</td>
<td>Table 290 on page 980</td>
<td>Trusted block name (key label)</td>
</tr>
<tr>
<td>X'14'</td>
<td>Table 291 on page 980</td>
<td>Trusted block information</td>
</tr>
<tr>
<td>X'15'</td>
<td>Table 295 on page 982</td>
<td>Trusted block application-defined data</td>
</tr>
</tbody>
</table>

Every trusted block starts with a token header. The first byte of the token header determines the key form:

- An external header (first byte X'1E'), created by the Trusted Block Create verb
- An internal header (first byte X'1F'), imported from an active external trusted block by the PKA Key Import verb

Following the token header of a trusted block is an unordered set of sections. A trusted block is formed by concatenating these sections to a trusted block header:
Key token formats

- An optional public-key section (trusted block section identifier X'11')
  The trusted block trusted RSA public key section includes the key itself in addition to a key-usage flag. No multiple sections are allowed.
- An optional rule section (trusted block section identifier X'12')
  A trusted block can have zero or more rule sections.
  1. A trusted block with no rule sections can be used by the PKA Key Token Change and PKA Key Import verbs. A trusted block with no rule sections can also be used by the Digital Signature Verify verb, provided there is an RSA public key section that has its key-usage flag bits set to allow digital signature operations.
  2. At least one rule section is required when the Remote Key Export verb is used to:
     - Generate an RKX key-token
     - Export an RKX key-token
     - Export a CCA DES key-token
     - Encrypt the clear generated or exported key using the provided vendor certificate
  3. If a trusted block has multiple rule sections, each rule section must have a unique 8-character Rule ID.
- An optional name (key label) section (trusted block section identifier X'13')
  The trusted block name section provides a 64-byte variable to identify the trusted block, just as key labels are used to identify other CCA keys. This name, or label, enables a host access-control system such as RACF® to use the name to verify that the application has authority to use the trusted block. No multiple sections are allowed.
- A required information section (trusted block section identifier X'14')
  The trusted block information section contains control and security information related to the trusted block. The information section is required while the others are optional. This section contains the cryptographic information that guarantees its integrity and binds it to the local system. No multiple sections are allowed.
- An optional application-defined data section (trusted block section identifier X'15')
  The trusted block application-defined data section can be used to include application-defined data in the trusted block. The purpose of the data in this section is defined by the application. CCA does not examine or use this data in any way. No multiple sections are allowed.

Trusted block integrity

An enciphered confounder and triple-length MAC key contained within the required information section of the trusted block is used to protect the integrity of the trusted block.

The randomly generated MAC key is used to calculate an ISO 16609 CBC mode TDES MAC of the trusted block contents. Together, the MAC key and MAC value provide a way to verify that the trusted block originated from an authorized source, and binds it to the local system.

An external trusted block has its MAC key enciphered under an IMP-PKA key-encrypting key. An internal trusted block has its MAC key enciphered under a variant of the PKA master key, and the master-key verification pattern is stored in the information section.
Number representation in trusted blocks

The number format in trusted blocks.
- All length fields are in binary.
- All binary fields (exponents, lengths, and so forth) are stored with the high-order byte first (big-endian format). Thus the least significant bits are to the right and preceded with zero-bits to the width of a field.
- In variable-length binary fields, that have an associated field-length value, leading bytes that would otherwise contain X'00' can be dropped. Thus, these fields can be shortened to contain only the significant bits.

Trusted block sections

At the beginning of every trusted block is a trusted block header.

The header contains the following information:
- A token identifier, which specifies if the token contains an external or internal key-token
- A token version number to allow for future changes
- A length in bytes of the trusted block, including the length of the header

The trusted block header is defined in Table 281.

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Token identifier (a flag that indicates token type)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'1E'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'1F'</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Token version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the key-token structure in bytes.</td>
</tr>
<tr>
<td>004</td>
<td>004</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

Following the header, in no particular order, are trusted block sections. There are five different sections defined, each identified by a one-byte section identifier (X'11' - X'15'). Two of the five sections have subsections defined. A subsection is a tag-length-value (TLV) object, identified by a two-byte subsection tag.

Only sections X'12' and X'14' have subsections defined; the other sections do not. A section and its subsections, if any, are one contiguous unit of data. The subsections are concatenated to the related section, but are otherwise in no particular order.

Section X'12' has five subsections defined (X'0001' - X'0005'). Section X'14' has two subsections, (X'0001' and X'0002'). Of all the subsections, only subsection X'0001' of section X'14' is required. Section X'14' is also required.

The trusted block sections and subsections are described in detail in the following topics.
Key token formats

**Trusted block section X'11'**
Trusted block section X'11' contains the trusted RSA public key in addition to a key-usage flag indicating whether the public key is usable in key-management operations, digital signature operations, or both.

Section X'11' is optional. No multiple sections are allowed. It has no subsections defined.

Table 282. Trusted block trusted RSA public key section (X'11')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Section identifier: Trusted block trusted RSA public key</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length (16 + xxx + yyy).</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Reserved, must be binary zero.</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>RSA public key exponent field length in bytes, xxx.</td>
</tr>
<tr>
<td>008</td>
<td>002</td>
<td>RSA public key modulus length in bits.</td>
</tr>
<tr>
<td>010</td>
<td>002</td>
<td>RSA public key modulus field length in bytes, yyy.</td>
</tr>
<tr>
<td>012</td>
<td>xxx</td>
<td>Public key exponent, $e$ (this field length is typically 1, 3, or 64 - 512 bytes). $e$ must be odd and $1 \leq e &lt; n$. ($e$ is frequently valued to 3 or $2^{16}+1 = 65537$, otherwise $e$ is of the same order of magnitude as the modulus).</td>
</tr>
<tr>
<td>012 + xxx</td>
<td>yyy</td>
<td>RSA public key modulus, $n$. $n=pq$, where $p$ and $q$ are prime and $2^{512} \leq n &lt; 2^{406}$. The field length is 64 - 512 bytes.</td>
</tr>
<tr>
<td>012 + xxx + yyy</td>
<td>004</td>
<td>Flags:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000000'</td>
<td>Trusted block public key can be used in digital signature operations only</td>
</tr>
<tr>
<td>X'80000000'</td>
<td>Trusted block public key can be used in both digital signature and key management operations</td>
</tr>
<tr>
<td>X'C0000000'</td>
<td>Trusted block public key can be used in key management operations only</td>
</tr>
</tbody>
</table>

**Trusted block section X'12'**
Trusted block section X'12' contains information that defines a rule.

TA trusted block can have zero or more rule sections.

1. A trusted block with no rule sections can be used by the PKA Key Token Change and PKA Key Import verbs. A trusted block with no rule sections can be used by the Digital Signature Verify verb, provided there is an RSA public key section that has its key-usage flag set to allow digital signature operations.

2. At least one rule section is required when the Remote Key Export verb is used to:
   - Generate an RKX key-token
   - Export an RKX key-token
   - Export a CCA DES key-token
   - Generate or export a key encrypted by a public key. The public key is contained in a vendor certificate and is the root certification key for the ATM vendor. It is used to verify the digital signature on public-key certificates for specific individual ATMs.
3. If a trusted block has multiple rule sections, each rule section must have a unique 8-character Rule ID.

Section X’12’ is the only section that can have multiple sections. Section X’12’ is optional.

**Note:** The overall length of the trusted block cannot exceed its maximum size of 3500 bytes.

Five subsections (TLV objects) are defined.

---

**Table 283. Trusted block rule section (X’12’)**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Section identifier: X’12’ Trusted block rule</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X’00’).</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length in bytes (20 + yyyy).</td>
</tr>
<tr>
<td>004</td>
<td>008</td>
<td>Rule ID (in ASCII). An 8-byte character string that uniquely identifies the rule within the trusted block. Valid ASCII characters are: A - Z, a - z, 0 - 9, - (hyphen), and _ (underscore), left-aligned and padded on the right with space characters.</td>
</tr>
<tr>
<td>012</td>
<td>004</td>
<td>Flags (undefined flag bits are reserved and must be zero). Value Description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’00000000’ Generate new key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’00000001’ Export existing key</td>
</tr>
<tr>
<td>016</td>
<td>001</td>
<td>Generated key length. Length in bytes of key to be generated when flags value (offset 012) is set to generate a new key; otherwise ignore this value. Valid values are 8, 16, or 24; return an error if not valid.</td>
</tr>
<tr>
<td>017</td>
<td>001</td>
<td>Key-check algorithm identifier (all others are reserved and must not be used): Value Description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’00’ Do not compute key-check value. Set the key_check_value_length variable to zero.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’01’ Encrypt an 8-byte block of binary zeros with the key. See “Encrypt zeros” on page 1024.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’02’ Compute the MDC-2 hash of the key. See “Modification Detection Code calculation” on page 1024.</td>
</tr>
</tbody>
</table>
Key token formats

Table 283. Trusted block rule section (X'12') (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>018</td>
<td>001</td>
<td>Symmetric encrypted output key format flag (all other values are reserved and must not be used). Return the indicated symmetric key-token using the <code>sym_encrypted_key_identifier</code> parameter.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td>Return an RKX key-token encrypted under a variant of the MAC key. <strong>Note:</strong> This key format is permitted when the flags value (offset 012) is set to either: 1. Generate a new key 2. Export an existing key</td>
</tr>
<tr>
<td>X'01'</td>
<td>Return a CCA DES key-token encrypted under a transport key. <strong>Note:</strong> This key format is not permitted if the flags value (offset 012) is set to generate a new key; it is only permitted when exporting an existing key.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>019</td>
<td>001</td>
<td>Asymmetric encrypted output key format flag (all other values are reserved and must not be used). Return the indicated asymmetric key-token in the <code>asym_encrypted_key</code> variable.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td>Do not return an asymmetric key. Set the <code>asym_encrypted_key_length</code> variable to zero.</td>
</tr>
<tr>
<td>X'01'</td>
<td>Output in PKCS-1.2 format.</td>
</tr>
<tr>
<td>X'02'</td>
<td>Output in RSA-OAEP format.</td>
</tr>
<tr>
<td>X'04'</td>
<td>Output in RSA-OAEP format (RSA PKCS #1 v2.0) using SHA-256.</td>
</tr>
</tbody>
</table>

| YYYY | Rule section subsections (tag-length-value objects). A series of zero - five objects in TLV format. |

Trusted block section X'12' subsections:

Section X'12' has five rule subsections (tag-length-value objects) defined.

These subsections are summarized in Table 284.

Table 284. Summary of trusted block X'12' subsections

<table>
<thead>
<tr>
<th>Rule subsection tag</th>
<th>TLV object</th>
<th>Optional or required</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0001'</td>
<td>Transport key variant</td>
<td>Optional</td>
<td>Contains variant to be XORed into the cleartext transport key.</td>
</tr>
<tr>
<td>X'0002'</td>
<td>Transport key rule reference</td>
<td>Optional; required to use an RKX key-token as a transport key</td>
<td>Contains the rule ID for the rule that must have been used to create the transport key.</td>
</tr>
<tr>
<td>X'0003'</td>
<td>Common export key parameters</td>
<td>Optional for key generation; required for key export of an existing key</td>
<td>Contains the export key and source key minimum and maximum lengths, an output key variant length and variant, a CV length, and a CV to be XORed with the cleartext transport key to control usage of the key.</td>
</tr>
<tr>
<td>X'0004'</td>
<td>Source key reference</td>
<td>Optional; required if the source key is an RKX key-token</td>
<td>Contains the rule ID for the rule used to create the source key. <strong>Note:</strong> Include all rules that will ever be needed when a trusted block is created. A rule cannot be added to a trusted block after it has been created.</td>
</tr>
</tbody>
</table>
Table 284. Summary of trusted block X’12’ subsections (continued)

<table>
<thead>
<tr>
<th>Rule subsection tag</th>
<th>TLV object</th>
<th>Optional or required</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>X’0005’</td>
<td>Export key CCA token parameters</td>
<td>Optional; used for export of CCA DES key tokens only</td>
<td>Contains mask length, mask, and CV template to limit the usage of the exported key. Also contains the template length and template that defines which source key labels are allowed. The key type of a source key input parameter can be &quot;filtered&quot; by using the export key CV limit mask (offset 005) and limit template (offset 005 + yyyy) in this subsection.</td>
</tr>
</tbody>
</table>

Note: See “Number representation in trusted blocks” on page 971.

Trusted block section X’12’ subsection X’0001’

Subsection X’0001’ of the trusted block rule section (X’12’) is the transport key variant TLV object. This subsection is optional. It contains a variant to be XORed into the cleartext transport key.

This subsection is defined in Table 285.

Table 285. Transport key variant subsection (X’0001’) of trusted block rule section (X’12’)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Subsection tag: X’0001’ Transport key variant TLV object</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Subsection length in bytes (8 + mnn).</td>
</tr>
<tr>
<td>004</td>
<td>001</td>
<td>Subsection version number (X’00’).</td>
</tr>
<tr>
<td>005</td>
<td>002</td>
<td>Reserved, must be binary zero.</td>
</tr>
<tr>
<td>007</td>
<td>001</td>
<td>Length of variant field in bytes (mnn).</td>
</tr>
<tr>
<td></td>
<td>mnn</td>
<td>Transport key variant.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XOR this variant into the cleartext transport key, provided: (1) the length of the variant field value (offset 007) is not zero, and (2) the symmetric encrypted output key format flag (offset 018 in section X’12’) is X’01’.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: A transport key is not used when the symmetric encrypted output key is in RKX key-token format.</td>
</tr>
</tbody>
</table>

Note: See “Number representation in trusted blocks” on page 971.

Trusted block section X’12’ subsection X’0002’

Subsection X’0002’ of the trusted block rule section (X’12’) is the transport key rule reference TLV object. This subsection is optional. It contains the rule ID for the rule that must have been used to create the transport key. This subsection must be present to use an RKX key-token as a transport key.

This subsection is defined in Table 286 on page 976.
## Key token formats

### Table 286. Transport key rule reference subsection (X'0002') of trusted block rule section (X'12')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Subsection tag: X'0002' Transport key rule reference TLV object</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Subsection length in bytes (14).</td>
</tr>
<tr>
<td>004</td>
<td>001</td>
<td>Subsection version number (X'00').</td>
</tr>
<tr>
<td>005</td>
<td>001</td>
<td>Reserved, must be binary zero.</td>
</tr>
<tr>
<td>006</td>
<td>008</td>
<td>Rule ID. Contains the rule identifier for the rule that must have been used to create the RKX key-token used as the transport key. The Rule ID is an 8-byte string of ASCII characters, left-aligned and padded on the right with space characters. Acceptable characters are A - Z, a - z, 0 - 9, - (X'2D'), and _ (X'5F'). All other characters are reserved for future use.</td>
</tr>
</tbody>
</table>

**Note:** See “Number representation in trusted blocks” on page 971.

### Trusted block section X'12' subsection X'0003'

Subsection X'0003' of the trusted block rule section X'12') is the common export key parameters TLV object. This subsection is optional, but is required for the key export of an existing source key (identified by the source_key_identifier parameter) in either RKX key-token format or CCA DES key-token format. For new key generation, this subsection applies the output key variant to the cleartext generated key, if such an option is desired. It contains the input source key and output export key minimum and maximum lengths, an output key variant length and variant, a CV length, and a CV to be XORed with the cleartext transport key.

This subsection is defined in Table 287.

### Table 287. Common export key parameters subsection (X'0003') of trusted block rule section (X'12')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Subsection tag: X'0003' Common export key parameters TLV object</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Subsection length in bytes (12 + xxx + yyyy).</td>
</tr>
<tr>
<td>004</td>
<td>001</td>
<td>Subsection version number (X'00').</td>
</tr>
<tr>
<td>005</td>
<td>002</td>
<td>Reserved, must be binary zero.</td>
</tr>
<tr>
<td>007</td>
<td>001</td>
<td>Flags (must be set to binary zero).</td>
</tr>
<tr>
<td>008</td>
<td>001</td>
<td>Export key minimum length in bytes. Length must be 0, 8, 16, or 24. Also applies to the source key. Not applicable for key generation.</td>
</tr>
<tr>
<td>009</td>
<td>001</td>
<td>Export key maximum length in bytes (yyyy). Length must be 0, 8, 16, or 24. Also applies to the source key. Not applicable for key generation.</td>
</tr>
<tr>
<td>010</td>
<td>001</td>
<td>Output key variant length in bytes (xxx). Valid values are 0 or 8 - 255. If greater than 0, the length must be at least as long as the longest key ever to be exported using this rule. If the variant is longer than the key, truncate it on the right to the length of the key prior to use. <strong>Note:</strong> The output key variant (offset 011) is not used if this length is zero.</td>
</tr>
</tbody>
</table>

976 Common Cryptographic Architecture Application Programmer's Guide
### Table 287. Common export key parameters subsection (X'0003') of trusted block rule section (X'12') (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>011</td>
<td>xxx</td>
<td>Output key variant. The variant can be any value. XOR this variant into the cleartext value of the output key.</td>
</tr>
</tbody>
</table>
| 011 + xxx      | 001            | CV length in bytes (yyy).  
  - If the length is not 0, 8, or 16, return an error.  
  - If the length is 0, and if the source key is a CCA DES key-token, preserve the CV in the symmetric encrypted output if the output is to be in the form of a CCA DES key-token.  
  - If a nonzero length is less than the length of the key identified by the source_key_identifier parameter, return an error.  
  - If the length is 16, and if the CV (offset 012 + xxx) is valued to 16 bytes of X'00' (ignoring the key-part bit), then:  
    1. Ignore all CV bit definitions  
    2. If CCA DES key-token format, set the flag byte of the symmetric encrypted output key to indicate a CV value is present.  
    3. If the source key is eight bytes in length, do not replicate the key to 16 bytes |
| 012 + xxx      | yyy            | CV. (See “Control vector table” on page 989)  
  Place this CV into the output exported key-token, provided that the symmetric encrypted output key format selected (offset 018 in rule section) is CCA DES key-token.  
  - If the symmetric encrypted output key format flag (offset 018 in section X'12') indicates return an RKX key-token (X'00'), then ignore this CV. Otherwise, XOR this CV into the cleartext transport key.  
  - XOR the CV of the source key into the cleartext transport key if the CV length (offset 011 + xxx) is set to 0. If a transport key to encrypt a source key has equal left and right key halves, return an error. Replicate the key halves of the key identified by the source_key_identifier parameter whenever all of these conditions are met:  
    1. The Key Generate - SINGLE-R command (offset X'00DB') is enabled in the active role  
    2. The CV length (offset 011 + xxx) is 16, and both CV halves are nonzero  
    3. The source_key_identifier parameter (contained in either a CCA DES key-token or RKX key-token) identifies an 8-byte key  
    4. The key-form bits (40 - 42) of this CV do not indicate a single-length key (are not set to zero)  
    5. Key-form bit 40 of this CV does not indicate the key is to have guaranteed unique halves (is not set to B'1'). See “Key Form Bits, fff” on page 995.  
  Note: A transport key is not used when the symmetric encrypted output key is in RKX key-token format. |

**Note:** See “Number representation in trusted blocks” on page 971.

### Trusted block section X'12' subsection X'0004'

Subsection X'0004' of the trusted block rule section (X'12') is the source key rule reference TLV object. This subsection is optional, but is required if using an RKX key-token as a source key (identified by source_key_identifier parameter). It contains the rule ID for the rule used to create the export key. If this subsection is not present, an RKX key-token format source key will not be accepted for use.

This subsection is defined in Table 288 on page 978.
Table 288. Source key rule reference subsection (X'0004') of trusted block rule section (X'12')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Subsection tag: X'0004'  Source key rule reference TLV object</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Subsection length in bytes (14).</td>
</tr>
<tr>
<td>004</td>
<td>001</td>
<td>Subsection version number (X'00').</td>
</tr>
<tr>
<td>005</td>
<td>001</td>
<td>Reserved, must be binary zero.</td>
</tr>
<tr>
<td>006</td>
<td>008</td>
<td>Rule ID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rule identifier for the rule that must have been used to create the source key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The Rule ID is an 8-byte string of ASCII characters, left-aligned and padded on the right with space characters. Acceptable characters are A - Z, a - z, 0 - 9, - (X'2D'), and _ (X'5F'). All other characters are reserved for future use.</td>
</tr>
</tbody>
</table>

Note: See “Number representation in trusted blocks” on page 971.

Trusted block section X'12' subsection X'0005'

Subsection X'0005' of the trusted block rule section (X'12') is the export key CCA token parameters TLV object. This subsection is optional. It contains a mask length, mask, and template for the export key CV limit. It also contains the template length and template for the source key label. When using a CCA DES key-token as a source key input parameter, its key type can be “filtered” by using the export key CV limit mask (offset 005) and limit template (offset 005+yyy) in this subsection.

This subsection is defined in Table 289.

Table 289. Export key CCA token parameters subsection (X'0005') of trusted block rule section (X'12')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Subsection tag: X'0005'  Export key CCA token parameters TLV object</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Subsection length in bytes (8 + yyy + yyy + zzz).</td>
</tr>
<tr>
<td>004</td>
<td>001</td>
<td>Subsection version number (X'00').</td>
</tr>
<tr>
<td>005</td>
<td>002</td>
<td>Reserved, must be binary zero.</td>
</tr>
<tr>
<td>007</td>
<td>001</td>
<td>Flags (must be set to binary zero).</td>
</tr>
<tr>
<td>008</td>
<td>001</td>
<td>Export key CV limit mask length in bytes (yyy).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Do not use CV limits if this CV limit mask length (yyy) is zero. Use CV limits if yyy is nonzero, in which case yyy:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Must be 8 or 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Must not be less than the export key minimum length (offset 008 in subsection X'0003')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Must be equal in length to the actual source key length of the key</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Example:</strong> An export key minimum length of 16 and an export key CV limit mask length of 8 returns an error.</td>
</tr>
</tbody>
</table>
### Table 289. Export key CCA token parameters subsection (X’0005’) of trusted block rule section (X’12’) (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>009</td>
<td><em>yyy</em></td>
<td>Export key CV limit mask (does not exist if <em>yyy</em>=0).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See “Control-vector-base bit maps” on page 991.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Indicates which CV bits to check against the source key CV limit template (offset 009 + <em>yyy</em>).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Examples:</strong> A mask of X’FF’ means check all bits in a byte. A mask of X’FE’ ignores the parity bit in a byte.</td>
</tr>
<tr>
<td>009 + <em>yyy</em></td>
<td><em>yyy</em></td>
<td>Export key CV limit template (does not exist if <em>yyy</em>=0).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specifies the required values for those CV bits that are checked based on the export key CV limit mask (offset 009). (See “Control-vector-base bit maps” on page 991).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The export key CV limit mask and template have the same length, <em>yyy</em>. This is because these two variables work together to restrict the acceptable CVs for CCA DES key tokens to be exported. The checks work as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. If the length of the key to be exported is less than <em>yyy</em>, return an error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Logical AND the CV for the key to be exported with the export key CV limit mask</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Compare the result to the export key CV limit template</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. Return an error if the comparison is not equal</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Examples:</strong> An export key CV limit mask of X’FF’ for CV byte 1 (key type) along with an export key CV limit template of X’3F’ (key type CVARENCE) for byte 1 filters out all key types except CVARENCE keys.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> Using the mask and template to permit multiple key types is possible, but cannot consistently be achieved with one rule section. For example, setting bit 10 to B’1’ in the mask and the template permits PIN processing keys and cryptographic variable encrypting keys, and only those keys. However, a mask to permit PIN-processing keys and key-encrypting keys, and only those keys, is not possible. In this case, multiple rule sections are required, one to permit PIN-processing keys and the other to permit key-encrypting keys.</td>
</tr>
<tr>
<td>009 + <em>yyy</em> + 001</td>
<td><em>yyy</em></td>
<td>Source key label template length in bytes (zzz).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values are 0 and 64. Return an error if the length is 64 and a source key label is not provided.</td>
</tr>
<tr>
<td>010 + <em>yyy</em> + <em>zzz</em></td>
<td><em>zzz</em></td>
<td>Source key label template (does not exist if <em>zzz</em>= 0).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If a key label is identified by the <code>source_key_identifier</code> parameter, verify that the key label name matches this template. If the comparison fails, return an error. The source key label template must conform to the following rules:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The key label template must be 64 bytes in length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The first character cannot be in the range X’00’ - X’1F’, nor can it be X’FF’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The first character cannot be numeric (X’30’ - X’39’)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• A key label name is terminated by a space character (X’20’) on the right and must be padded on the right with space characters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The only special characters permitted are #, $, @, and * (X’23’, X’24’, X’40’, and X’2A’)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The wildcard X’2A’ (*) is permitted only as the first character, the last character, or the only character in the template</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Only alphanumeric characters (a - z, A - Z, 0 - 9), the four special characters (X’23’, X’24’, X’40’, and X’2A’), and the space character (X’20’) are allowed</td>
</tr>
</tbody>
</table>

**Note:** See “Number representation in trusted blocks” on page 971.

**Trusted block section X’13’**

Trusted block section X’13’ contains the name (key label).
Key token formats

The trusted block name section provides a 64-byte variable to identify the trusted block, just as key labels are used to identify other CCA keys. This name, or label, enables a host access-control system such as RACF to use the name to verify that the application has authority to use the trusted block.

Section X'13' is optional. No multiple sections are allowed. It has no subsections defined.

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Section identifier: X'13' Trusted block name (key label)</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length in bytes (68).</td>
</tr>
<tr>
<td>004</td>
<td>064</td>
<td>Name (key label).</td>
</tr>
</tbody>
</table>

**Table 290. Trusted block key label (name) section X'13'**

Trusted block section X'14'

Trusted block section X'14' contains control and security information related to the trusted block.

This information section is separate from the public key and other sections because this section is required while the others are optional. This section contains the cryptographic information that guarantees its integrity and binds it to the local system.

Section X'14' is required. No multiple sections are allowed. Two subsections are defined.

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Section identifier: X'14' Trusted block information</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length in bytes (10+xxx).</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>006</td>
<td>004</td>
<td>Flags: Value Description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00000000' Trusted block is in the inactive state</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00000001' Trusted block is in the active state</td>
</tr>
<tr>
<td>010</td>
<td>xxx</td>
<td>Information section subsections (tag-length-value objects). One or two objects in TLV format.</td>
</tr>
</tbody>
</table>

**Table 291. Trusted block information section (X'14')**

Trusted block section X'14' subsections:

Section X'14' has two information subsections (tag-length-value objects) defined.
These subsections are summarized in Table 292. See also “Number representation in trusted blocks” on page 971.

Table 292. Summary of trusted block information subsections

<table>
<thead>
<tr>
<th>Rule subsection tag</th>
<th>TLV object</th>
<th>Optional or required</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0001'</td>
<td>Protection information</td>
<td>Required</td>
<td>Contains the encrypted 8-byte confounder and triple-length (24-byte) MAC key, the ISO-16609 TDES CBC MAC value, and the MKVP of the PKA master key (computed using MDC4).</td>
</tr>
<tr>
<td>X'0002'</td>
<td>Activation and expiration dates</td>
<td>Optional</td>
<td>Contains flags indicating whether or not the coprocessor is to validate dates, and contains the activation and expiration dates that are considered valid for the trusted block.</td>
</tr>
</tbody>
</table>

Trusted block section X'14' subsection X'0001'

Subsection X'0001' of the trusted block information section (X'14') is the protection information TLV object. This subsection is required. It contains the encrypted 8-byte confounder and triple-length (24-byte) MAC key, the ISO-16609 TDES CBC MAC value, and the MKVP of the PKA master key (computed using MDC4).

This subsection is defined in Table 293.

Table 293. Protection information subsection (X'0001') of trusted block information section (X'14')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Subsection tag: X'0001' Trusted block information TLV object</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Subsection length in bytes (62).</td>
</tr>
<tr>
<td>004</td>
<td>001</td>
<td>Subsection version number (X'00').</td>
</tr>
<tr>
<td>005</td>
<td>001</td>
<td>Reserved, must be binary zero.</td>
</tr>
<tr>
<td>006</td>
<td>032</td>
<td>Encrypted MAC key. Contains the encrypted 8-byte confounder and triple-length (24-byte) MAC key in the following format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Offset Description</td>
</tr>
<tr>
<td>00 - 07</td>
<td>Confounder</td>
<td></td>
</tr>
<tr>
<td>08 - 15</td>
<td>Left key</td>
<td></td>
</tr>
<tr>
<td>16 - 23</td>
<td>Middle key</td>
<td></td>
</tr>
<tr>
<td>24 - 31</td>
<td>Right key</td>
<td></td>
</tr>
<tr>
<td>038</td>
<td>008</td>
<td>MAC. Contains the ISO-16609 TDES CBC Message Authentication Code value.</td>
</tr>
<tr>
<td>046</td>
<td>016</td>
<td>MKVP. Contains the PKA master-key verification pattern, computed using MDC4, when the trusted block is in internal form, otherwise contains binary zero.</td>
</tr>
</tbody>
</table>

Trusted block section X'14' subsection X'0002'
Subsection X'0002' of the trusted block information section (X'14') is the activation and expiration dates TLV object. This subsection is optional. It contains flags indicating whether or not the coprocessor is to validate dates, and contains the activation and expiration dates that are considered valid for the trusted block.

This subsection is defined in Table 294.

Table 294. Activation and expiration dates subsection (X'0002') of trusted block information section (X'14')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Subsection tag: X'0002' Activation and expiration dates TLV object</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Subsection length in bytes (16).</td>
</tr>
<tr>
<td>004</td>
<td>001</td>
<td>Subsection version number (X'00').</td>
</tr>
<tr>
<td>005</td>
<td>001</td>
<td>Reserved, must be binary zero.</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>Flags:</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>X'0000'</td>
<td>The coprocessor does not check dates.</td>
<td></td>
</tr>
<tr>
<td>X'0001'</td>
<td>The coprocessor checks dates.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Compare the activation date (offset 008) and the expiration date (offset 012) to the coprocessor's internal real-time clock. Return an error if the coprocessor date is before the activation date or after the expiration date.</td>
<td></td>
</tr>
<tr>
<td>008</td>
<td>004</td>
<td>Activation date.</td>
</tr>
<tr>
<td></td>
<td>Contains the first date that the trusted block can be used for generating or exporting keys. Format of the date is YYMDD, where:</td>
<td></td>
</tr>
<tr>
<td>YY</td>
<td>Big-endian year (return an error if greater than 9999)</td>
<td></td>
</tr>
<tr>
<td>MM</td>
<td>Month (return an error if any value other than X'01' - X'0C')</td>
<td></td>
</tr>
<tr>
<td>DD</td>
<td>Day of month (return an error if any value other than X'01' - X'1F'. Day must be valid for given month and year, including leap years).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Return an error if the activation date is after the expiration date or is not valid.</td>
<td></td>
</tr>
<tr>
<td>012</td>
<td>004</td>
<td>Expiration date.</td>
</tr>
<tr>
<td></td>
<td>Contains the last date that the trusted block can be used. Same format as activation date (offset 008). Return an error if date is not valid.</td>
<td></td>
</tr>
</tbody>
</table>

**Trusted block section X'15'**

Trusted block section X'15' contains application-defined data.

The trusted block application-defined data section can be used to include application-defined data in the trusted block. The purpose of the data in this section is defined by the application; it is neither examined nor used by CCA in any way.

Section X'15' is optional. No multiple sections are allowed. It has no subsections defined.

Table 295. Trusted block application-defined data section (X'15')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Section identifier: X'15' Application-defined data</td>
</tr>
</tbody>
</table>
Table 295. Trusted block application-defined data section (X'15') (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length (6 + xxx)</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Application data length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The value of xxx must be between 0 and N, where N does not cause the overall length of the trusted block to exceed its maximum size of 3500 bytes.</td>
</tr>
<tr>
<td>006</td>
<td>xxx</td>
<td>Application-defined data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Could be used to hold a public-key certificate for the trusted public key.</td>
</tr>
</tbody>
</table>
Key token formats
Chapter 20. Key forms and types used in the Key Generate verb

The Key Generate verb is the most complex of all the CCA verbs. You can generate different key forms and key types with the Key Generate verb.

Generating an operational key

There are different methods that you can use to generate an operational key.

Choose one of the following methods:

- For operational keys, call the Key Generate (CSNBKGN) verb. Table 59 on page 219 and Table 60 on page 219 show the key type and key form combinations for a single key and for a key pair.

- For data-encrypting keys, call the Random Number Generate (CSNBRNG) verb and specify the form parameter as ODD. Then pass the generated value to the Clear Key Import (CSNBCKI) verb or the Multiple Clear Key Import (CSNBCKM) verb. The DATA key type is now in operational form.

You cannot generate a PIN verification (PINVER) key in operational form because the originator of the PIN generation (PINGEN) key generates the PINVER key in exportable form, which is sent to you to be imported.

Generating an importable key

To generate an importable key form, call the Key Generate (CSNBKGN) verb.

If you want a DATA, MAC, PINGEN, DATAM, or DATAC key type in importable form, obtain it directly by generating a single key. If you want any other key type in importable form, request a key pair where either the first or second key type is importable (IM). Discard the generated key form that you do not need.

Generating an exportable key

To generate an exportable key form, call the Key Generate (CSNBKGN) verb.

If you want a DATA, MAC, PINGEN, DATAM, or DATAC key type in exportable form, obtain it directly by generating a single key. If you want any other key type in exportable form, request a key pair where either the first or second key type is exportable (EX). Discard the generated key form that you do not need.

Examples of single-length keys in one form only

An example of single-length keys.

<table>
<thead>
<tr>
<th>Key Form</th>
<th>Key Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP DATA</td>
<td>Encipher or Decipher data. Use Key Export or Key Export to send encrypted key to another cryptographic partner. Then communicate the ciphertext.</td>
</tr>
<tr>
<td>OP MAC</td>
<td>MAC Generate. Because no MACVER key exists, there is no secure communication of the MAC with another cryptographic partner.</td>
</tr>
</tbody>
</table>
Examples of OPEX single-length, double-length, and triple-length keys in two forms

The first two letters of the key form indicate the form that key type 1 parameter is in, and the second two letters indicate the form that key type 2 parameter is in.

<table>
<thead>
<tr>
<th>Key Form</th>
<th>Type 1</th>
<th>Type 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPEX DATA DATA</td>
<td>Use the OP form in Encipher. Use Key Export with the OP form to communicate ciphertext and key with another cryptographic partner. Use Key Import at a later time to use Encipher or Decipher with the same key again.</td>
<td></td>
</tr>
<tr>
<td>OPEX MAC MAC</td>
<td>Single-length MAC Generate key. Use the OP form in MAC Generate. You have no corresponding verb MACVER key, but you can call the MAC Verify verb with the MAC key directly. Use the Key Import verb and then compute the MAC again using the MAC Verify verb, which compares the MAC it generates with the MAC supplied with the message and issues a return code indicating whether they compare.</td>
<td></td>
</tr>
</tbody>
</table>

Examples of OPEX single-length, double-length, and triple-length keys in two forms

Examples of OPEX single-length, double-length, and triple-length keys in two forms.

<table>
<thead>
<tr>
<th>Key Form</th>
<th>Type 1</th>
<th>Type 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPEX DATA DATA</td>
<td>Use the OP form in Encipher. Send the EX form and the ciphertext to another cryptographic partner.</td>
<td></td>
</tr>
<tr>
<td>OPEX MAC MAC</td>
<td>Single-length MAC generation key. Use the OP form in both MAC Generate and MAC Verify. Send the EX form to a cryptographic partner to be used in the MAC Generate or MAC Verify verbs.</td>
<td></td>
</tr>
<tr>
<td>OPEX MAC MACVER</td>
<td>Single-length MAC generation and MAC verification keys. Use the OP form in MAC Generate. Send the EX form to a cryptographic partner where it will be put into Key Import, and then MAC Verify, with the message and MAC that you have also transmitted.</td>
<td></td>
</tr>
<tr>
<td>OPEX PINGEN PINVER</td>
<td>Use the OP form in Clear PIN Generate. Send the EX form to a cryptographic partner where it is put into Key Import, and then Encrypted PIN Verify, along with an IPINENC key.</td>
<td></td>
</tr>
<tr>
<td>OPEX IMPORTER EXPORTER</td>
<td>Use the OP form in Key Import or Key Generate. Send the EX form to a cryptographic partner where it is used in Key Export, Data Key Export, or Key Generate, or put in the CCA key storage file.</td>
<td></td>
</tr>
</tbody>
</table>
| OPEX EXPORTER IMPORTER | Use the OP form in Key Export, Data Key Export,
or Key Generate. Send the EX form to a cryptographic partner where it is put into the CCA Key storage file or used in Key Import or Key Generate.

When you and your partner have the OPEX IMPORTER EXPORTER, OPEX EXPORTER IMPORTER pairs of keys in “Examples of OPEX single-length, double-length, and triple-length keys in two forms” on page 986 installed, you can start key and data exchange.

### Examples of IMEX single-length and double-length keys in two forms

Examples of IMEX single-length and double-length keys in two forms.

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Type</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMEX</td>
<td>DATA</td>
<td>Use the Key Import verb to import IM form and use the OP form in Encipher. Send the EX form to a cryptographic partner.</td>
</tr>
<tr>
<td>IMEX</td>
<td>MAC</td>
<td>Use the Key Import verb to import the IM form and use the OP form in MAC Generate. Send the EX form to a cryptographic partner who can verify the MAC.</td>
</tr>
<tr>
<td>IMEX</td>
<td>IMPORTER</td>
<td>Use the Key Import verb to import the IM form and send the EX form to a cryptographic partner. This establishes a new IMPORTER/EXPORTER key between you and your partner.</td>
</tr>
<tr>
<td>IMEX</td>
<td>PINGEN</td>
<td>Use the Key Import verb to import the IM form and send the EX form to a cryptographic partner. This establishes a new PINGEN/PINVER key between you and your partner.</td>
</tr>
</tbody>
</table>

### Examples of EXEX single-length and double-length keys in two forms

Examples of IMEX single-length and double-length keys in two forms.

For the keys shown in the following list, you are providing key distribution services for other nodes in your network, or other cryptographic partners. Neither key type can be used in your installation.

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Type</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXEX</td>
<td>DATA</td>
<td>Send the first EX form to a cryptographic partner with the corresponding IMPORTER and</td>
</tr>
<tr>
<td>EXEX</td>
<td>MAC</td>
<td>send the second EX form to another</td>
</tr>
<tr>
<td>EXEX</td>
<td>IMPORTER</td>
<td>cryptographic partner with the corresponding IMPORTER. This exchange establishes a key</td>
</tr>
<tr>
<td>EXEX</td>
<td>OPINENC</td>
<td>between two partners.</td>
</tr>
</tbody>
</table>
Chapter 21. Control vectors and changing control vectors with the Control Vector Translate verb

A control vector table shows the default value of the control vector associated with each type of key.

This information unit also describes how to change control vectors with the Control Vector Translate verb.

Control vector table

The control vector values that CCA uses to XOR key halves depend on the type of key.

Note: The control vectors descriptions here build on the descriptions used for earlier IBM products supporting CCA, each in turn: 4765, 4764, 4758, and TSS.

The master key enciphers all keys operational on your system. A transport key enciphers keys distributed off your system. Before a master key or transport key enciphers a key, CCA XORs both halves of the master key or transport key with a control vector. The same control vector is XORed to the left and right half of a master key or transport key.

Also, if you are entering a key part, CCA XORs each half of the key part with a control vector before placing the key part into the key storage file.

Each type of CCA key (except the master key) has either one or two unique control vectors associated with it. The master key or transport key CCA XORs with the control vector depending on the type of key the master key or transport key is enciphering. For double-length keys, a unique control vector exists for each half of a specific key type. For example, there is a control vector for the left half of an input PIN-encrypting key, and a control vector for the right half of an input PIN-encrypting key.

If you are entering a cleartext key part, CCA XORs the key part with the unique control vector(s) associated with the key type. CCA also enciphers the key part with two master key variants for a key part. One master key variant enciphers the left half of the key part and another master key variant enciphers the right half of the key part. CCA creates the master key variants for a key part by XORing the master key with the control vectors for key parts. These procedures protect key separation.

Table 296 displays the default value of the control vector associated with each type of key. Some key types do not have a default control vector. For keys that are double-length, CCA enciphers using a unique control vector on each half.

Table 296. Default control vector values

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Control Vector Value (Hex) Value for Single-length Key or Left Half of Double-length Key</th>
<th>Control Vector Value (Hex) Value for Right Half of Double-length Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>00 00 00 00 00 00 00 00 00 00 00</td>
<td>00 00 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>Key Type</td>
<td>Control Vector Value (Hex) Value for Single-length Key or Left Half of Double-length Key</td>
<td>Control Vector Value (Hex) Value for Right Half of Double-length Key</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------</td>
</tr>
<tr>
<td>AESTOKEN</td>
<td>00 00 00 00 00 00 00 00 00 00 00</td>
<td>00 00 00 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>CIPHERXI</td>
<td>00 0C 50 00 03 C0 00 00</td>
<td>00 0C 50 00 03 A0 00 00</td>
</tr>
<tr>
<td>CIPHERXO</td>
<td>00 0C 60 00 03 C0 00 00</td>
<td>00 0C 60 00 03 A0 00 00</td>
</tr>
<tr>
<td>CIPHERXLI</td>
<td>00 0C 71 00 03 C0 00 00</td>
<td>00 0C 71 00 03 A0 00 00</td>
</tr>
<tr>
<td>CIPHER</td>
<td>00 03 71 00 03 00 00 00</td>
<td></td>
</tr>
<tr>
<td>CIPHER (double length)</td>
<td>00 03 71 00 03 41 00 00</td>
<td>00 03 71 00 03 21 00 00</td>
</tr>
<tr>
<td>CVARDEC</td>
<td>00 3F 42 00 03 00 00 00</td>
<td></td>
</tr>
<tr>
<td>CVARENC</td>
<td>00 3F 48 00 03 00 00 00</td>
<td></td>
</tr>
<tr>
<td>CVARPINE</td>
<td>00 3F 41 00 03 00 00 00</td>
<td></td>
</tr>
<tr>
<td>CVARXCVL</td>
<td>00 3F 44 00 03 00 00 00</td>
<td></td>
</tr>
<tr>
<td>CVARXCVR</td>
<td>00 3F 47 00 03 00 00 00</td>
<td></td>
</tr>
<tr>
<td>DATA (external)</td>
<td>00 00 00 00 00 00 00 00 00 00 00</td>
<td>00 00 00 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>DATA (internal)</td>
<td>00 00 7D 00 03 41 00 00 00 00</td>
<td>00 00 7D 00 03 21 00 00</td>
</tr>
<tr>
<td>DATA</td>
<td>00 00 00 00 00 00 00 00 00 00</td>
<td></td>
</tr>
<tr>
<td>DATAC</td>
<td>00 00 71 00 03 41 00 00 00 00</td>
<td>00 00 71 00 03 21 00 00</td>
</tr>
<tr>
<td>DATAM generation key</td>
<td>00 00 4D 00 03 41 00 00 00 00</td>
<td>00 00 4D 00 03 21 00 00</td>
</tr>
<tr>
<td>(external)</td>
<td>00 00 4D 00 03 00 00 00 00 00</td>
<td>00 00 4D 00 03 00 00 00 00 00 00</td>
</tr>
<tr>
<td>DATAM key (internal)</td>
<td>00 05 4D 00 03 00 00 00 00 00</td>
<td>00 05 4D 00 03 00 00 00 00 00 00</td>
</tr>
<tr>
<td>DATAMV MAC verification key (external)</td>
<td>00 00 44 00 03 41 00 00 00 00 00</td>
<td>00 00 44 00 03 21 00 00 00 00 00 00</td>
</tr>
<tr>
<td>DATAMV MAC verification key (internal)</td>
<td>00 05 44 00 03 00 00 00 00 00 00</td>
<td>00 05 44 00 03 21 00 00 00 00 00 00</td>
</tr>
<tr>
<td>DATAXLAT</td>
<td>00 06 71 00 03 00 00 00 00 00</td>
<td></td>
</tr>
<tr>
<td>DECIPHER</td>
<td>00 03 50 00 03 00 00 00 00 00</td>
<td></td>
</tr>
<tr>
<td>DECIPHER (double-length)</td>
<td>00 03 50 00 03 41 00 00 00 00 00</td>
<td>00 03 50 00 03 21 00 00 00 00 00</td>
</tr>
<tr>
<td>DKYGENKY</td>
<td>00 71 44 00 03 31 00 00 00 00</td>
<td>00 71 44 00 03 21 00 00</td>
</tr>
<tr>
<td>DKYL0</td>
<td>This control vector has the DKYL0 set by default.</td>
<td></td>
</tr>
<tr>
<td>DKYL1</td>
<td>00 72 44 00 03 41 00 00 00 00</td>
<td>00 71 44 00 03 21 00 00</td>
</tr>
<tr>
<td>DKYL2</td>
<td>00 74 44 00 03 41 00 00 00 00</td>
<td>00 71 44 00 03 21 00 00</td>
</tr>
<tr>
<td>DKYL3</td>
<td>00 77 44 00 03 41 00 00 00 00</td>
<td>00 71 44 00 03 21 00 00</td>
</tr>
<tr>
<td>DKYL4</td>
<td>00 78 44 00 03 41 00 00 00 00</td>
<td>00 71 44 00 03 21 00 00</td>
</tr>
<tr>
<td>DKYL5</td>
<td>00 7B 44 00 03 41 00 00 00 00</td>
<td>00 71 44 00 03 21 00 00</td>
</tr>
<tr>
<td>DKYL6</td>
<td>00 7D 44 00 03 41 00 00 00 00</td>
<td>00 71 44 00 03 21 00 00</td>
</tr>
<tr>
<td>DKYL7</td>
<td>00 7E 44 00 03 41 00 00 00 00</td>
<td>00 71 44 00 03 21 00 00</td>
</tr>
<tr>
<td>ENCIIPHER</td>
<td>00 03 60 00 03 00 00 00 00 00</td>
<td></td>
</tr>
<tr>
<td>ENCIIPHER (double-length)</td>
<td>00 03 60 00 03 41 00 00 00 00</td>
<td>00 03 60 00 03 21 00 00 00 00 00</td>
</tr>
<tr>
<td>EXPORTER</td>
<td>00 41 7D 00 03 41 00 00 00 00</td>
<td>00 41 7D 00 03 21 00 00</td>
</tr>
<tr>
<td>IKEYXLAT</td>
<td>00 42 42 00 03 41 00 00 00 00</td>
<td>00 42 42 00 03 21 00 00</td>
</tr>
<tr>
<td>IMP-PKA</td>
<td>00 42 05 00 03 41 00 00 00 00</td>
<td>00 42 05 00 03 21 00 00</td>
</tr>
</tbody>
</table>
### Table 296. Default control vector values (continued)

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Control Vector Value (Hex) Value for Single-length Key or Left Half of Double-length Key</th>
<th>Control Vector Value (Hex) Value for Right Half of Double-length Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMPORTER</td>
<td>00 42 7D 00 00 03 41 00 00</td>
<td>00 42 7D 00 00 03 21 00 00</td>
</tr>
<tr>
<td>IPINENC</td>
<td>00 21 5F 00 00 03 41 00 00</td>
<td>00 21 5F 00 00 03 21 00 00</td>
</tr>
<tr>
<td>MAC</td>
<td>00 05 4D 00 00 03 00 00 00</td>
<td></td>
</tr>
<tr>
<td>MAC (double-length)</td>
<td>00 05 4D 00 00 03 41 00 00</td>
<td>00 05 4D 00 00 03 21 00 00</td>
</tr>
<tr>
<td>MACVER</td>
<td>00 05 44 00 00 03 00 00 00</td>
<td></td>
</tr>
<tr>
<td>MACVER (double-length)</td>
<td>00 05 44 00 00 03 41 00 00</td>
<td>00 05 44 00 00 03 21 00 00</td>
</tr>
<tr>
<td>OKEYXLAT</td>
<td>00 41 42 00 00 03 41 00 00</td>
<td></td>
</tr>
<tr>
<td>OPINENC</td>
<td>00 24 77 00 00 03 41 00 00</td>
<td>00 24 77 00 00 03 21 00 00</td>
</tr>
<tr>
<td>PINGEN</td>
<td>00 22 7E 00 00 03 41 00 00</td>
<td>00 22 7E 00 00 03 21 00 00</td>
</tr>
<tr>
<td>PINVER</td>
<td>00 22 42 00 00 03 41 00 00</td>
<td>00 22 42 00 00 03 21 00 00</td>
</tr>
<tr>
<td>SECMSG with SMPIN set</td>
<td>00 0A 50 00 00 03 41 00 00</td>
<td>00 0A 50 00 00 03 21 00 00</td>
</tr>
<tr>
<td>SECMSG with SMKEY set</td>
<td>00 0A 60 00 00 03 41 00 00</td>
<td>00 0A 60 00 00 03 21 00 00</td>
</tr>
</tbody>
</table>

**Note:** The external control vectors for DATA, DATAM MAC generation, and DATAMV MAC verification keys are also referred to as data compatibility control vectors.

**Control-vector-base bit maps**

Details of the control vector base bit maps.
Figure 15. Control vector base bit map (common bits and key-encrypting keys)
**Control-Vector Base Bits**

<table>
<thead>
<tr>
<th>0 0 0 0</th>
<th>0 1 1 1</th>
<th>1 1 2 2</th>
<th>2 2 3 3</th>
<th>3 3 3 3</th>
<th>4 4 4 4</th>
<th>4 5 5 5</th>
<th>5 5 6 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 4 6</td>
<td>0 0 2 4</td>
<td>1 6 0 2</td>
<td>2 2 6 2</td>
<td>2 4 6 8</td>
<td>4 4 6 8</td>
<td>0 2 4 6</td>
<td>0 2 4 6</td>
</tr>
</tbody>
</table>

Most Significant Bit

**Data Operation Keys**

<table>
<thead>
<tr>
<th>Key-Form</th>
<th>e = ENCIPHER</th>
<th>d = DECIPHER</th>
<th>m = MACGEN</th>
<th>v = MACVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>00000000</td>
<td>00000000</td>
<td>00000001</td>
<td>00000000</td>
</tr>
<tr>
<td>DATAC</td>
<td>00000000</td>
<td>00000000</td>
<td>00000011</td>
<td>00000000</td>
</tr>
<tr>
<td>DATAM</td>
<td>00000000</td>
<td>00000000</td>
<td>00000011</td>
<td>00000000</td>
</tr>
<tr>
<td>DATAMV</td>
<td>00000000</td>
<td>00000000</td>
<td>00000011</td>
<td>00000000</td>
</tr>
<tr>
<td>CIPHER</td>
<td>00000011</td>
<td>00000011</td>
<td>00000011</td>
<td>00000000</td>
</tr>
<tr>
<td>CIPHERXI</td>
<td>00001100</td>
<td>00001100</td>
<td>00001100</td>
<td>00000000</td>
</tr>
<tr>
<td>CIPHERXO</td>
<td>00001100</td>
<td>00001100</td>
<td>00001100</td>
<td>00000000</td>
</tr>
<tr>
<td>CIPHERXL</td>
<td>00001100</td>
<td>00001100</td>
<td>00001100</td>
<td>00000000</td>
</tr>
<tr>
<td>SECMSG</td>
<td>00001010</td>
<td>00001010</td>
<td>00001010</td>
<td>00000000</td>
</tr>
<tr>
<td>MAC</td>
<td>00000101</td>
<td>00000101</td>
<td>00000101</td>
<td>00000000</td>
</tr>
<tr>
<td>MACVER</td>
<td>00000101</td>
<td>00000101</td>
<td>00000101</td>
<td>00000000</td>
</tr>
</tbody>
</table>

**Figure 16. Control vector base bit map (data operation keys)**

Chapter 21. Control vectors with the Control Vector Translate verb

---

**Chapter 21. Control vectors with the Control Vector Translate verb**

**MACVER**

- 0000 ANY
- 0001 ANSI X9.9
- 0010 CVV KEY-A
- 0011 CVV KEY-B
- 0100 AMEX-CSC

**MAC**

- CIPHERXI
- CIPHERXO
- CIPHERXL

**SECMSG**

- 01 PIN encryption
- 10 Key encryption
**Control-Vector Base Bits**

<table>
<thead>
<tr>
<th>0 0 0 0</th>
<th>0 1 1 1</th>
<th>1 1 2 2</th>
<th>2 2 2 3</th>
<th>3 3 3 3</th>
<th>4 4 4 4</th>
<th>4 5 5 5</th>
<th>5 5 6 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 2 4 6</td>
<td>0 8 2 4</td>
<td>6 8 0 2</td>
<td>4 6 8 0</td>
<td>2 4 6 8</td>
<td>0 2 4 6</td>
<td>8 0 2 4</td>
<td>6 8 0 2</td>
</tr>
</tbody>
</table>

Most Significant Bit

**PIN Processing Keys**

0000 NO-SPEC
0001 IBM-PIN/IBM-PINO
0010 VISA-PVV
0011 INBK-PIN
0100 GBP-PIN/GBP-PINO
0101 NL-PIN-1

Prohibit offsets: NOOFFSET

**Cryptographic Variable-Encrypting Keys**

0000 uu.P 00111111 0EvvvvvP 00000000 00000011 fff0K00P 00000000 HTC00u0P

Key-form

Figure 17. Control vector base bit map (PIN processing keys and cryptographic variable-encrypting keys)
Key Form Bits, \textit{fff}

The key form bits, 40-42, and for a double-length key, bits 104-106, are designated \textit{fff} in the preceding illustration.

These bits can have the following values:

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{Value} & \textbf{Description} \\
\hline
000 & Single length key \\
010 & Double length key, left half \\
001 & Double length key, right half \\
\hline
\end{tabular}
\end{center}

The following values could exist in some CCA implementations:

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{Value} & \textbf{Description} \\
\hline
110 & Double-length key, left half, halves guaranteed unique \\
101 & Double-length key, right half, halves guaranteed unique \\
\hline
\end{tabular}
\end{center}

**Specifying a control-vector-base value**

You can determine the value of a control vector by working through a series of questions.

**Procedure**

Work through this series of questions:

1. Begin with a field of 64 bits (eight bytes) set to B’0’. The most significant bit is referred to as bit 0. Define the key type and subtype (bits 8 - 14) as follows:
- The main key type bits (bits 8 - 11). Set bits 8 - 11 to one of the following values:

<table>
<thead>
<tr>
<th>Bits 8 - 11</th>
<th>Main Key Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Data operation keys</td>
</tr>
<tr>
<td>0010</td>
<td>PIN keys</td>
</tr>
<tr>
<td>0011</td>
<td>Cryptographic variable-encrypting keys</td>
</tr>
<tr>
<td>0100</td>
<td>Key-encrypting keys</td>
</tr>
<tr>
<td>0101</td>
<td>Key-generating keys</td>
</tr>
<tr>
<td>0111</td>
<td>Diversified key-generating keys</td>
</tr>
</tbody>
</table>

- The key subtype bits (bits 12 - 14). Set bits 12 - 14 to one of the following values:

**Note:** For Diversified Key Generating Keys, the subtype field specifies the hierarchical level of the DKYGENKY. If the subtype is nonzero, the DKYGENKY can generate only another DKYGENKY key with the hierarchy level decremented by one. If the subtype is zero, the DKYGENKY can generate only the final diversified key (a non-DKYGENKY key) with the key type specified by the usage bits.

<table>
<thead>
<tr>
<th>Bits 12 - 14</th>
<th>Key Subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Operation Keys</strong></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>Compatibility key (DATA)</td>
</tr>
<tr>
<td>001</td>
<td>Confidentiality key (CIPHER, DECRYPTER, or ENCRYPTER)</td>
</tr>
<tr>
<td>010</td>
<td>MAC key (MAC or MACVER)</td>
</tr>
<tr>
<td>101</td>
<td>Secure messaging keys</td>
</tr>
<tr>
<td><strong>Key-Encrypting Keys</strong></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>Transport-sending keys (EXPORTER and OKEYXLAT)</td>
</tr>
<tr>
<td>001</td>
<td>Transport-receiving keys (IMPORTER and IKEYXLAT)</td>
</tr>
<tr>
<td><strong>PIN Keys</strong></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>PIN-generating key (PINGEN, PINVER)</td>
</tr>
<tr>
<td>000</td>
<td>Inbound PIN-block decrypting key (IPINENC)</td>
</tr>
<tr>
<td>010</td>
<td>Outbound PIN-block encrypting key (OPINENC)</td>
</tr>
<tr>
<td><strong>Cryptographic Variable-Encrypting Keys</strong></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>Cryptographic variable-encrypting key (CVAR...)</td>
</tr>
<tr>
<td><strong>Diversified Key Generating Keys</strong></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>DKY Subtype 0</td>
</tr>
<tr>
<td>001</td>
<td>DKY Subtype 1</td>
</tr>
<tr>
<td>010</td>
<td>DKY Subtype 2</td>
</tr>
<tr>
<td>011</td>
<td>DKY Subtype 3</td>
</tr>
<tr>
<td>100</td>
<td>DKY Subtype 4</td>
</tr>
<tr>
<td>101</td>
<td>DKY Subtype 5</td>
</tr>
<tr>
<td>110</td>
<td>DKY Subtype 6</td>
</tr>
</tbody>
</table>
Table 298. Key subtype bits (continued)

<table>
<thead>
<tr>
<th>Bits 12 - 14</th>
<th>Key Subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>DKY Subtype 7</td>
</tr>
</tbody>
</table>

2. For key-encrypting keys, set the following bits:
   - The key-generating usage bits (gks, bits 18 - 20). Set the gks bits to B’111’ to indicate the Key Generate verb can use the associated key-encrypting key to encipher generated keys when the Key Generate verb is generating various key-pair key-form combinations (see the Key-Encrypting Keys section of Figure 15 on page 992). Without any of the gks bits set to B’1’, the Key Generate verb cannot use the associated key-encrypting key. The Key Token Build verb can set the gks bits to B’1’ when you supply the OPIM, IMEX, IMIM, OPEX, and EXEX keywords.
   - The IMPORT and EXPORT bit and the XLATE bit (ix, bits 21 and 22). If the ‘i’ bit is set to B’1’, the associated key-encrypting key can be used in the Data Key Import, Key Import, Data Key Export, and Key Export verbs. If the ‘x’ bit is set to B’1’, the associated key-encrypting key can be used in the Key Translate and Key Translate2 verbs.
   - The key-form bits (fff, bits 40 - 42). The key-form bits indicate how the key was generated and how the control vector participates in multiple-enciphering. To indicate the parts can be the same value, set these bits to B’010’. For information about the value of the key-form bits in the right half of a control vector, see Step 8 on page 998.

3. For MAC and MACVER keys, set the following bits:
   - The MAC control bits (bits 20 and 21). For a MAC-generate key, set bits 20 and 21 to B’11’. For a MAC-verify key, set bits 20 and 21 to B’01’.
   - The key-form bits (fff, bits 40 - 42). For a single-length key, set the bits to B’000’. For a double-length key, set the bits to B’010’.

4. For PINGEN and PINVER keys, set the following bits:
   - The PIN calculation method bits (aaaa, bits 0 - 3). Set these bits to one of the following values:

Table 299. Calculation method keyword bits

<table>
<thead>
<tr>
<th>Bits 0 - 3</th>
<th>Calculation Method Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>NO-SPEC</td>
<td>A key with this control vector can be used with any PIN calculation method.</td>
</tr>
<tr>
<td>0001</td>
<td>IBM-PIN or IBM-PINO</td>
<td>A key with this control vector can be used only with the IBM PIN or PIN Offset calculation method.</td>
</tr>
<tr>
<td>0010</td>
<td>VISA-PVV</td>
<td>A key with this control vector can be used only with the VISA-PVV calculation method.</td>
</tr>
<tr>
<td>0100</td>
<td>GBP-PIN or GBP-PINO</td>
<td>A key with this control vector can be used only with the German Banking Pool PIN or PIN Offset calculation method.</td>
</tr>
<tr>
<td>0011</td>
<td>INBK-PIN</td>
<td>A key with this control vector can be used only with the Interbank PIN calculation method.</td>
</tr>
</tbody>
</table>
- The prohibit-offset bit (o, bit 37) to restrict operations to the PIN value. If set to B'1', this bit prevents operation with the IBM 3624 PIN Offset calculation method and the IBM German Bank Pool PIN Offset calculation method.

5. For PINGEN, IPINENC, and OPINENC keys, set bits 18 - 22 to indicate whether the key can be used with the following verbs:

<table>
<thead>
<tr>
<th>Service Allowed</th>
<th>Bit Name</th>
<th>Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear PIN Generate</td>
<td>CPINGEN</td>
<td>18</td>
</tr>
<tr>
<td>Encrypted PIN Generate Alternate</td>
<td>EPINGENA**</td>
<td>19</td>
</tr>
<tr>
<td>Encrypted PIN Generate</td>
<td>EPINGEN</td>
<td>20 for PINGEN, 19 for OPINENC</td>
</tr>
<tr>
<td>Clear PIN Generate Alternate</td>
<td>CPINGENA</td>
<td>21 for PINGEN, 20 for IPINENC</td>
</tr>
<tr>
<td>Encrypted PIN Verify</td>
<td>EPINVER</td>
<td>19</td>
</tr>
<tr>
<td>Clear PIN Encrypt</td>
<td>CPINENC</td>
<td>18</td>
</tr>
</tbody>
</table>

** EPINGENA is no longer supported, although the bit retains this definition for compatibility. There is no Encrypted Pin Generate Alternate verb.

6. For the IPINENC (inbound) and OPINENC (outbound) PIN-block ciphering keys, do the following:
   - Set the TRANSLAT bit (t, bit 21) to B'1' to permit the key to be used in the PIN Translate verb. The Control Vector Generate verb can set the TRANSLAT bit to B'1' when you supply the TRANSLAT keyword.
   - Set the REFORMAT bit (r, bit 22) to B'1' to permit the key to be used in the PIN Translate verb. The Control Vector Generate verb can set the REFORMAT bit and the TRANSLAT bit to B'1' when you supply the REFORMAT keyword.

7. For the cryptographic variable-encrypting keys (bits 18 - 22), set the variable-type bits (bits 18 - 22) to one of the following values:

<table>
<thead>
<tr>
<th>Bits 18 - 22</th>
<th>Generic Key Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>CVARPINE</td>
<td>Used in the Encrypted PIN Generate Alternate verb to encrypt a clear PIN.</td>
</tr>
<tr>
<td>00010</td>
<td>CVARXCVL</td>
<td>Used in the Control Vector Translate verb to decrypt the left mask array.</td>
</tr>
<tr>
<td>00011</td>
<td>CVARXCVR</td>
<td>Used in the Control Vector Translate verb to decrypt the right mask array.</td>
</tr>
</tbody>
</table>

8. For key-generating keys, set the following bits:
   - For KEYGENKY, set bit 18 for UKPT usage and bit 19 for CLR8-ENC usage.
   - For DKYGENKY, bits 12–14 will specify the hierarchical level of the DKYGENKY key. If the subtype CV bits are nonzero, the DKYGENKY can generate only another DKYGENKY key with the hierarchical level
decremented by one. If the subtype CV bits are zero, the DKYGENKY can generate only the final diversified key (a non-DKYGENKY key) with the key type specified by usage bits.

To specify the subtype values of the DKYGENKY, keywords DKYL0, DKYL1, DKYL2, DKYL3, DKYL4, DKYL5, DKYL6, and DKYL7 will be used.

- For DKYGENKY, bit 18 is reserved and must be zero.
- Usage bits 18-22 for the DKYGENKY key type are defined as follows. They will be encoded as the final key type that the DKYGENKY key generates.

<table>
<thead>
<tr>
<th>Bits 19 - 22</th>
<th>Keyword</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>DDATA</td>
<td>DATA, DATAC, single or double length</td>
</tr>
<tr>
<td>0010</td>
<td>DMAC</td>
<td>MAC, DATAM</td>
</tr>
<tr>
<td>0011</td>
<td>DMV</td>
<td>MACVER, DATAMV</td>
</tr>
<tr>
<td>0100</td>
<td>DIMP</td>
<td>IMPORTER, IKEYXLAT</td>
</tr>
<tr>
<td>0101</td>
<td>DEXP</td>
<td>EXPORTER, OKEYXLAT</td>
</tr>
<tr>
<td>0110</td>
<td>DPVR</td>
<td>PINVER</td>
</tr>
<tr>
<td>1000</td>
<td>DMKEY</td>
<td>Secure message key for encrypting keys</td>
</tr>
<tr>
<td>1001</td>
<td>DMPIN</td>
<td>Secure message key for encrypting PINs</td>
</tr>
<tr>
<td>1111</td>
<td>DALL</td>
<td>All key types can be generated except DKYGENKY and KEYGENKY keys. Usage of the DALL keyword is controlled by a separate access control point.</td>
</tr>
</tbody>
</table>

9. For secure messaging keys, set the following bits:
   - Set bit 18 to B’1’ if the key will be used in the secure messaging for PINs service. Set bit 19 to B’1’ if the key will be used in the secure messaging for keys service.

10. For all keys, set the following bits:
   - The export bit (E, bit 17). If set to B’0’, the export bit prevents a key from being exported. By setting this bit to B’0’, you can prevent the receiver of a key from exporting or translating the key for use in another cryptographic subsystem. After this bit is set to B’0’, it cannot be set to B’1’ by any service other than Control Vector Translate. The Prohibit Export verb can reset the export bit.
   - The key-part bit (K, bit 44). Set the key-part bit to B’1’ in a control vector associated with a key part. When the final key part is combined with previously accumulated key parts, the key-part bit in the control vector for the final key part is set to B’0’. The Control Vector Generate verb can set the key-part bit to B’1’ when you supply the KEY-PART keyword.
   - The anti-variant bits (bit 30 and bit 38). Set bit 30 to B’0’ and bit 38 to B’1’. Many cryptographic systems have implemented a system of variants where a 7-bit value is XORed with each 7-bit group of a key-encrypting key before enciphering the target key. By setting bits 30 and 38 to opposite values, control vectors do not produce patterns that can occur in variant-based systems.
• Control vector bits 64 - 127. If bits 40 - 42 are B’000’ (single-length key), set bits 64 - 127 to B’0’. Otherwise, copy bits 0 - 63 into bits 64 - 127 and set bits 105 and 106 to B’01’.

• Set the parity bits (low-order bit of each byte, bits 7, 15, ..., 127). These bits contain the parity bits (P) of the control vector. Set the parity bit of each byte so the number of zero-value bits in the byte is an even number.

• For secure messaging keys, usage bit 18 on will enable the encryption of keys in a secure message and usage bit 19 on will enable the encryption of PINs in a secure message.

---

**Changing control vectors with the Control Vector Translate verb**

What you need to do when you use the Control Vector Translate verb.

**About this task**

Do the following when using the verb:

• Provide the control information for testing the control vectors of the source, target, and key-encrypting keys to ensure that only sanctioned changes can be performed

• Select the key-half processing mode.

**Note:** This verb is not available for compliance-tagged key tokens.

**Providing the control information for testing the control vectors**

To minimize your security exposure, the Control Vector Translate verb requires control information (*mask array* information) to limit the range of allowable control vector changes.

To ensure that this verb is used only for authorized purposes, the source-key control vector, target-key control vector, and key-encrypting key (KEK) control vector must pass specific tests. The tests on the control vectors are performed within the secured cryptographic engine.

The tests consist of evaluating four logic expressions, the results of which must be a string of binary zeros. The expressions operate bitwise on information that is contained in the mask arrays and in the portions of the control vectors associated with the key or key-half that is being processed. If any of the expression evaluations do not result in all zero bits, the verb is ended with a *control vector violation* return and reason code (8/39). See [Figure 19 on page 1002](#). Only the 56-bit positions that are associated with a key value are evaluated. The low-order bit that is associated with key parity in each key byte is not evaluated.

**Mask array preparation**

A mask array consists of seven 8-byte elements: A₁, B₁, A₂, B₂, A₃, B₃, and B₄.

You choose the values of the array elements such that each of the following four expressions evaluates to a string of binary zeros. (See [Figure 19 on page 1002](#).) Set the A bits to the value you require for the corresponding control vector bits. In expressions 1 on page 1001 through 3 on page 1001, set the B bits to select the control vector bits to be evaluated. In expression 4 on page 1001, set the B bits to select the source and target control vector bits to be evaluated. Also, use the following control vector information:
$C_1$ is the control vector associated with the left half of the KEK.
$C_2$ is the control vector associated with the source key or selected source-key half/halves.
$C_3$ is the control vector associated with the target key or selected target-key half/halves.

1. $(C_1 \text{ XOR } A_1) \text{ logical-AND } B_1$
   This expression tests whether the KEK used to encipher the key meets your criteria for the desired translation.

2. $(C_2 \text{ XOR } A_2) \text{ logical-AND } B_2$
   This expression tests whether the control vector associated with the source key meets your criteria for the desired translation.

3. $(C_3 \text{ XOR } A_3) \text{ logical-AND } B_3$
   This expression tests whether the control vector associated with the target key meets your criteria for the desired translation.

4. $(C_2 \text{ XOR } C_3) \text{ logical-AND } B_4$
   This expression tests whether the control vectors associated with the source key and the target key meet your criteria for the desired translation.

Encipher two copies of the mask array, each under a different cryptographic-variable key (key type CVARENC). Use two different keys so the enciphered-array copies are unique values. When using the Control Vector Translate verb, the mask_array_left parameter and the mask_array_right parameter identify the enciphered mask arrays. The array_key_left parameter and the array_key_right parameter identify the internal keys for deciphering the mask arrays. The array_key_left parameter must have a key type of CVARXCVL and the array_key_right parameter must have a key type of CVARXCVR. The cryptographic process deciphers the arrays and compares the results; for the service to continue, the deciphered arrays must be equal. If the results are not equal, the service returns the return and reason code for data that is not valid (8/385).

Use the Key Generate verb to create the key pairs CVARENC-CVARXCVL and CVARENC-CVARXCVR. Each key in the key pair must be generated for a different node. The CVARENC keys are generated for, or imported into, the node where the mask array will be enciphered. After enciphering the mask array, you should destroy the enciphering key. The CVARXCVL and CVARXCVR keys are generated for, or imported into, the node where the Control Vector Translate verb will be performed.

If using the BOTH keyword to process both halves of a double-length key, remember that bits 41, 42, 104, and 105 are different in the left and right halves of the CCA control vector and must be ignored in your mask-array tests (that is, make the corresponding $B_2$ and/or $B_3$ bits equal to zero).

When the control vectors pass the masking tests, the verb does the following:

- Deciphers the source key. In the decipher process, the service uses a key that is formed by the XOR of the KEK and the control vector in the key token variable the source_key_token parameter identifies.
- Enciphers the deciphered source key. In the encipher process, the verb uses a key that is formed by the XOR of the KEK and the control vector in the key token variable the target_key_token parameter identifies.
- Places the enciphered key in the key field in the key token variable the target_key_token parameter identifies.
Selecting the key-half processing mode

Use the Control Vector Translate verb to change a control vector associated with a key.

rule_array keywords determine which key halves are processed in the call, as shown in Figure 20 on page 1003.
Keyword Description

SINGLE This keyword causes the control vector of the left half of the source key to be changed. The updated key half is placed into the left half of the target key in the target key token. The right half of the target key is unchanged.

The SINGLE keyword is useful when processing a single-length key or when first processing the left half of a double-length key (to be followed by processing the right half).

RIGHT This keyword causes the control vector of the right half of the source key to be changed. The updated key half is placed into the right half of the target key of the target key token. The left half of the source key is copied unchanged into the left half of the target key in the target key token.

BOTH This keyword causes the control vector of both halves of the source key to be changed. The updated key is placed into the target key in the target key token.

A single set of control information must permit the control vector changes applied to each key half. Normally, control vector bit positions 41, 42, 105, and 106 are different for each key half. Therefore, set bits 41 and 42 to B’00’ in mask array elements $B_1$, $B_2$, and $B_3$.

You can verify that the source and target key tokens have control vectors with matching bits in bit positions 40-42 and 104-106, the “form field” bits. Ensure bits 40-42 of mask array $B_4$ are set to B’111’.

LEFT This keyword enables you to supply a single-length key and obtain a double-length key. The source key token must contain:

- The KEK-enciphered single-length key
- The control vector for the single-length key (often this is a null value)
- A control vector, stored in the source token where the right-half control vector is normally stored, used in decrypting the single-length source key when the key is being processed for the target right half of the key.

The verb first processes the source and target tokens as with the SINGLE keyword. Then the source token is processed using the single-length enciphered key and the source token right-half control vector to obtain the actual key value. The key value is then enciphered using the KEK and the control vector in the target token for the right-half of the key.
This approach is frequently of use when you must obtain a double-length CCA key from a system that supports only a single-length key, for example when processing PIN keys or key-encrypting keys received from non-CCA systems.

To prevent the verb from ensuring each key byte has odd parity, you can specify the NOADJUST keyword. If you do not specify the NOADJUST keyword, or if you specify the ADJUST keyword, the verb ensures each byte of the target key has odd parity.

**When the target key-token CV is null**

When you use any of the LEFT, BOTH, or RIGHT keywords, and when the control vector in the target key token is null (all B'0'), bit 3 in byte 59 will be set to B'1' to indicate this is a double-length DATA key.

**Control vector translate example**

As an example, consider the case of receiving a single-length PIN-block encrypting key from a non-CCA system.

Often such a key will be encrypted by an unmodified transport key (no control vector or variant is used). In a CCA system, an inbound PIN encrypting key is double-length.

First use the Key Token Build verb to insert the single-length key value into the left-half key-space in a key token. Specify USE-CV as a key type and a control vector value set to 16 bytes of X'00'. Also specify EXTERNAL, KEY, and CV keywords in the rule_array. This key token will be the source key key-token.

Second, the target key token can also be created using the Key Token Build verb. Specify a key type of IPINENC and the NO-EXPORT rule_array keyword.

Then call the Control Vector Translate verb and specify a rule_array keyword of LEFT. The mask arrays can be constructed as follows:

- \( A_1 \) is set to the value of the KEK’s control vector, most likely the value of an IMPORTER key, perhaps with the NO-EXPORT bit set. \( B_1 \) is set to eight bytes of X'FF' so all bits of the KEK's control vector will be tested.
- \( A_2 \) is set to eight bytes of X'00', the (null) value of the source key control vector. \( B_2 \) is set to eight bytes of X'FF' so all bits of the source-key “control vector” will be tested.
- \( A_3 \) is set to the value of the target key’s left-half control vector. \( B_3 \) is set to X'FFFF FFFF FF9F FFFF'. This will cause all bits of the control vector to be tested except for the two (“ff”) bits used to distinguish between the left-half and right-half target-key control vector.
- \( B_4 \) is set to eight bytes of X'00' so no comparison is made between the source and target control vectors.
Chapter 22. PIN formats and algorithms

Personal identification number (PIN) notation, PIN block formats, PIN extraction rules, and PIN algorithms.

For PIN calculation procedures, see IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface Reference.

PIN notation

The content of PIN blocks are assigned a letter that represents the content.

The following notations describe the contents of PIN blocks:

- **P** = A 4-bit decimal digit that is one digit of the PIN value.
- **C** = A 4-bit hexadecimal control value. The valid values are X'0', X'1', and X'2'.
- **L** = A 4-bit hexadecimal value that specifies the number of PIN digits. This value is in the range 4 - 12.
- **F** = A 4-bit field delimiter of value X'F'.
- **f** = A 4-bit delimiter filler that is either P or F, depending on the length of the PIN.
- **D** = A 4-bit decimal padding value. All pad digits in the PIN block have the same value.
- **X** = A 4-bit hexadecimal padding value. All pad digits in the PIN block have the same value.
- **x** = A 4-bit hexadecimal filler that is either P or X, depending on the length of the PIN.
- **R** = A 4-bit hexadecimal random digit. The sequence of R digits can each take a different value.
- **r** = A 4-bit random filler that is either P or R, depending on the length of the PIN.
- **Z** = A 4-bit hexadecimal zero (X'0').
- **z** = A 4-bit zero filler that is either P or Z, depending on the length of the PIN.
- **S** = A 4-bit hexadecimal digit that constitutes one digit of a sequence number.
- **A** = A 4-bit decimal digit that constitutes one digit of a user-specified constant.

PIN block formats

All PIN block formats have a code assigned to them.

**ANSI X9.8**

This format is also named ISO format 0, VISA format 1, VISA format 4, and ECI format 1.

- P1 = CLPPPPffffffffFF
- P2 = ZZZZAAAAAAAAAAAA
PIN Block = P1 XOR P2

where C = X'0'
L = X'4' to X'C'

**Programming Note:** The rightmost 12 digits (excluding the check digit) in P2 are the rightmost 12 digits of the account number for all formats except VISA format 4. For VISA format 4, the rightmost 12 digits (excluding the check digit) in P2 are the leftmost 12 digits of the account number.

### ISO Format 1

Example code for ISO Format 1.

This format is also named ECI format 4.

PIN Block = CLPPPPrrrrrrrrRR

where C = X'1'
L = X'4' to X'C'

### ISO Format 2

Example code for ISO Format 2.

PIN Block = CLPPPPffffffffFF

where C = X'2'
L = X'4' to X'C'

### ISO Format 3

The formats of the intermediate PIN-block, the PAN block, and the ISO-3 PIN-block.

An ISO-3 PIN-block format is equivalent to the ANSI X9.8, VISA-1, and ECI-1 PIN-block formats in length. A PIN that is longer than 12 digits is truncated on the right.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
</table>

Intermediate PIN-Block = IPB

|   | 0 | 0 | 0 | 0 | PAN | PAN | PAN | PAN | PAN | PAN | PAN | PAN | PAN | PAN | PAN |
|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|

PAN Block

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>L</th>
<th>P</th>
<th></th>
<th>P/R</th>
<th>P/R</th>
<th>P/R</th>
<th>P/R</th>
<th>P/R</th>
<th>P/R</th>
<th>P/R</th>
<th>R</th>
<th>R</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>L</td>
<td>P</td>
<td></td>
<td>PAN</td>
<td>PAN</td>
<td>PAN</td>
<td>PAN</td>
<td>PAN</td>
<td>PAN</td>
<td>PAN</td>
<td>PAN</td>
<td>PAN</td>
<td>PAN</td>
<td>PAN</td>
<td>PAN</td>
<td>PAN</td>
<td>PAN</td>
</tr>
</tbody>
</table>

PIN Block = IPB XOR PAN Block

*Figure 21. ISO-3 PIN-block format*

where:

3 Is the value X’3’ for ISO-3.
L  Is the length of the PIN, which is a 4-bit value from X'4' - X'C'.

P  Is a PIN digit, which is a 4-bit value from X'0' - X'9'. The values of the PIN
digits are independent.

P/R  Is a PIN digit or pad value. A PIN digit has a 4-bit value from X'0' - X'9'. A
pad value has a random 4-bit value of X'A' - X'F'. The number of pad
values in the intermediate PIN block (IPB) is from 2 - 10.

R  Is the random value X'A' - X'F' for the pad value.

PAN  Is twelve 4-bit digits that represent one of the following:
   • The rightmost 12 digits of the primary account-number (excluding the
     check digit) if the format of the PIN block is ISO-3, ANSI X9.8, VISA-1,
     or ECI-1.
   • The leftmost 12 digits of the primary account-number (excluding the
     check digit) if the format of the PIN block is VISA-4.

Each PAN digit has a value from X'0' - X'9'.

The PIN block is the result of XORing the 64-bit IPB with the 64-bit PAN block.

Example:

L = 6, PIN = 123456, Personal Account Number = 111222333444555
36123456AFBECD00 : IPB
0000222333444555 : PAN block for ISO-3 (ANSI X9.8, VISA-1,
36121679CFA88889 : PIN block for ISO-3 (ANSI X9.8, VISA-1, ECI-1) format

Visa Format 2

Example code for Visa Format 2.

PIN Block = LPPPPzzDDDDDDDDD

where L = X'4' to X'6'

Visa Format 3

Example code for Visa Format 3.

This format specifies that the PIN length can be in the range of 4 - 12 digits. The
PIN starts from the leftmost digit and ends by the delimiter ('F'), and the
remaining digits are padding digits.

An example of a 6-digit PIN:

PIN Block = PPPPPFFxxxxxxxx

IBM 4700 Encrypting PINPAD Format

Example code for IBM 4700 Encrypting PINPAD Format. This format uses the
value X'F' as the delimiter for the PIN.

PIN Block = LPPPPPPPPPPPPPPPPPPFSS

where L = X'4' to X'C'

IBM 3624 Format

This format requires the program to specify the delimiter, X, for determining the
PIN length.

PIN Block = PPPxxxxxxxxxxxx
IBM 3621 Format
This format requires the program to specify the delimiter, X, for determining the PIN length.

PIN Block = SSSSSPPPxxxxxxxx

ECI Format 2
This format defines the PIN to be 4 digits.

PIN Block = PPPPRRRRRRRRRRRR

ECI Format 3
Example code for ECI Format 3.

PIN Block = LPPPPPzzRRRRRRRRRR

where L = X'4' to X'6'

PIN extraction rules

There are PIN extraction rules for the Encrypted PIN Verify and Encrypted PIN Translate verbs.

Encrypted PIN Verify verb
This verb extracts the customer-entered PIN from the input PIN block.

It extracts the PIN according to the following rules:

- If the input PIN block format is ANSI X9.8, ISO format 0, VISA format 1, VISA format 4, ECI format 1, ISO format 1, ISO format 2, ISO format 3, VISA format 2, IBM Encrypting PINPAD format, or ECI format 3, the verb extracts the PIN according to the length specified in the PIN block.
- If the input PIN block format is VISA format 3, the specified delimiter (padding) determines the PIN length. The search starts at the leftmost digit in the PIN block. If the input PIN block format is 3624, the specification of a PIN extraction method for the 3624 is supported through rule_array keywords. If no PIN extraction method is specified in the rule_array, the specified delimiter (padding) determines the PIN length.
- If the input PIN block format is 3621, the specification of a PIN extraction method for the 3621 is supported through rule_array keywords. If no PIN extraction method is specified in the rule_array, the specified delimiter (padding) determines the PIN length.
- If the input PIN block format is ECI format 2, the PIN is the leftmost 4 digits.

For the VISA algorithm, if the extracted PIN length is less than 4, the verb sets a reason code that indicates verification failed. If the length is greater than or equal to 4, the verb uses the leftmost 4 digits as the referenced PIN.

For the IBM German Banking Pool algorithm, if the extracted PIN length is not 4, the verb sets a reason code that indicates verification failed.

For the IBM 3624 algorithm, if the extracted PIN length is less than the PIN check length, the verb sets a reason code that indicates verification failed.
Clear PIN Generate Alternate verb

This verb extracts the customer-entered PIN from the input PIN block.

It extracts the PIN from the input PIN block according to the following rules:

- This verb supports the specification of a PIN extraction method for the 3624 and 3621 PIN block formats through the use of the rule_array keyword. The rule_array points to an array of one or two 8-byte elements. The first element in the rule_array specifies the PIN calculation method. The second element in the rule_array (if specified) indicates the PIN extraction method. Refer to the “Clear PIN Generate Alternate (CSNBCPA)” on page 518 for an explanation of PIN extraction method keywords.

Encrypted PIN Translate verb

This verb extracts the customer-entered PIN from the input PIN block.

It extracts the PIN from the input PIN block according to the following rules:

- If the input PIN block format is ANSI X9.8, ISO format 0, VISA format 1, VISA format 4, ECI format 1, ISO format 1, ISO format 2, ISO format 3, VISA format 2, IBM Encrypting PINPAD format, or ECI format 3 and, if the specified PIN length is less than 4, the verb sets a reason code to reject the operation. If the specified PIN length is greater than 12, the operation proceeds to normal completion with unpredictable contents in the output PIN block. Otherwise, the verb extracts the PIN according to the specified length.

- If the input PIN block format is VISA format 3, the specified delimiter (padding) determines the PIN length. The search starts at the leftmost digit in the PIN block. If the input PIN block format is 3624, the specification of a PIN extraction method for the 3624 is supported through rule_array keywords. If no PIN extraction method is specified in the rule_array, the specified delimiter (padding) determines the PIN length.

- If the input PIN block format is 3621, the specification of a PIN extraction method for the 3621 is supported through rule_array keywords. If no PIN extraction method is specified in the rule_array, the specified delimiter (padding) determines the PIN length.

- If the input block format is ECI format 2, the PIN is always the leftmost 4 digits.

If the maximum PIN length allowed by the output PIN block is shorter than the extracted PIN, only the leftmost digits of the extracted PIN that form the allowable maximum length are placed in the output PIN block. The PIN length field in the output PIN block, if it exists, specifies the allowable maximum length.

PIN Change/Unblock verb

The PIN Block calculation PIN Change/Unblock.

1. Form three 8-byte, 16-digit blocks, -1, -2, and -3, and set all digits to X’0’.
2. Replace the rightmost four bytes of block-1 with the authentication code described in the previous section.
3. Set the second digit of block-2 to the length of the new PIN (4 to 12), followed by the new PIN, and padded to the right with X’F’.
4. Include any current PIN by placing it into the leftmost digits of block-3.
5. Exclusive-OR blocks -1, -2, and -3 to form the 8-byte PIN block.
6. Pad the PIN block with other portions of the message for the smart card:
   - prepend X’08’
• append X'80'
• append an additional six bytes of X'00'.

The resulting message is ECB-mode triple-encrypted with an appropriate session key.

**IBM PIN algorithms**

There are several PIN algorithms, including the IBM PIN generation algorithms, IBM PIN offset generation algorithm, and IBM PIN verification algorithms.

**3624 PIN generation algorithm**

This algorithm generates an n-digit PIN based on account-related data or person-related data, namely the validation data. The assigned PIN length parameter specifies the length of the generated PIN.

The algorithm requires the following input parameters:
• A 64-bit validation data
• A 64-bit decimalization table
• A 4-bit assigned PIN length
• A 128-bit PIN-generation key

The service uses the PIN generation key to encipher the validation data. Each digit of the enciphered validation data is replaced by the digit in the decimalization table whose displacement from the leftmost digit of the table is the same as the value of the digit of the enciphered validation data. The result is an intermediate PIN. The leftmost n digits of the intermediate PIN are the generated PIN, where n is specified by the assigned PIN length.

*Figure 22 on page 1011* illustrates the 3624 PIN generation algorithm.
German Banking Pool PIN Generation algorithm

This algorithm generates a 4-digit PIN based on account-related data or person-related data, namely the validation data.

The algorithm requires the following input parameters:

- A 64-bit validation data
- A 64-bit decimalization table
- A 128-bit PIN-generation key

The validation data is enciphered using the PIN generation key. Each digit of the enciphered validation data is replaced by the digit in the decimalization table whose displacement from the leftmost digit of the table is the same as the value of the digit of enciphered validation data. The result is an intermediate PIN. The rightmost 4 digits of the leftmost 6 digits of the intermediate PIN are extracted. The leftmost digit of the extracted 4 digits is checked for zero. If the digit is zero, the digit is changed to one; otherwise, the digit remains unchanged. The resulting four digits is the generated PIN.

Figure 23 on page 1012 illustrates the German Banking Pool (GBP) PIN generation algorithm.
PIN offset generation algorithm

To allow the customer to select his own PIN, a PIN offset is used by the IBM 3624 and GBP PIN generation algorithms to relate the customer-selected PIN to the generated PIN.

The PIN offset generation algorithm requires two parameters in addition to those used in the 3624 PIN generation algorithm. They are a customer-selected PIN and a 4-bit PIN check length. The length of the customer-selected PIN is equal to the assigned-PIN length, n.

The "3624 PIN generation algorithm" on page 1010 is performed. The offset data value is the result of subtracting (modulo 10) the leftmost n digits of the intermediate PIN from the customer-selected PIN. The modulo 10 subtraction ignores borrows. The rightmost m digits of the offset data form the PIN offset, where m is specified by the PIN check length. Note that n cannot be less than m.

To generate a PIN offset for a GBP PIN, m is set to 4 and n is set to 6.

Figure 24 on page 1013 illustrates the PIN offset generation algorithm.
3624 PIN verification algorithm

This algorithm generates an intermediate PIN based on the specified validation data. A part of the intermediate PIN is adjusted by adding an offset data. A part of the result is compared with the corresponding part of the customer-entered PIN.

The algorithm requires the following input parameters:

- A 64-bit validation data
- A 64-bit decimalization table
- A 128-bit PIN-verification key
- A 4-bit PIN check length
- An offset data
- A customer-entered PIN

The rightmost m digits of the offset data form the PIN offset, where m is the PIN check length.

1. The validation data is enciphered using the PIN verification key. Each digit of the enciphered validation data is replaced by the digit in the decimalization...
table whose displacement from the leftmost digit of the table is the same as the value of the digit of enciphered validation data.

2. The leftmost $n$ digits of the result is added (modulo 10) to the offset data value, where $n$ is the length of the customer-entered PIN. The modulo 10 addition ignores carries.

3. The rightmost $m$ digits of the result of the addition operation form the PIN check number. The PIN check number is compared with the rightmost $m$ digits of the customer-entered PIN. If they match, PIN verification is successful; otherwise, verification is unsuccessful.

When a nonzero PIN offset is used, the length of the customer-entered PIN is equal to the assigned PIN length.

Figure 25 on page 1015 illustrates the PIN verification algorithm.
German Banking Pool PIN Verification algorithm

This algorithm generates an intermediate PIN based on the specified validation data.

A part of the intermediate PIN is adjusted by adding an offset data. A part of the result is extracted. The extracted value might or might not be modified before it compares with the customer-entered PIN.

The algorithm requires the following input parameters:
• A 64-bit validation data
• A 64-bit decimalization table
• A 128-bit PIN verification key
• An offset data
• A customer-entered PIN

The rightmost 4 digits of the offset data form the PIN offset.
1. The validation data is enciphered using the PIN verification key. Each digit of the enciphered validation data is replaced by the digit in the decimalization table whose displacement from the leftmost digit of the table is the same as the value of the digit of enciphered validation data.
2. The leftmost 6 digits of the result is added (modulo 10) to the offset data. The modulo 10 addition ignores carries.
3. The rightmost 4 digits of the result of the addition (modulo 10) are extracted.
4. The leftmost digit of the extracted value is checked for zero. If the digit is zero, the digit is set to one; otherwise, the digit remains unchanged. The resulting four digits are compared with the customer-entered PIN. If they match, PIN verification is successful; otherwise, verification is unsuccessful.

[Figure 26 on page 1017](#) illustrates the GBP PIN verification algorithm.
VISA PIN algorithms

The VISA PIN verification algorithm performs a multiple encipherment of a value, called the transformed security parameter (TSP), and a extraction of a 4-digit PIN verification value (PVV) from the ciphertext.

The calculated PVV is compared with the referenced PVV and stored on the plastic card or data base. If they match, verification is successful.
PVV Generation algorithm

The algorithm generates a 4-digit PIN verification value (PVV) based on the transformed security parameter (TSP).

The algorithm requires the following input parameters:

- A 64-bit TSP
- A 128-bit PVV generation key

1. A multiple encipherment of the TSP using the double-length PVV generation key is performed.
2. The ciphertext is scanned from left to right. Decimal digits are selected during the scan until four decimal digits are found. Each selected digit is placed from left to right according to the order of selection. If four decimal digits are found, those digits are the PVV.
3. If, at the end of the first scan, less than four decimal digits have been selected, a second scan is performed from left to right. During the second scan, all decimal digits are skipped and only non-decimal digits can be processed. Non-decimal digits are converted to decimal digits by subtracting 10. The process proceeds until four digits of PVV are found.

Figure 27 illustrates the PVV generation algorithm.

Programming Note: For VISA PVV algorithms, the leftmost 11 digits of the TSP are the personal account number (PAN), the leftmost 12th digit is a key table index to select the PVV generation key, and the rightmost 4 digits are the PIN. The key table index should have a value in the range 1 - 6.
**PVV Verification algorithm**

The PVV verification algorithm requires specific parameters.

The algorithm requires the following input parameters:

- A 64-bit TSP
- A 16-bit referenced PVV
- A 128-bit PVV verification key

A PVV is generated using the PVV generation algorithm, except a PVV verification key rather than a PVV generation key is used. The generated PVV is compared with the referenced PVV. If they match, verification is successful.

**Interbank PIN Generation algorithm**

A description of the Interbank PIN calculation method.

The Interbank PIN calculation method consists of the following steps:

1. Let X denote the *transaction_security* parameter element converted to an array of 16 4-bit numeric values. This parameter consists of (in the following sequence) the 11 rightmost digits of the customer PAN (excluding the check digit), a constant of 6, a 1-digit key indicator, and a 3-digit validation field.

2. Encrypt X with the double-length PINGEN (or PINVER) key to get 16 hexadecimal digits (64 bits).

3. Perform decimalization on the result of the previous step by scanning the 16 hexadecimal digits from left to right, skipping any digit greater than X'9' until 4 decimal digits (for example, digits that have values from X'0' - X'9') are found.
   If all digits are scanned but 4 decimal digits are not found, repeat the scanning process, skipping all digits that are X'9' or less and selecting the digits that are greater than X'9'. Subtract 10 (X'A') from each digit selected in this scan.
   If the 4 digits that were found are all zeros, replace the 4 digits with 0100.

4. Concatenate and use the resulting digits for the Interbank PIN. The 4-digit PIN consists of the decimal digits in the sequence in which they are found.
Chapter 23. Cryptographic algorithms and processes

CCA algorithms and processes include key verification algorithms, data-encryption processes, as well as key format and encryption processes.

These processing details are described for these aspects:

- “Cryptographic key-verification techniques”
- “Modification Detection Code calculation” on page 1024
- “Ciphering methods” on page 1026
- “MAC calculation methods” on page 1033
- “RSA key-pair generation” on page 1035
- “Multiple decipherment and encipherment” on page 1036
- “PKA92 key format and encryption process” on page 1042
- “Formatting hashes and keys in public-key cryptography” on page 1044

Cryptographic key-verification techniques

The key-verification implementations described in this document employ several mechanisms for assuring the integrity and value of the key.

Information is presented about these topics:

- “Master-key verification algorithms”
- “CCA DES-key verification algorithm” on page 1023
- “Encrypt zeros AES-key verification algorithm” on page 1023
- “Encrypt zeros DES-key verification algorithm” on page 1024

Master-key verification algorithms

The CEX*C implementations employ triple-length DES and PKA master keys (three DES keys) that are internally represented in 24 bytes (168 bits).

Beginning with Release 4.1.0, the CEX*C employs an APKA master key represented in 32 bytes (256 bits). Verification patterns on the contents of the new, current, and old master-key registers can be generated and verified when the selected register is not in the empty state. For the AES master key, the SHA-256 verification method is used.

The CEX*C employ several verification pattern generation methods.

SHA-1 based master-key verification method

A SHA-1 hash algorithm is calculated on the quantity X’01’ prepended to the 24-byte register contents.

The resulting 20-byte hash value is used in the following ways:

- The Key Test and Key Test2 verbs use the first eight bytes of the 20-byte hash value as the random_number variable, and uses the second eight bytes as the verification_pattern.
- A SHA-1 based master-key verification pattern stored in a two-byte or an eight-byte master-key verification pattern field in a key token consists of the first two or the first eight bytes of the calculated SHA-1 value, respectively.
**z/OS-based master-key verification method**

When the first and third portions of the symmetric master key have the same value, the master key is effectively a double-length DES key.

In this case, the master-key verification pattern (MKVP) is based on this algorithm:

- \( C = \text{'4545454545454545'} \)
- \( \text{IR} = \text{MK}_{\text{first-part}} \text{ XOR } e_{C}(\text{MK}_{\text{first-part}}) \)
- \( \text{MKVP} = \text{MK}_{\text{second-part}} \text{ XOR } e_{\text{IR}}(\text{MK}_{\text{second-part}}) \)

where:

- \( e_{x}(Y) \) is the DES encoding of \( Y \) using \( x \) as a key
- \( \text{XOR} \) means bitwise exclusive OR

Version \( \text{'00'} \) internal CCA DES key tokens use this eight-byte master-key verification pattern.

**SHA-256 based master-key verification method**

A SHA-256 hash algorithm is calculated on the quantity \( \text{'01'} \) prep-ended to the 24-byte register contents.

For AES, there will be verification patterns for both the AES master key and for AES operational keys that are used to encipher or decipher data. The verification pattern on the master key is called the MKVP. The verification pattern on operational keys is referred to as a key-verification pattern (KVP).

Both the MKVP and KVP for AES will use the same algorithm. Both will be computed with the following process.

1. Compute the SHA-256 hash of the string formed by prepending the byte \( \text{'01'} \) to the cleartext key value.
2. Take the leftmost eight bytes of the hash as the verification pattern.

This value is truncated to eight bytes because this is the length allocated for the verification in several CCA structures and APIs. For example, the AES key token has eight bytes for the MKVP, and the Key Test and Key Test2 verbs have an eight-byte parameter for the verification pattern.

**Asymmetric master key MDC-based verification method**

The verification pattern for the asymmetric master keys is based on hashing the value of the master-key using the MDC-4 hashing algorithm.

The master key is not parity adjusted.

The RSA private key sections \( \text{'06'} \) and \( \text{'08'} \) use this 16-byte master-key version number.

**Key-token verification patterns**

These verification pattern techniques are used in the several types of CCA key tokens.

The techniques are:

- AES and ECC key tokens: leftmost 8 bytes of SHA-256 hash of the string formed by pre-pending \( \text{'01'} \) to the cleartext key value.
- DES key tokens:
– Triple-length master key, key token version X'00': leftmost 8 bytes of SHA-1 hash
– Triple-length master key, key token version X'03': leftmost 2 bytes of SHA-1 hash
– Double-length master key, key token version X'00': leftmost 8 bytes of z/OS hash
– Double-length master key, key token version X'03': leftmost 2 bytes of SHA-1 hash

• RSA key tokens:
  – Private-key section types X'06' and X'08': 16-byte MDC-4 value
  – Private-key section types X'02' and X'05': leftmost 2 bytes of SHA-1 hash
• Trusted blocks: 16-byte MDC-4 value

CCA DES-key verification algorithm

The cryptographic engines provide a method for verifying the value of a DES cryptographic key or key part without revealing information about the value of the key or key part.

The CCA verification method first creates a random number. A one-way cryptographic function combines the random number with the key or key part. The verification method returns the result of this one-way cryptographic function (the verification pattern) and the random number.

Note: A one-way cryptographic function is a function in which it is easy to compute the output from a given input, but it is not computationally feasible to compute the input given an output.

For information about how you can use an application program to invoke this verification method, see “Key Test (CSNBKYT)” on page 245.

The CCA DES key verification algorithm does the following:

1. Sets KKR′ = KKR XOR RN
2. Sets K1 = X'4545454545454545'
3. Sets X1 = DES encoding of KKL using key K1
4. Sets K2 = X1 XOR KKL
5. Sets X2 = DES encoding of KKR′ using key K2
6. Sets VP = X2 XOR KKR′

where:

RN Is the random number generated or provided
KKL Is the value of the single-length key, or is the left half of the double-length key
KKR Is XL8'00' if the key is a single-length key, or is the value of the right half of the double-length key
VP Is the verification pattern

Encrypt zeros AES-key verification algorithm

The cryptographic engine provides a method for verifying the value of an AES cryptographic key or key part without revealing information about the value of the key or key part.
In this method, the AES key data encryption algorithm encodes a 128-bit value that is all zero bits. The leftmost 32 bits of the result are compared to the trial input value or returned from the Key Test2 verb in an 8-byte variable that is padded with bits valued to zero.

Encrypt zeros DES-key verification algorithm
The cryptographic engine provides a method for verifying the value of a DES cryptographic key or key part without revealing information about the value of the key or key part.

In this method the single-length or double-length key DEA encodes a 64-bit value that is all zero bits. The leftmost 32 bits of the result are compared to the trial input value or returned from the Key Test and Key Test2 verbs.

For a single-length key, the key DEA encodes an 8-byte, all-zero-bits value.

For a double-length key, the key DEA triple-encodes an 8-byte, all-zero-bits value. The left half (high-order half) key encodes the zero-bit value, this result is DEA decoded by the right half key, and that result is DEA encoded by the left key half.

SHAVP1 algorithm
This algorithm is used by the Key Test2 callable service to generate and verify the verification pattern.

\[ VP = Trunc128( SHA256( KA || KT || KL || K )) \]

where:

\( VP \) Is the 128-bit verification pattern

\( \text{TruncN}(x) \) is truncation of the string \( x \) to the left most \( N \) bits

\( \text{SHA256}(x) \) is the SHA-256 hash of the string \( x \)

\( KA \) Is the one-byte CCA variable-length key token constant for the algorithm of key (HMAC X'03')

\( KT \) Is the two-byte CCA variable-length key token constant for the type of key (MAC X'0002')

\( KL \) Is the two-byte bit length of the clear key value

\( K \) Is the clear key value left-aligned and padded on the right with binary zeros to byte boundary

\( || \) Is string concatenation

Modification Detection Code calculation
The Modification Detection Code (MDC) calculation method defines a one-way cryptographic function.

A one-way cryptographic function is a function in which it is easy to compute the input into output (a digest) but very difficult to compute the output into input. MDC uses DES encryption only and a default key of X'5252 5252 5252 5252 5252 5252 5252 2525 2525 2525 2525 2525'.

1024 Common Cryptographic Architecture Application Programmer's Guide
The MDC Generate verb supports four versions of the MDC calculation method that you specify by using one of the keywords shown in Table 303. All versions use the MDC-1 calculation.

Table 303. Versions of the MDC calculation method

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Version of the MDC calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDC-2, PADMDC-2</td>
<td>Specifies two encipherments for each 8-byte input data block. These versions use the MDC-2 calculation procedure described in Table 304.</td>
</tr>
<tr>
<td>MDC-4, PADMDC-4</td>
<td>Specifies four encipherments for each 8-byte input data block. These versions use the MDC-4 calculation procedure described in Table 304.</td>
</tr>
</tbody>
</table>

When the keywords PADMDC-2 and PADMDC-4 are used, the supplied text is always padded as follows:

- If the total supplied text is less than 16 bytes in length, pad bytes are appended to make the text length equal to 16 bytes. A length of zero is allowed.
- If the total supplied text is a minimum of 16 bytes in length, pad bytes are appended to make the text length equal to the next-higher multiple of eight bytes. One or more pad bytes are always added.
- All appended pad bytes, other than the last pad byte, are set to X'FF'.
- The last pad byte is set to a binary value equal to the count of all appended pad bytes (X'01' - X'10').

Use the resulting pad text in the Table 304. The MDC Generate verb uses these MDC calculation methods. See “MDC Generate (CSNBMDG)” on page 441 for more information.

Table 304. MDC calculation procedures

<table>
<thead>
<tr>
<th>Calculation</th>
<th>Procedure</th>
</tr>
</thead>
</table>
| MDC-1       | MDC-1(KD1, KD2, IN1, IN2, OUT1, OUT2);
|             | Set K01mod := set KD1 bit 1 to B'1' and bit 2 to B'0' (bits 0-7) |
|             | Set K02mod := set KD2 bit 1 to B'0' and bit 2 to B'1' (bits 0-7) |
|             | Set F1 := IN1 XOR eKD1mod(IN1) |
|             | Set F2 := IN2 XOR eKD2mod(IN2) |
|             | Set OUT1 := (bits 0..31 of F1) || (bits 32..63 of F2) |
|             | Set OUT2 := (bits 0..31 of F2) || (bits 32..63 of F2) |
|             | End procedure |
| MDC-2       | MDC-2(n, text, KEY1, KEY2, MDC);
|             | For i := 1, 2, ..., n do |
|             | Call MDC-1(KEY1, KEY2, T8<i>, T8<i>, OUT1, OUT2) |
|             | Set KEY1 := OUT1 |
|             | Set KEY2 := OUT2 |
|             | End do |
|             | Set output MDC := (KEY1 || KEY2) |
|             | End procedure |
### Ciphering methods

The Data Encryption Standard (DES) algorithm defines operations on 8-byte data strings.

The DES algorithm is used in many different processes within CCA:
- Encrypting and decrypting general data
- Triple-encrypting and triple-decrypting PIN blocks
- Triple-encrypting and triple-decrypting CCA DES keys
- Triple-encrypting and triple-decrypting RSA private keys with several processes
- Deriving keys, hashing data, generating CVV values, and so forth

The Encipher and Decipher verbs describe how you can request encryption or decryption of application data. See the following topic: "General data-encryption processes" on page 1027 for a description of the two standardized processes you can use.

In CCA, PIN blocks are encrypted with double-length keys. The PIN block is encrypted with the left-half key, for which the result is decrypted with the right-half key and this result is encrypted with the left-half key.

See "Triple-DES ciphering algorithms" on page 1030 and "Ciphering methods," which describe how CCA DES keys are enciphered.
**General data-encryption processes**

Although the fundamental concepts of enciphering and deciphering data are simple, different methods exist to process data strings that are not a multiple of eight bytes in length.

Two widely used methods for enciphering general data are defined in these ANSI standards:

- ANSI X3.106 cipher block chaining (CBC)
- ANSI X9.23

These methods also differ in how they define the initial chaining value (ICV).

This section describes how the Encipher and Decipher verbs implement these methods.

**Single-DES and Triple-DES encryption algorithms for general data**

Using the CEX*C, you can use the triple-DES algorithm in addition to the classical single-DES algorithm.

In the subsequent descriptions of the CBC method and ANSI X9.23 method, the actions of the Encipher and Decipher verbs encompass both single-DES and triple-DES algorithms. The triple-DES processes are depicted in Figure 28 where “left key” and “right key” refer to the two halves of a double-length DES key.

![Figure 28. Triple-DES data encryption and decryption](image)

**ANSI X3.106 Cipher Block Chaining (CBC) method**

ANSI standard X3.106 defines four modes of operation for ciphering, and one of these modes, Cipher Block Chaining (CBC), defines the basic method for ciphering multiple 8-byte data strings.
Figure 29 and Figure 30 on page 1029 show CBC using the Encipher and Decipher verbs. A plaintext data string that must be a multiple of eight bytes is processed as a series of 8-byte blocks. The ciphered result from processing an 8-byte block is XORed with the next block of 8 input bytes. The last 8-byte ciphered result is defined as an output chaining value (OCV). The security server stores the OCV in bytes 0 - 7 of the chaining_vector variable.

An ICV is XORed with the first block of eight bytes. When you call the Encipher or Decipher verb, specify the INITIAL or CONTINUE keywords. If you specify the INITIAL keyword, the default, the initialization vector from the verb parameter is XORed with the first eight bytes of data. If you specify the CONTINUE keyword, the OCV identified by the chaining_vector parameter is XORed with the first eight bytes of data.

![Diagram](image-url)

Figure 29. Enciphering using the ANSI X3.106 CBC method
ANSI X9.23 cipher block chaining

ANSI X9.23 defines an enhancement to the basic cipher block chaining (CBC) mode of ANSI X3.106 so that the system can process data with a length that is not an exact multiple of eight bytes.

The ANSI X9.23 method always appends from 1 - 8 bytes to the plaintext before encipherment. The last appended byte is the count of the added bytes and is in the range of X'01' - X'08'. The standard defines that any other added bytes, or pad characters, be random.

When the coprocessor enciphers the plaintext, the resulting ciphertext is always 1 - 8 bytes longer than the plaintext. See Figure 31 on page 1030. This is true even if the length of the plaintext is a multiple of eight bytes. When the coprocessor deciphers the ciphertext, it uses the last byte of the deciphered data as the number of bytes to remove from the end (pad bytes, if any, and count byte). The result is the original plaintext. See Figure 32 on page 1030.

The output chaining vector can be used as feedback with this method in the same way as with the X3.106 method.

The ANSI X9.23 method requires the caller to supply an initialization vector, and it does not allow specification of a pad character.

Note: The ANSI X9.23 standard has been withdrawn, but the X9.23 padding method is retained in CCA for compatibility with applications that rely on this method.
Triple-DES ciphering algorithms

A triple-DES (TDES) algorithm is used to encrypt keys, PIN blocks, and general data.

Several techniques are employed:

TDES ECB

DES keys, when triple encrypted under a double-length DES key, are ciphered using an e-d-e scheme without feedback.
TDES CBC

Encryption of general data, and RSA section type X’08’ CRT-format private keys and OPK keys, employs the scheme depicted in Figure 33 and Figure 34 on page 1032. This is often referred to as “outer CBC mode.”

This CCA supports double-length DES keys for triple-DES data encryption using the Encipher and Decipher verbs. The triple-length asymmetric master key is used to CBC encrypt CRT-format OPK keys.

EDE / DEDx

CCA employs EDE processes for encrypting several of the RSA private key formats (section types X’02’, X’05’, and X’06’) and the OPK key in section type X’06’. The EDE processes make successive use of single-key DES CBC processes. EDE2, EDE3, and EDE5 processes have been defined, based on the number of keys and initialization vectors used in the process. See Figure 35 on page 1032 and Figure 36 on page 1033. K1, K2, and K3 are true keys while “K4” and “K5” are initialization vectors. See Figure 35 on page 1032 and Figure 36 on page 1033.

For 2-key triple-DES, Kc = Ka

Figure 33. Triple-DES CBC encryption process
For 2-key triple-DES, $K_c = K_a$

*Figure 34. Triple-DES CBC decryption process*

*Figure 35. EDE algorithm*
MAC calculation methods

Four variations of DES-based message authentication can be used by the MAC Generate and MAC Verify verbs.

These variations are:
- "ANSI X9.9 MAC"
- "ANSI X9.19 Optional Procedure 1 MAC" on page 1034
- "EMV MAC" on page 1034
- "ISO 16609 TDES MAC" on page 1034

A keyed-hash MAC (HMAC) based message authentication can be used by the HMAC Generate and HMAC Verify verbs.

ANSI X9.9 MAC

The Financial Institution (Wholesale) Message Authentication Standard (ANSI X9.9-1986) defines a process for the authentication of messages from originator to recipient, and this process is called the Message Authentication Code (MAC) calculation method.

The ANSI X9.9 standard defines five options. The MAC Generate and MAC Verify verbs implement option 1, binary data.
Figure 37 shows the MAC calculation for binary data. In this figure, KEY is a 64-bit key, and \( T_1 \) - \( T_n \) are 64-bit data blocks of text. If \( T_n \) is less than 64 bits long, binary zeros are appended to the right of \( T_n \). Data blocks \( T_1 \)...\( T_n \) are DES CBC-encrypted with all output discarded except for the final output block, \( O_n \).

**ANSI X9.19 Optional Procedure 1 MAC**

The Financial Institution (Retail) Message Authentication Standard, ANSI X9.19 Optional Procedure 1 (ISO/IEC 9797-1, Algorithm 3) specifies additional processing of the 64-bit \( O_n \) MAC value.

The CCA “X9.19OPT” process employs a double-length DES key. After calculating the 64-bit MAC as above with the left half of the double-length key, the result is decrypted using the right half of the double-length key. This result is then encrypted with the left half of the double-length key. The resulting MAC value is processed according to other specifications supplied to the verb call.

**EMV MAC**

The EMV smart card standards define MAC generation and verification processes that are the same as ANSI X9.9 and ANSI X9.19 Optional Procedure 1 (ISO/IEC 9797-1, Algorithm 3), except for padding added to the end of the message.

Append one byte of \( \text{X'80'} \) to the original message. Then append additional bytes, as required, of \( \text{X'00'} \) to form an extended message, which is a multiple of eight bytes in length.

In the ANSI X9.9 and ANSI X9.19 Optional Procedure 1 standards, the leftmost 32 bits (4 bytes) of \( O_n \) are taken as the MAC. In the EMV standards, the MAC value is between four and eight bytes in length. CCA provides support for the leftmost four, six, and eight bytes of MAC value.

**ISO 16609 TDES MAC**

ISO 16609 defines a process for protecting the integrity of transmitted banking messages and for verifying that a message has originated from an authorized source and this process is called the ISO 16609 TDES MAC method.
The ISO 16609 TDES MAC method corresponds to ISO/IEC 9797-1, algorithm 1 using T-DEA (ANSI X9.52:1998). ISO/FDIS 16609 identifies this method as one of the recommended ways to generate a MAC using symmetric techniques.

The ISO 16609 TDES MAC method uses a double-length DES key and operates on data blocks that are a multiple of eight bytes. If the last input data block is not a multiple of eight bytes, binary zeros are appended to the right of the block. A CBC mode triple-DES (TDES) encryption operation is performed on the data, with all output discarded except for the final output block.

The resulting MAC value is processed according to other specifications supplied to the verb call.

**Keyed-hash MAC (HMAC)**

The Keyed-Hash Message Authentication Code (HMAC) standard (FIPS PUB 198-1) describes a mechanism for message authentication using cryptographic hash functions.

HMAC can be used with a hash function in combination with a shared secret key.

To see how to compute a MAC over the data, see FIPS PUB 198-1 available at:

**RSA key-pair generation**

RSA key-pair generation is determined based on user input of the modulus bit length, public exponent, and key type.

The output is based on creating primes \( p \) and \( q \) in conformance with ANSI X9.31 requirements as follows:

- \( p \) bit length = \((\text{modulus\_bit\_length} +1)/2\)
- \( q \) bit length = \text{modulus\_bit\_length} - \( p \)_bit_length
- \( p \) and \( q \) are randomly chosen prime numbers
- \( p > q \)
- The Rabin-Miller Probabilistic Primality Test is iterated 8 times for each prime. This test determines that a false prime is produced with probability no greater than \( 1/4^c \), where \( c \) is the number of iterations. Refer to the ANSI X9.31 standard and see the section entitled “Miller-Rabin Probabilistic Primality Test.”
- Primes \( p \) and \( q \) are relatively prime with the public exponent.
- Primes \( p \) and \( q \) are different in at least one of the first 100 most significant bits, that is, \(|p-q| > 2^{(\text{prime\_bit\_length} \cdot 100)}\). For example, when the modulus bit length is 1024, then both primes bit length are 512 bits and the difference of the two primes is \(|p-q| > 2^{412}\).

1. For each key generation, and for any size of key, the PKA manager seeds an internal FIPS-approved, SHA-1 based psuedo random number generator (PRNG) with the first 24 bytes of information that it receives from three successive calls to the random number generator (RNG) manager's PRNG interface.

2. The RNG manager can supply random number in two ways, but with the CCA Support Program only one way is used, namely, the PRNG method. The PKA manager seeds an internal FIPS-approved, SHA-1 based PRNG with 24 bytes obtained.
The RNG manager can respond to requests for random numbers from other processes with such responses interspersed between responses to PKA manager requests. An RSA key is generated from random information obtained from two cascaded SHA-1 PRNGs.

3. An RSA key is based on one or more 24-byte seeds from the RNG manager source, depending on the dynamic mix of tasks running inside the coprocessor.

There exists a system RNG manager (ANSI X9.31 compliant) that is used as the source for pseudo random numbers. The PKA manager also has a PRNG that is DSA compliant for generating primes. The PKA manager PRNG is re-seeded from the system RNG manager, for every new key pair generation, which is for every generation of a public/private key pair.

**Multiple decipherment and encipherment**

CCA uses multiple encipherment and decipherment to protect and retrieve cryptographic keys and PIN data.

CCA uses multiple encipherment whenever it enciphers a key under a key-encrypting key such as the master key or the transport key and in triple-DES encipherment for data privacy. Multiple encipherment is superior to single encipherment because multiple encipherment increases the work needed to "break" a key. CCA provides extra protection for a key by enciphering it under an enciphering key multiple times rather than once. The multiple encipherment method for keys enciphered under a key-encrypting key uses a double-length (128-bit) key split into two 64-bit halves. Like single encipherment, multiple encipherment uses a DES based on the electronic code book (ECB) mode of encipherment.

Keys can either be double-length or single-length depending on the installation and their cryptographic function. When a single-length key is encrypted under a double-length key, multiple encipherment is performed on the key. In the multiple encipherment method, the key is encrypted under the left half of the enciphering key. The result is then decrypted under the right half of the enciphering key. Finally, this result is encrypted under the left half of the enciphering key again.

When a double-length key is encrypted with multiple encipherment, the method is similar, except CCA uses two enciphering keys. One enciphering key encrypts each half of the double-length key. Double-length keys active on the system have two master key variants used when enciphering them.

Multiple encipherment and decipherment is not only used to protect or retrieve a cryptographic key, but they are also used to protect or retrieve 64-bit data in the area of PIN applications. For example, the following two sections use a double-length *KEK as an example to cipher a single-length key even though the same algorithms apply to cipher 64-bit data by a double-length PIN-related cryptographic key.

CCA also supports triple-DES encipherment for data privacy using double-length and triple-length DATA keys. For this procedure the data is first enciphered using the first DATA key. The result is then deciphered using the second DATA key. This second result is then enciphered using the third DATA key when a triple-length key is provided or reusing the first DATA key when a double-length key is provided.
Note that an asterisk (*) preceding the key means the key is double-length. Notations in this chapter have the following meaning:

- $e_K(x)$, where $x$ is enciphered under $K$
- $d_K(y)$ represents plaintext, where $K$ is the key and $y$ is the ciphertext

Therefore, $d_K(e_K(x))$ equals $x$ for any 64-bit key $K$ and any 64-bit plaintext $x$.

When a key (*$K$) to be protected is double-length, two double-length *KEKs are used. One *KEK is used for protecting the left half of the key (*$K$); another is for the right half. Multiple encipherment is used with the appropriate *KEK for protecting each half of the key.

**Multiple encipherment of single-length keys**

Definition of the multiple encipherment of a single-length key ($K$) using a double-length *KEK.

The multiple encipherment of a single-length key ($K$) using a double-length *KEK is defined as follows:

$$e^{*KEK}(K) = e_{KEKL}(d_{KEKR}(e_{KEKL}(K)))$$

where $KEKL$ is the left 64 bits of *KEK and $KEKR$ is the right 64 bits of *KEK. Figure 38 illustrates the definition.

![Figure 38. Multiple encipherment of single-length keys](image)

**Multiple decipherment of single-length keys**

Definition of the multiple decipherment of an encrypted single-length key ($Y = e^{*KEK}(K)$) using a double-length *KEK.

The multiple decipherment of an encrypted single-length key ($Y = e^{*KEK}(K)$) using a double-length *KEK is defined as follows:

$$d^{*KEK}(Y) = d_{KEKL}(e_{KEKR}(d_{KEKL}(Y)))$$

$$= d^{*KEK}(e^{*KEK}(K))$$

$$= K$$
where KEKL is the left 64 bits of *KEK and KEKR is the right 64 bits of *KEK.

Figure 39 illustrates the definition.

Multiple encipherment of double-length keys

Definition of the multiple encipherment of a double-length key (*K) using two double-length *KEKs, *KEKa, and *KEKb.

The multiple encipherment of a double-length key (*K) using two double-length *KEKs, *KEKa, and *KEKb is defined as follows:
\[
e^{*KEK}(K) = \left|\left| \begin{align*}
e^{KEKaL}(dKEKaR(e^{KEKaL}(KL))) \\ e^{KEKbL}(dKEKbR(e^{KEKbL}(KR)))
\end{align*} \right|\right|
\]

where:
- KL is the left 64 bits of *K
- KR is the right 64 bits of *K
- KEKaL is the left 64 bits of *KEKa
- KEKaR is the right 64 bits of *KEKa
- KEKbL is the left 64 bits of *KEKb
- KEKbR is the right 64 bits of *KEKb
- || means concatenation

Figure 40 on page 1039 illustrates the definition.
Multiple decipherment of double-length keys

Definition of the multiple decipherment of an encrypted double-length key, \( *Y = e^{*KEKa}(KL) \parallel e^{*KEKb}(KR) \), using two double-length *KEKs, *KEKa, and *KEKb.

The multiple decipherment of an encrypted double-length key, \( *Y = e^{*KEKa}(KL) \parallel e^{*KEKb}(KR) \), using two double-length *KEKs, *KEKa, and *KEKb, is defined as follows:

\[
D^{*KEKa}(Y_L) \parallel d^{*KEKb}(Y_R) = d^{KEKaL}(e^{KEKaR}(d^{KEKaL}(Y_L))) \parallel \\
\quad d^{KEKbL}(e^{KEKbR}(d^{KEKbL}(Y_R))) = d^{*KEKa}(e^{*KEKa}(KL)) \parallel \\
\quad d^{*KEKb}(e^{*KEKb}(KR)) = *K
\]

where
- \( Y_L \) is the left 64 bits of \( *Y \)
- \( Y_R \) is the right 64 bits of \( *Y \)
- \( KEKaL \) is the left 64 bits of \( *KEKa \)
- \( KEKaR \) is the right 64 bits of \( *KEKa \)
- \( KEKbL \) is the left 64 bits of \( *KEKb \)
- \( KEKbR \) is the right 64 bits of \( *KEKb \)
- \( \parallel \) means concatenation

Figure 41 on page 1040 illustrates the definition.
Multiple encipherment of triple-length keys

Definition of the multiple encipherment of a triple-length key (**K) using two double-length *KEKs, *KEKa, and *KEKb.

The multiple encipherment of a triple-length key (**K) using two double-length *KEKs, *KEKa, and *KEKb is defined as follows:

\[ e^{*KEKa}(KL) || e^{*KEKb}(KM) || e^{*KEKa}(KR) = e^{KEKaL}(d^{KEKaR}(e^{KEKaL}(KL))) || e^{KEKbL}(d^{KEKbR}(e^{KEKbL}(KM))) || e^{KEKaL}(d^{KEKaR}(e^{KEKaL}(KR))) \]

where:

- KL is the left 64 bits of **K
- KM is the next 64 bits of **K
- KR is the right 64 bits of **K
- KEKaL is the left 64 bits of *KEKa
- KEKaR is the right 64 bits of *KEKa
- KEKbL is the left 64 bits of *KEKb
- KEKbR is the right 64 bits of *KEKb
- || means concatenation

Figure 42 on page 1041 illustrates the definition.
Multiple decipherment of triple-length keys

Definition of the multiple decipherment of an encrypted triple-length key $**Y = e^{*KEK_a(KL)} || e^{*KEK_b(KM)} || e^{*KEK_a(KR)}$, using two double-length *KEKs, *KEK_a, and *KEK_b.

The multiple decipherment of an encrypted triple-length key $**Y = e^{*KEK_a(KL)} || e^{*KEK_b(KM)} || e^{*KEK_a(KR)}$, using two double-length *KEKs, *KEK_a, and *KEK_b, is defined as follows:

$$d^{*KEK_a}(YL) || d^{*KEK_b}(YM) || d^{*KEK_a}(YR)$$

where:
- $YL$ is the left 64 bits of $**Y$
- $YM$ is the next 64 bits of $**Y$
- $YR$ is the right 64 bits of $**Y$
- $KEK_aL$ is the left 64 bits of *KEK_a
- $KEK_aR$ is the right 64 bits of *KEK_a
- $KEK_bL$ is the left 64 bits of *KEK_b
- $KEK_bR$ is the right 64 bits of *KEK_b
- $||$ means concatenation

Figure 43 on page 1042 illustrates the definition.
PKA92 key format and encryption process

The Symmetric Key Generate and the Symmetric Key Import verbs optionally support a **PKA92** method of encrypting a DES key with an RSA public key.

This format is adapted from the IBM Transaction Security System (TSS) 4753 and 4755 product's implementation of **PKA92**. The verbs do not create or accept the complete PKA92 AS key token as defined for the TSS products. Rather, the verbs support only the actual RSA-encrypted portion of a TSS PKA92 key token, the **AS External Key Block**.

**Forming an external key block** - The PKA96 implementation forms an AS External Key Block by RSA-encrypting a key block using a public key. The key block is formed by padding the key record detailed in Table 305 with zero bits on the left, high-order end of the key record. The process completes the key block with three sub-processes: masking, overwriting, and RSA encrypting.

Table 305. **PKA96 clear DES key record**

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>005</td>
<td>Header and flags: X’01 0000 0000.’</td>
</tr>
<tr>
<td>005</td>
<td>016</td>
<td>Environment Identifier (EID), encoded in ASCII.</td>
</tr>
<tr>
<td>021</td>
<td>008</td>
<td>Control vector base for the DES key.</td>
</tr>
<tr>
<td>029</td>
<td>008</td>
<td>Repeat of the CV data at offset 021.</td>
</tr>
</tbody>
</table>
Table 305. PKA96 clear DES key record (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>037</td>
<td>008</td>
<td>The single-length DES key or the left half of a double-length DES key.</td>
</tr>
<tr>
<td>045</td>
<td>008</td>
<td>The right half of a double-length DES key or a random number. This value is locally designated &quot;K.&quot;</td>
</tr>
<tr>
<td>053</td>
<td>008</td>
<td>Random number, &quot;IV.&quot;</td>
</tr>
<tr>
<td>061</td>
<td>001</td>
<td>Ending byte, X'00.'</td>
</tr>
</tbody>
</table>

Masking Sub-process - Create a mask by CBC encrypting a multiple of eight bytes of binary zeros using K as the key and IV as the initialization vector as defined in the key record at offsets 45 and 53. XOR the mask with the key record and call the result PKR.

Overwriting Sub-process - Set the high-order bits of PKR to B'01' and set the low-order bits to B'0110'.

XOR K and IV and write the result at offset 45 in PKR.

Write IV at offset 53 in PKR. This causes the masked and overwritten PKR to have IV at its original position.

Encrypting Sub-process - RSA encrypt the overwritten PKR masked key record using the public key of the receiving node.

Recovering a key from an external key block - Recover the encrypted DES key from an AS External Key Block by performing decrypting, validating, unmasking, and extraction sub-processes.

Decrypting Sub-process - RSA decrypt the AS External Key Block using an RSA private key and call the result of the decryption PKR. The private key must be usable for key management purposes.

Validating Sub-process - Verify the high-order two bits of the PKR record are valued to B'01' and the low-order four bits of the PKR record are valued to B'0110'.

Unmasking Sub-process - Set IV to the value of the eight bytes at offset 53 of the PKR record. Note that there is a variable quantity of padding prior to offset 0. See Table 305 on page 1042

Set K to the XOR of IV and the value of the eight bytes at offset 45 of the PKR record.

Create a mask equal in length to the PKR record by CBC encrypting a multiple of eight bytes of binary zeros using K as the key and IV as the initialization vector. XOR the mask with PKR and call the result the key record.

Copy K to offset 45 in the PKR record.

Extraction Sub-process. Confirm that:
- The four bytes at offset 1 in the key record are valued to X'0000 0000'.
- The two control vector fields at offsets 21 and 29 are identical.
• If the control vector is an IMPORTER or EXPORTER key class, the Environment Identifier (EID) in the key record is not the same as the EID stored in the cryptographic engine.

The control vector base of the recovered key is the value at offset 21. If the control vector base bits 40 - 42 are valued to B'010' or B'110', the key is double length. Set the right half of the received key's control vector equal to the left half and reverse bits 41 and 42 in the right half.

The recovered key is at offset 37 and is either 8 or 16 bytes long based on the control vector base bits 40 - 42. If these bits are valued to B'000', the key is single length. If these bits are valued to B'010' or B'110', the key is double length.

---

**Formatting hashes and keys in public-key cryptography**

The Digital Signature Generate and Digital Signature Verify verbs support several methods for formatting a hash and, in some cases, a descriptor for the hashing method, into a bit-string to be processed by the cryptographic algorithm.

This section provides information about the ANSI X9.31 and PKCS #1 methods. The ISO 9796-1 method can be found in the ISO standard.

This section also describes the PKCS #1, version 1, 1.5, and 2.0, methods for placing a key in a bit string for RSA ciphering as part of a key exchange.

**ANSI X9.31 hash format**

With ANSI X9.31, the string that is processed by the RSA algorithm is formatted by the concatenation of a header, padding, the hash value and a trailer, from the most significant bit to the least significant bit, so that the resulting string is the same length as the modulus of the key.

For CCA, the modulus length must be a multiple of 8 bits.

• The header consists of the value X'6B'.
• The padding consists of the value X'BB', repeated as many times as required, and ended with X'BA'.
• The hash value follows the padding.
• The trailer consists of a hashing mechanism specifier and final byte. The hashing mechanism specifier is defined as one of the following values:

  - X'31'  RIPEMD-160
  - X'32'  RIPEMD-128
  - X'33'  SHA-1
  - X'34'  SHA-256
• The final byte is X'CC'.

**PKCS #1 hash formats**

Version 2.0 and 2.1 of the PKCS #1 standard defines methods for formatting keys and hashes prior to RSA encryption of the resulting data structures.
The earlier versions of the PKCS #1 standard defined block types 0, 1, and 2, but in the current standard that terminology is dropped.

CCA implemented these processes using the terminology of the Version 2.0 standard:

- For formatting keys for secured transport (Symmetric Key Export, Digital Signature Generate, Symmetric Key Import):
  - RSAES-OAEP, the preferred method for key-encipherment when exchanging DATA keys between systems. Keyword PKCSOAEP (Version 2.0) and PKOAEP2 (Version 2.1) is used to invoke this formatting technique. The \( P \) parameter described in the standard is not used and its length is set to zero.
  - RSAES-PKCS1-v1_5, is an older method for formatting keys. It is included for compatibility with existing applications. Keyword PKCS-1.2 is used to invoke this formatting technique.

- For formatting hashes for digital signatures (Digital Signature Generate and Digital Signature Verify):
  - RSASSA-PKCS1-v1_5, the newer name for the block-type 1 format. Keyword PKCS-1.1 is used to invoke this formatting technique.
  - The PKCS #1 specification no longer describes the use of block-type 0. Keyword PKCS-1.0 is used to invoke this formatting technique. Use of block-type 0 is discouraged.

Using the terminology from older versions of the PKCS #1 standard, block types 0 and 1 are used to format a hash and block type 2 is used to format a DES key. The blocks consist of the following ("\( \| \)" means concatenation):

- \( X'00' \| BT \| PS \| X'00' \| D \)

  where:

  - \( BT \) is the block type,\( X'00'\), \( X'01'\), or \( X'02'\).
  - \( PS \) is the padding of as many bytes as required to make the block the same length as the modulus of the RSA key. Padding of \( X'00' \) is used for block type 0, \( X'FF' \) for block type 1, and random and non-\( X'00' \) for block type 2. The length of \( PS \) must be a minimum of eight bytes.
  - \( D \) is the key, or the concatenation of the BER-encoded hash identifier and the hash value.

You can create the ASN.1 BER encoding of an MD5, SHA-1, or SHA-256 value by attaching these strings at the beginning of the 16-byte or 20-byte hash values, respectively:

- **MD5**
  
  \[
  X'3020300C 06082A86 4886F70D 02050500 0410' 
  \]

- **SHA-1**
  
  \[
  X'30213009 06052B0E 03021A05 000414' 
  \]

- **SHA-256**
  
  \[
  X'3031300D 0609660E 48016503 04020105 000420' 
  \]

---

2. PKCS standards can be retrieved from [http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/public-key-cryptography-standards.htm](http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/public-key-cryptography-standards.htm).

3. The PKA 92 method and the method incorporated into the SET standard are other examples of the Optimal Asymmetric Encryption Padding (OAEP) technique. The OAEP technique is attributed to Bellare and Rogaway.
Chapter 24. Access control points and verbs

Verbs use access control points (ACPs). ACPs are also referred to as commands.

Important: By default, you should disable commands that allow an action and enable commands that disallow an action. Enabling or disabling ACPs requires knowledge about the underlying action, so that you are aware why to set the ACP on or off.

For instructions on how to enable and disable these ACPs using the TKE workstation, see z/OS Cryptographic Services ICSF Trusted Key Entry Workstation User’s Guide.

For systems that do not use the optional TKE workstation, most ACPs (current and new) are enabled in the default role with the appropriate licensed internal code on the CEX*C.

Note:
1. Each domain in the CEX*C (with hardware enforced access permissions) starts out with its own default role with the default ACP values as shown. However, it is possible to use the TKE to change ACP values in the default role or to define other roles.
2. With the panel.exe, you can show the settings of all or specific ACPs (see “Using panel.exe to show the active role and ACPs” on page 1129).

As described in Chapter 25, “Access control data structures,” on page 1073, you can assign a role to a user. The user’s permissions (permitted or disallowed operations) are attached to each role in the form of an access control point (ACP) list. Thus, the assigned role determines the commands (or ACPs) available to that user.

Full coverage of TKE use for configuration is outside the scope of this document. For details, see z/OS Cryptographic Services ICSF Trusted Key Entry Workstation User’s Guide.

Table 306 on page 1048 lists the CCA ACPs. The name of each ACP is given as it appears on the panels of the TKE user interface. Note that the group names are also given to aid locating the ACPs. The table includes the following columns:

ACP number
The hexadecimal offset, or ACP code, for the command. Offsets between X’0000’ and X’FFFF’ that are not listed in this table are reserved.

Name of ACP from TKE interface
The name of the ACP as it appears on the TKE interface

Verb name
The names of the verbs that require that ACP to be enabled; for example, the Encipher (CSNBENC) verb fails without permission to use the Encipher - DES ACP.

Entry point
The entry-point name of the verb.
**Initial setting**

Whether the ACP is ON or OFF by default.

**Usage** Usage recommendations for the ACP. The abbreviations in this column are explained at the end of the table.

See the **Restrictions**, **Required commands**, or **Usage notes** sections at the end of each verb description for access control information.

**Table 306. Access Control Points and corresponding CCA verbs**

<table>
<thead>
<tr>
<th>ACP number (hex)</th>
<th>Name of ACP from TKE interface</th>
<th>Verb name</th>
<th>Entry point</th>
<th>Initial setting</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0001 GROUP: ISPF Services</strong> Note: This group name refers to ISPF, a z/OS feature. Although ISPF is not relevant to Linux on Z, it is listed here as shown on the TKE panels to avoid confusion.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'0018'</td>
<td>DES Master Key - Load first key part</td>
<td>Master Key Process¹</td>
<td>CSNBMKP</td>
<td>ON</td>
<td>SC, SEL</td>
</tr>
<tr>
<td>X'0019'</td>
<td>DES Master Key - Combine key parts</td>
<td>Master Key Process¹</td>
<td>CSNBMKP</td>
<td>ON</td>
<td>SC, SEL</td>
</tr>
<tr>
<td>X'001A'</td>
<td>DES Master Key - Set master key</td>
<td>Master Key Process¹</td>
<td>CSNBMKP</td>
<td>ON</td>
<td>SC, SEL</td>
</tr>
<tr>
<td>X'001E'</td>
<td>Reencipher CKDS Note: The TKE name for this ACP refers to z/OS key storage (CKDS). However z/OS key storage is not impacted. This ACP refers to a service for Linux, see verb for details.</td>
<td>Key Token Change</td>
<td>CSNBKTC</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0032'</td>
<td>DES Master Key - Clear new master key register</td>
<td>Master Key Process¹</td>
<td>CSNBMKP</td>
<td>ON</td>
<td>O, SUP</td>
</tr>
<tr>
<td>X'0053'</td>
<td>RSA Master Key - Load first key part</td>
<td>Master Key Process¹</td>
<td>CSNBMKP</td>
<td>ON</td>
<td>SC, SEL</td>
</tr>
<tr>
<td>X'0054'</td>
<td>RSA Master Key - Combine key parts</td>
<td>Master Key Process¹</td>
<td>CSNBMKP</td>
<td>ON</td>
<td>SC, SEL</td>
</tr>
<tr>
<td>X'0057'</td>
<td>RSA Master Key - Set master key</td>
<td>Master Key Process¹</td>
<td>CSNBMKP</td>
<td>ON</td>
<td>SC, SEL</td>
</tr>
<tr>
<td>X'0060'</td>
<td>RSA Master Key - Clear new master key register</td>
<td>Master Key Process¹</td>
<td>CSNBMKP</td>
<td>ON</td>
<td>SC, SEL</td>
</tr>
<tr>
<td>X'00F0'</td>
<td>Reencipher CKDS2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'0124'</td>
<td>AES Master Key - Clear new master key register</td>
<td>Master Key Process¹</td>
<td>CSNBMKP</td>
<td>ON</td>
<td>O, SUP</td>
</tr>
<tr>
<td>X'0125'</td>
<td>AES Master Key - Load first key part</td>
<td>Master Key Process¹</td>
<td>CSNBMKP</td>
<td>ON</td>
<td>O, SUP</td>
</tr>
<tr>
<td>X'0126'</td>
<td>AES Master Key - Combine key parts</td>
<td>Master Key Process¹</td>
<td>CSNBMKP</td>
<td>ON</td>
<td>O, SUP</td>
</tr>
<tr>
<td>X'0128'</td>
<td>AES Master Key - Set master key</td>
<td>Master Key Process¹</td>
<td>CSNBMKP</td>
<td>ON</td>
<td>O, SUP</td>
</tr>
<tr>
<td>X'0146'</td>
<td>CKDS Conversion2 - Allow wrapping override keywords</td>
<td>Key Token Change</td>
<td>CSNBKTC</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0147'</td>
<td>CKDS Conversion2 - Convert from enhanced to original</td>
<td>Key Token Change</td>
<td>CSNBKTC</td>
<td>ON</td>
<td>O, NRP</td>
</tr>
</tbody>
</table>
Table 306. Access Control Points and corresponding CCA verbs (continued)

<table>
<thead>
<tr>
<th>ACP number (hex)</th>
<th>Name of ACP from TKE interface</th>
<th>Verb name</th>
<th>Entry point</th>
<th>Initial setting</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'014B'</td>
<td>Key Translate2 - Allow use of REFORMAT</td>
<td>Key Translate2</td>
<td>CSNBKTR2</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>X'014C'</td>
<td>CKDS Conversion2 - Allow use of REFORMAT</td>
<td>Key Token Change</td>
<td>CSNBKTC</td>
<td>ON O</td>
<td></td>
</tr>
<tr>
<td>X'0240'</td>
<td>Authorize UDX</td>
<td>- no verb -</td>
<td>- no verb -</td>
<td>ON O</td>
<td></td>
</tr>
<tr>
<td>X'0241'</td>
<td>Reencipher PKDS</td>
<td>PKA Key Token Change</td>
<td>CSNDKTC</td>
<td>ON O, R</td>
<td></td>
</tr>
<tr>
<td>X'0303'</td>
<td>PCF CKDS conversion utility</td>
<td>PCF/CUSP Key Conversion - indirect verb</td>
<td>CSNBDHD</td>
<td>OFF O, R</td>
<td></td>
</tr>
<tr>
<td>X'031F'</td>
<td>ECC Master Key - Clear new master key register</td>
<td>Master Key Process</td>
<td>CSNBMPK</td>
<td>ON O</td>
<td></td>
</tr>
<tr>
<td>X'0320'</td>
<td>ECC Master Key - Load first key part</td>
<td>Master Key Process</td>
<td>CSNBMPK</td>
<td>ON O</td>
<td></td>
</tr>
<tr>
<td>X'0321'</td>
<td>ECC Master Key - Combine key parts</td>
<td>Master Key Process</td>
<td>CSNBMPK</td>
<td>ON O</td>
<td></td>
</tr>
<tr>
<td>X'0322'</td>
<td>ECC Master Key - Set master key</td>
<td>Master Key Process</td>
<td>CSNBMPK</td>
<td>ON O</td>
<td></td>
</tr>
<tr>
<td>X'0330'</td>
<td>DES master key - 24-byte key</td>
<td>Master Key Process</td>
<td>CSNBMPK</td>
<td>OFF O</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** This ACP is included for reference only. The service impacted is available only for IBM System x, IBM System p, or using the TKE interface.

**Note:** This ACP forces the SYM and ASYM master keys to be full 24 byte DES keys.

**0002 GROUP: Coprocessor Configuration**

<table>
<thead>
<tr>
<th>ACP number (hex)</th>
<th>Name of ACP from TKE interface</th>
<th>Verb name</th>
<th>Entry point</th>
<th>Initial setting</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0034'</td>
<td>Log Query: System</td>
<td>Log Query</td>
<td>CSUALGQ</td>
<td>ON O</td>
<td></td>
</tr>
<tr>
<td>X'0035'</td>
<td>Log Query: CCA</td>
<td>Log Query</td>
<td>CSUALGQ</td>
<td>ON O</td>
<td></td>
</tr>
<tr>
<td>X'0036'</td>
<td>Log Query: Set Log Level -4-</td>
<td>Log Query</td>
<td>CSUALGQ</td>
<td>ON O</td>
<td></td>
</tr>
<tr>
<td>X'0037'</td>
<td>Log Query: Set Log Level -8-</td>
<td>Log Query</td>
<td>CSUALGQ</td>
<td>ON O</td>
<td></td>
</tr>
<tr>
<td>X'0116'</td>
<td>Access Control Manager - Read role</td>
<td>Note: This ACP is included for reference only. The service impacted is available only for IBM System x, IBM System p, or using the TKE interface.</td>
<td>CSUALGQ</td>
<td>ON O</td>
<td></td>
</tr>
<tr>
<td>X'0139'</td>
<td>Symmetric token wrapping - internal enhanced method</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>CSNDMKE</td>
<td>ON O</td>
<td></td>
</tr>
<tr>
<td>X'013A'</td>
<td>Symmetric token wrapping - internal original method</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>CSNDMKE</td>
<td>ON O</td>
<td></td>
</tr>
<tr>
<td>X'013B'</td>
<td>Symmetric token wrapping - external enhanced method</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>CSNDMKE</td>
<td>ON O</td>
<td></td>
</tr>
<tr>
<td>X'013C'</td>
<td>Symmetric token wrapping - external original method</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>CSNDMKE</td>
<td>ON O</td>
<td></td>
</tr>
<tr>
<td>X'0328'</td>
<td>Variable-length Symmetric Token - disallow weak wrap</td>
<td>EC Diffie-Hellman1 Key Generate2 PKA Key Generate3 Symmetric Key Export4</td>
<td>CSNDMKE</td>
<td>OFF O, R</td>
<td></td>
</tr>
<tr>
<td>ACP number (hex)</td>
<td>Name of ACP from TKE interface</td>
<td>Verb name</td>
<td>Entry point</td>
<td>Initial setting</td>
<td>Usage</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>----------------</td>
<td>-------</td>
</tr>
<tr>
<td>X'032C'</td>
<td>Variable-length Symmetric Token - warn when weak wrap</td>
<td>EC Diffie-Hellman(^1) Key Generate(^2) Symmetric Key Export(^3) Symmetric Key Import(^2)</td>
<td>CSNDEDH CSNBDGM2 CSNDKSM CSNDSTYJ</td>
<td>OFF</td>
<td>O, R</td>
</tr>
<tr>
<td>X'032D'</td>
<td>Disallow 24-byte DATA wrapped with 16-byte Key</td>
<td>PKA Key Generate</td>
<td>CSNDPKG</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0332'</td>
<td>Warn when weak wrap - Master keys</td>
<td>Clear Key Import Data Key Import Diversified Key Generate EC Diffie-Hellman Key Generate Key Generate(^2) Key Import Key Part Import Key Part Import(^2) Key Token Change Key Token Change(^2) Master Key Process Multiple Clear Key Import PKA Key Generate PKA Key Import PKA Key Token Change Prohibit Export Restrict Key Attribute Symmetric Key Generate Symmetric Key Import Symmetric Key Import(^2) TR31 Key Import Unique Key Derive</td>
<td>CSNBCKI CSNBDMK CSNBDMG CSNDEHH CSNBKGM CSNBKGN CSNBKPN CSNBKPM CSNBKPI CSNBKPI(^2) CSNBKTC CSNBKTC(^2) CSNBKTP CSNBKTP(^2) CSNBKTR CSNBKYM CSNBKYM(^2) CSNBKTN CSNBKTN(^2) CSNBKTM CSNBKTR</td>
<td>ON</td>
<td>O, R</td>
</tr>
<tr>
<td>X'0333'</td>
<td>Prohibit weak wrapping - Master keys</td>
<td>Same as ACP X'0332'</td>
<td>Same as ACP X'0332'</td>
<td>ON</td>
<td>O, R</td>
</tr>
<tr>
<td>0003 GROUP: API Cryptographic Services</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'000E'</td>
<td>Encipher - DES</td>
<td>Encipher</td>
<td>CSNBPNC</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'000F'</td>
<td>Decipher - DES</td>
<td>Decipher</td>
<td>CSNBDEC</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0010'</td>
<td>MAC Generate</td>
<td>MAC Generate</td>
<td>CSNBMDG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0011'</td>
<td>MAC Verify</td>
<td>MAC Verify</td>
<td>CSNBMMV</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0012'</td>
<td>Key Import</td>
<td>Key Import</td>
<td>CSNBKIM</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0013'</td>
<td>Key Export</td>
<td>Key Export</td>
<td>CSNBKEX</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'001B'</td>
<td>Key Part Import - first key part</td>
<td>Key Part Import(^1)</td>
<td>CSNBKPI</td>
<td>ON</td>
<td>SC, SEL</td>
</tr>
<tr>
<td>X'001C'</td>
<td>Key Part Import - middle and last</td>
<td>Key Part Import(^1)</td>
<td>CSNBKPI</td>
<td>ON</td>
<td>SC, SEL</td>
</tr>
<tr>
<td>X'001D'</td>
<td>Compute Verification Pattern</td>
<td>Key Storage Initialization Key Test Key Test(^2) Key Test Extended</td>
<td>CSNBKSI CSNBKSTY CSNBKyT2 CSNBKyTX</td>
<td>ON</td>
<td>R</td>
</tr>
<tr>
<td>X'001F'</td>
<td>Key Translate</td>
<td>Key Translate</td>
<td>CSNBKTR</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0021'</td>
<td>Key Test(^2) - AES, ENC-ZERO</td>
<td>Key Test(^2)</td>
<td>CSNBKyT2</td>
<td>ON</td>
<td>O</td>
</tr>
</tbody>
</table>
### Table 306. Access Control Points and corresponding CCA verbs (continued)

<table>
<thead>
<tr>
<th>ACP number (hex)</th>
<th>Name of ACP from TKE interface</th>
<th>Verb name</th>
<th>Entry point</th>
<th>Initial setting</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0022'</td>
<td>Key Test2 - AES, CMAC-ZERO</td>
<td>Key Test2</td>
<td>CSNBKYT2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0023'</td>
<td>Key Test2 - DES, CMAC-ZERO</td>
<td>Key Test2</td>
<td>CSNBKYT2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'003A'</td>
<td>Public Key Import: Disallow Clear Key Import</td>
<td>PKA Key Import</td>
<td>CSNDPKI</td>
<td>OFF</td>
<td>O, SC</td>
</tr>
<tr>
<td>X'0040'</td>
<td>Diversified Key Generate - CLR8-ENC</td>
<td>Diversified Key Generate</td>
<td>CSNBKDGA</td>
<td>ON</td>
<td>O, SEL</td>
</tr>
<tr>
<td>X'0041'</td>
<td>Diversified Key Generate - TDES-ENC</td>
<td>Diversified Key Generate</td>
<td>CSNBKDGA</td>
<td>ON</td>
<td>O, SEL</td>
</tr>
<tr>
<td>X'0042'</td>
<td>Diversified Key Generate - TDES-DEC</td>
<td>Diversified Key Generate</td>
<td>CSNBKDGA</td>
<td>ON</td>
<td>O, SEL</td>
</tr>
<tr>
<td>X'0043'</td>
<td>Diversified Key Generate - SESS-XOR</td>
<td>Diversified Key Generate</td>
<td>CSNBKDGA</td>
<td>ON</td>
<td>O, SEL</td>
</tr>
<tr>
<td>X'0044'</td>
<td>Diversified Key Generate - Single length or same halves</td>
<td>Diversified Key Generate</td>
<td>CSNBKDGA</td>
<td>ON</td>
<td>SC, SEL</td>
</tr>
<tr>
<td>X'0045'</td>
<td>Diversified Key Generate - TDES-XOR</td>
<td>Diversified Key Generate</td>
<td>CSNBKDGA</td>
<td>ON</td>
<td>O, SEL</td>
</tr>
<tr>
<td>X'0046'</td>
<td>Diversified Key Generate - TDESEMV2/TDESEMV4</td>
<td>Diversified Key Generate</td>
<td>CSNBKDGA</td>
<td>ON</td>
<td>O, SEL</td>
</tr>
<tr>
<td>X'0048'</td>
<td>Log Query: Set secure log range</td>
<td>Log Query</td>
<td>CSUALGQ</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'0049'</td>
<td>Log Query: Secure log clear range inactive</td>
<td>Log Query</td>
<td>CSUALGQ</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'004A'</td>
<td>Log Query: Secure log clear range active</td>
<td>Log Query</td>
<td>CSUALGQ</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'004B'</td>
<td>Log Query: Secure log clear all inactive</td>
<td>Log Query</td>
<td>CSUALGQ</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'004C'</td>
<td>Log Query: Secure log clear all active</td>
<td>Log Query</td>
<td>CSUALGQ</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'0070'</td>
<td>Public Infrastructure Certificate</td>
<td>Public Infrastructure Certificate</td>
<td>CSNDPIC</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>X'0071'</td>
<td>PIC Signature Algorithm SHA+RSA</td>
<td>Public Infrastructure Certificate</td>
<td>CSNDPIC</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'0072'</td>
<td>PIC Signature Algorithm ECDSA</td>
<td>Public Infrastructure Certificate</td>
<td>CSNDPIC</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>ACP number (hex)</td>
<td>Name of ACP from TKE interface</td>
<td>Verb name</td>
<td>Entry point</td>
<td>Initial setting</td>
<td>Usage</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>-----------------</td>
<td>-------</td>
</tr>
<tr>
<td>X'0073'</td>
<td>PIC Signature Algorithm RSASSA_PSS</td>
<td>Public Infrastructure Certificate</td>
<td>CSNDPIC</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'0076'</td>
<td>PIC Signature Algorithm SHA-1</td>
<td>Public Infrastructure Certificate</td>
<td>CSNDPIC</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'0077'</td>
<td>PIC Signature Algorithm SHA-224</td>
<td>Public Infrastructure Certificate</td>
<td>CSNDPIC</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'0078'</td>
<td>PIC Signature Algorithm SHA-256</td>
<td>Public Infrastructure Certificate</td>
<td>CSNDPIC</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'0079'</td>
<td>PIC Signature Algorithm SHA-384</td>
<td>Public Infrastructure Certificate</td>
<td>CSNDPIC</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'007A'</td>
<td>PIC Signature Algorithm SHA-512</td>
<td>Public Infrastructure Certificate</td>
<td>CSNDPIC</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'007B'</td>
<td>Create x509 certificate</td>
<td>Public Infrastructure Certificate</td>
<td>CSNDPIC</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'007C'</td>
<td>Public Infrastructure Certificate - PKI8SNRQ</td>
<td>Public Infrastructure Certificate</td>
<td>CSNDPIC</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>X'008A'</td>
<td>MDC Generate</td>
<td>MDC Generate</td>
<td>CSNBMDG</td>
<td>OFF</td>
<td>R</td>
</tr>
<tr>
<td>X'008C'</td>
<td>Key Generate - Key set</td>
<td>Key Generate²</td>
<td>CSNBKGN</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'008E'</td>
<td>Key Generate - OP</td>
<td>Key Generate² Random Number Generate</td>
<td>CSNBKGN</td>
<td>ON</td>
<td>R</td>
</tr>
<tr>
<td>X'0090'</td>
<td>Symmetric Key Token Change - RTCMK</td>
<td>Key Token Change</td>
<td>CSNKTC</td>
<td>ON</td>
<td>R</td>
</tr>
<tr>
<td>X'00A0'</td>
<td>Clear PIN Generate - 3624</td>
<td>Clear PIN Generate</td>
<td>CSNBPGN</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00A1'</td>
<td>Clear PIN Generate - GBP</td>
<td>Clear PIN Generate</td>
<td>CSNBPGN</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00A2'</td>
<td>Clear PIN Generate - VISA PVV</td>
<td>Clear PIN Generate</td>
<td>CSNBPGN</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00A3'</td>
<td>Clear PIN Generate - Interbank</td>
<td>Clear PIN Generate</td>
<td>CSNBPGN</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00A4'</td>
<td>Clear PIN Generate Alternate - 3624 Offset</td>
<td>Clear PIN Generate Alternate¹</td>
<td>CSNBCPA</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00AB'</td>
<td>Encrypted PIN Verify - 3624</td>
<td>Encrypted PIN Verify¹</td>
<td>CSNBPVVR</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00AC'</td>
<td>Encrypted PIN Verify - GBP</td>
<td>Encrypted PIN Verify¹</td>
<td>CSNBPVVR</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00AD'</td>
<td>Encrypted PIN Verify - VISA PVV</td>
<td>Encrypted PIN Verify¹</td>
<td>CSNBPVVR</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00AE'</td>
<td>Encrypted PIN Verify - Interbank</td>
<td>Encrypted PIN Verify¹</td>
<td>CSNBPVVR</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00AF'</td>
<td>Clear PIN Encrypt</td>
<td>Clear PIN Encrypt</td>
<td>CSNBCEPE</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00B0'</td>
<td>Encrypted PIN Generate - 3624</td>
<td>Encrypted PIN Generate³</td>
<td>CSNBEPG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00B1'</td>
<td>Encrypted PIN Generate - GBP</td>
<td>Encrypted PIN Generate³</td>
<td>CSNBEPG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00B2'</td>
<td>Encrypted PIN Generate - Interbank</td>
<td>Encrypted PIN Generate³</td>
<td>CSNBEPG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00B3'</td>
<td>Encrypted PIN Translate - Translate</td>
<td>Encrypted PIN Translate³</td>
<td>CSNBPTR</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00B7'</td>
<td>Encrypted PIN Translate - Reformat</td>
<td>Encrypted PIN Translate³</td>
<td>CSNBPTR</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00BB'</td>
<td>Clear PIN Generate Alternate - VISA PVV</td>
<td>Clear PIN Generate Alternate¹</td>
<td>CSNBCPA</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>ACP number (hex)</td>
<td>Name of ACP from TKE interface</td>
<td>Verb name</td>
<td>Entry point</td>
<td>Initial setting</td>
<td>Usage</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>----------------</td>
<td>-------</td>
</tr>
<tr>
<td>X'00BC'</td>
<td>PIN Change/Unblock - change EMV PIN with OPINENC</td>
<td>PIN Change/Unblock&lt;sup&gt;1&lt;/sup&gt;</td>
<td>CSNBPCT</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00BD'</td>
<td>PIN Change/Unblock - change EMV PIN with IPINENC</td>
<td>PIN Change/Unblock&lt;sup&gt;1&lt;/sup&gt;</td>
<td>CSNBPCT</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00C3'</td>
<td>Clear Key Import/Multiple Clear Key Import - DES</td>
<td>Clear Key Import Multiple Clear Key Import</td>
<td>CSNBCKI&lt;br&gt;CSNBCKM</td>
<td>ON</td>
<td>SC</td>
</tr>
<tr>
<td>X'00C4'</td>
<td>Secure Key Import - DES, OP</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>ON</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'00CD'</td>
<td>Prohibit Export</td>
<td>Prohibit Export</td>
<td>CSNBPCT</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00D6'</td>
<td>Control Vector Translate</td>
<td>Control Vector Translate</td>
<td>CSNBCVT</td>
<td>ON</td>
<td>SC</td>
</tr>
<tr>
<td>X'00D7'</td>
<td>Key Generate - Key set extended</td>
<td>Key Generate&lt;sup&gt;2&lt;/sup&gt;</td>
<td>CSNBCKG</td>
<td>ON</td>
<td>SC, SUP</td>
</tr>
<tr>
<td>X'00DA'</td>
<td>Cryptographic Variable Encipher</td>
<td>Cryptographic Variable Encipher</td>
<td>CSNBCVE</td>
<td>ON</td>
<td>NRP, O, SUP</td>
</tr>
<tr>
<td>X'00DB'</td>
<td>Key Generate - SINGLE-R</td>
<td>Key Generate&lt;sup&gt;2&lt;/sup&gt;&lt;br&gt;Remote Key Export&lt;sup&gt;2&lt;/sup&gt;</td>
<td>CSNBCKG&lt;br&gt;CSNBCKR</td>
<td>ON</td>
<td>NR, SC</td>
</tr>
<tr>
<td>X'00DC'</td>
<td>Secure Key Import - DES, IM</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>ON</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'00DF'</td>
<td>VISA CVV Generate</td>
<td>CVV Generate</td>
<td>CSNBCSG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00E0'</td>
<td>VISA CVV Verify</td>
<td>CVV Verify</td>
<td>CSNBCSV</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00E1'</td>
<td>UKPT - PIN Verify_PIN Translate</td>
<td>Encrypted PIN Translate&lt;sup&gt;1&lt;/sup&gt;&lt;br&gt;Encrypted PIN Verify&lt;sup&gt;1&lt;/sup&gt;</td>
<td>CSNBPTR&lt;br&gt;CSNBPVR</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00E4'</td>
<td>HMAC Generate - SHA-1</td>
<td>HMAC Generate</td>
<td>CSNBHMG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00E5'</td>
<td>HMAC Generate - SHA-224</td>
<td>HMAC Generate</td>
<td>CSNBHMG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00E6'</td>
<td>HMAC Generate - SHA-256</td>
<td>HMAC Generate</td>
<td>CSNBHMG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00E7'</td>
<td>HMAC Generate - SHA-384</td>
<td>HMAC Generate</td>
<td>CSNBHMG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00E8'</td>
<td>HMAC Generate - SHA-512</td>
<td>HMAC Generate</td>
<td>CSNBHMG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00E9'</td>
<td>Restrict Key Attribute - Export Control</td>
<td>Restrict Key Attribute</td>
<td>CSNBGKA</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00EA'</td>
<td>Key Generate2 - OP</td>
<td>Key Generate2</td>
<td>CSNBKGN2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00EB'</td>
<td>Key Generate2 - Key set</td>
<td>Key Generate2</td>
<td>CSNBKGN2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00EC'</td>
<td>Key Generate2 - Key set extended</td>
<td>Key Generate2</td>
<td>CSNBKGN2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00F1'</td>
<td>Symmetric Key Token Change2 - RTCMK</td>
<td>Key Token Change2</td>
<td>CSNBKTC2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00F2'</td>
<td>Secure Key Import2 - OP</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>ON</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'00F3'</td>
<td>Secure Key Import2 - IM</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>ON</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>ACP number (hex)</td>
<td>Name of ACP from TKE interface</td>
<td>Verb name</td>
<td>Entry point</td>
<td>Initial setting</td>
<td>Usage</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>----------------</td>
<td>-------</td>
</tr>
<tr>
<td>X'00F4'</td>
<td>Symmetric Key Import2 - HMAC, PKCSOAEP2</td>
<td>Symmetric Key Import2</td>
<td>CSNDSYI2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00F5'</td>
<td>Symmetric Key Export - HMAC, PKCSOAEP2</td>
<td>Symmetric Key Export</td>
<td>CSNDSYX</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00F7'</td>
<td>HMAC Verify - SHA-1</td>
<td>HMAC Verify</td>
<td>CSNBHMOV</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00F8'</td>
<td>HMAC Verify - SHA-224</td>
<td>HMAC Verify</td>
<td>CSNBHMOV</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00F9'</td>
<td>HMAC Verify - SHA-256</td>
<td>HMAC Verify</td>
<td>CSNBHMOV</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00FA'</td>
<td>HMAC Verify - SHA-384</td>
<td>HMAC Verify</td>
<td>CSNBHMOV</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00FB'</td>
<td>HMAC Verify - SHA-512</td>
<td>HMAC Verify</td>
<td>CSNBHMOV</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00FC'</td>
<td>Symmetric Key Export - AES, PKOAEP2</td>
<td>Symmetric Key Export</td>
<td>CSNDSYX</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00FD'</td>
<td>Symmetric Key Import2 - AES, PKOAEP2</td>
<td>Symmetric Key Import2</td>
<td>CSNDSYI2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00FE'</td>
<td>PKA Key Translate - Translate internal key token</td>
<td>PKA Key Translate</td>
<td>CSNDPKT</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'00FF'</td>
<td>PKA Key Translate - Translate external key token</td>
<td>PKA Key Translate</td>
<td>CSNDPKT</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0100'</td>
<td>Digital Signature Generate</td>
<td>Digital Signature Generate</td>
<td>CSNDDSG</td>
<td>ON, SC</td>
<td></td>
</tr>
<tr>
<td>X'0101'</td>
<td>Digital Signature Verify</td>
<td>Digital Signature Verify</td>
<td>CSNDDSV</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0102'</td>
<td>PKA Key Token Change RTCMK</td>
<td>PKA Key Token Change</td>
<td>CSNDKTC</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0103'</td>
<td>PKA Key Generate</td>
<td>PKA Key Generate</td>
<td>CSNDPKG</td>
<td>ON, SUP</td>
<td></td>
</tr>
<tr>
<td>X'0104'</td>
<td>PKA Key Import</td>
<td>PKA Key Import</td>
<td>CSNDPKI</td>
<td>ON, SUP</td>
<td></td>
</tr>
<tr>
<td>X'0105'</td>
<td>Symmetric Key Export - DES, PKCS-1.2</td>
<td>Symmetric Key Export</td>
<td>CSNDSYX</td>
<td>ON</td>
<td>SC</td>
</tr>
<tr>
<td>X'0106'</td>
<td>Symmetric Key Import - DES, PKCS-1.2</td>
<td>Symmetric Key Import</td>
<td>CSNDSYI</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0109'</td>
<td>Data Key Import</td>
<td>Data Key Import</td>
<td>CSNBDKM</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'010A'</td>
<td>Data Key Export</td>
<td>Data Key Export</td>
<td>CSNBDKX</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'010B'</td>
<td>SET Block Compose</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>ON</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'010C'</td>
<td>SET Block Decompose</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>ON</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'010D'</td>
<td>Symmetric Key Generate - DES, PKA92</td>
<td>Symmetric Key Generate</td>
<td>CSNDSYG</td>
<td>ON</td>
<td>SC</td>
</tr>
<tr>
<td>X'011E'</td>
<td>PKA Encrypt</td>
<td>PKA Encrypt</td>
<td>CSNDPKE</td>
<td>ON, SEL</td>
<td></td>
</tr>
<tr>
<td>X'011F'</td>
<td>PKA Decrypt</td>
<td>PKA Decrypt</td>
<td>CSNDPKD</td>
<td>ON, SC, SEL</td>
<td></td>
</tr>
<tr>
<td>X'0121'</td>
<td>SET Block Decompose - PIN Extension IPINENC</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>ON</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'0129'</td>
<td>Multiple Clear Key Import/Multiple Secure Key Import - AES</td>
<td>Multiple Clear Key Import</td>
<td>CSNBCKM</td>
<td>ON</td>
<td>SC</td>
</tr>
<tr>
<td>ACP number (hex)</td>
<td>Name of ACP from TKE interface</td>
<td>Verb name</td>
<td>Entry point</td>
<td>Initial setting</td>
<td>Usage</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>-----------------</td>
<td>-------</td>
</tr>
<tr>
<td>X'012A'</td>
<td>Symmetric Algorithm Encipher - secure AES keys</td>
<td>Symmetric Algorithm Encipher</td>
<td>CSNBSAE</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'012B'</td>
<td>Symmetric Algorithm Decipher - secure AES keys</td>
<td>Symmetric Algorithm Decipher</td>
<td>CSNBSAD</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'012C'</td>
<td>Symmetric Key Generate - AES, PKCSOAEP, PKCS-1.2</td>
<td>Symmetric Key Generate</td>
<td>CSNDSYG</td>
<td>ON</td>
<td>SC</td>
</tr>
<tr>
<td>X'012D'</td>
<td>Symmetric Key Generate - AES, ZERO-PAD</td>
<td>Symmetric Key Generate</td>
<td>CSNDSYG</td>
<td>ON</td>
<td>SC</td>
</tr>
<tr>
<td>X'012E'</td>
<td>Symmetric Key Import - AES, PKCSOAEP, PKCS-1.2</td>
<td>Symmetric Key Import</td>
<td>CSNDSYI</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'012F'</td>
<td>Symmetric Key Import - AES, ZERO-PAD</td>
<td>Symmetric Key Import</td>
<td>CSNDSYI</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0130'</td>
<td>Symmetric Key Export - AES, PKCSOAEP, PKCS-1.2</td>
<td>Symmetric Key Export</td>
<td>CSNDSYX</td>
<td>ON</td>
<td>SC</td>
</tr>
<tr>
<td>X'0131'</td>
<td>Symmetric Key Export - AES, ZERO-PAD</td>
<td>Symmetric Key Export</td>
<td>CSNDSYX</td>
<td>ON</td>
<td>SC</td>
</tr>
<tr>
<td>X'013D'</td>
<td>Diversified Key Generate - Allow wrapping override keywords</td>
<td>Diversified Key Generate</td>
<td>CSNBDKG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'013E'</td>
<td>Symmetric Key Generate - Allow wrapping override keywords</td>
<td>Symmetric Key Generate</td>
<td>CSNDSYG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'013F'</td>
<td>Remote Key Export - include RKX in default wrap config</td>
<td>Remote Key Export</td>
<td>CSNDRKX</td>
<td>ON</td>
<td>SC</td>
</tr>
<tr>
<td>X'0140'</td>
<td>Key Part Import - Allow wrapping override keywords</td>
<td>Key Part Import</td>
<td>CSNBKPI</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0141'</td>
<td>Multiple Clear Key Import - Allow wrapping override keywords</td>
<td>Multiple Clear Key Import</td>
<td>CSNBCKM</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0142'</td>
<td>Multiple Secure Key Import - Allow wrapping override keywords</td>
<td>This ACP is for verbs CSNBSKI / CSNBSKM but these are not supported on Linux on Z, but are supported on z/OS.</td>
<td>ON</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'0144'</td>
<td>Symmetric Key Import - Allow wrapping override keywords</td>
<td>Symmetric Key Import</td>
<td>CSNDSYI</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0149'</td>
<td>Key Translate2</td>
<td>Key Translate2</td>
<td>CSNBKTR2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'014A'</td>
<td>Key Translate2 - Allow wrapping override keywords</td>
<td>Key Translate2</td>
<td>CSNBKTR2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'014D'</td>
<td>TR31 Export - Permit version A TR-31 key blocks</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'014E'</td>
<td>TR31 Export - Permit version B TR-31 key blocks</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'014F'</td>
<td>TR31 Export - Permit version C TR-31 key blocks</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0150'</td>
<td>TR31 Import - Permit version A TR-31 key blocks</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0151'</td>
<td>TR31 Import - Permit version B TR-31 key blocks</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>ACP number (hex)</td>
<td>Name of ACP from TKE interface</td>
<td>Verb name</td>
<td>Entry point</td>
<td>Initial setting</td>
<td>Usage</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>----------------</td>
<td>-------</td>
</tr>
<tr>
<td>X'0152'</td>
<td>TR31 Import - Permit version C TR-31 key blocks</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0153'</td>
<td>TR31 Import - Permit override of default wrapping method</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>ON</td>
<td>O, SC</td>
</tr>
<tr>
<td>X'0154'</td>
<td>Restrict Key Attribute - Permit setting the TR-31 export bit</td>
<td>Restrict Key Attribute</td>
<td>CSNBRKA</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0155'</td>
<td>CVV Key Combine</td>
<td>CVV Key Combine</td>
<td>CSNBCKC</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0156'</td>
<td>CVV Key Combine - Allow wrapping override keywords</td>
<td>CVV Key Combine</td>
<td>CSNBCKC</td>
<td>ON</td>
<td>O, SC</td>
</tr>
<tr>
<td>X'0157'</td>
<td>CVV Key Combine - Permit mixed key types</td>
<td>CVV Key Combine</td>
<td>CSNBCKC</td>
<td>ON</td>
<td>O, SC</td>
</tr>
<tr>
<td>X'0158'</td>
<td>TR31 Export - Permit any CCA key if INCL-CV is specified</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O, SC</td>
</tr>
<tr>
<td>X'0159'</td>
<td>TR31 Import - Permit C0 to MAC/MACVER:CVVKEY-A</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O, SC</td>
</tr>
<tr>
<td>X'015B'</td>
<td>TR31 Import - Permit C0 to MAC/MACVER:AMEX-CSC</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O, SC</td>
</tr>
<tr>
<td>X'015C'</td>
<td>TR31 Import - Permit K0:E to EXPORTER/OKEYXLAT</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O, SC</td>
</tr>
<tr>
<td>X'015D'</td>
<td>TR31 Import - Permit K0:D to IMPORTER/IKEYXLAT</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'015E'</td>
<td>TR31 Import - Permit K0:B to EXPORTER/OKEYXLAT</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'015F'</td>
<td>TR31 Import - Permit K1:E to IMPORTER/IKEYXLAT</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0160'</td>
<td>TR31 Import - Permit K1:D to IMPORTER/IKEYXLAT</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0161'</td>
<td>TR31 Import - Permit K1:B to IMPORTER/OKEYXLAT</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0162'</td>
<td>TR31 Import - Permit K0:E to EXPORTER/OKEYXLAT</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0163'</td>
<td>TR31 Import - Permit K0:D to IMPORTER/IKEYXLAT</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0164'</td>
<td>TR31 Import - Permit M0/M1/M3 to MAC/MACVER:ANY-MAC</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0165'</td>
<td>TR31 Import - Permit P0:E to OPINENC</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0166'</td>
<td>TR31 Import - Permit P0:D to IPINENC</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0167'</td>
<td>TR31 Import - Permit V0 to PINGEN:NO-SPEC</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0168'</td>
<td>TR31 Import - Permit V0 to PINVER:NO-SPEC</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0169'</td>
<td>TR31 Import - Permit V1 to PINGEN:IBM-PIN/IBM-PINO</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>ACP number (hex)</td>
<td>Name of ACP from TKE interface</td>
<td>Verb name</td>
<td>Entry point</td>
<td>Initial setting</td>
<td>Usage</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>-----------------</td>
<td>-------</td>
</tr>
<tr>
<td>X'016A'</td>
<td>TR31 Import - Permit V1 to PINVER:IBM-PIN/IBM-PINO</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'016B'</td>
<td>TR31 Import - Permit V2 to PINGEN:VISA-PVV</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'016C'</td>
<td>TR31 Import - Permit V2 to PINGEN:VISA-PVV</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'016D'</td>
<td>TR31 Import - Permit E0 to DKYGENKY:DKYL0+DMAC</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'016E'</td>
<td>TR31 Import - Permit E0 to DKYGENKY:DKYL0+DMV</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'016F'</td>
<td>TR31 Import - Permit E0 to DKYGENKY:DKYL1+DMAC</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0170'</td>
<td>TR31 Import - Permit E0 to DKYGENKY:DKYL1+DMV</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0171'</td>
<td>TR31 Import - Permit E1 to DKYGENKY:DKYL0+DMPIN</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0172'</td>
<td>TR31 Import - Permit E1 to DKYGENKY:DKYL0+DDATA</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0173'</td>
<td>TR31 Import - Permit E1 to DKYGENKY:DKYL1+DMPIN</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0174'</td>
<td>TR31 Import - Permit E1 to DKYGENKY:DKYL1+DDATA</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0175'</td>
<td>TR31 Import - Permit E2 to DKYGENKY:DKYL0+DDATA</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0176'</td>
<td>TR31 Import - Permit E2 to DKYGENKY:DKYL1+DMAC</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0177'</td>
<td>TR31 Import - Permit E3 to ENCIPHER</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0178'</td>
<td>TR31 Import - Permit E4 to DKYGENKY:DKYL0+DDATA</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0179'</td>
<td>TR31 Import - Permit E5 to DKYGENKY:DKYL0+DMAC</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'017A'</td>
<td>TR31 Import - Permit E5 to DKYGENKY:DKYL0+DDATA</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'017B'</td>
<td>TR31 Import - Permit E5 to DKYGENKY:DKYL0+DEXP</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'017C'</td>
<td>TR31 Import - Permit V0/V1/V2:N to PINGEN/PINVER</td>
<td>TR31 Key Import</td>
<td>CSNBT31I</td>
<td>OFF</td>
<td>O, SC</td>
</tr>
<tr>
<td>X'0180'</td>
<td>TR31 Export - Permit KEYGENKY:UKPT to B0</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0181'</td>
<td>TR31 Export - Permit MAC/MACVER:AMEX-CSC to C0:G/C/V</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0182'</td>
<td>TR31 Export - Permit MAC/MACVER:CVV-KEYA to C0:G/C/V</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
</tbody>
</table>
### Table 306. Access Control Points and corresponding CCA verbs (continued)

<table>
<thead>
<tr>
<th>ACP number (hex)</th>
<th>Name of ACP from TKE interface</th>
<th>Verb name</th>
<th>Entry point</th>
<th>Initial setting</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0183'</td>
<td>TR31 Export - Permit MAC/MACVER:ANY-MAC to C0:G/C/V</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0184'</td>
<td>TR31 Export - Permit DATA to C0:G/C</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0185'</td>
<td>TR31 Export - Permit ENCIPHER/DECIPHER/CRYPTO function to DO:E/D/B</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0186'</td>
<td>TR31 Export - Permit DATA to DO:B</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0187'</td>
<td>TR31 Export - Permit KEYXLAT to K0:E</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0188'</td>
<td>TR31 Export - Permit KEYXLAT to K0:D</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0189'</td>
<td>TR31 Export - Permit KEYXLAT to K1:E</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'018A'</td>
<td>TR31 Export - Permit KEYXLAT to K1:D</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'018B'</td>
<td>TR31 Export - Permit MAC/MACVER to M0:G/C</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'018C'</td>
<td>TR31 Export - Permit MAC/MACVER to M0:V</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'018D'</td>
<td>TR31 Export - Permit MAC/MACVER to M1:G/C</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'018E'</td>
<td>TR31 Export - Permit MAC/MACVER to M1:V</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'018F'</td>
<td>TR31 Export - Permit MAC/MACVER to M3:G/C</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0190'</td>
<td>TR31 Export - Permit MAC/MACVER to M3:V</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0191'</td>
<td>TR31 Export - Permit PINENC to P0:E</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0192'</td>
<td>TR31 Export - Permit PINENC to P0:D</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0193'</td>
<td>TR31 Export - Permit PINVER:NO-SPEC to V0</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0194'</td>
<td>TR31 Export - Permit PINENC:NO-SPEC to V0</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0195'</td>
<td>TR31 Export - Permit PINVER:NO-SPEC/IBM-PIN/IBM-PIN to V1</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0196'</td>
<td>TR31 Export - Permit PINENC:NO-SPEC/IBM-PIN/IBM-PIN to V1</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0197'</td>
<td>TR31 Export - Permit PINVER:NO-SPEC/NO-PIN to V2</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
</tbody>
</table>
Table 306. Access Control Points and corresponding CCA verbs (continued)

<table>
<thead>
<tr>
<th>ACP number (hex)</th>
<th>Name of ACP from TKE interface</th>
<th>Verb name</th>
<th>Entry point</th>
<th>Initial setting</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0198'</td>
<td>TR31 Export - Permit PINGEN:NO-SPEC/VISA-PVV to V2</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0199'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DMAC to E0</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'019A'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DMV to E0</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'019B'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DALL to E0</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'019C'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DMAC to E0</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'019D'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DMV to E0</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'019E'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DALL to E0</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'019F'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DDATA to E1</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'01A0'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DMPIN to E1</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'01A1'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DALL to E1</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'01A2'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DDATA to E1</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'01A3'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DMPIN to E1</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'01A4'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DALL to E1</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'01A5'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DMAC to E2</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'01A6'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DDATA to E2</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'01A7'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DALL to E2</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'01A8'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DMAC to E2</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'01A9'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL1+DALL to E2</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'01AA'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DDATA to E3</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'01AB'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DMPIN to E3</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'01AC'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DALL to E3</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'01AD'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DDATA to E4</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'01AE'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DMPIN to E4</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O</td>
</tr>
</tbody>
</table>

Chapter 24. Access control points and verbs 1059
Table 306. Access Control Points and corresponding CCA verbs (continued)

<table>
<thead>
<tr>
<th>ACP number (hex)</th>
<th>Name of ACP from TKE interface</th>
<th>Verb name</th>
<th>Entry point</th>
<th>Initial setting</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'01AF'</td>
<td>TR31 Export - Permit DKYGENKY:DKYL0+DALL to E5</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'01B0'</td>
<td>TR31 Export - Permit PINGEN/PINVER to VO/V1/V2:N</td>
<td>Key Export to TR31</td>
<td>CSNBT31X</td>
<td>OFF</td>
<td>O, SC</td>
</tr>
<tr>
<td>X'01B1'</td>
<td>Public Infrastructure Management</td>
<td>Public Infrastructure Manage</td>
<td>CSNDPIM</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>Note: This ACP is required for a TKE service.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'01B2'</td>
<td>PIM: Load Root Certificate</td>
<td>Public Infrastructure Manage</td>
<td>CSNDPIM</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>Note: This ACP is required for a TKE service.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'01B3'</td>
<td>PIM: Activate Root Certificate</td>
<td>Public Infrastructure Manage</td>
<td>CSNDPIM</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>Note: This ACP is required for a TKE service.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'01B4'</td>
<td>PIM: Renew Certificate</td>
<td>Public Infrastructure Manage</td>
<td>CSNDPIM</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>Note: This ACP is required for a TKE service.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'01B5'</td>
<td>Change Certificate Label</td>
<td>Public Infrastructure Manage</td>
<td>CSNDPIM</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>Note: This ACP is required for a TKE service.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'01B6'</td>
<td>Delete Certificate</td>
<td>Public Infrastructure Manage</td>
<td>CSNDPIM</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>Note: This ACP is required for a TKE service.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'01B7'</td>
<td>Signature Algorithm SHA+RSA</td>
<td>Public Infrastructure Manage</td>
<td>CSNDPIM</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>Note: This ACP is required for a TKE service.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'01B8'</td>
<td>Signature Algorithm ECDSA</td>
<td>Public Infrastructure Manage</td>
<td>CSNDPIM</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>Note: This ACP is required for a TKE service.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'01B9'</td>
<td>Signature Algorithm RSASSA_PSS</td>
<td>Public Infrastructure Manage</td>
<td>CSNDPIM</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>Note: This ACP is required for a TKE service.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACP number (hex)</td>
<td>Name of ACP from TKE interface</td>
<td>Verb name</td>
<td>Entry point</td>
<td>Initial setting</td>
<td>Usage</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>-----------------</td>
<td>-------</td>
</tr>
<tr>
<td>X'01BA'</td>
<td>Signature Algorithm SHA-1</td>
<td>Public Infrastructure Manage</td>
<td>CSNDPIM Note: This ACP is required for a TKE service.</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'01BB'</td>
<td>Signature Algorithm SHA-224</td>
<td>Public Infrastructure Manage</td>
<td>CSNDPIM Note: This ACP is required for a TKE service.</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'01BC'</td>
<td>Signature Algorithm SHA-256</td>
<td>Public Infrastructure Manage</td>
<td>CSNDPIM Note: This ACP is required for a TKE service.</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'01BD'</td>
<td>Signature Algorithm SHA-384</td>
<td>Public Infrastructure Manage</td>
<td>CSNDPIM Note: This ACP is required for a TKE service.</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'01BE'</td>
<td>Signature Algorithm SHA-512</td>
<td>Public Infrastructure Manage</td>
<td>CSNDPIM Note: This ACP is required for a TKE service.</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'01BF'</td>
<td>Load Sub-CA Certificate</td>
<td>Public Infrastructure Manage</td>
<td>CSNDPIM Note: This ACP is required for a TKE service.</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'01C0'</td>
<td>Cipher Text Translate2</td>
<td>Cipher Text Translate2</td>
<td>CSNBCTT2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'01C1'</td>
<td>Cipher Text Translate2 - Allow translate from AES to TDES</td>
<td>Cipher Text Translate2</td>
<td>CSNBCTT2</td>
<td>ON</td>
<td>SC</td>
</tr>
<tr>
<td>X'01C2'</td>
<td>Cipher Text Translate2 - Allow translate to weaker AES</td>
<td>Cipher Text Translate2</td>
<td>CSNBCTT2</td>
<td>ON</td>
<td>SC</td>
</tr>
<tr>
<td>X'01C3'</td>
<td>Cipher Text Translate2 - Allow translate to weaker DES</td>
<td>Cipher Text Translate2</td>
<td>CSNBCTT2</td>
<td>ON</td>
<td>SC</td>
</tr>
<tr>
<td>X'01C4'</td>
<td>Cipher Text Translate2 - Allow only cipher text translate types</td>
<td>Cipher Text Translate2</td>
<td>CSNBCTT2</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'01C8'</td>
<td>Unique Key Derive</td>
<td>Unique Key Derive</td>
<td>CSNBUKD</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'01C9'</td>
<td>Unique Key Derive - Allow PIN-DATA processing</td>
<td>Unique Key Derive</td>
<td>CSNBUKD</td>
<td>OFF</td>
<td>NR</td>
</tr>
<tr>
<td>X'01CA'</td>
<td>Unique Key Derive - Override default wrapping</td>
<td>Unique Key Derive</td>
<td>CSNBUKD</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'01CB'</td>
<td>Key Test - Warn when keyword inconsistent with key length</td>
<td>Key Test Extended</td>
<td>CSNBKYTX</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'01CC'</td>
<td>Access Control Tracking - Enable</td>
<td>Access Control Tracking</td>
<td>CSUAACCT</td>
<td>OFF</td>
<td>O</td>
</tr>
</tbody>
</table>
Table 306. Access Control Points and corresponding CCA verbs  (continued)

<table>
<thead>
<tr>
<th>ACP number (hex)</th>
<th>Name of ACP from TKE interface</th>
<th>Verb name</th>
<th>Entry point</th>
<th>Initial setting</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>X’01CD’</td>
<td>Symmetric Algorithm Encipher - AES GCM</td>
<td>Symmetric Algorithm Encipher</td>
<td>CSNBSAE</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X’01CE’</td>
<td>Symmetric Algorithm Decipher - AES GCM</td>
<td>Symmetric Algorithm Decipher</td>
<td>CSNBSAD</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X’0203’</td>
<td>Retained Key Delete</td>
<td>Retained Key Delete</td>
<td>CSNDRKD</td>
<td>ON</td>
<td>O, SEL</td>
</tr>
<tr>
<td>X’0204’</td>
<td>PKA Key Generate - Clone</td>
<td>PKA Key Generate^1</td>
<td>CSNDPKG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X’0205’</td>
<td>PKA Key Generate - Clear</td>
<td>PKA Key Generate^1</td>
<td>CSNDPKG</td>
<td>ON</td>
<td>O, SUP</td>
</tr>
<tr>
<td>X’0206’</td>
<td>PKA Encipher - Clear Key Disallow PKCS-1.2</td>
<td>PKA Encrypt</td>
<td>CSNDPKE</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X’0207’</td>
<td>PKA Encipher - Clear Key Disallow ZERO-PAD</td>
<td>PKA Encrypt</td>
<td>CSNDPKE</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X’0208’</td>
<td>PKA Encipher - Clear Key Disallow MRP</td>
<td>PKA Encrypt</td>
<td>CSNDPKE</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X’0209’</td>
<td>PKA Encipher - Clear Key Disallow PKCSOAEP</td>
<td>PKA Encrypt</td>
<td>CSNDPKE</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X’020A’</td>
<td>PKA Decipher - Key Data Disallow PKCS-1.2</td>
<td>PKA Decrypt</td>
<td>CSNDPKD</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X’020B’</td>
<td>PKA Decipher - Key Data Disallow ZERO-PAD</td>
<td>PKA Decrypt</td>
<td>CSNDPKD</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X’020C’</td>
<td>PKA Decipher - Key Data Disallow PKCSOAEP</td>
<td>PKA Decrypt</td>
<td>CSNDPKD</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X’0230’</td>
<td>Retained Key List</td>
<td>Retained Key List</td>
<td>CSNDRKL</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X’0235’</td>
<td>Symmetric Key Import - DES, PKA92 KEK</td>
<td>Symmetric Key Import^1</td>
<td>CSNSYI</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X’023C’</td>
<td>Symmetric Key Generate - DES, ZERO-PAD</td>
<td>Symmetric Key Generate^3</td>
<td>CSNDYG</td>
<td>ON</td>
<td>O, SC</td>
</tr>
<tr>
<td>X’023D’</td>
<td>Symmetric Key Import - DES, ZERO-PAD</td>
<td>Symmetric Key Import^1</td>
<td>CSNSYI</td>
<td>ON</td>
<td>O, SC</td>
</tr>
<tr>
<td>X’023E’</td>
<td>Symmetric Key Export - DES, ZERO-PAD</td>
<td>Symmetric Key Export^1</td>
<td>CSNSYX</td>
<td>ON</td>
<td>O, SC</td>
</tr>
<tr>
<td>X’023F’</td>
<td>Symmetric Key Generate - DES, PKCS-1.2</td>
<td>Symmetric Key Generate^3</td>
<td>CSNDYG</td>
<td>ON</td>
<td>O, SC</td>
</tr>
<tr>
<td>X’0261’</td>
<td>TKE Authorization for domain 0</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X’0262’</td>
<td>TKE Authorization for domain 1</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X’0263’</td>
<td>TKE Authorization for domain 2</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X’0264’</td>
<td>TKE Authorization for domain 3</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>ACP number (hex)</td>
<td>Name of ACP from TKE interface</td>
<td>Verb name</td>
<td>Entry point</td>
<td>Initial setting</td>
<td>Usage</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>----------------</td>
<td>-------</td>
</tr>
<tr>
<td>X'0265'</td>
<td>TKE Authorization for domain 4</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'0266'</td>
<td>TKE Authorization for domain 5</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'0267'</td>
<td>TKE Authorization for domain 6</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'0268'</td>
<td>TKE Authorization for domain 7</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'0269'</td>
<td>TKE Authorization for domain 8</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'026A'</td>
<td>TKE Authorization for domain 9</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'026B'</td>
<td>TKE Authorization for domain 10</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'026C'</td>
<td>TKE Authorization for domain 11</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'026D'</td>
<td>TKE Authorization for domain 12</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'026E'</td>
<td>TKE Authorization for domain 13</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'026F'</td>
<td>TKE Authorization for domain 14</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'0270'</td>
<td>TKE Authorization for domain 15</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'0273'</td>
<td>Secure Messaging for Keys</td>
<td>Secure Messaging for Keys</td>
<td>CSNBSKY</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0274'</td>
<td>Secure Messaging for PINs</td>
<td>Secure Messaging for PINs</td>
<td>CSNBSNP</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0275'</td>
<td>DATAM Key Management Control</td>
<td>Diversified Key Generate Data Key Import Data Key Export Key Export Key Generate Key Import</td>
<td>CSNBDKG CSNBDKM CSNBDXX CSNBKEX CSNBKGN CSNBRKM</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0276'</td>
<td>Key Export - Unrestricted</td>
<td>Key Export</td>
<td>CSNBKEX</td>
<td>ON</td>
<td>O, SC</td>
</tr>
<tr>
<td>X'0277'</td>
<td>Data Key Export - Unrestricted</td>
<td>Data Key Export</td>
<td>CSNBDXX</td>
<td>ON</td>
<td>O, SC</td>
</tr>
<tr>
<td>X'0278'</td>
<td>Key Part Import - ADD-PART</td>
<td>Key Part Import</td>
<td>CSNKBPI</td>
<td>ON</td>
<td>SC, SEL</td>
</tr>
<tr>
<td>ACP number (hex)</td>
<td>Name of ACP from TKE interface</td>
<td>Verb name</td>
<td>Entry point</td>
<td>Initial setting</td>
<td>Usage</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>----------------</td>
<td>-------</td>
</tr>
<tr>
<td>X'0279'</td>
<td>Key Part Import - COMPLETE</td>
<td>Key Part Import</td>
<td>CSNBKPI</td>
<td>ON</td>
<td>SC, SEL</td>
</tr>
<tr>
<td>X'027A'</td>
<td>Key Part Import - Unrestricted</td>
<td>Key Part Import</td>
<td>CSNBKPI</td>
<td>ON</td>
<td>O, SC</td>
</tr>
<tr>
<td>X'027B'</td>
<td>Key Import - Unrestricted</td>
<td>Key Import</td>
<td>CSNBKIM</td>
<td>ON</td>
<td>O, SC</td>
</tr>
<tr>
<td>X'027C'</td>
<td>Data Key Import - Unrestricted</td>
<td>Data Key Import</td>
<td>CSNBDKM</td>
<td>ON</td>
<td>O, SC</td>
</tr>
<tr>
<td>X'027D'</td>
<td>PKA Key Generate - Permit</td>
<td>PKA Key Generate</td>
<td>CSNBDKG</td>
<td>ON</td>
<td>O, NRP, SC</td>
</tr>
<tr>
<td>X'027E'</td>
<td>PKA Key Generate - Permit</td>
<td>PKA Key Generate</td>
<td>CSNBDKG</td>
<td>ON</td>
<td>O, NRP, SC</td>
</tr>
<tr>
<td>X'0290'</td>
<td>Diversified Key Generate -</td>
<td>Diversified Key Generate</td>
<td>CSNBDKG</td>
<td>OFF</td>
<td>O, SC</td>
</tr>
<tr>
<td>X'0291'</td>
<td>Transaction Validation -</td>
<td>Transaction Validation</td>
<td>CSNBDKG</td>
<td>ON</td>
<td>O, SEL</td>
</tr>
<tr>
<td>X'0292'</td>
<td>Transaction Validation -</td>
<td>Transaction Validation</td>
<td>CSNBDKG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0293'</td>
<td>Transaction Validation -</td>
<td>Transaction Validation</td>
<td>CSNBDKG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0294'</td>
<td>Transaction Validation -</td>
<td>Transaction Validation</td>
<td>CSNBDKG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0295'</td>
<td>High-performance secure DES</td>
<td>Enables CPACF key translation for DES keys.</td>
<td>N/A</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0296'</td>
<td>High-performance secure AES</td>
<td>Enables CPACF key translation for AES keys.</td>
<td>N/A</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0297'</td>
<td>Key Part Import2 - Load first</td>
<td>Key Part Import2</td>
<td>CSNBKPI2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0298'</td>
<td>Key Part Import2 - Load first</td>
<td>Key Part Import2</td>
<td>CSNBKPI2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0299'</td>
<td>Key Part Import2 - Load first</td>
<td>Key Part Import2</td>
<td>CSNBKPI2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'029A'</td>
<td>Key Part Import2 - Add second</td>
<td>Key Part Import2</td>
<td>CSNBKPI2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'029B'</td>
<td>Key Part Import2 - Add last</td>
<td>Key Part Import2</td>
<td>CSNBKPI2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'029C'</td>
<td>Key Part Import2 - Add optional</td>
<td>Key Part Import2</td>
<td>CSNBKPI2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'029D'</td>
<td>Key Part Import2 - Complete</td>
<td>Key Part Import2</td>
<td>CSNBKPI2</td>
<td>ON</td>
<td>SEL</td>
</tr>
<tr>
<td>X'029E'</td>
<td>Operational Key Load -</td>
<td>Key Part Import2</td>
<td>CSNBKPI2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'029F'</td>
<td>Recover PIN from Offset</td>
<td>Recover PIN from Offset</td>
<td>CSNBKPI2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02B1'</td>
<td>Authentication Parameter</td>
<td>Authentication Parameter Generate</td>
<td>CSNBKPI2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02B2'</td>
<td>Authentication Parameter</td>
<td>Authentication Parameter Generate</td>
<td>CSNBKPI2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02B3'</td>
<td>Symmetric Key Export - AESKWCV</td>
<td>Symmetric Key Export</td>
<td>CSNBKPI2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>ACP number (hex)</td>
<td>Name of ACP from TKE interface</td>
<td>Verb name</td>
<td>Entry point</td>
<td>Initial setting</td>
<td>Usage</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>----------------</td>
<td>-------</td>
</tr>
<tr>
<td>X'02B4'</td>
<td>Symmetric Key Import2 - AESKCV</td>
<td>Symmetric Key Import2</td>
<td>CSNDSYI2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02B5'</td>
<td>Symmetric Key Export with Data</td>
<td>Symmetric Key Export with Data</td>
<td>CSNDSXD</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02B6'</td>
<td>Symmetric Key Export with Data - Special</td>
<td>Symmetric Key Export with Data</td>
<td>CSNDSXD</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02B8'</td>
<td>Diversified Key Generate - TDES-CBC</td>
<td>Diversified Key Generate</td>
<td>CSNBDKG2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02B9'</td>
<td>Symmetric Key Import2 - Allow wrapping override keywords</td>
<td>Symmetric Key Import2</td>
<td>CSNDSYI2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02BA'</td>
<td>Remote Key Export - Allow wrapping override keywords</td>
<td>Remote Key Export</td>
<td>CSNDRKX</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02BB'</td>
<td>Key Generate2 - DK PIN key set PINPROT</td>
<td>Key Generate2</td>
<td>CSNBKGN2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02BC'</td>
<td>Key Generate2 - DK PIN print key PINPROT</td>
<td>Key Generate2</td>
<td>CSNBKGN2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02BD'</td>
<td>Key Generate2 - DK PIN admin1 key set PINPROT</td>
<td>Key Generate2</td>
<td>CSNBKGN2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02BE'</td>
<td>Key Generate2 - DK PIN admin1 key set MAC</td>
<td>Key Generate2</td>
<td>CSNBKGN2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02BF'</td>
<td>Key Generate2 - DK PIN admin2 key set MAC</td>
<td>Key Generate2</td>
<td>CSNBKGN2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02C0'</td>
<td>DK Random PIN Generate</td>
<td>DK Random PIN Generate</td>
<td>CSNBDRPG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02C1'</td>
<td>DK PIN Verify</td>
<td>DK PIN Verify</td>
<td>CSNBPDV</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02C2'</td>
<td>DK PIN Change</td>
<td>DK PIN Change</td>
<td>CSNBPDPC</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02C3'</td>
<td>DK PRW Card Number Update</td>
<td>DK PRW Card Number Update</td>
<td>CSNBPDNU</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02C4'</td>
<td>DK PRW CMAC Generate</td>
<td>DK PRW CMAC Generate</td>
<td>CSNBPDPCG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02C5'</td>
<td>DK PAN Modify in Transaction</td>
<td>DK PAN Modify in Transaction</td>
<td>CSNBPDPM</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02C6'</td>
<td>DK Deterministic PIN Generate</td>
<td>DK Deterministic PIN Generate</td>
<td>CSNBDDPG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02C7'</td>
<td>DK PAN Translate</td>
<td>DK PAN Translate</td>
<td>CSNBPDPT</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02C8'</td>
<td>DK Regenerate PRW</td>
<td>DK Regenerate PRW</td>
<td>CSNBDRP</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02CC'</td>
<td>Diversified Key Generate2 - AES EMV1 SESS</td>
<td>Diversified Key Generate2</td>
<td>CSNBDKG2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02CD'</td>
<td>Diversified Key Generate2 - DALL</td>
<td>Diversified Key Generate2</td>
<td>CSNBDKG2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02CE'</td>
<td>DK Migrate PIN</td>
<td>DK Migrate PIN</td>
<td>CSNBDMG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02CF'</td>
<td>FPE Encrypt</td>
<td>FPE Encrypt</td>
<td>CSNBFPDE</td>
<td>ON</td>
<td>ID, R</td>
</tr>
<tr>
<td>X'02D0'</td>
<td>FPE Decrypt</td>
<td>FPE Decrypt</td>
<td>CSNBFPED</td>
<td>ON</td>
<td>ID, R</td>
</tr>
<tr>
<td>X'02D1'</td>
<td>FPE Translate</td>
<td>FPE Translate</td>
<td>CSNBFPET</td>
<td>ON</td>
<td>ID, R</td>
</tr>
<tr>
<td>X'02D2'</td>
<td>Diversified Key Generate2 - MK-OPTC</td>
<td>Diversified Key Generate2</td>
<td>CSNBDKG2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02D3'</td>
<td>Diversified Key Generate2 - KDFFM-DK</td>
<td>Diversified Key Generate2</td>
<td>CSNBDKG2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'02D4'</td>
<td>Diversified Key Generate2 - KDFFM-DK</td>
<td>Diversified Key Generate2</td>
<td>CSNBDKG2</td>
<td>ON</td>
<td>O</td>
</tr>
</tbody>
</table>
Table 306. Access Control Points and corresponding CCA verbs (continued)

<table>
<thead>
<tr>
<th>ACP number (hex)</th>
<th>Name of ACP from TKE interface</th>
<th>Verb name</th>
<th>Entry point</th>
<th>Initial setting</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'02D5'</td>
<td>Encrypted PIN Translate Enhanced</td>
<td>Encrypted PIN Translate Enhanced</td>
<td>CSNBPTRE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'02D6'</td>
<td>DM load role</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>Pri</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'02D7'</td>
<td>DM load profile</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>Pri</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'02D8'</td>
<td>DM load role cos</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>Pri</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'02D9'</td>
<td>DM load profile cos</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>Pri</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'02DA'</td>
<td>DM delete role</td>
<td>Access Control Maintenance</td>
<td>CSUAACM</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'02DB'</td>
<td>DM delete profile</td>
<td>Access Control Maintenance</td>
<td>CSUAACM</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'02DC'</td>
<td>DM delete role cos</td>
<td>Access Control Maintenance</td>
<td>CSUAACM</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'02DD'</td>
<td>DM delete profile cos</td>
<td>Access Control Maintenance</td>
<td>CSUAACM</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'02E0'</td>
<td>CFC:COMPIMPR</td>
<td>Note: This ACP is included for TKE reference only.</td>
<td>Pri</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'02E1'</td>
<td>CFC:COMPIMPR cos</td>
<td>Note: This ACP is included for TKE reference only.</td>
<td>Pri</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'02E2'</td>
<td>CFC:COMP-SET</td>
<td>Note: This ACP is included for TKE reference only.</td>
<td>Pri</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'02E3'</td>
<td>CFC:COMP-SET cos</td>
<td>Note: This ACP is included for TKE reference only.</td>
<td>Pri</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'02E4'</td>
<td>CFC:COMP-RMV</td>
<td>Note: This ACP is included for TKE reference only.</td>
<td>Pri</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'02E5'</td>
<td>CFC:COMP-RMV cos</td>
<td>Note: This ACP is included for TKE reference only.</td>
<td>Pri</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'02E6'</td>
<td>CFC:COMP-RMV imprint mode</td>
<td>Note: This ACP is included for TKE reference only.</td>
<td>Pri</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'02E7'</td>
<td>CFC:COMPMIGB</td>
<td>Note: This ACP is included for TKE reference only.</td>
<td>Pri</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'02E8'</td>
<td>CFC:COMPMIGB cos</td>
<td>Note: This ACP is included for TKE reference only.</td>
<td>Pri</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'02E9'</td>
<td>CFC:COMPMIGE</td>
<td>Note: This ACP is included for TKE reference only.</td>
<td>Pri</td>
<td>OFF</td>
<td></td>
</tr>
</tbody>
</table>
Table 306. Access Control Points and corresponding CCA verbs (continued)

<table>
<thead>
<tr>
<th>ACP number (hex)</th>
<th>Name of ACP from TKE interface</th>
<th>Verb name</th>
<th>Entry point</th>
<th>Initial setting</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'02EA'</td>
<td>CFC:COMPMIGE cos</td>
<td>Note: This ACP is included for TKE reference only.</td>
<td>OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'02EB'</td>
<td>IGN_EFF_MK (Allow weak wrapping of compliance-tagged keys by DES MK)</td>
<td>All callable services that use PCI-HSM 2016 compliant-tagged DES key tokens.</td>
<td>OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'02EC'</td>
<td>IGN_RKA_DATAIXMAC</td>
<td>Reserved for future use.</td>
<td>OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'02ED'</td>
<td>CMD_RKA_DATAIXCIP</td>
<td>Reserved for future use.</td>
<td>OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'02EE'</td>
<td>CMD_PKT_INTUSCHG</td>
<td>Reserved for future use.</td>
<td>OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'02EF'</td>
<td>CMD_PKT_EXTUSCHG</td>
<td>Reserved for future use.</td>
<td>OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'02F0'</td>
<td>PUB_X_MACDPUB</td>
<td>Reserved for future use.</td>
<td>OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'02F1'</td>
<td>RSAPRV_X_MACDPUB</td>
<td>Reserved for future use.</td>
<td>OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'02F2'</td>
<td>ECCPRV_X_MACDPUB</td>
<td>Reserved for future use.</td>
<td>OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'02F3'</td>
<td>X509_X_MACDPUB</td>
<td>Reserved for future use.</td>
<td>OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'02F4'</td>
<td>ALLOW_SHA1_X509</td>
<td>Multiple services that accept the X.509 certificates.</td>
<td>OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'02F5'</td>
<td>Authorized Key Export - SEITNKEY</td>
<td>Note: This ACP is included for TKE reference only.</td>
<td>ON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'02F6'</td>
<td>Authorized Key Export - DRVITKEY</td>
<td>Note: This ACP is included for TKE reference only.</td>
<td>ON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'02F7'</td>
<td>Authorized Key Export - EXPTSK</td>
<td>Note: This ACP is included for TKE reference only.</td>
<td>ON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'02F8'</td>
<td>Key Translate2 - COMP-TAG</td>
<td>Key Translate2</td>
<td>CSNBKTR2</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>X'02F9'</td>
<td>Key Translate2 - COMP-CHECK</td>
<td>Key Translate2</td>
<td>CSNBKTR2</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>X'0300'</td>
<td>NOCV KEK usage for export-related functions</td>
<td>Data Key Export</td>
<td>CSNBKEX</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key Export</td>
<td>CSNBKEX</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key Generate</td>
<td>CSNBKGN</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remote Key Export</td>
<td>CSNDRKX</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>X'0301'</td>
<td>Prohibit Export Extended</td>
<td>Prohibit Export Extended</td>
<td>CSNBPEXX</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0309'</td>
<td>Operational Key Load</td>
<td>Key Part Import</td>
<td>CSNBKPI</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'030A'</td>
<td>NOCV KEK usage for import-related functions</td>
<td>Data Key Import</td>
<td>CSNBKGM</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key Import</td>
<td>CSNBKGM</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key Generate</td>
<td>CSNBKGN</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remote Key Export</td>
<td>CSNDRKX</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>X'030C'</td>
<td>DSG ZERO-PAD unrestricted hash length</td>
<td>Digital Signature Generate</td>
<td>CSNDDSG</td>
<td>OFF</td>
<td>O, SC</td>
</tr>
<tr>
<td>X'030D'</td>
<td>Key Encryption Translate - CBC to ECB</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>ON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'030E'</td>
<td>Key Encryption Translate - ECB to CBC</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>ON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'030F'</td>
<td>Trusted Block Create - Create Block in inactive form</td>
<td>Trusted Block Create</td>
<td>CSNDTBC</td>
<td>ON</td>
<td>O, SUP</td>
</tr>
<tr>
<td>ACP number (hex)</td>
<td>Name of ACP from TKE interface</td>
<td>Verb name</td>
<td>Entry point</td>
<td>Initial setting</td>
<td>Usage</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>----------------</td>
<td>-------</td>
</tr>
<tr>
<td>X'0310'</td>
<td>Trusted Block Create - Activate an inactive block</td>
<td>Trusted Block Create</td>
<td>CSNDTBC</td>
<td>ON</td>
<td>O, SUP</td>
</tr>
<tr>
<td>X'0311'</td>
<td>PKA Key Import - Import an external trusted block</td>
<td>PKA Key Import</td>
<td>CSNDPKI</td>
<td>ON</td>
<td>O, SEL</td>
</tr>
<tr>
<td>X'0312'</td>
<td>Remote Key Export - Gen or export a non-CCA node key</td>
<td>Remote Key Export</td>
<td>CSNDRKX</td>
<td>ON</td>
<td>O, SEL</td>
</tr>
<tr>
<td>X'0318'</td>
<td>PKA Key Translate - from CCA RSA to SC Visa Format</td>
<td>PKA Key Translate</td>
<td>CSNDPKT</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0319'</td>
<td>PKA Key Translate - from CCA RSA to SC ME Format</td>
<td>PKA Key Translate</td>
<td>CSNDPKT</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'031A'</td>
<td>PKA Key Translate - from CCA RSA to SC CRT Format</td>
<td>PKA Key Translate</td>
<td>CSNDPKT</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'031B'</td>
<td>PKA Key Translate - from source EXP KEK to target EXP KEK</td>
<td>PKA Key Translate</td>
<td>CSNDPKT</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'031C'</td>
<td>PKA Key Translate - from source IMP KEK to target EXP KEK</td>
<td>PKA Key Translate</td>
<td>CSNDPKT</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'031D'</td>
<td>PKA Key Translate - from source IMP KEK to target IMP KEK</td>
<td>PKA Key Translate</td>
<td>CSNDPKT</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0326'</td>
<td>PKA Key Generate - Clear ECC keys</td>
<td>PKA Key Generate</td>
<td>CSNDPKG</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0327'</td>
<td>Symmetric Key Export - AESKW</td>
<td>Symmetric Key Export</td>
<td>CSNDSYX</td>
<td>ON</td>
<td>O, R</td>
</tr>
<tr>
<td>X'0329'</td>
<td>Symmetric Key Import2 - AESKW</td>
<td>Symmetric Key Import2</td>
<td>CSNDSYI2</td>
<td>ON</td>
<td>O, R</td>
</tr>
<tr>
<td>X'032A'</td>
<td>Key Translate2 - Disallow AES ver 5 to ver 4 conversion</td>
<td>Key Translate2</td>
<td>CSNBKTR2</td>
<td>OFF</td>
<td>O, R</td>
</tr>
<tr>
<td>X'032B'</td>
<td>Symmetric Key Import2 - disallow weak import</td>
<td>Symmetric Key Import2</td>
<td>CSNDSYI2</td>
<td>OFF</td>
<td>O, R</td>
</tr>
<tr>
<td>X'032E'</td>
<td>TBC - Disallow triple-length MAC key</td>
<td>Trusted Block Create</td>
<td>CSNDTBC</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0331'</td>
<td>Allow weak DES wrap of RSA</td>
<td>PKA Key Generate</td>
<td>CSNDPKG</td>
<td>OFF</td>
<td>O, R</td>
</tr>
<tr>
<td>X'0334'</td>
<td>Key Translate2 - Translate fixed to variable payload</td>
<td>Key Translate2</td>
<td>CSNBKTR2</td>
<td>ON</td>
<td>SC</td>
</tr>
<tr>
<td>X'0335'</td>
<td>Unique Key Derive - K3IPEK</td>
<td>Unique Key Derive</td>
<td>CSNBUKD</td>
<td>ON</td>
<td>SC</td>
</tr>
<tr>
<td>X'0336'</td>
<td>MAC Generate2 - AES CMAC</td>
<td>MAC Generate2</td>
<td>CSNBMGN2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0337'</td>
<td>MAC Verify2 - AES CMAC</td>
<td>MAC Verify2</td>
<td>CSNBMVR2</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0338'</td>
<td>PKA Key Translate - from CCA RSA CRT to EMV DDA format</td>
<td>PKA Key Translate</td>
<td>CSNDPKT</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>ACP number (hex)</td>
<td>Name of ACP from TKE interface</td>
<td>Verb name</td>
<td>Entry point</td>
<td>Initial setting</td>
<td>Usage</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>----------------</td>
<td>-------</td>
</tr>
<tr>
<td>X'0339'</td>
<td>PKA Key Translate - from CCA RSA CRT to EMV DDAE format</td>
<td>PKA Key Translate(^1)</td>
<td>CSNDPKT</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'033A'</td>
<td>PKA Key Translate - from CCA RSA CRT to EMV CRT format</td>
<td>PKA Key Translate(^1)</td>
<td>CSNDPKT</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'033B'</td>
<td>Digital Signature Verify - PKCS-PSS allow not exact salt length</td>
<td>Digital Signature Verify</td>
<td>CSNDDSV</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'033C'</td>
<td>Digital Signature Verify - PKCS-PSS allow small salt</td>
<td>Digital Signature Verify</td>
<td>CSNDDSV</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>X'0350'</td>
<td>ANSI X9.8 PIN - Enforce PIN block restrictions</td>
<td>Clear PIN Generate Alternate Encrypted PIN Translate Secure Messaging for PINs</td>
<td>CSNBCPA CSNBPRTR CSNBPSPN</td>
<td>OFF, O, R</td>
<td></td>
</tr>
<tr>
<td>X'0351'</td>
<td>ANSI X9.8 PIN - Allow modification of PAN_01_0350</td>
<td>Encrypted PIN Translate Secure Messaging for PINs</td>
<td>CSNBPRTR CSNBPSPN</td>
<td>OFF, O, SC</td>
<td></td>
</tr>
<tr>
<td>X'0352'</td>
<td>ANSI X9.8 PIN - Allow only ANSI PIN blocks</td>
<td>Encrypted PIN Translate Secure Messaging for PINs</td>
<td>CSNBPRTR CSNBPSPN</td>
<td>OFF, O, SC</td>
<td></td>
</tr>
<tr>
<td>X'0353'</td>
<td>ANSI X9.8 PIN - Load Decimalization Tables</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td></td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0354'</td>
<td>ANSI X9.8 PIN - Delete Decimalization Tables</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td></td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0355'</td>
<td>ANSI X9.8 PIN - Activate Decimalization Tables</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td></td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0356'</td>
<td>ANSI X9.8 PIN - Use stored decimalization tables only</td>
<td>Clear PIN Generate(^1) Clear PIN Generate Alternate(^1) Encrypted PIN Generate(^1) Encrypted PIN Verify(^1)</td>
<td>CSNBPGN CSNBCPA CSNBEPG CSNBPRTR</td>
<td>OFF, O, R</td>
<td></td>
</tr>
<tr>
<td>X'035F'</td>
<td>ECC Diffie-Hellman - Allow DERIV02</td>
<td>EC Diffie-Hellman</td>
<td>CSNDEDH</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0360'</td>
<td>ECC Diffie-Hellmann</td>
<td>EC Diffie-Hellman(^1)</td>
<td>CSNDEDH</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0361'</td>
<td>EC Diffie-Hellman - Allow PASSTHRU</td>
<td>EC Diffie-Hellman(^1)</td>
<td>CSNDEDH</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0362'</td>
<td>ECC Diffie-Hellmann - Allow key wrap override</td>
<td>EC Diffie-Hellman(^1)</td>
<td>CSNDEDH</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0363'</td>
<td>ECC Diffie-Hellmann - Allow Prime Curve 192</td>
<td>EC Diffie-Hellman(^1)</td>
<td>CSNDEDH</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0364'</td>
<td>ECC Diffie-Hellmann - Allow Prime Curve 224</td>
<td>EC Diffie-Hellman(^1)</td>
<td>CSNDEDH</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0365'</td>
<td>ECC Diffie-Hellmann - Allow Prime Curve 256</td>
<td>EC Diffie-Hellman(^1)</td>
<td>CSNDEDH</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0366'</td>
<td>ECC Diffie-Hellmann - Allow Prime Curve 384</td>
<td>EC Diffie-Hellman(^1)</td>
<td>CSNDEDH</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0367'</td>
<td>ECC Diffie-Hellmann - Allow Prime Curve 521</td>
<td>EC Diffie-Hellman(^1)</td>
<td>CSNDEDH</td>
<td>ON</td>
<td>O</td>
</tr>
</tbody>
</table>
Table 306. Access Control Points and corresponding CCA verbs (continued)

<table>
<thead>
<tr>
<th>ACP number (hex)</th>
<th>Name of ACP from TKE interface</th>
<th>Verb name</th>
<th>Entry point</th>
<th>Initial setting</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0368'</td>
<td>ECC Diffie-Hellmann - Allow BP Curve 160</td>
<td>EC Diffie-Hellman</td>
<td>CSNDEDH</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'0369'</td>
<td>ECC Diffie-Hellmann - Allow BP Curve 192</td>
<td>EC Diffie-Hellman</td>
<td>CSNDEDH</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'036A'</td>
<td>ECC Diffie-Hellmann - Allow BP Curve 224</td>
<td>EC Diffie-Hellman</td>
<td>CSNDEDH</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'036B'</td>
<td>ECC Diffie-Hellmann - Allow BP Curve 256</td>
<td>EC Diffie-Hellman</td>
<td>CSNDEDH</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'036C'</td>
<td>ECC Diffie-Hellmann - Allow BP Curve 320</td>
<td>EC Diffie-Hellman</td>
<td>CSNDEDH</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'036D'</td>
<td>ECC Diffie-Hellmann - Allow BP Curve 384</td>
<td>EC Diffie-Hellman</td>
<td>CSNDEDH</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'036E'</td>
<td>ECC Diffie-Hellmann - Allow BP Curve 512</td>
<td>EC Diffie-Hellman</td>
<td>CSNDEDH</td>
<td>ON</td>
<td>O</td>
</tr>
<tr>
<td>X'036F'</td>
<td>ECC Diffie-Hellman - Prohibit weak key generate</td>
<td>EC Diffie-Hellman</td>
<td>CSNDEDH</td>
<td>OFF</td>
<td>O</td>
</tr>
<tr>
<td>X'0370'</td>
<td>CSNBKPIT: Allow load 1st key part for a key with min 3 key parts</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'0371'</td>
<td>CSNBKPIT: Allow load 1st key part for a key with min 2 key parts</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'0372'</td>
<td>CSNBKPIT: Allow load 1st key part for a key with min 1 key part</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'0373'</td>
<td>CSNBKPIT: Allow load 2nd and later key part for a key requiring more key parts</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'0374'</td>
<td>CSNBKPIT: Allow load last key part for a key</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'0375'</td>
<td>CSNBKPIT: Allow load an optional key part for a key</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'0376'</td>
<td>CSNBKPIT: Allow completing a key that has all key parts loaded</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'0377'</td>
<td>CSNBKPIT: Allow clearing a key part register</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'0378'</td>
<td>CSNBKPIT: Allow HMAC load 1st key part for a key with min 3 key parts</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'0379'</td>
<td>CSNBKPIT: Allow HMAC load 1st key part for a key with min 2 key parts</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
</tbody>
</table>
### Table 306. Access Control Points and corresponding CCA verbs (continued)

<table>
<thead>
<tr>
<th>ACP number (hex)</th>
<th>Name of ACP from TKE interface</th>
<th>Verb name</th>
<th>Entry point</th>
<th>Initial setting</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'037A'</td>
<td>CSNBKPIT: Allow HMAC load 1st key part for a key with min 1 key part</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'037B'</td>
<td>CSNBKPIT: Allow HMAC load 2nd and later key part for a key requiring more key parts</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'037C'</td>
<td>CSNBKPIT: Allow HMAC load last key part for a key</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'037D'</td>
<td>CSNBKPIT: Allow HMAC load an optional key part for a key</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'037E'</td>
<td>CSNBKPIT: Allow HMAC completing a key that has all key parts loaded</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>X'037F'</td>
<td>CSNBKPIT: Allow HMAC clearing a key part register</td>
<td>Note: This ACP is included for TKE reference only, the service impacted is available only (for IBM Z) on z/OS.</td>
<td>OFF</td>
<td>O</td>
<td></td>
</tr>
</tbody>
</table>

The following codes are used in this table:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Initial default.</td>
</tr>
<tr>
<td>O</td>
<td>Usage of this command is optional; enable it as required for authorized usage.</td>
</tr>
<tr>
<td>R</td>
<td>Enabling this command is recommended.</td>
</tr>
<tr>
<td>NR</td>
<td>Enabling this command is not recommended.</td>
</tr>
<tr>
<td>NRP</td>
<td>Enabling this command is not recommended for production.</td>
</tr>
<tr>
<td>SC</td>
<td>Usage of this command requires special consideration.</td>
</tr>
<tr>
<td>SEL</td>
<td>Usage of this command is normally restricted to one or more selected roles.</td>
</tr>
<tr>
<td>SUP</td>
<td>This command is normally restricted to one or more supervisory roles.</td>
</tr>
<tr>
<td>1</td>
<td>This verb performs more than one function, as determined by the keyword in the rule_array parameter of the verb call. Not all functions of the verb require the command in this row.</td>
</tr>
<tr>
<td>2</td>
<td>This verb does not always require the command in this row. Use as determined by the control vector for the key and the action being performed.</td>
</tr>
</tbody>
</table>

### Managing ACPs using a TKE workstation

The TKE workstation allows you to enable or disable access control points for verbs.

For systems that do not use the optional TKE workstation, most access control points (current and new) are enabled in the default role with the appropriate licensed internal code on the CEX*C. For more information about the TKE workstation, see [z/OS Cryptographic Services ICSF Trusted Key Entry Workstation User’s Guide](https://www.ibm.com/support/docview.wss?uid=swg21690487).

For information about required TKE versions for accessing the various CEX*C features, see "CEX6C information" on page 1117.
Use of particular cryptographic or key management verb functions with the CEX*C are controlled through access control points. You can see the default settings of an access control point in Table 306 on page 1048 in column Initial setting.

Note:
1. Access control points DKYGENKY-DALL and DSG ZERO-PAD unrestricted hash length are always disabled in the default role for all customers (TKE and non-TKE). A TKE workstation is required to enable these access control points.
2. When you modify the setting of an access control point, please be sure to use a procedure according to your organization’s security policy. TKE workstation versions earlier than V6.0 do not show the current setting of the access control points. TKE workstation versions 6.0 and higher show the current setting, but neither show the default settings nor a change history of the listed access control points. If you do not remember the change history, note that using the Zeroize function of the card or the domain to reset all access control points to their default values, discards all keys.
3. The TKE can save a current setting of ACPs under a given name. So if something fails with ACP changes, you can restore the old setting on the TKE.
Chapter 25. Access control data structures

Learn about the data structures that are used in the access control system. You can also view examples of the access-control data structures.

Read the following topics which provide more detailed information in their contained subtopics:

- "Role structures"
- "Examples of the access control data structures" on page 1078

Unless otherwise noted, all 2-byte and 4-byte integers are in big-endian format. The high-order byte of the value is in the lowest-numbered address in memory.

Role structures

Read the description of the data structures of a role, which controls the permitted operations a user can perform, and when those operations can be performed.

You can also use a defined default role. Its characteristics are described in "Default role contents" on page 1077.

Basic structure of a role

View a table that describes how the role data is structured.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>01</td>
<td>Role structure major version number: X'01'.</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>Role structure minor version number: X'00'.</td>
</tr>
<tr>
<td>02</td>
<td>02</td>
<td>Role structure length (big endian).</td>
</tr>
<tr>
<td>04</td>
<td>20</td>
<td>Comment.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A 20-character variable padded on the right with spaces, containing a comment which describes the role. This variable is not null (X'00') terminated.</td>
</tr>
<tr>
<td>24</td>
<td>02</td>
<td>Checksum (big endian).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The checksum value is not used in the current role structure. It can be verified by the IBM Cryptographic Coprocessor with a future version of the role structure.</td>
</tr>
<tr>
<td>26</td>
<td>02</td>
<td>Reserved (big endian): X'0000'.</td>
</tr>
<tr>
<td>28</td>
<td>08</td>
<td>Role ID.</td>
</tr>
<tr>
<td>36</td>
<td>02</td>
<td>Required authentication strength (big endian).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A 2-byte integer defining how secure the user authentication must be in order to authorize this role.</td>
</tr>
</tbody>
</table>
Table 307. Access-control system: basic structure of a role (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>02</td>
<td>Lower time limit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The earliest time of day that this role can be used. Format is $h : m$ where:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$h$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$m$</td>
</tr>
<tr>
<td>40</td>
<td>02</td>
<td>Upper time limit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The latest time of day that this role can be used. Format is the same as lower time limit (offset 38).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the lower time limit and upper time limit are identical, the role is valid for use at any time of the day.</td>
</tr>
<tr>
<td>42</td>
<td>01</td>
<td>Valid days of week (DOW):</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'0xxx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'1xxx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'x0xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'x1xx xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xx0x xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xx1x xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxx0 xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxx1 xxxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx 0xxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx 1xxx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx x0xx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx x1xx'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx xx0x'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx xx1x'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx xxx0'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B'xxxx xxx1'</td>
</tr>
<tr>
<td>43</td>
<td>01</td>
<td>Reserved (X'00')</td>
</tr>
<tr>
<td>44</td>
<td>variable</td>
<td>Access-control-point list (permitted or disallowed operations). The permitted operations are defined by the access control point list, described in Table 308 on page 1075</td>
</tr>
</tbody>
</table>
**Access control point list**

The user's permissions (permitted or disallowed operations) are attached to each role in the form of an access control point list (ACP list).

This ACP list is a map of bits, and every bit is uniquely identified by its offset. A single bit for each primitive function can be independently controlled. Refer to Chapter 24, "Access control points and verbs," on page 1047 for a list of all ACP offsets that are defined, their command names, which verbs use a particular command, and recommendations on command usage.

The **Required commands** section of each verb defines which bits of the ACP list, if any, control the functions of that verb. Each offset that a verb defines has a mnemonic command name. If an offset in an ACP list is B'1', the command is enabled. Otherwise, the command is disabled. Typically, when a command is enabled, the function associated with the command is permitted, provided that all other access conditions are also satisfied. However, in some cases, enabling a command disallows the function associated with the command. An offset that disallows a function has "Disallow" as part of its mnemonic command name.

Each ACP identifier (offset) is a two-byte integer (X'0000' - X'FFFF'). This allows addressability of $2^{16}$ (64K) bits. Only a small fraction of these addressable bits are used, so storing the entire 64K bit (8K byte) table in each role would waste memory space. Instead, the table is stored as a sparse matrix, where only the necessary bits are included.

To accomplish this, each bit map is stored as a series of one or more bit-map segments, where each can hold a variable number of bits. Each segment must start with a bit that is the high-order bit in a byte, and each must end with a bit that is the low order bit in a byte. This restriction results in segments that have no partial bytes at the beginning or end. Any bits that do not represent defined access control points must be set to zero, indicating that the corresponding function is not permitted.

The bit-map portion of each segment is preceded by a header, providing information about the segment. The header contains the following fields:

**Starting bit number**
- The index of the first bit contained in the segment. The index of the first access control point in the table is zero (X'0000').

**Ending bit number**
- The index of the last bit contained in the segment.

**Number of bytes in segment**
- The number of bytes of bit-map data contained in this segment.

The entire access control point structure is comprised of a header, followed by one or more access control point segments. The header indicates how many segments are contained in the entire structure.

The layout of this structure is illustrated in **Table 308**.

*Table 308. Access-control-point list structure*

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>2</td>
<td>Number of segments n in big endian format.</td>
</tr>
</tbody>
</table>
Table 308. Access-control-point list structure (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
<td>2</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

Bit-map segment 1 (refer to Table 309)

Bit-map segment 2 (optional)

... Bit-map segment n

An ACP list contains one or more bit-map segments. Table 309 defines the layout of a bit-map segment.

Table 309. Bit-map segment structure

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>2</td>
<td>Start bit number of bit-map segment in big endian format.</td>
</tr>
<tr>
<td>02</td>
<td>2</td>
<td>End bit number of bit-map segment in big endian format.</td>
</tr>
<tr>
<td>04</td>
<td>2</td>
<td>Number of bit-map bytes in big endian format.</td>
</tr>
<tr>
<td>06</td>
<td>2</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>08</td>
<td>variable</td>
<td>Bit-map data of segment.</td>
</tr>
</tbody>
</table>

For a specific role, you can display the ACP segments using the `panel.exe`, for example:

```
panel.exe --show-role --role=DEFAULT02
```

As output, you receive the specific bit list of enabled/disabled ACPs:
Default role contents

Read about the purpose and characteristics of the default role.

You can use the default role from a CEX*C coprocessor to access a certain domain in order to process the appropriate verbs that are allowed by the ACP list assigned to that role. Using the TKE, you can adapt the default role according to your needs for each domain that you want to access and thus use one role for each domain. The name that you apply to a default role for a domain depends on the domain number.

There are two variants for naming the default role for a domain on an S390 system: DEFALTXX and DFLTXXXX, where the XX or XXXX stand for a multi-digit decimal number indicating the domain number. The DFLTXXXX naming scheme was introduced for z13 machines.

Examples:
- DEFALT02 is the default role ID for domain 2 on a z10 machine (CEX3C or CEX4C).
- DFLT0052 is the default role ID for domain 52 on a z13 (CEX5C) or on a z14 (CEX6C).

The role ID names are always 8 characters, ASCII. Role names on some platforms and through some interfaces may end with ASCII space characters (0x20), therefore every input mechanism is designed to explicitly allow space characters.

See a list of the current roles:

```
# panel.exe --list-roles
```

The default role has the following characteristics:
- The required authentication strength level is zero.
- The role is valid at all times and on all days of the week.
- The only functions that are permitted are those related to access control initialization. This guarantees that the owner initializes the coprocessor before any cryptographic work can be done. This requirement prevents security accidents in which unrestricted default authority might accidentally be left intact when the system is put into service.

### Examples of the access control data structures

View examples for access control data structures as they are used in the Common Cryptographic Architecture.

Examples are provided in the following subtopics:
- “Access control point list - data structure example”
- “Role data structure example” on page 1079

#### Access control point list - data structure example

View an example of an access control point list together with explanations of the contents.

```
Figure 44 shows the contents of a sample access control point list.
```

```
00 02 00 00 00 00 01 17 00 23 00 00 f0 ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
00 02 17 00 03 00 00 8f 99 fe
```

*Figure 44. Access-control-point list example*

The access-control-point list contains the following data fields:
- **00 02** The number of segments of data in the access control point list. In this list, there are two discontiguous segments of access control points. One starts at access-control point 0, and the other starts at access control point X'200'.
- **00 00** A reserved field, which must be filled with zeros.
- **00 00** The number of the first access-control point in this segment.
- **01 17** The number of the last access control point in this segment. The segment starts at access control point 0, and ends with access control point X'0117', which is decimal 279.
00 23  The number of bytes of data in the access control points for this segment. There are X'23' bytes, which is 35 decimal.

00 00  A reserved field, which must be filled with zeros.

F0 FF FF FF ... FF FF (35 bytes)
The first set of access control points, with one bit corresponding to each point. Thus, the first byte contains bits 0-7, the next byte contains 8-15, and so on.

02 00  The number of the first access control point in the second segment.

02 17  The number of the last access control point in this segment. The segment starts at access control point X'200' (decimal 512), and ends with access control point X'217' (decimal 535).

00 03  The number of bytes of data in the access control points for this segment. There are 3 bytes for the access control points from 512 through 535.

00 00  A reserved field, which must be filled with zeros.

8F 99 FE
The second set of access control points, with one bit corresponding to each point. Thus, the first byte contains the bits 512-519, the second byte contains bits 520-527, and the third byte contains bits 528-535.

For a specific role, you can display the ACP segments using the panel.exe, as shown in “Access control point list” on page 1075.

**Role data structure example**
View an example of a role data structure together with explanations of the contents.

*New default role 1*
The 20-character comment describing this role.

AB CD The checksum for the role.

**Note:** The checksum value is not used.

00 00  A reserved field, which must be filled with zeros.

DEFAULT
The Role ID for this role. The role in this example replaces the DEFAULT role.

---

**Figure 45. Role data structure example**

This structure contains the following data fields:

01 00  The role structure version number.

00 66  The length of the role structure, including the length field itself.

*New default role 1*
The 20-character comment describing this role.

AB CD
The checksum for the role.

**Note:** The checksum value is not used.

00 00  A reserved field, which must be filled with zeros.

DEFAULT
The Role ID for this role. The role in this example replaces the DEFAULT role.
23 45  The required authentication strength field.
01 0F  The lower time limit. X'01' is the hour, and X'0F' is the minute (decimal 15), so the lower time limit is 1:15 AM, UTC.
17 1E  The upper time limit. X'17' is the hour (decimal 23), and X'1E' is the minute (30), so the upper time limit is 23:30 UTC.
7C  This byte maps the valid days of the week for the role. The first (high order) bit represents Sunday, the second represents Monday, and so on. Hex 7C is binary 01111100, and enables the weekdays Monday through Friday.
00  This byte is a reserved field. It must be zero.

**Access-control-point list**

The remainder of the role structure contains the access control point list described in "Access control point list - data structure example" on page 1078.
Chapter 26. Using verbs and applications in PCI-HSM 2016 compliance mode

Beginning with the Crypto Express6 adapter, when configured as a CCA coprocessor, such a coprocessor is capable of running in the PCI-HSM 2016 compliance mode. In order for the requirements of PCI-HSM 2016 to apply to a workload, the workload must be using compliant-tagged key tokens.

The PCI-HSM 2016 compliance mode places certain restrictions on the use of compliant-tagged key tokens. Thus, not all verbs can make use of such keys. Table 312 on page 1083 provides a list of PCI-HSM 2016 compliant verbs. Non-compliant verbs are listed in Table 311 on page 1082.

Furthermore, you may want to migrate applications to work in PCI-HSM 2016 compliance mode. This means, that you can adapt existing applications prior to CCA release 6.0 that exploit verbs that are now PCI-HSM 2016 compliant, to apply to PCI-HSM 2016 compliance mode. For more information, see “Migrating applications to PCI-HSM 2016 compliance mode” on page 1088.

Generating PCI-HSM 2016 compliant keys

A cryptographic coprocessor running in PCI-HSM 2016 compliance mode can use compliant-tagged key tokens. Read the contained information about how to generate PCI-HSM 2016 compliant keys.

A compliant-tagged key token is a key token that must adhere to the requirements of a compliance mode. The compliant tag is indicated by bit 58 in the control vector of the key token. For more information about the compliant tag, see the description of the COMP-TAG keyword in Chapter 21, “Control vectors and changing control vectors with the Control Vector Translate verb,” on page 989.

In turn, a compliance mode places certain restrictions on the use of compliant-tagged key tokens, as described in “Restrictions on creation and use of PCI-HSM 2016 compliant key tokens” on page 1084.

To generate a compliant-tagged key token, you must at first build a skeleton token with the compliant tag (bit 58) switched on. The Control Vector Generate (CSNBCCVG) and Key Token Build (CSNBKTB) services allow that. This skeleton token can then be passed to any callable service that generates key tokens and supports compliant-tagged key tokens, for example, Key Generate (CSNBKGN).

The TR31 Key Import (CSNBT31I) and EMV services are exceptions as they do not support skeleton tokens as input. When importing a TR-31 key block, if the key encrypting key is compliant-tagged, the resulting key is also compliant-tagged.

Note: Only internal, fixed-length TDES key tokens can be compliant-tagged.
Impact of the PCI-HSM 2016 compliance mode on the callable verbs

When compliant-tagged key tokens are used, the request is processed according to the compliance mode in effect. A subset of callable services support compliant-tagged key tokens. Any attempt to use a compliant-tagged key token in a cryptographic operation within a service that does not accept compliant-tagged key tokens results in a failure.

Table 310 shows a list of callable services (verbs) that are not compliant with PCI-HSM 2016 compliance mode.

Table 310. Callable services not compliant with PCI-HSM 2016

<table>
<thead>
<tr>
<th>Verb</th>
<th>Service name</th>
<th>Possible compliance alternatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNBCKC</td>
<td>CVV Key Combine</td>
<td>Create double-length keys instead of single-length keys. This eliminates a need for the service. Note: Though this service does not accept or create compliant-tagged key tokens, the key tokens that are created by this service can be converted to compliant-tagged key tokens.</td>
</tr>
<tr>
<td>CSNBCKI</td>
<td>Clear Key Import</td>
<td>• Use the TKE workstation to create a compliant-tagged key token from key parts. • Use the Key Generate (CSNBKGN) callable service to create a compliant-tagged key token with a random key value.</td>
</tr>
<tr>
<td>CSNBCKM</td>
<td>Multiple Clear Key Import</td>
<td>• Use the TKE workstation to create a compliant-tagged key token from key parts. • Use the Key Generate (CSNBKGN) callable service to create a compliant-tagged key token with a random key value.</td>
</tr>
<tr>
<td>CSNBCVT</td>
<td>Control Vector Translate</td>
<td>• Recreate the source key token with the wanted control vector. • Wrap the source key such that the wanted attributes are bound to the key. You can use the Key Import/Key Export (CSNBKIM/CSNBKEX) or TR31 Key Import/Key Export to TR31 (CSNBT31I/CSNBT31X) verbs.</td>
</tr>
<tr>
<td>CSNBCVE</td>
<td>Cryptographic Variable Encipher</td>
<td>Create double-length keys instead of single-length keys. This eliminates a need for the service.</td>
</tr>
<tr>
<td>CSNBDKX</td>
<td>Data Key Export</td>
<td>DATA keys are not compliant. Create compliant-tagged CIPHER or MAC keys that can be exported by using the Key Export (CSNBKEX) or Key Export to TR31 (CSNBT31X) callable services.</td>
</tr>
<tr>
<td>CSNBDKM</td>
<td>Data Key Import</td>
<td>DATA keys are not compliant. Create compliant-tagged CIPHER or MAC keys that can be imported by using the Key Import (CSNBKIM) or TR31 Key Import (CSNBT31I) callable services.</td>
</tr>
<tr>
<td>CSNBSKY</td>
<td>Secure Messaging for Keys</td>
<td>None.</td>
</tr>
<tr>
<td>CSNDSXD</td>
<td>Symmetric Key Export with Data</td>
<td>You can use the Key Export to TR31 (CSNBT31X) to export a compliant-tagged symmetric key token with data.</td>
</tr>
</tbody>
</table>

In addition to the callable services listed in Table 310, there are callable services that do not support compliant-tagged key tokens as shown in Table 311.

Table 311. Callable services that do not support compliant-tagged key tokens

<table>
<thead>
<tr>
<th>Verb</th>
<th>Service name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNBDDPG</td>
<td>DK Deterministic PIN Generate</td>
</tr>
<tr>
<td>CSNBDMP</td>
<td>DK Migrate PIN</td>
</tr>
<tr>
<td>CSNBDPC</td>
<td>DK PIN Change</td>
</tr>
</tbody>
</table>
Table 311. Callable services that do not support compliant-tagged key tokens (continued)

<table>
<thead>
<tr>
<th>Verb</th>
<th>Service name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNBDPG</td>
<td>DK PRW CMAC Generate</td>
</tr>
<tr>
<td>CSNBDMT</td>
<td>DK PAN Modify in Transaction</td>
</tr>
<tr>
<td>CSNBDPNU</td>
<td>DK PRW Card Number Update</td>
</tr>
<tr>
<td>CSNBDPT</td>
<td>DK PAN Translate</td>
</tr>
<tr>
<td>CSNBDPV</td>
<td>DK PIN Verify</td>
</tr>
<tr>
<td>CSNBDRP</td>
<td>DK Regenerate PRW</td>
</tr>
<tr>
<td>CSNBDRPG</td>
<td>DK Random PIN Generate</td>
</tr>
<tr>
<td>CSNDEDH</td>
<td>EC Diffie-Hellman</td>
</tr>
<tr>
<td>CSNDSYX</td>
<td>Symmetric Key Export</td>
</tr>
<tr>
<td>CSNDSYG</td>
<td>Symmetric Key Generate</td>
</tr>
<tr>
<td>CSNDSYI</td>
<td>Symmetric Key Import</td>
</tr>
<tr>
<td>CSNDSYI2</td>
<td>Symmetric Key Import2</td>
</tr>
</tbody>
</table>

Table 312 shows the list of services that support compliant-tagged key tokens when running in PCI-HSM 2016 compliance mode.

Table 312. Callable services that support compliant-tagged key tokens in cryptographic operations

<table>
<thead>
<tr>
<th>Verb</th>
<th>Service name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNBAAPG</td>
<td>Authentication Parameter Generate</td>
</tr>
<tr>
<td>CSNBCPA</td>
<td>Clear PIN Generate Alternate</td>
</tr>
<tr>
<td>CSNBCPE</td>
<td>Clear PIN Encrypt</td>
</tr>
<tr>
<td>CSNBCSG</td>
<td>CVV Generate</td>
</tr>
<tr>
<td>CSNBCSV</td>
<td>CVV Verify</td>
</tr>
<tr>
<td>CSNBCCTT2</td>
<td>Cipher Text Translate2</td>
</tr>
<tr>
<td>CSNBDDEC</td>
<td>Decipher</td>
</tr>
<tr>
<td>CSNBDKG</td>
<td>Diversified Key Generate</td>
</tr>
<tr>
<td>CSNBNENC</td>
<td>Encipher</td>
</tr>
<tr>
<td>CSNBEPG</td>
<td>Encrypted PIN Generate</td>
</tr>
<tr>
<td>CSNBFPED</td>
<td>FPE Decipher</td>
</tr>
<tr>
<td>CSNBFPEE</td>
<td>FPE Encipher</td>
</tr>
<tr>
<td>CSNBFPET</td>
<td>FPE Translate</td>
</tr>
<tr>
<td>CSNBEKEX</td>
<td>Key Export</td>
</tr>
<tr>
<td>CSNBEKGN</td>
<td>Key Generate</td>
</tr>
<tr>
<td>CSNBEKIM</td>
<td>Key Import</td>
</tr>
<tr>
<td>CSNBKTR</td>
<td>Key Translate</td>
</tr>
<tr>
<td>CSNBKTR2</td>
<td>Key Translate2</td>
</tr>
<tr>
<td>CSNBEKYT</td>
<td>Key Test</td>
</tr>
<tr>
<td>CSNBEKYTX</td>
<td>Key Test Extended</td>
</tr>
<tr>
<td>CSNBEKYT2</td>
<td>Key Test2</td>
</tr>
<tr>
<td>CSNBMGN</td>
<td>MAC Generate</td>
</tr>
</tbody>
</table>
Table 312. Callable services that support compliant-tagged key tokens in cryptographic operations (continued)

<table>
<thead>
<tr>
<th>Verb</th>
<th>Service name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNBMVR</td>
<td>MAC Verify</td>
</tr>
<tr>
<td>CSNBPCU</td>
<td>PIN Change/Unblock</td>
</tr>
<tr>
<td>CSNBPEX</td>
<td>Prohibit Export</td>
</tr>
<tr>
<td>CSNBPEXX</td>
<td>Prohibit Export Extended</td>
</tr>
<tr>
<td>CSNBPFO</td>
<td>Recover PIN from Offset</td>
</tr>
<tr>
<td>CSNBPGN</td>
<td>Clear PIN Generate</td>
</tr>
<tr>
<td>CSNBPTR</td>
<td>Encrypted PIN Translate</td>
</tr>
<tr>
<td>CSNBPTRE</td>
<td>Encrypted PIN Translate Enhanced</td>
</tr>
<tr>
<td>CSNBPVX</td>
<td>Encrypted PIN Verify</td>
</tr>
<tr>
<td>CSNBPCA</td>
<td>Restrict Key Attribute</td>
</tr>
<tr>
<td>CSNBSFPN</td>
<td>Secure Messaging for PINs</td>
</tr>
<tr>
<td>CSNBTRI1</td>
<td>TR31 Key Import</td>
</tr>
<tr>
<td>CSNBTRI2</td>
<td>Key Export to TR31</td>
</tr>
<tr>
<td>CSNBTRV</td>
<td>Transaction Validation</td>
</tr>
<tr>
<td>CSNBUKD</td>
<td>Unique Key Derive</td>
</tr>
</tbody>
</table>

Restrictions on creation and use of PCI-HSM 2016 compliant key tokens

Read about the restrictions that apply to PCI-HSM 2016 compliant key tokens.

- CCA 6.0 introduces support for compliant-tagged TDES key tokens. Other key tokens are not impacted at this time.
- Only the enhanced wrapping method can be used.
- Single-length key tokens cannot be compliant-tagged.
- Key-encrypting keys (KEKs), including master keys, must be at least as strong as the keys they protect. Therefore, a KEK with replicated key halves cannot wrap a key with unique key halves.
- Default DATA key tokens (including zero CV DATA key tokens) cannot be compliant-tagged. They are multi-use keys capable of performing both encipherment and MAC generation. One alternative is to upgrade to CIPHER or MAC keys, or both CIPHER and MAC keys. Currently, triple-length DATA keys with unique key parts have no migration path to becoming compliant-tagged without downgrading the key to a double-length key.
- NOCV-KEKs cannot be compliant-tagged. The NOCV flag is not bound to the key of the KEK, so it is possible to change this flag for a key outside the HSM. KEKs with the NOCV flag set (done externally) and the compliance tag set in the control vector are not usable.
- Compliant-tagged key tokens are restricted in terms of the verification patterns that can be calculated. Only the ENC-ZERO and CMACZERO verification patterns are allowed. See the Key Test (CSNBKYYT), Key Test Extended (CSNBKYYXT), and Key Test2 (CSNBKYYT2) callable services for more detail. In addition, for the Key Test and Key Test Extended services, the KEY-ENCD keyword is required.
• Generally, compliant-tagged key tokens cannot be used with non-compliant-tagged key tokens. The only exception is the Cipher Text Translate2 (CSNBCTT2) callable service.

Restrictions on PIN block translation operations for compliant-tagged key tokens

In general, the described restrictions are concerned with binding/unbinding of the PAN to the PIN data, or changes to the PAN.

See Table 313 for a summary of PIN block translation restrictions that apply to compliant-tagged key tokens.

Table 313. Using compliant-tagged key tokens to translate between PIN block formats

<table>
<thead>
<tr>
<th>Translate from:</th>
<th>ECI-2 (4 digit PIN)</th>
<th>ECI-3 (4-6 digit PIN)</th>
<th>ISO-0, ANSI X9.8, VISA-1, VISA-4, and ECI-1</th>
<th>ISO-1 and ECI-4</th>
<th>ISO-2</th>
<th>ISO-3</th>
<th>VISA-2 (4-6 digit PIN)</th>
<th>VISA-3 (4-12 digit PIN)</th>
<th>3621 (4-12 digit PIN)</th>
<th>3624 (4-12 digit PIN)</th>
<th>4704-EPP (4-12 digit PIN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECI-2</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>ECI-3</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>ISO-0, ANSI X9.8, VISA-1, VISA-4, and ECI-1</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>If not changing PAN</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>ISO-1 and ECI-4</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>ISO-2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>ISO-3</td>
<td>N</td>
<td>N</td>
<td>If not changing PAN</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>VISA-2</td>
<td>If PIN is 4 digits</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>VISA-3</td>
<td>If PIN is 4 digits</td>
<td>If PIN is 4-6 digits</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>If PIN is 4-6 digits</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>3621</td>
<td>If PIN is 4 digits</td>
<td>If PIN is 4-6 digits</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>If PIN is 4-6 digits</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>3624</td>
<td>If PIN is 4 digits</td>
<td>If PIN is 4-6 digits</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>If PIN is 4-6 digits</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>4704-EPP</td>
<td>If PIN is 4 digits</td>
<td>If PIN is 4-6 digits</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>If PIN is 4-6 digits</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Generating warning events

You use the CSU_CMP_WARN_MODE environment variable to cause CCA to issue warning events for operations that need modifications to meet PCI-HSM 2016 compliance mode requirements. Warning events are created as entries to SYSLOG in a certain layout, specifying information about the required actions.
Setting the CSU_CMP_WARN_MODE environment variable

If set, the CSU_CMP_WARN_MODE environment variable indicates that you want to have PCI-HSM 2016 compliance mode warnings generated. The values can be one of the following:

**OFF**  This is the default value for the variable. No warning mode is set and no warning events are created.

**SYSLOG**  The warning mode output is sent in a log content stream to the system log and creates an entry in the SYSLOG file as described in "Warning log message layout.”

Note:
- No log entry is written when an operation is completely compliant (compliant verb and compliant keys).
- Warning mode responses are sent to the configured system log.

**Warning log message layout**

The overall syntax of a generated warning message looks like shown:

```plaintext
```

The name of the application which causes the triggering of the warning is not displayed in the message, but is captured by the system log itself. The components have the following meanings:

**preamble**  Content that a SYSLOG entry typically puts at the start of each message. If the SYSLOG is not the output target, then no preamble is posted.

A part of the preamble posted by the SYSLOG is the ASCII name of the application that was processing when the warning triggered. See the following example for details.

**CSUCMPW**  ASCII eye-catcher to mark the start of each entry

**serialnum**  ASCII representation of adapter serial number that generated the reply containing the warning response.

**KDF**  Key Derivation Function (KDF) value used in the host-to-card communication buffer. It corresponds to the compliance level of the crypto card. This field is one binary byte. It is printed as two hexadecimal characters in ASCII.

**cprbflags**  Warning flag value returned from the adapter for the verb. This field is one binary byte. It is printed as two hexadecimal characters in ASCII.

**Flag value**

<table>
<thead>
<tr>
<th>Flag value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40</td>
<td>CPRBFO_CMP_BAD_VERB: The requested verb is not compliant with the current target.</td>
</tr>
<tr>
<td>0x20</td>
<td>CPRBFO_CMP_BAD_OP: The requested verb (keyword or data) is not compliant with the current target</td>
</tr>
</tbody>
</table>
0x10 CPRBFO_CMP_BAD_KEY: The request included one or more keys that are not compliant with the current target.

0x08 CPRBFO_CMP_UNK: The request for the warning mode processing could not be honored due to problems with the input. The main reason is an unknown value supplied by the host library to the adapter as the target KDF: This is an internal error.

verb_code
Name of the verb encoded as two ASCII characters. See column Subcommand code in Table 317 on page 1135 for the encoding reference.

keyword_count
ASCII number for the count of verb keywords contained in the request.

keywords
Keywords that were in request when sent to the adapter. Some API calls are split up into smaller requests when sent to the adapter to overcome bus transfer size limitations. Therefore, keywords may be slightly different when sent to the adapter. Also, a single API request may generate more than one warning log entry.

If the keyword_count is 0x00, this field is empty, such that only the delimiters on either side are visible (::).

key_count
Count of keys that were sent to the adapter.

Below is a warning message from the CSNBCTT2 verb, using a double length DES DATA token which can not be COMP-TAGGED in compliance mode.


CSUCMPW is the eye-catcher you should search for. All material previous to this ASCII string is the so-called preamble. As a part of this preamble, the SYSLOG captures the name of the application executing when warning triggered. In our example, this is s6ctt2.e (running on the test05 system.

DV753302 is the serial number of the adapter that generated the warning.

01 is the compliance level in the card (KDF parameter).

10 specifies that the request included one or more keys that are not compliant with the current target (cprbflags0 parameter).

TT is the verb code for CSNBCTT2 as provided in Table 317 on page 1135 (verb-code parameter). TT stands for CSNBCTT2.

5 is the number of 8 byte keywords (keywordcount parameter).

IKEY-DESOKEY-DESI-CBC O-CBC INITIAL are the five 8 byte keywords provided by the keywords parameter:

IKEY-DES
OKEY-DES
I-CBC
O-CBC
INITIAL

This part can be empty (::) if no keywords are supplied in the request.
Note that sometimes requests are split into smaller requests when sent to the adapter.

2 is the number of keys supplied to CSNBCTT2 (key_count parameter).

Another example shows how to use the CSNBKYT verb to obtain SHA-1 hashes of master keys in legacy mode. The warning mode is logging that the requested verb service (keyword or data) is not compliant with current target:

```
Jan 12 09:49:04 test05 load.e[24427]: IBM Crypto ./f_sec2/sefcmpwarn.c:627,  
Jan 12 09:49:04 test05 load.e[24427]: IBM Crypto ./f_sec2/sefcmpwarn.c:627,  
Jan 12 09:49:04 test05 load.e[24427]: IBM Crypto ./f_sec2/sefcmpwarn.c:627,  
Jan 12 09:49:04 test05 load.e[24427]: IBM Crypto ./f_sec2/sefcmpwarn.c:627,  
```

**Migrating applications to PCI-HSM 2016 compliance mode**

You can migrate existing applications to PCI-HSM 2016. This involves converting the keys that are used by the application to key tokens that are tagged as compliant to PCI-HSM 2016.

The migration process completes with existing key tokens that are converted to being compliant-tagged. Before you complete the migration, it is important that the necessary steps are taken so that upon completion of the migration, future key tokens can be created as compliant-tagged.

**Identifying key tokens to be converted using compliance warning events**

Except for the Cipher Text Translate2 (CSNBCTT2) verb, a compliant-tagged key cannot be used in a cryptographic operation with a non-compliant-tagged key. Therefore, with that one exception, the key migration process needs to ensure that for each key that is to be compliant-tagged, all keys it can be used with, must also be compliant-tagged.

One procedure for migrating keys is to put them into one of the following five categories in order from top to bottom:

1. Keys that are used in non-compliant or unsupported ways. These keys are included in compliance warning events with any of the following results:
   - Non-compliant service
   - Compliance not supported
   - Non-compliant service operation.
   Before these keys can be compliant-tagged, there must be a change to use them in a supported, compliant way.

2. Keys that are used with a key in category 1. Since keys in this category are used with keys that should not be compliant-tagged, by extension they should not be compliant-tagged.

3. Keys that are not compliant. The overall warning event result would indicate that a non-compliant key was used and this key is marked as a non-compliant key. The reason for the non-compliance needs to be determined and fixed before the key can be compliant-tagged. For more information, see “Restrictions on creation and use of PCI-HSM 2016 compliant key tokens” on page 1084.
4. Keys that are used with a key in category (3 on page 1088). Since keys in this category are used with keys that cannot be compliant-tagged, by extension these keys should not be compliant-tagged.

5. Keys that are compliant and only used in compliant ways. These keys are only included in warning events with an overall result of compliance.

For help interpreting the compliance warning events, see section “Warning log message layout” on page 1086.

As the issues in categories 1 on page 1088 to 4 are resolved, you are left only with keys in category 5. At this point, you can successfully compliant-tag all your keys.

Further information is in “Migration mode” on page 1090.

It might be the case that not all keys can become compliant-tagged, for instance, when a key is used in a verb that does not support compliance. In this case, you have a key or keys that remain in category 1 on page 1088 and possibly also in category 2 on page 1088 that should not be compliant-tagged. The keys in category 5 can still be compliant-tagged. A similar situation arises when a key that is used in an otherwise compliant way is itself non-compliant. In this case, you have a key or keys that are remaining in category 3 on page 1088 and possibly also in category 4 that should not be compliant-tagged. Once again, the keys in category 5 can still be compliant-tagged. These scenarios are not mutually exclusive so you might end up with keys in categories 1 on page 1088 to 5 at migration time. If the categories are strictly adhered to, you can compliant-tag the keys in category 5.

All requests that are selected for compliance warning processing should be routed to a CEX6C coprocessor by the application. The CCA host library does not change application routing decisions if a CEX5C cryptographic coprocessor has been selected for processing. To avoid application changes, it is recommended to disable any CEX5C instances using the chzcrypt -d command before making use of the warning mode with CEX6C adapters. This will avoid any errors from routing issues.

Identifying key tokens to be converted outside of compliance warning events

Since compliance warnings are based on actual usage, be especially aware of operations (for example, infrequent operations) which might not be started during the period where warnings are being collected. These operations would not show up in a warning log and so must be discovered and analyzed independently.

It might be necessary to do an analysis of the key storage to identify the key tokens to be converted. If the key labels follow a naming convention, the DES Key Record List (CSNBKRL) callable service can be used to produce a list of labels according to a filter. You can use the panel.exe utility to list all the DES key tokens from key storage. The panel.exe utility also uses the CSNBKRL service.

Key tokens that are identified in this way do not have any information about how they are used. So either the usage of such key tokens must be understood, or all the key tokens that are used must be converted together.

Ensure the key tokens identified can become compliant-tagged

When the key tokens to be converted have been identified, you need to ensure that the process of compliant-tagging the key tokens is successful. Compliant-tagged key tokens cannot be used with non-compliant-tagged key tokens. So you do not
want the conversion of some tokens to fail while others succeed. You do this by compliance-checking the key tokens first. Key tokens are compliance-checked by using the Key Translate2 (CSNBKTR2) verb with the COMP-CHK keyword.

**Migration mode**

Existing keys are transformed into compliant-tagged keys using the migration mode of the adapter domain. This migration mode is a temporary sub-state of the PCI-HSM 2016 compliance mode.

The process is like follows:
1. Your cryptographic coprocessor starts in a normal, non-compliant mode.
2. Take one or more domains to the PCI-HSM 2016 compliance mode, using the TKE workstation for administrative steps.
3. Now you can create new tokens with the compliance-tag. You cannot migrate legacy tokens by adding the compliance-tag while in the operational compliance mode, because this is a sensitive operation that could compromise security.
4. Take one or more domains to migration mode, using the TKE workstation for administrative steps.

**Note:** You can only enter the migration mode from the PCI-HSM 2016 compliance mode.
5. Now you can migrate legacy tokens using the CSNBKTR2 verb with the COMP-TAG keyword.

The migration mode has an inactivity timeout. If no key migration actions (initiated from the CSNBKTR2J verb using the COMP-TAG keyword) are received within 30 minutes, then the domain automatically exits migration mode and returns to the PCI-HSM 2016 compliance mode. Note that in migration mode you are allowed to migrate legacy tokens by adding the compliance-tag. However, no compliance-tagged key operations other than key migration are allowed in migration mode, as required for PCI-HSM 2016 compliance.

**Converting key tokens to become compliant-tagged**

Make sure to identify all key tokens that must be converted, because they cannot be used in a non-compliant way. Then verify that they can be converted to compliant-tagged key tokens. If your verification is successful, you can start the conversion.

At this point, you must decide what backup strategy, if any, to pursue. Depending on the nature of your workloads, your backup strategy can include, but is not be limited to:
1. Making a backup copy of the key storage.
2. Retrieving the key tokens to be converted from the key storage and storing them in a file.

Instead of making a backup copy of the key storage, you can write the compliant-tagged key tokens to new key storage labels.

Key tokens are converted to compliant-tagged tokens by using the Key Translate2 (CSNBKTR2) verb with the COMP-TAG keyword.
To begin the conversion, at least one CCA coprocessor must be placed in migration mode by using the TKE workstation. To confirm the compliance mode of a CCA coprocessor, view the hardware status panel. Also, take note of the number of CCA coprocessors in PCI-HSM 2016 compliance mode without being in migration mode. A coprocessor in migration mode cannot handle requests that contain compliant-tagged key tokens. Therefore, if workloads that use compliant-tagged key tokens are already in use (for example, you previously converted some key tokens to compliant-tagged), you must keep one or more coprocessors in PCI-HSM 2016 compliance mode, but not in migration mode.

At this point in the process, none of the key tokens should fail because they were previously compliance-checked. However, if for some reason there is a failure such that some of the key tokens were converted and others were not, the key tokens should be brought back to a consistent state as soon as possible. This means that the key tokens should be updated such that they are all compliant-tagged or all non-compliant-tagged. If a backup copy of the CKDS was made, it can be used to make all the key tokens non-compliant-tagged until the issue can be resolved. If the process involves creating compliant-tagged key tokens under new key labels, the old labels can still be used until the issue is resolved.
Chapter 27. Sample verb call routines

This appendix contains sample verb call routines for both C and Java.

**Important:** The user must load the Symmetric Master Key before the verb calls complete successfully. Otherwise return code 12 and reason code 764 is returned.

To illustrate the practical application of CCA verb calls, this appendix describes the sample routines included with the RPM. A sample in C, and one in Java is included.

The sample routines generate a Message Authentication Code (MAC) on a text string, and then verifies the MAC. To accomplish this, the routine:

- Calls the Key Generate (CSNBKGN or CSNBKGNJ) verb to create a MAC/MACVER key pair.
- Calls the MAC Generate (CSNBMGN or CSNBMGNJ) verb to generate a MAC on a text string with the MAC key.
- Calls the MAC Verify (CSNBMVR or CSNBMVRJ) verb to verify the text string MAC with the MACVER key.

As you review the sample routines shown in [Figure 46](#) and [Figure 47 on page 1098](#), refer to the chapters in this book for descriptions of the called verbs and their parameters. These verbs are listed in Table 314.

**Table 314. Verbs called by the sample routines**

<table>
<thead>
<tr>
<th>Verb</th>
<th>Entry point name for C and Java versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Generate</td>
<td>CSNBKGN or CSNBKGNJ</td>
</tr>
<tr>
<td>MAC Generate</td>
<td>CSNBMGN or CSNBMGNJ</td>
</tr>
<tr>
<td>MAC Verify</td>
<td>CSNBMVR or CSNBMVRJ</td>
</tr>
</tbody>
</table>

Sample program in C

This sample code, which consists of a C program (mac.c) and a makefile (makefile.Inx), can be found in the /opt/IBM/CCA/samples directory.

For reference, a copy of the sample routine is shown in [Figure 46](#).

**Figure 46. Syntax, sample routine in C**

```
#*----------------------------------------------------------------------*
#* Module Name: mac.c                                                   *
#* DESRIPTIVE NAME: Cryptographic Coprocessor Support Program          *
#* C language source code example                                      *
#*----------------------------------------------------------------------*
#* Licensed Materials - Property of IBM                                *
#* (C) Copyright IBM Corp. 1997-2001 All Rights Reserved              *
```
/* US Government Users Restricted Rights - Use duplication or */
/* disclosure restricted by GSA ADP Schedule Contract with IBM Corp. */
/* ...........................................................................*/
/* */
/* NOTICE TO USERS OF THE SOURCE CODE EXAMPLES */
/* */
/* The source code examples provided by IBM are only intended to */
/* assist in the development of a working software program. The */
/* source code examples do not function as written: additional */
/* code is required. In addition, the source code examples may */
/* not compile and/or bind successfully as written. */
/* */
/* International Business Machines Corporation provides the source */
/* code examples, both individually and as one or more groups, */
/* "as is" without warranty of any kind, either expressed or */
/* implied, including, but not limited to the implied warranties of */
/* merchantability and fitness for a particular purpose. The entire */
/* risk as to the quality and performance of the source code */
/* examples, both individually and as one or more groups, is with */
/* you. Should any part of the source code examples prove defective, */
/* you (and not IBM or an authorized dealer) assume the entire cost */
/* of all necessary servicing, repair or correction. */
/* */
/* IBM does not warrant that the contents of the source code */
/* examples, whether individually or as one or more groups, will */
/* meet your requirements or that the source code examples are */
/* error-free. */
/* */
/* IBM may make improvements and/or changes in the source code */
/* examples at any time. */
/* */
/* Changes may be made periodically to the information in the */
/* source code examples; these changes may be reported, for the */
/* sample code included herein, in new editions of the examples. */
/* */
/* References in the source code examples to IBM products, programs, */
/* or services do not imply that IBM intends to make these */
/* available in all countries in which IBM operates. Any reference */
/* to the IBM licensed program in the source code examples is not */
/* intended to state or imply that IBM's licensed program must be */
/* used. Any functionally equivalent program may be used. */
/* */
/* ************************************************************************/
/* */
/* This example program: */
/* */
/* 1) Calls the Key_Generate verb (CSNBKGN) to create a MAC (message */
/* authentication code) key token and a MACVER key token. */
/* */
/* 2) Calls the MAC_Generate verb (CSNBMGNM) using the MAC key token */
/* from step 1 to generate a MAC on the supplied text string */
/* [INPUT_TEXT]. */
/* */
/* 3) Calls the MAC_Verify verb (CSNBMVR) to verify the MAC for the */
/* same text string, using the MACVER key token created in */
/* step 1. */
/* */
/**************************************************************************/
#include <stdio.h>
#include <string.h>

#ifdef _AIX
    #include <csufincl.h>
#else __WINDOWS__
    #include "csunincl.h"
#endif
#include "csulincl.h" /* else linux */
#endif

/* Defines */
#define KEY_FORM "OPOP"
#define KEY_LENGTH "SINGLE"
#define KEY_TYPE_1 "MAC"
#define KEY_TYPE_2 "MACVER"
#define INPUT_TEXT "abcdefhgijklmn0987654321"
#define MAC_PROCESSING_RULE "X9.9-1"
#define SEGMENT_FLAG "ONLY"
#define MAC_LENGTH "HEX-9"
#define MAC_BUFFER_LENGTH 10

void main()
{
    static long return_code;
    static long reason_code;
    static unsigned char key_form[4];
    static unsigned char key_length[8];
    static unsigned char mac_key_type[8];
    static unsigned char macver_key_type[8];
    static unsigned char kek_key_id_1[64];
    static unsigned char kek_key_id_2[64];
    static unsigned char mac_key_id[64];
    static unsigned char macver_key_id[64];
    static long text_length;
    static unsigned char text[26];
    static long rule_array_count;
    static unsigned char rule_array[3][8];
    static unsigned char chaining_vector[18];
    static unsigned char mac_value[MAC_BUFFER_LENGTH];

    /* Print a banner */
    printf("Cryptographic Coprocessor Support Program example program.\n");
    /* Set up initial values for Key_Generate call */
    return_code = 0;
    reason_code = 0;
    memcp (key_form, KEY_FORM, 4); /* OPOP key pair */
    memcp (key_length, KEY_LENGTH, 8); /* Single-length keys */
    memcp (mac_key_type, KEY_TYPE_1, 8); /* 1st token, MAC key type */
    memcp (macver_key_type, KEY_TYPE_2, 8); /* 2nd token, MACVER key type */
    memset (kek_key_id_1, 0x00, sizeof(kek_key_id_1)); /* 1st KEK not used */
    memset (kek_key_id_2, 0x00, sizeof(kek_key_id_2)); /* 2nd KEK not used */
    memset (mac_key_id, 0x00, sizeof(mac_key_id)); /* Init 1st key token */
    memset (macver_key_id, 0x00, sizeof(macver_key_id)); /* Init 2nd key token */

    /* Generate a MAC/MACVER operational key pair */
    CSNBKGN(&return_code,
            &reason_code,
            NULL, 
            NULL, /* exit_data_length */
            NULL, /* exit_data */
            key_form,
            key_length,
            mac_key_type,
            macver_key_type,
            kek_key_id_1,
            kek_key_id_2,
            mac_key_id,
            macver_key_id);

    /* Check the return/reason codes. Terminate if there is an error. */
    if (return_code != 0 || reason_code != 0) {
        printf ("Key_Generate failed: "); /* Print failing verb */
        printf ("return_code = %ld", return_code); /* Print return code */
        printf ("reason_code = %ld:\n", reason_code); /* Print reason code */
        return;
    }

    /* Print a banner */
    printf("Cryptographic Coprocessor Support Program example program.\n");
    /* Set up initial values for Key_Generate call */
    return_code = 0;
    reason_code = 0;
    memcp (key_form, KEY_FORM, 4); /* OPOP key pair */
    memcp (key_length, KEY_LENGTH, 8); /* Single-length keys */
    memcp (mac_key_type, KEY_TYPE_1, 8); /* 1st token, MAC key type */
    memcp (macver_key_type, KEY_TYPE_2, 8); /* 2nd token, MACVER key type */
    memset (kek_key_id_1, 0x00, sizeof(kek_key_id_1)); /* 1st KEK not used */
    memset (kek_key_id_2, 0x00, sizeof(kek_key_id_2)); /* 2nd KEK not used */
    memset (mac_key_id, 0x00, sizeof(mac_key_id)); /* Init 1st key token */
    memset (macver_key_id, 0x00, sizeof(macver_key_id)); /* Init 2nd key token */

    /* Generate a MAC/MACVER operational key pair */
    CSNBKGN(&return_code,
            &reason_code,
            NULL, 
            NULL, /* exit_data_length */
            NULL, /* exit_data */
            key_form,
            key_length,
            mac_key_type,
            macver_key_type,
            kek_key_id_1,
            kek_key_id_2,
            mac_key_id,
            macver_key_id);

    /* Check the return/reason codes. Terminate if there is an error. */
    if (return_code != 0 || reason_code != 0) {
        printf ("Key_Generate failed: "); /* Print failing verb */
        printf ("return_code = %ld", return_code); /* Print return code */
        printf ("reason_code = %ld:\n", reason_code); /* Print reason code */
        return;
    }

    /* Print a banner */
    printf("Cryptographic Coprocessor Support Program example program.\n");
    /* Set up initial values for Key_Generate call */
    return_code = 0;
    reason_code = 0;
    memcp (key_form, KEY_FORM, 4); /* OPOP key pair */
    memcp (key_length, KEY_LENGTH, 8); /* Single-length keys */
    memcp (mac_key_type, KEY_TYPE_1, 8); /* 1st token, MAC key type */
    memcp (macver_key_type, KEY_TYPE_2, 8); /* 2nd token, MACVER key type */
    memset (kek_key_id_1, 0x00, sizeof(kek_key_id_1)); /* 1st KEK not used */
    memset (kek_key_id_2, 0x00, sizeof(kek_key_id_2)); /* 2nd KEK not used */
    memset (mac_key_id, 0x00, sizeof(mac_key_id)); /* Init 1st key token */
    memset (macver_key_id, 0x00, sizeof(macver_key_id)); /* Init 2nd key token */

    /* Generate a MAC/MACVER operational key pair */
    CSNBKGN(&return_code,
            &reason_code,
            NULL, 
            NULL, /* exit_data_length */
            NULL, /* exit_data */
            key_form,
            key_length,
            mac_key_type,
else
    printf("Key_Generate successful.\n");

/* Set up initial values for MAC_Generate call */
return_code = 0;
reason_code = 0;
text_length = sizeof(INPUT_TEXT) - 1; /* Length of MAC text */
memcpy (text, INPUT_TEXT, text_length); /* Define MAC input text */
rule_array_count = 3; /* 3 rule array elements */
memset (rule_array, ' ', sizeof(rule_array)); /* Clear rule array */
memcpy (rule_array[0], MAC_PROCESSING_RULE, 8); /* 1st rule array element */
memcpy (rule_array[1], SEGMENT_FLAG, 8); /* 2nd rule array element */
memcpy (rule_array[2], MAC_LENGTH, 8); /* 3rd rule array element */
memset (chaining_vector, 0x00, 18); /* Clear chaining vector */
memset (mac_value, 0x00, sizeof(mac_value)); /* Clear MAC value */
/* Generate a MAC based on input text */
CSNBMGN (&return_code,
    &reason_code,
    NULL, /* exit_data_length */
    NULL, /* exit_data */
    mac_key_id, /* Output from Key_Generate */
    &text_length,
    text,
    &rule_array_count,
    &rule_array[0][0],
    chaining_vector,
    mac_value);

/* Check the return/reason codes. Terminate if there is an error. */
if (return_code != 0 || reason_code != 0) {
    printf("MAC_Generate Failed: "); /* Print failing verb */
    printf("return_code = %ld, " , return_code); /* Print return code */
    printf("reason_code = %ld.\n", reason_code); /* Print reason code */
    return;
} else {
    printf("MAC_Generate successful.\n");
    printf("MAC_value = %s\n", mac_value); /* Print MAC value (HEX-9) */
}

/* Set up initial values for MAC_Verify call */
return_code = 0;
reason_code = 0;
rule_array_count = 1; /* 1 rule array element */
memset (rule_array, ' ', sizeof(rule_array)); /* Clear rule array */
memcpy (rule_array[0], MAC_LENGTH, 8); /* Rule array element */
    /* (use default Ciphering */
    /* Method and Segmenting */
    /* Control) */
memset (chaining_vector, 0x00, 18); /* Clear the chaining vector */
/* Verify MAC value */
CSNBMVR (&return_code,
    &reason_code,
    NULL, /* exit_data_length */
    NULL, /* exit_data */
    macver_key_id, /* Output from Key_Generate */
    &text_length, /* Same as for MAC_Generate */
    text,
    &rule_array_count,
    &rule_array[0][0],
    chaining_vector,
    mac_value); /* Output from MAC_Generate */

/* Check the return/reason codes. Terminate if there is an error. */
if (return_code != 0 || reason_code != 0) {
    printf("MAC_Verify failed: "); /* Print failing verb */
Sample program in Java

Before running this program, review the information about the JNI interface.

You can find this information in “Building Java applications using the CCA JNI” on page 28.

This sample code consists of a Java program named mac.java. For reference, a copy of the sample routine is shown in Figure 47 on page 1098. Another sample program named RNG.java is included with the distribution at the same location, but is not copied here because it is a very simple JNI reference exercise to call the Random Number Generate verb.

The default distribution location of the sample code is:

**SUSE Linux**

/opt/IBM/CCA/samples

**Red Hat Linux**

/opt/IBM/CCA/samples

Invoke the following command from the directory that contains the sample source code to compile the program:

```
javac -classpath /opt/IBM/CCA/cnm/HIKM.zip mac.java
```

**Note:**
1. The classpath option points to the HIKM.zip file because the hikmNativeNumber class is in this file.
2. The path shown for the HIKM.zip file is the default distribution location of that file.

When it is compiled, you can run the sample Java program from the directory that contains the compiled output, with these commands.

**For a Red Hat Linux system:**

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib64
/export/ibm/java-i386-60/jre/bin/java -classpath /opt/IBM/CCA/cnm/HIKM.zip:. mac
```

**For a SUSE Linux system:**

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib64
java -classpath /opt/IBM/CCA/cnm/HIKM.zip:. mac
```

**Note:**
1. The path shown for the HIKM.zip file is the default distribution location of that file.
2. The libcsulcca.so library for Linux also contains the C support for the CCA Java Native Interface (JNI).
Figure 47. Syntax, sample routine in Java

```java
/*
  Module Name: mac.java
  *
  DESCRIPTIVE NAME: Cryptographic Coprocessor Support Program
  *
  JNI example code
  *
  LICENSED MATERIALS - Property of IBM
  *
  Copyright IBM Corp. 2008, 2011 All Rights Reserved
  *
  US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
  *
  NOTICE TO USERS OF THE SOURCE CODE EXAMPLES
  *
  The source code examples provided by IBM are only intended to assist in the development of a working software program. The source code examples do not function as written: additional code is required. In addition, the source code examples may not compile and/or bind successfully as written.
  *
  International Business Machines Corporation provides the source code examples, both individually and as one or more groups, "as is" without warranty of any kind, either expressed or implied, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the source code examples, both individually and as one or more groups, is with you. Should any part of the source code examples prove defective, you (and not IBM or an authorized dealer) assume the entire cost of all necessary servicing, repair or correction.
  *
  IBM does not warrant that the contents of the source code examples, whether individually or as one or more groups, will meet your requirements or that the source code examples are error-free.
  *
  IBM may make improvements and/or changes in the source code examples at any time.
  *
  Changes may be made periodically to the information in the source code examples; these changes may be reported, for the sample code included herein, in new editions of the examples.
  *
  References in the source code examples to IBM products, programs, or services do not imply that IBM intends to make these products, programs, or services available in all countries in which IBM operates. Any reference to the IBM licensed program in the source code examples is not intended to state or imply that IBM's licensed program must be used. Any functionally equivalent program may be used.
  *
  This example program:
  *
  1) Calls the KeyGenerate verb (CSNBKGN) to create a MAC (message authentication code) key token and a MACVER key token.
  *
  2) Calls the MACGenerate verb (CSNBMGN) using the MAC key token
```
/* from step 1 to generate a MAC on the supplied text string */
/* (INPUT_TEXT). */
/* */
/* 3) Calls the MAC_Verify verb (CSNBMVNR) to verify the MAC for the */
/* same text string, using the MACVER key token created in */
/* step 1. */
/* */
/***************************************************************************/
import java.io.*;
import com.ibm.crypto.cca.jni.*;
public class mac {
  static final String KEY_FORM = "OPOP";
  static final String KEY_LENGTH = "SINGLE ";
  static final String KEY_TYPE_1 = "MAC ";
  static final String KEY_TYPE_2 = "MACVER ";
  static final String INPUT_TEXT = "abcdefhgijklmnopqrstuvwx";
  static final String MAC_PROCESSING_RULE = "X9.9-1 ";
  static final String SEGMENT_FLAG = "ONLY ";
  static final String MAC_LENGTH = "HEX-9 ";

  public static void main (String args[])
  {
    byte [] ByteExitData = new byte [4];
    byte [] Byte_key_form = new byte [4];
    byte [] Byte_key_length = new byte [8];
    byte [] Byte_mac_key_type = new byte [8];
    byte [] Byte_macver_key_type = new byte [8];
    byte [] Byte_mac_value = new byte [10];
    byte [] Byte_chaining_vector = new byte [18];
    byte [] Byte_rule_array = new byte [24];
    byte [] Byte_text = new byte [26];
    byte [] Byte_kek_key_id_1 = new byte [64];
    byte [] Byte_kek_key_id_2 = new byte [64];
    byte [] Byte_mac_key_id = new byte [64];
    byte [] Byte_macver_key_id = new byte [64];

    try
    {
      //setup to pause on non-zero return/reason code
      //and require enter key to continue
      BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));

      himNativeLong IntReturncode = new himNativeLong(0);
      himNativeLong IntReasoncode = new himNativeLong(0);
      himNativeLong IntExitDataLength = new himNativeLong(0);

      /* Print beginning banner */
      System.out.println("Cryptographic Coprocessor Support Program JAVA example program.\n");

      /* Set up initial values for KeyGenerate call */
      Byte_key_form = new String(KEY_FORM).getBytes(); // OPOP key pair
      Byte_key_length = new String(KEY_LENGTH).getBytes(); // Single-length keys
      Byte_mac_key_type = new String(KEY_TYPE_1).getBytes(); // 1st token, MAC key type
      Byte_macver_key_type = new String(KEY_TYPE_2).getBytes(); // 2nd token, MACVER key type

      /* Generate a MAC/MACVER operational key pair */
      new HIKM().CSNBKGNJ (IntReturncode,
                       IntReasoncode,
                       IntExitDataLength,
                       ByteExitData,
                       Byte_key_form,
                       Byte_key_length,
                       Byte_mac_key_type,
                       Byte_macver_key_type,
                       Byte_kek_key_id_1,
                       Byte_return_reason,
                       Byte_exit_reason,
                       Byte_exit_data_length,
                       Byte_key_form,
                       Byte_key_length,
                       Byte_mac_key_type,
                       Byte_macver_key_type,
                       Byte_kek_key_id_1,
                       Byte_chaining_vector,
                       Byte_rule_array,
                       Byte_text,
                       Byte_kek_key_id_1,
                       Byte_kek_key_id_2,
                       Byte_mac_key_id,
                       Byte_macver_key_id,
                       Byte_mac_value,
                       Byte藐查LogicalGist段落
Byte_kek_key_id_2,
Byte_mac_key_id,
Byte_macver_key_id);

if (0 != IntReturncode.getValue() || 0 != IntReasoncode.getValue())
{
    System.out.println("\nKey Generate Failed");            /* Print failing verb. */
    System.out.println("Return_code = "+ IntReturncode.getValue()); /* Print return code. */
    System.out.println("Reason_code = "+ IntReasoncode.getValue()); /* Print reason code. */
    System.out.println("Press ENTER to continue...");        /* Print Pause message */
    stdin.readLine();
}
else
{
    System.out.println("Key_Generate successful.");
}

/* Set up initial values for MAC_Generate call */
IntReturncode = new hikmNativeLong(0);
IntReasoncode = new hikmNativeLong(0);
IntExitDataLength = new hikmNativeLong(0);

hikmNativeLong Int_rule_array_count = new hikmNativeLong(3);
hikmNativeLong Int_text_length = new hikmNativeLong(24);

byte[] temp_array = new String(SEGMENT_FLAG).getBytes();      /* Define MAC input text */
System.arraycopy(temp_array, 0, Byte_rule_array, 8, temp_array.length);  /* 1st rule array element */

temp_array = new String(MAC_LENGTH).getBytes();                 /* 2nd rule array element */
System.arraycopy(temp_array, 0, Byte_rule_array, 16, temp_array.length); /* 2nd rule array element */

/* Generate a MAC based on input text */
new HIKM().CSNBMGNJ (IntReturncode,
IntReasoncode,
IntExitDataLength,
ByteExitData,
Byte_mac_key_id,
Int_text_length,
Byte_text,
Int_rule_array_count,
Byte_rule_array,
Byte_chaining_vector,
Byte_mac_value);

if (0 != IntReturncode.getValue() || 0 != IntReasoncode.getValue())
{
    System.out.println("\nMAC Generate Failed");            /* Print failing verb. */
    System.out.println("Return_code = "+ IntReturncode.getValue()); /* Print return code. */
    System.out.println("Reason_code = "+ IntReasoncode.getValue()); /* Print reason code. */
    System.out.println("Press ENTER to continue...");        /* Print Pause message */
    stdin.readLine();
}
else
{
    System.out.println("MAC_Generate successful.");
    System.out.println("MAC_value = [" + new String(Byte_mac_value) + "]");
}

/* Set up initial values for MAC_Verify call */
IntReturncode = new hikmNativeLong(0);
IntReasoncode = new hikmNativeLong(0);
IntExitDataLength = new hikmNativeLong(0);
Byte_rule_array = new String (MAC_LENGTH).getBytes(); /* Rule array element */
Int_rule_array_count = new hikmNativeLong(1);

new HIKM().CSNBMVRJ (IntReturncode,
    IntReasoncode,
    IntExitDataLength,
    ByteExitData,
    Byte_macver_key_id,
    Int_text_length,
    Byte_text,
    Int_rule_array_count,
    Byte_rule_array,
    Byte_chaining_vector,
    Byte_mac_value);

if (0 != IntReturncode.getValue() || 0 != IntReasoncode.getValue())
{
    System.out.println ("\nMAC_Verify Failed"); /* Print failing verb. */
    System.out.println ("Return_code = " + IntReturncode.getValue()); /* Print return code. */
    System.out.println ("Reason_code = " + IntReasoncode.getValue()); /* Print reason code. */
    System.out.println ("Press ENTER to continue..."); /* Print Pause message */
    stdin.readLine();
} else
{
    System.out.println ("MAC_Verify successful.");
}

} catch (Exception anException)
{
    System.out.println(anException);
}

/* Print ending banner */
System.out.println("\nCryptographic Coprocessor Support Program JAVA example program finished.\n");
//end main
//end mac class
Chapter 28. Initial system set-up tips

Use these tips to help you set up your system for the first time.

The name of the CCA package is:

\[ \text{csulcca-<version>.<release>.<maintenanceLevel>-<fixLevel>.s390x.rpm} \]
\[ \text{csulcca_<version>.<release>.<maintenanceLevel>-<fixLevel>_s390x.deb} \]

where:

- **version** represents the major version, typically the same for an adapter life
- **release** represents the function release available, updated to add function
- **maintenanceLevel** represents the level of increment on top of the function release for small functional changes
- **fixLevel** represents a level of fix applied, typically incremented during internal package development.

Examples:

- `csulcca-6.0.9-11.s390x.rpm`
- `csulcca_6.0.9-11_s390x.deb`

In order to use the full set of CCA functions, a CEX*C feature in CCA coprocessor mode is required. This feature must have a CCA code level appropriate for the desired function provided by the RPM or DEB package.

Consult the README.txt file in the `/opt/IBM/CCA/doc/` directory for this information:

- Release-specific information, if there is any
- Pointers to helpful tools

**Note:** If the version and release level of the coprocessor feature is lower than the version and release level of the CCA package (RPM or DEB), then only the functions supported by the feature are supported by the package.

Installing and loading the cryptographic device driver

The cryptographic device driver is included in the regular kernel package shipped with your Linux distribution.

In earlier Linux distributions, the cryptographic device driver is shipped as a single module called `z90crypt`. In more recent distributions, the cryptographic device driver is shipped as set of modules with the `ap` module being the main module that triggers loading all required sub-modules. There is, however, an alias name `z90crypt` that links to the `ap` main module.

There might be distributions using kernel levels starting with 4.10, that have basic cryptographic device driver support as part of the kernel (that is, the `ap` module is already compiled in the kernel). In this case, the subsequently mentioned `lsmod` and `modprobe` commands do not work as described. In addition, the domain and
poll_thread parameters are no longer module parameters, but kernel parameters. In this case, you can change the values directly via sysfs, or change as kernel parameters. Refer to the Device Drivers, Features, and Commands for kernel 4.12 or later on the developerWorks website for further information.

For installations with a loadable cryptographic device driver, use the `lsmod` command to find out if either the `z90crypt` or the `ap` module is already loaded.

If required, use the `modprobe` command to load the `z90crypt` or `ap` module. When loading the `z90crypt` or `ap` module, you can use the following optional module parameters:

- **domain=**
  - specifies a particular cryptographic domain. By default, the device driver attempts to use the domain with the maximum number of devices. To use all CCA 6.0 functions, the domain must include at least one CEX6C feature.
  - After loading the device driver, use the `lszcrypt` command with the `-b` option to confirm that the correct domain is used. If your distribution does not include this command, see the version of Device Drivers, Features, and Commands that applies to your distribution about how to use the sysfs interface to find out the domain. This publication also provides more information about loading and configuring the cryptographic device driver.
  - To change the domain, you must unload the `z90crypt` or `ap` module (see “Unloading the cryptographic device driver”) and reload it.
  - You should also read the information presented in “Domain selection capabilities” on page 10.

- **poll_thread=**
  - enables the polling thread for instances of Linux on z/VM and for Linux instances that run in LPAR mode on an IBM mainframe earlier than z10.
  - For Linux instances that run in LPAR mode on a z10 or later mainframe, this setting is ignored and AP interrupts are used instead.
  - For more information about these module parameters, the polling thread, and AP interrupts, see the version of Device Drivers, Features, and Commands that applies to your distribution.

---

### Unloading the cryptographic device driver

You might need to unload the cryptographic device driver, for example, to change the domain setting.

Unloading the device driver is complicated slightly because the catcher.exe daemon is always running, to be ready to receive TKE requests. To unload the device driver, for example, in preparation for a reload, you must stop the catcher.exe daemon. This can be done with the service management script `/etc/init.d/CSUTKEcat` using the `stop` argument, or by using the `ps` command to find the PID for the daemon, and then using the `kill -9 <PID>` command to kill it.

Unloading of the device driver is only successful, if you stop all applications that access the main `ap` module and any of the running sub-modules before the unload. Unloading fails, if you miss to stop an involved application. In this case, the device driver requires a restart to be able to function as desired.
After reloading the device driver, you can restart the catcher.exe daemon (restarting TKE access) using the /etc/init.d/CSUTKEcat start command. See “Files in the RPM or DEB” on page 1109 for more details.

**Note:** If the cryptographic device driver is part of the kernel, you cannot unload it. In this case, you can directly edit domain settings via sysfs (see “Checking the adapter settings”).

### Confirming your cryptographic devices

Use the lszcrypt command to display a list of available cryptographic devices and their online status.

<table>
<thead>
<tr>
<th>CARD.DOMAIN</th>
<th>TYPE</th>
<th>MODE</th>
<th>STATUS</th>
<th>REQUEST_CNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>04</td>
<td>CEX5C</td>
<td>CCA-Coproc</td>
<td>online</td>
<td>2174</td>
</tr>
<tr>
<td>04.0013</td>
<td>CEX5C</td>
<td>CCA-Coproc</td>
<td>online</td>
<td>2174</td>
</tr>
<tr>
<td>04.0014</td>
<td>CEX5C</td>
<td>CCA-Coproc</td>
<td>offline</td>
<td>0</td>
</tr>
<tr>
<td>05</td>
<td>CEX5P</td>
<td>EP11-Coproc</td>
<td>online</td>
<td>0</td>
</tr>
<tr>
<td>05.0013</td>
<td>CEX5P</td>
<td>EP11-Coproc</td>
<td>online</td>
<td>0</td>
</tr>
<tr>
<td>05.0014</td>
<td>CEX5P</td>
<td>EP11-Coproc</td>
<td>online</td>
<td>0</td>
</tr>
<tr>
<td>0a</td>
<td>CEX6C</td>
<td>CCA-Coproc</td>
<td>online</td>
<td>15420</td>
</tr>
<tr>
<td>0a.0013</td>
<td>CEX6C</td>
<td>CCA-Coproc</td>
<td>online</td>
<td>15420</td>
</tr>
<tr>
<td>0a.0014</td>
<td>CEX6C</td>
<td>CCA-Coproc</td>
<td>offline</td>
<td>0</td>
</tr>
</tbody>
</table>

In the example, you see one CEX5C device and one CEX6C device offline. All other devices are online.

You can use the chzcrypt command to set a cryptographic device online. For example, to set a cryptographic domain X’0014’ on device X’0a’ online, issue:

```
chzcrypt -e 0a.0014
```

For more details about the lszcrypt and chzcrypt commands, see the man pages.

If your distribution does not include these commands, see the version of Device Drivers, Features, and Commands that applies to your distribution about how to use the sysfs interface to find out this information and to set devices online.

### Checking the adapter settings

Use the lszcrypt command with the -b option to display information about the settings for your cryptographic adapters.

```
# lszcrypt -b
ap_domain=0x4
ap_max_domain_id=0x54
ap_interrups are disabled
config_time=30 (seconds)
poll_thread is disabled
poll_timeout=1500000 (nanoseconds)
```

As seen in the example, the command output shows:

- the used domain in hexadecimal notation

**Note:** To change the domain you must unload the z90crypt module and reload it with the appropriate domain= module parameter. In installations with the in-kernel ap module and multi-domain support, you achieve this directly by changing the ap_domain value via sysfs.
- whether ap interrupts are used
- whether the polling thread is used and, if so, at which frequency
Note: Polling threads cannot be enabled if AP interrupts are available.

If your distribution does not include the `lscrypt` command, see the version of *Device Drivers, Features, and Commands* that applies to your distribution about how to use the `sysfs` interface to retrieve this information.

---

### Performance tuning

If no AP interrupts are available to your Linux instance, you can use the settings for the polling thread and the high resolution polling timer to tune the performance of your cryptographic adapters.

See the version of *Device Drivers, Features, and Commands* that applies to your distribution for more information about these settings.

---

### Running secure key under a z/VM guest

In order to use the CEX*C feature under z/VM, you need to apply specific APAR fixes.

These fixes are described in “Hardware requirements” on page xxi.

Also, to get secure key running under a z/VM guest, a directory control statement (CRYPTO APDED) for a given z/VM guest needs to be used. This requires that the APs with this domain are owned by the LPAR. There is no virtualization done by z/VM.

For secure key, z/VM does not virtualize the AP’s. The AP’s need to be dedicated, which is done by the user statement:

```
CRYPTO DOMAIN 12 APDED 5 7
```

This statement dedicates AP’s 5 and 7 for domain 12 to one Linux guest.

Note: Shared crypto adapters, as defined with the z/VM user directory statement CRYPTO APVIRT, cannot be used for secure key cryptographic operations. Because dedicated and shared cryptographic adapters cannot be mixed in a z/VM guest virtual machine, additional crypto adapters for use with clear key cryptography, coprocessors or accelerators, must be defined as dedicated adapters.
Chapter 29. CCA installation instructions

Use these instructions when you install, configure, or uninstall CCA release 6.0.0 and later.

You can install the new release either on a new system or over an earlier release, such as CCA 4.* or 5.*

Before you begin

In the installation instructions, the term CCA 6.x is used to describe CCA packages referring to CCA 6.0.0 and subsequent versions.

Before you begin the CCA installation, review these points:

- If you are upgrading from a prior installation, be sure to review “Default installation directory” on page 1108.
- Ensure that you are using supported hardware. See “Hardware requirements” on page xxi.
- Ensure that your Linux distribution has the cryptographic device driver support. For details, see “Installing and loading the cryptographic device driver” on page 1103.
- If you are going to use the Java Native Interface (JNI), see “Building Java applications using the CCA JNI” on page 28 for supported Java levels and installation instructions.

Ubuntu installation considerations

The installation process for Ubuntu is very similar to that for the other distributions. Read the provided information about special Ubuntu considerations.

Package
IBM CCA Host Libraries and Tools

Name csulcca-version-arch.deb

Target Architecture
tested and supported for 64-bit s390x; 31-bit is not supported.

Further information
There is a web location presenting a file with eventually updated Ubuntu installation instructions:
Software packages for IBM z Systems servers running Linux

1. Click on the Software version that matches your environment (such as CCA Releases 6.0 for CEX6C ... for RHEL, SLES, and Ubuntu)
2. Select Ubuntu installation instructions to open and download a text file with the most recent instructions and also with a list of known issues.

Note: Debian, and by extension Ubuntu, use a different model for DSO Linking when compared to RHEL and SLES. Ubuntu adds --as-needed for gcc when gcc calls the linker. This can cause a link to fail on Ubuntu with undefined reference issues for symbols that are defined in libcsulcca.so. The remedy is to add this argument to the gcc call
When building our sample program, as copied from this documentation to a file named sample.c, an appropriate compile call would look like the following:

```
gcc -g -Wl,--no-as-needed -I/opt/IBM/CCA/include -lcsulcca -lcsulccamk -o sample sample.c
```

### Default installation directory

The CCA RPM or DEB by default choose overall installation directories for their contents.

- `/usr/lib64/` (RPM) and `/usr/lib/` (DEB)
  - all libraries to be used by applications

- `/opt/IBM/CCA/
  - all other actual files from the release are placed in sub-directories of this path

In the past a part of the path name was the name of the specific CCA coprocessor being supported, such as CEX3C. As of CEX4C support, multiple CCA coprocessors are supported from the same install tree, and it is no longer appropriate to use the CCA coprocessor name in the path. The new default install path is:

```
/opt/IBM/CCA/
```

Soft-links to the following paths are added for ease of use and migration:

- `/opt/ibm/cca`
- `/opt/IBM/CEX3C/`

If you are upgrading, note that the RPMs copy your key storage files from the old default location to the new default location. The old directory is kept and renamed to `/opt/IBM/CEX3C-old/`.

### Download and install the RPM or DEB file

RPM is the installation package format for Red Hat Enterprise Linux (RHEL) and SUSE Linux Enterprise Server (SLES) distributions. DEB is the package format for the Ubuntu distribution. The CCA RPM or DEB packages contain files, samples, and groups.

#### About this task

To download the RPM or DEB package, complete these steps:

**Procedure**

1. Point your Web browser at this location:
   

2. Click the appropriate link for your cryptographic coprocessor / HSM from the Cryptocards navigation bar on the left hand side.

3. Click Download software package from the left navigation bar on the appearing web page.

4. On the next web page, in column Platform, find the entry IBM Z servers running Linux.
5. In the corresponding entry in column **Software / Firmware available**, click on the link reading **software-package selection page**. On the subsequent page, you find instructions and links to continue the download the following parts:

- the README file for the RPM or DEB
- Release Notes
- the RPM or DEB installation file that installs the host code.

**Files in the RPM or DEB**

These files are included in the RPM or DEB.

```bash
/etc/profile.d/csulcca.sh
```

Environment variables are created in this file. Customers should read this file for up-to-date information. The key storage environment variables are added here. See "Environment variables for the key storage file” on page 450.

```bash
/etc/profile.d/csulcca.csh
```

Environment variables are created in this file. Customers should read this file for up-to-date information. The key storage environment variables are added here. See "Environment variables for the key storage file” on page 450.

**Note:** `/etc/profile.d/csulcca.sh` and `/etc/profile.d/csulcca.csh` are exactly the same in what they do, but have syntax differences. Only one of these two files is used, depending on the configuration of the particular user running an application.

```bash
/etc/init.d/CSUTKEcat
```

System initialization script that automatically starts the `catcher.exe` daemon when Linux starts. You can also use this script to start or stop `catcher.exe` from the command line. To start `catcher.exe`, issue:

```bash
/etc/init.d/CSUTKEcat start
```

To stop `catcher.exe`, issue:

```bash
/etc/init.d/CSUTKEcat stop
```

You can also start the `catcher.exe` using system services, if available:

```bash
]# service CSUTKEcat start
]# service CSUTKEcat stop
```

```bash
/etc/rc.d/rc2.d/S14CSUTKEcat
```

Link to: `/etc/init.d/CSUTKEcat`

```bash
/etc/rc.d/rc3.d/S14CSUTKEcat
```

Link to: `/etc/init.d/CSUTKEcat`

```bash
/etc/rc.d/rc5.d/S14CSUTKEcat
```

Link to: `/etc/init.d/CSUTKEcat`

```bash
/opt/IBM/CCA/bin/TKECM.dat
```

```bash
/opt/IBM/CCA/bin/acpoints.dat
```

```bash
/opt/IBM/CCA/bin/catcher.exe
```

Daemon executable that is controlled by the `catcher.exe` script. This daemon listens on TCP port 50003 for requests from the Trusted Key Entry workstation (TKE) for secure commands to administer any adapters configured as available to the system. See the TKE documentation for usage and capabilities. The daemon depends on libcsulcca.so and on the
z90crypt or ap device driver. While the daemon is running, it has an open file handle to the /dev/z90crypt device node maintained by the cryptographic module. Therefore, catcher.exe as a running process affects whether you can load or unload z90crypt or ap respectively. Use /etc/init.d/CSUETKEcat to start catcher.exe or to stop it if you want to unload z90crypt or ap respectively. Remember that TKE secure administration requires a running catcher.exe process.

/opt/IBM/CCA/bin/panel.exe
Utility: run with no arguments for help. See also Chapter 31, “Utilities,” on page 1119

/opt/IBM/CCA/bin/ivp.e
Utility: run with no arguments for install verification. See also Chapter 31, “Utilities,” on page 1119

/opt/IBM/CCA/cnm/CNMM.jar
jar archive of the Java packaged version of the JNI interface, implementing the Java versions of most CCA calls.

/opt/IBM/CCA/cnm/CNMMD.jar
jar archive of the Java packaged version of the JNI interface, implementing the Java versions of CCA calls for changing the Master Key in the CEX*C

/opt/IBM/CCA/doc/README.txt
contains release-specific information, if applicable and pointers to helpful tools

/opt/IBM/CCA/doc/license.txt
entry point to license documentation identifying release and product specific terms

/opt/IBM/CCA/doc/
contains files that describe classes added by the JNI support for CCA:
• hikmNativeNumber.html
• hikmNativeInteger.html
• hikmNativeLong.html

/opt/IBM/CCA/include/csulincl.h
C header file that is listing the names of the verbs and their parameters.

/usr/lib64/libcsulcca.so (RPM) and /usr/lib/libcsulcca.so (DEB)
This file is a link to the most recent library file, with name
libcsulcca.so.V.R.M where: V = version, R = release, and M = maintenance level.

/usr/lib64/libcsulccamk.so (RPM) and /usr/lib/libcsulccamk.so (DEB)
This file is a link to the most recent library file, with name
libcsulccamk.so.V.R.M where: V = version, R = release, and M = maintenance level.

/opt/IBM/CCA/keys/README.keys
Samples in the RPM or DEB
These samples are included in the RPM or DEB.

/opt/IBM/CCA/samples/mac.c
C code sample

/opt/IBM/CCA/samples/makefile.lnx
Used to build mac.c
Groups in the RPM or DEB

These groups are created for the purpose of loading master keys.

They are added during RPM or DEB installation as updates to /etc/groups. See Table 315 on page 1115

- cca_admin
- cca_clrmk
- cca_lfmkp
- cca_cmkp
- cca_setmk

Install and configure the RPM or DEB

Use the following steps to install and configure the CCA RPM or DEB.

Procedure

1. Copy the RPM or DEB to the host where it will be installed. For example, /root on your host image.
2. Login to the host as root. Change to the directory where the installation package is located by issuing these commands:
   <login to host>
   cd /root/
3. Install either the RPM or DEB by issuing one of the following commands:
   rpm -i <rpm_name>  /* for RPM */
   dpkg -i <deb_name>  /* for DEB */

Note:

a. For compatibility reasons a softlink is created from /opt/IBM/CCA to /opt/IBM/CEX3C. This is not valid when installing the DEB package.
b. If this is an upgrade, you can use this command:
   rpm -Uvh <rpm_name>  /* for RPM */
   dpkg -i <deb_name>  /* for DEB, same as for installation */
c. If you are installing the RPM on a SUSE Linux distribution, you might receive the following warning messages because of an unsupported groupadd option.

   groupadd: You are using an undocumented option (-f)!
   groupadd: You are using an undocumented option (-f)!
   groupadd: You are using an undocumented option (-f)!
   groupadd: You are using an undocumented option (-f)!

   No action on your part is needed. The installation proceeds with another call if this happens.
4. Reboot the host by issuing the following command: shutdown -r now This is necessary because of the defaults added to /etc/profile.d/csulcca.sh and /etc/profile.d/csulcca.csh for using CCA must be propagated to all user login sessions.

Chapter 29. CCA installation instructions  1111
If all users that use the CCA logout and then login again, and if all applications that use CCA are re-started, then a reboot may be avoided. It is always recommended to reboot to ensure for new users or new system administrators that the updated profiles are actually in force after the install procedure is completed.

5. Login to the host as root. Change to the directory where the RPM or DEB binaries are installed by issuing the following command:

```
<login as root to host>
cd /opt/IBM/CCA/bin/
```

6. Verify that at least one card is present and active:

**Note:** Starting with kernel 4.10, the cryptographic device driver (zcrypt) is statically built into the kernel. So there is no need to load the ap module any longer.

a. If required, you can use the `lszcrypt` command to check if the zcrypt device driver is available:

```bash
$ lszcrypt
```

If the command displays the message Crypto device driver not available, the zcrypt device driver is not loaded. To load zcrypt, issue the following command:

```bash
$ modprobe ap
```

or on older kernels, issue:

```bash
$ modprobe z90crypt
```

With the zcrypt device driver loaded, you can use `lszcrypt` to see the installed and configured crypto adapters for your LPAR or z/VM guest:

<table>
<thead>
<tr>
<th>CARD.DOMAIN</th>
<th>TYPE</th>
<th>MODE</th>
<th>STATUS</th>
<th>REQUEST_CNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>CEX6A Accelerator</td>
<td>online</td>
<td>2174</td>
<td>Adapted 0</td>
</tr>
<tr>
<td>00.0005</td>
<td>CEX6A Accelerator</td>
<td>online</td>
<td>1347</td>
<td>Domain 0 on Adapter 0</td>
</tr>
<tr>
<td>00.001f</td>
<td>CEX6A Accelerator</td>
<td>online</td>
<td>827</td>
<td>Domain 31 (X'1F')</td>
</tr>
<tr>
<td>01</td>
<td>CEX6C CCA-Coproc</td>
<td>online</td>
<td>15635</td>
<td>Adapter 1</td>
</tr>
<tr>
<td>01.0005</td>
<td>CEX6C CCA-Coproc</td>
<td>online</td>
<td>9187</td>
<td></td>
</tr>
<tr>
<td>01.001f</td>
<td>CEX6C CCA-Coproc</td>
<td>online</td>
<td>6448</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>CEX6P EP11-Coproc</td>
<td>online</td>
<td>17</td>
<td>Adapter 5</td>
</tr>
<tr>
<td>05.0005</td>
<td>CEX6P EP11-Coproc</td>
<td>online</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>05.001f</td>
<td>CEX6P EP11-Coproc</td>
<td>online</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

The latest version of the zcrypt device driver can deal with more than one domain. The previous screen shot shows two domains (with IDs 5 and 31). By default the device driver uses the domain with the lowest ID. You can change this by setting a new default domain value using the command

```bash
$ chzcrypt --default-domain=x
```

or with a kernel boot parameter `ap.domain=x` (where x must be in hexadecimal notation).

For CCA applications, an adapter in CCA coprocessor mode is needed. So the `lszcrypt` output should show some CEX*C cards, for example, CEX6C.
b. When you are sure that the device driver is loaded, you can display details of all the available CCA coprocessors and verify their accessibility by running one of the RPM or DEB installed utilities:

1)  /opt/IBM/CCA/bin/ivp.e
   This command performs a health check for all active cards.

2)  /opt/IBM/CCA/bin/panel.exe -status
   This command shows the serial numbers and master key register states of all active cards running CCA that are visible to this Linux host. The total number of active cards and any errors are also reported. You also see the CCA numbering of the CEXnC adapters and the default card is shown.

Note:
1) To be able to use /opt/IBM/CCA/panel.exe the user must be either root or a member of the cca_admin group, that is, the owner of
   /usr/lib64/libcsulccamk.so /* for RPM */
   /usr/lib/libcsulccamk.so /* for DEB */

2) If there is not at least one active card at this point, double check earlier steps and, if necessary, involve IBM service because the rest of the setup is designed around having active cards.

3) Unloading the device driver requires killing the catcher.exe program, and then restarting it when the driver is reloaded. See the note in “Installing and loading the cryptographic device driver” on page 1103 for specific instructions.

    Note: Unloading the device driver is not possible with the ap module being statically in the kernel.

7. Master key load - This procedure is for using the Linux on Z native API or the utility (panel.exe) to load the master keys for the active cards.

There are several methods available to load the master keys, and it is important to choose the correct method for your production environment. For more information, see "CCA Master Key administration: choosing the right method or tool" on page 1119.

If you want to use TKE instead of the panel.exe utility, refer to a TKE manual, such as the IBM Redbooks publication Exploiting S/390 Hardware Cryptography with Trusted Key Entry for proper use. After completing this step using the TKE procedure, go to Step 8 on page 1116.

a. Setup the groups for the users who will be loading the master keys to the cards. Each part of the load process is owned by a different Linux group created by the RPM or DEB install procedure, and verified in the host library implementing the API allowing master key processing. To complete a specific step the user must have membership in the proper group. There are a couple ways to change group membership depending on your Linux distribution. A third option is to create the users specifically for these roles. If a user does not have the proper group membership for a particular master key operation, the error X'0008005a' is returned and an error message is printed to the system log.

    Note: To be able to use /opt/IBM/CCA/panel.exe, the user must be either root or a member of the cca_admin group, that is, the owner of
   /usr/lib64/libcsulccamk.so /* for RPM */
   /usr/lib/libcsulccamk.so /* for DEB */

1) Group membership for Red Hat and Ubuntu based Linux distributions:
a) Use the `groups` command to see a list of the user's current group membership:

```bash
groups <user name>
---output is
   <user name> : <grouplist>
   <grouplist> is a single-space separated list
```

b) `<grouplist>` must be passed along with the new group to the `usermod` command as a comma-separated list, followed by the `<user name>`. For example, if you wanted to add `cca_lfmkp` membership to user named `admin`, you would use the following commands:

```bash
groups admin
---output:
   admin : admin bin daemon sys wheel
usermod -G admin,bin,daemon,sys,wheel,cca_lfmkp admin
---output:
[none if successful]
```

**Note:** Ensure the user logs out and logs back in, otherwise the group membership in the active session will not be updated.

2) Group membership for SUSE-based Linux distributions:

Use the `usermod` command to add membership for a specific group for a specific user. For example, if you wanted to add `cca_lfmkp` membership to user `admin`, you would use the following commands:

```bash
usermod -G cca_lfmkp admin
```

**Note:** Ensure the user logs out and logs back in, otherwise the group membership in the active session will not be updated.

3) Create users for each role with correct group memberships (Same commands for Red Hat, SUSE, and Ubuntu):

a) Create user `cca_user`, which will own default key storage by issuing the following commands:

i. `useradd -g cca_admin -d /home/cca_user -m cca_user`

   This command creates the user with primary group `cca_admin` and a new home directory.

ii. `passwd cca_user`

   This command sets the new user's password.

b) Create user `cca_lfmkp` by issuing the following commands:

i. `useradd -g cca_admin -d /home/cca_lfmkp -G cca_admin,cca_lfmkp -m cca_lfmkp`

   This command creates the user with primary group `cca_admin`, secondary group `cca_lfmkp`, and a new home directory.

ii. `passwd cca_lfmkp`

   This command sets the new user's password.

c) Create user `cca_cmkp` by issuing the following commands:

i. `useradd -g cca_admin -d /home/cca_cmkp -G cca_admin,cca_cmkp -m cca_cmkp`

   This command creates the user with primary group `cca_admin`, secondary group `cca_cmkp`, and a new home directory.

ii. `passwd cca_cmkp`
This command sets the new user's password.

d) Create user cca_clrmk by issuing the following commands:
   i. `useradd -g cca_admin -d /home/cca_clrmk`  
      `-G cca_admin,cca_clrmk`  
      `-m cca_clrmk`

   This command creates the user with primary group cca_admin,  
   secondary group cca_clrmk, and a new home directory.
   ii. `passwd cca_clrmk`

   This command sets the new user's password.

e) Create user cca_setmk by issuing the following commands:
   i. `useradd -g cca_admin -d /home/cca_setmk`
      `-G cca_admin,cca_setmk`
      `-m cca_setmk`

   This command creates the user with primary group cca_admin,  
   secondary group cca_setmk, and a new home directory.
   ii. `passwd cca_setmk`

   This command sets the new user's password.

b. Add group membership privileges to users based on their required  
function.

<table>
<thead>
<tr>
<th>Group Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cca_admin</td>
<td>All users who will run part of the master key load process must be in this group because the library itself is owned by root.cca_admin, with no permissions for ‘world’ as a protective measure. Reasons for this separate group also include allowing one owner of /usr/lib64/libcsulccamk.so and of /usr/lib/libcsulccamk.so for DEB, and allowing use of panel.exe without allowing any of the master key processing calls.</td>
</tr>
<tr>
<td>cca_lfmkp</td>
<td>The user to LOAD the first key part must be in this group.</td>
</tr>
<tr>
<td>cca_cmkp</td>
<td>The users to LOAD the middle and last key parts must be in this group.</td>
</tr>
<tr>
<td>cca_clrmk</td>
<td>The new master-key register can be CLEARed using the same Master Key Process call in case a mistake was made entering a key part (use the key verification patterns to check for this). To perform the clear, the user must be a member of this group.</td>
</tr>
<tr>
<td>cca_setmk</td>
<td>The user to call SET after the last key part has been successfully loaded must be a member of this Linux group.</td>
</tr>
</tbody>
</table>

c. Load FIRST, MIDDLE (optional), and LAST key parts for the AES, SYM,  
ASYM, and APKA master keys and then call SET for each master key. This  
step can be done using the panel.exe utility provided or by writing your  
own application to call the Master Key Process (CSNBMKP) verb directly.  
The application must link with the correct library (installed to /usr/lib64/libcsulccamk.so by RPM and to /usr/lib/libcsulccamk.so by DEB), and must be executed at each step by a user with the appropriate group memberships. The utility supports scripted as well as prompt-driven access.  
Repeat this step for each configured adapter. See “Changing the master  
key for two or more adapters that have the same master key, with shared  
CCA key storage” on page 454.  
For details about panel.exe, see “The panel.exe utility” on page 1121.  
See “Master Key Process (CSNBMKP)” on page 157 about parity requirements for master key parts.
Note: Loading master key parts modifies state information inside the card. For example you cannot load a 'FIRST' master key part twice in a row without clearing the new master-key register in between attempts. The same goes for setting the 'LAST' register. Any number of 'MIDDLE' parts can be loaded - with each call changing the contents of the new master-key register. Similarly a 'SET' operation changes the state of the 'new' register back to 'empty', while updating the 'current' register.

8. **Key storage initialization** - To perform this step, see "Using panel.exe for key storage initialization" on page 1126.

9. **Key storage re-encipher when changing the master key** - To perform this step, see "Using panel.exe for key storage re-encipher when changing the master key" on page 1128.

10. If you are going to be using Central Processor Assist for Cryptographic Functions (CPACF), it must be configured. See "CPACF support" on page 14.

---

**Uninstall the RPM or DEB**

Use the following steps to uninstall the CCA RPM or DEB.

**Procedure**

1. Uninstall any RPMs or DEBs that depend on the CCA RPM or DEB packages. If you try to uninstall the CCA RPM or DEB and dependent RPMs or DEBs are still installed, the uninstall RPM or DEB command will fail and list the names of dependent packages. Therefore, you can skip to Step 2 and come back to this step if Step 2 fails for that reason.

2. Uninstall the CCA RPM or DEB.
   a. Login as root. You must be root to uninstall the RPM or DEB.
   b. You must use the full name. You can find the name by issuing one of the following commands:

   ```
   rpm -qa | grep csulcca /* for RPM */
   dpkg -l | grep csulcca /* for DEB */
   ```

   c. Uninstall the RPM or DEB with the following command:

   ```
   rpm -e <rpm_name> /* for RPM */
   dpkg -P <deb_name> or dpkg --purge <deb_name> /* for DEB */
   ```

   **Note:**

   a. Groups are no longer deleted during the uninstall of CCA RPM or DEB. If you created any users with one of the groups created by the package install as their primary (note that the package install does NOT create any users, just groups), you can delete those users/groups yourself after uninstall, or remove such users before the uninstall of the RPM or DEB. This will remove any potential security holes.

   b. Card master keys (and other state information) are untouched by the host-side uninstall of the RPM or DEB.

   c. Key storage files are not deleted by the uninstall. All default and non-default key storage files are left as is. If you reinstall or install an upgraded package and load any new cards with the same master keys, you still can use your old key storage (old cards still have the old keys, see Step 7b on page 1115 of "Install and configure the RPM or DEB" on page 1111).
Chapter 30. Coexistence of CEX6C and previous CEX*C features

While only CEX6S and CEX5S coprocessors are supported on processors starting with IBM z14, the CCA 6.0 RPM is also useful on IBM z13 and other legacy systems. The CCA 6.0 rpm specifically adds support for the new CCA 5.3 firmware on z13. This information unit discusses co-existence considerations for the CCA 5.2 rpm.

- Use CCA 6.0 to update existing 5.* installations or to install it to a fresh client. Running CCA 6.0 in parallel with a prior 5.* rpm is not supported.
- The latest CCA RPM supersedes and replaces earlier RPMs if installed in the default manner. It is not recommended to manually alter the installation in order to run newer versions of CCA RPMs in parallel with older versions. This configuration is not supported.

Concurrent installations

These are background considerations for installation of the CEX6S RPM alongside with earlier CEX*C RPMs.

1. The libcsulcca.so and libcsulsapi.so libraries for CCA 6.0 and CCA 5.* have many symbols with the same names. An application cannot deterministically link with both libraries. The first library in the link statement is used for all symbols that can be resolved there, after that the second library will be examined. At this point, either the linker will not allow the link to continue by throwing an error on the duplicate symbols, or will produce a hybrid-linked application. Either case gives the user the wrong answer.

A new or updated library cannot itself resolve this kind of conflict because:

- There is no way to have a default set of symbols or card support in an updated host library. The link operation is a fundamental step in building the customer application and outside the control of the library or library installation process.
- One way to resolve name collisions is to change all of the function names in the new library. However, this would have greatly impacted the customer's ability to port applications forward, and this option was rejected.

2. The key storage environment variables in the default user profile (/etc/profile.d/csulcca.sh and /etc/profile.d/csulcca.csh) are changed at installation time to point to the /opt/IBM/CCA/keys/ path. Softlinks are added to older default installation paths to assist migration, but double-check that all of your key storage is accessible.

3. See “Interaction between the default card and use of protected key CPACF” on page 19 for a concurrency and CPACF.

CEX6C information

Since the TKE catcher daemon listens on one particular numbered port, it is impossible to allow a legacy catcher daemon to co-exist with a newer catcher daemon. This must be the new TKE catcher to support the newest adapters.

There are the following impacts:

- The newest TKE catcher must support all in-service CPRB types:
- T5 (CCA 4.*)
- T6 (CCA 5.* and CCA 6.4*)
  - libculca.so supports access to both types of adapters.

**CEX3C - CEX5C information**

The TKE catcher daemon listens on a single port for management communication. This port number has not changed for the CEX3C or later release. Therefore, the new daemon supports TKE management communication to the CEX3C - CEX5C or later adapters. Special steps are taken in the install/uninstall and daemon management for the CEX3C or later release to ensure that the new daemon is running when it is available.

**TKE versions for CEX*Cs**

- A TKE V6.0 or higher workstation is required to see supported CEX3Cs or CEX4Cs: If you are running in toleration mode, and Linux reports CEX4Cs as CEX3Cs, then TKE 6.0 is able to manage them as CEX3Cs.
- TKE V7.2: CEX4Cs are reported by TKE 7.2 as CEX3Cs. CEX4Cs are only supported by TKE 7.2 or later if the Linux driver reports them as CEX4s.
- A TKE V8.0 or higher workstation is required to see CEX5Cs. CEX4Cs, and CEX3Cs.
- A TKE V9.0 or higher workstation is required to see CEX6Cs and to exploit PCI-HSM 2016 compliance mode.

For more information about the TKE workstation, see [z/OS Cryptographic Services](http://www.ibm.com) [ICSF Trusted Key Entry Workstation User’s Guide](http://www.ibm.com)
Chapter 31. Utilities

You can use a utility to verify an installation. An additional utility is available to administer cryptographic coprocessors in setups without a TKE workstation.

These two utilities are called **ivp.e** and **panel.exe**.

**ivp.e**  This is an easy-to-use utility used to verify an installation.

It calls the Cryptographic Facility Query verb for all available adapters to report the firmware and configuration details.

This utility is installed by default to the following path in the Linux system:

```
/opt/IBM/CCA/bin/ivp.e
```

You can invoke this utility without any arguments. It presents information for all available CEX*C features on the system.

**panel.exe**

This utility provides a Linux native mechanism for administering and initializing certain characteristics of active cryptographic coprocessors. It is intended as a basic administration tool for Linux-only IBM Z configurations, where a Trusted Key Entry (TKE) solution is not available.

Note that **panel.exe** only covers some basic functions, but not the complete scope of a TKE workstation.

For mixed z/OS and Linux configurations, it is recommended that administration be accomplished using the z/OS TSO panels as described in the [z/OS Cryptographic Services ICSF Administrator’s Guide](#). The utility is installed by the Linux on Z cryptographic coprocessor (RPM or DEB install package) to this path in the Linux system:

```
/opt/IBM/CCA/bin/panel.exe
```

**Important:** As of CCA 6.0, the **panel.exe** interface has changed to offer a new default interface. The former interface is still available if the environment variable CSU_LEGACY_PANEL is set as follows:

```
export CSU_LEGACY_PANEL=TRUE
```

Go back to the new **panel.exe** interface:

```
unset CSU_LEGACY_PANEL
```

Be aware that new CEX6C functionality of **panel.exe** is only available from the default interface.

There is more detailed information in “The panel.exe utility” on page 1121.

CCA Master Key administration: choosing the right method or tool

Read the provided information why it is important to choose the correct tool to administer your cryptographic coprocessors.

There are several factors that influence the procedure of CCA master key administration:

- security requirements
Available methods are listed in Table 316.

### Table 316. Tools and methods for CCA master key administration

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Security</th>
<th>Environment</th>
</tr>
</thead>
</table>
| TKE               | • A TKE can be used to change the master keys for all the CEX*C adapters in a group, across multiple domains and IBM Z.  
• TKE administration is a fully tested and supported solution for CCA for Linux on Z installations. | • The TKE workstation is typically deployed in a secure location.  
• A TKE can leverage smart card credentials for a fully standards compliant remote administration solution. | • No extra pieces are needed to support TKE administration, the required daemon is installed by default from the CCA rpm as an automated service.  
• TKE administration does not conflict with any key storage approach you might use, but application use of key storage should be taken into account as you update keys. |
| z/OS exchange  
method | An operator temporarily assigns the domains to a z/OS partition, where ICSF user IDs and configuration panels are used to configure the master keys. | Since the z/OS tool is a host utility the users' key parts are potentially exposed:  
• in host memory on the system where they are entered  
• on the network channel for communication to the z host  
• in host memory for the z/OS partition | There is no conflict with any key storage approach you might use, but application use of key storage should be taken into account as you update keys and re-assign domains. |
| user application | A user application built to use the `libsulcamk.so` library for this purpose, which can be programmed to:  
• Allocate a CEX*C by invoking the CSUACRA verb (see "Cryptographic Resource Allocate (CSUACRA)" on page 144).  
• Change the master key.  
• Deallocate each adapter in the group before exiting, by invoking the CSUACRD verb (see "Cryptographic Resource Deallocate (CSUACRD)" on page 147). | Security features would depend on the implementation, but may have host memory or communication channel exposures. | Environment considerations would depend on the implementation. |
### Table 316. Tools and methods for CCA master key administration (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Security</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>panel.exe (included in RPM or DEB)</td>
<td>A general purpose simple utility that can be used to set the master keys. Keys are set one part at a time to one card at a time, which has some implications. <strong>Note:</strong> If a domain on a cryptographic coprocessor is set to PCI-HSM 2016 compliance mode, you can perform master key changes only on a TKE. You cannot use panel.exe for this purpose. Since panel.exe is a host utility that runs natively on the Linux instance, a local terminal and communication session are required. The users’ key parts are potentially exposed: • in host memory on the system where they are entered • on the network channel for communication to the z host • in host memory for the Linux partition</td>
<td>While panel.exe is installed by default from the CCA RPM, because of the simple nature of panel.exe, conflicts can occur in a multi-card environment. Especially, key storage conflicts occurred in prior releases when loading a master key to multiple adapters. This has been fixed as of CCA 6.0 (see “Changing the master key for two or more adapters that have the same master key, with shared CCA key storage” on page 454).</td>
<td></td>
</tr>
</tbody>
</table>

### The panel.exe utility

The **panel.exe** utility is installed by the Linux on Z cryptographic coprocessor install package or RPM.

It is installed to this path in the Linux system:

```
/opt/IBM/CCA/bin/panel.exe
```

The **panel.exe** utility numbers cards from card0 to card63, while verbs such as Cryptographic Resource Allocate number cards from CRP01 to CRP64, and therefore card0 corresponds to CRP01, card1 corresponds to CRP02, and so forth.

### panel.exe default syntax as of CCA 6.0

Precise usage information can be obtained by running the panel.exe utility with no arguments on the Linux shell command line.

```bash
>>> You must specify an operation <<<
New panel command-line arguments:

GENERAL

[Services]
-?, --help          Usage
--usage          Usage
-v, --version      Print CCA host library version
-x, --status       List ALL crypto resources and basic status
-k, --query-tke    Query whether TKE can use the card
--list-cpacf       List CPACF (local CPU crypto) resources

[Options]
-a=CARDC#, --adapter=CARD# Specity target device (default is '0') for single device services
-o, --disable-stdout Limit output to STDOUT to critical errors
--acps=0xAAAA,0xBBBB,0xCCCC... List of hex access-control points

COMPLIANCE

[Options]
--vsig         Optionally validate signature on reply
This will verify the compliance info signature using CCA’s epoch key
--nonce="NON"  32-byte hexadecimal nonce entered as 64 characters ASCII
--output-file="outputfile" Specify output file for health information

[Services]
```
Common Cryptographic Architecture Application Programmer's Guide

CERTIFICATE
[Options]
   --cert <value> <value>="NAME" Quoted name of certificate to process
[Services]
   --epoc-cert-quick Briefly display CCA epoch certificate
   --epoc-cert-validate Verify CCA epoch certificate
   --epoc-cert-dump Hex dump CCA epoch certificate
   --epoc-cert-show Display fully-parsed CCA epoch certificate
   --oa-cert-quick <cert> Briefly display the specified OA/MB certificate
   --oa-cert-dump <cert> Hex dump of the specified OA/MB certificate
   --oa-cert-show <cert> Display fully-parsed OA/MB certificate

ROLE
[Options]
   --role="ROLEID", "ROLEID", "ROLEID"
   (whether one <IR> or multiple roles <MR> are accepted depends
   on the operation being performed)
   --all
   Perform action for all roles, valid where noted below
[Services]
   -mrl, --list-roles
     --show-role
     --query-acps
     --uts, --show-tracking-state Show ACP tracking state for role
     --utr, --dump-tracking-state Dump ACP tracking state for role

The following also require either [--role=<MR>] or [--all]:
   --enable-tracking Enable ACP-tracking for role
   --disable-tracking Disable ACP-tracking for role
   --clear-tracking Clear ACP-tracking for role
   --tracking-size ACP tracking data size for role
   --show-tracking Show ACP tracking data for role
   --dump-tracking Raw dump ACP tracking data for role

SYSLOG/CCAALOG
[Services]
   -y, --card-log-info Get sizes of all card logs and log level
   --card-log-level=[4|8|12] Set card logging level
   -yc, --show-cca-log
     --show-syslog=[0|1|2|3|4] Dump card's CCA log to stdout
     Dump card's syslog to stdout
     (0 is most recent boot cycle)

MASTER KEY
[Options]
   --mktype=[ASYM|SYM|AES|APKA]
   --mkregister=[NEW|CURRENT|OLD]
   --mkpart=[FIRST|MIDDLE|LAST]
[Services]
   --mk-load-interactive Interactively load a master key part
   --mk-set-interactive Interactively set a master key
   --mk-clear-interactive Interactively clear a master key
   --mk-query-interactive Interactively query a master key verification pattern
   --mk-load="KEYPART" Load a master key part (requires --mktype, --mkpart)
   where "KEYPART" is a hexadecimal key value entered as text
   --mk-set
     Set a master key (requires --mktype)
   --mk-clear
     Clear a master key (requires --mktype)
   --mk-query
     Query master key verification pattern
     (requires --mktype, --mkregister)

KEY STORAGE
[Options]
   --ks-type=[AES|DES|PKA]
[Services]
   --ks-init Initialize key storage file (requires --ks-type)
---BASIC ADMIN---
-?  -h: Usage
-v: Print CCA Host Library version
-x: List crypto resources (and basic status)
-m: List CPACF (local CPU crypto) resources
-ml: List roles from card
-mrp "ID": Print role for "ID"
-mrq "ID" <0xAAAA>: Query ACP bit
--0xAAAA is hexadecimal ACP bit number
--quote IDs with spaces: "DEFAULT"
-ue "ID list": Enable ACP tracking for role "ID list"
-ud "ID list": Disable ACP tracking for role "ID list"
-uc "ID list": CLEAR ACP tracking for role "ID list"
-us "ID list": ACP tracking data size: role "ID list"
-usg "ID list": ACP tracking data show: role "ID list"
-uqg "ID list": ACP tracking data (raw): role "ID list"
-uts "ID list": ACP tracking state show: role "ID list"
-utr "ID list": ACP tracking state (raw): role "ID list"
if no "ID list": all roles will be processed

-y: Get SYSLOG or CCALOG info
-y [no options] :: sizes of all card logs and card logging level
-yl [4|8|12]:: sets 4, 8 or 12 as card logging level
4,8,12 are CCA return code trigger levels
-yc:: dump card CCA log to stdout
-ys [0|1|2|3|4]:: dump card SYS log to stdout
0 = current/default, 4 = oldest
-a <num> -y [options]:: perform on non-default card

---MASTER KEY (MK)---

Note: For security reasons, only a root user with correct group membership (real user ID equal to '0') is allowed to use panel.exe to load master key parts or to clear previously loaded master key parts. This is enforced at the shared library level in the implementation of the Master Key Process verb, not in the utility itself. Additionally, only the user who created a set of key storage files or the root user is able to take actions with respect to those key storage files, based on Linux file system permissions.

panel.exe legacy syntax

The panel.exe legacy syntax is available if you set the environment variable CSU_LEGACY_PANEL to TRUE. Precise usage information can be obtained by running the panel.exe utility with no arguments on the Linux shell command line.

The legacy panel.exe command-line arguments:
Panel usage ([-k,-a <num>,-o,-g]
[CC] Arg >> arg must precede non-[CC] args
[CC] -k: Can TKE administer a card?
[CC] -a <num>: use non-default card
<num> is the card number [0 - 63]
[CC] -o: Disable output to stdout:
[CC] -g <level>: Set the log level:
<level> can be NONE, TRANSACTIONS, NONZERO, ALL, DEBUG, and FUNCTIONS

>> non-[CC] Args >>: (all are mutually exclusive)
To LOAD a Master Key (MK) PART:
-1 (for interactive)
OR====>
-1 -t [A|S|E|P] -p [F|M|L] KEYPART
where: -t [A|S|E|P] is which MK: A=ASYM, S=SYM, E=AES P=APKA
where: -p [F|M|L] is the part: F=FIRST, M=MIDDLE, L=LAST
where: KEYPART is string in hex 2* size of key
(recall: 2 text chars = 1 binary Byte)

To SET a Master Key:
-s (for interactive)
OR====>
-s -t [A|S|E|P]
where: -t [A|S|E|P] is which MK: A=ASYM, S=SYM, E=AES P=APKA

To CLEAR a Master Key 'New' Register:
-c (for interactive)
OR====>
c -t [A|S|E|P]
where: -t [A|S|E|P] is which MK: A=ASYM, S=SYM, E=AES P=APKA

To QUERY a Master Key Verification Pattern:
-q (for interactive)
OR====>
-q -t [A|S|E|P] -r [N|C|O]
where: -t [A|S|E|P] is which MK: A=ASYM, S=SYM, E=AES P=APKA
where: -r [N|C|O] is which register: N=NEW, C=CURRENT, O=OLD

---KEY STORAGE---

To INIT a KEY STORAGE file:
-t <type> -i
To REENCipher KEY STORAGE:
-t <type> -r
To LIST a KEY STORAGE:
-t <type> -p
where:
<type> can be AES, DES, PKA
**environment variables for key storage files**
--the key DISK STORAGE (DS) variables are:
CSUDESDS, CSUAESDS, CSUPKADS
**the path must exist and specify a filename. The file
will be created on key storage initialization.
--the key LIST DIRECTORY (LD) variables are:
CSUDESLD, CSUAESLD, CSUPKALD
**the full path must exist.

---RETAINED KEYS---

To LIST RETAINED KEYS (this domain ONLY):
-n

panel.exe functions

Read about the different uses for the panel.exe utility.

The panel.exe utility can be used to:

- show the compliance state of the current domain.

```bash
# panel.exe --qcomp
```

- show and verify the outbound authentication (OA) certificate chain that
  represents the chain of trust for the adapter firmware.
Example:

```
# panel.exe --epoc-cert-validate
```

- determine if a TKE is currently able to administer a specific active coprocessor.

Example:

```
# panel.exe --query-tke
```

- list the labels and key types for all the keys in a designated key storage file.

Example:

```
# panel.exe --ks-list --ks-type=DES
# panel.exe --ks-list --ks-type=AES
# panel.exe --ks-list --ks-type=PKA
```

- list the labels for all of the retained keys (RSA private keys stored in the adapter) in the current domain of the CEX*C.

Example:

```
# panel.exe --ks-list-retained
```

- list the coprocessors currently active in the Linux system and their master key status.

Example:

```
# panel.exe -x or # panel.exe --status
```

- load master key parts to the coprocessor.

Example:

```
# panel.exe --mk-load-interactive
```

- set a master key that was loaded to the coprocessor. Note that `panel.exe` is not designed to change the master keys for all the cards in a group at the same time, because this is a more sophisticated operation.

Example:

```
# panel.exe --mk-set-interactive
```

- clear master key parts which were previously loaded to the coprocessor but not yet set or confirmed (used for when a mistake in entering master key parts has been detected).

Example:

```
# panel.exe --mk-clear-interactive
```

- list serial numbers and master key register states of all active cards running CCA that are visible to this Linux host. The total number of active cards and any errors will also be reported.

Example:

```
# panel.exe -x or # panel.exe --status
```

- query the master key verification pattern for any master-key register in the current domain.
Example:
```
# panel.exe --mk-query --mktype=AES --mkregister=CURRENT
```

- initialize a local host key storage file. For an example, see "Using panel.exe for key storage initialization."
- re-encipher a local host key storage file (use this when the master key has been changed to ensure currency with key storage). For an example, see "Using panel.exe for key storage re-encipher when changing the master key" on page 1128.
- list available CPACF functions, and whether they are supported in the current system image.

Example:
```
# panel.exe --list-cpacf
```

- check ACP settings and tracking of ACP usage. For an example, see "Using panel.exe to show the active role and ACPs" on page 1129.
- get SYSLOG or CCALOG information.

Example:
```
# panel.exe -y or # panel.exe --card-log-info
# panel.exe -yc or # panel.exe --show-cca-log
```

The `panel.exe` utility does not support access control point manipulation or more sophisticated administration. Refer to “Trusted Key Entry support” on page 57 for that functionality.

**Note:** You cannot perform all of the above functions using the legacy `panel.exe` syntax. Especially the new CEX6C-related functions are only available with the default syntax.

## Using panel.exe for key storage initialization

Each application using CCA typically creates key objects that are stored in the host, protected by the master key stored inside the card. Perform the steps described in this topic for key storage initialization.

1. The default locations for the files are setup by the CCA package in environment variables added in the new profile files `/etc/profile.d/csulcca.sh` and `/etc/profile.d/csulcca.csh` during installation. Key storage is unsupported without a master key loaded, so **master key load** (Step 7 on page 1113) must be completed before this step. The utility `panel.exe` can be used to initialize both the default key storage and any separate key storage you might want to set up. The full topic is too lengthy for this explanation (see the key storage topics elsewhere in this manual, including the verb "Key Storage Initialization (CSNBKSI)" on page 149). In brief, an application can specify a particular key storage location. That non-default key storage can be initialized now (or later) by using `panel.exe` or with a program using the Key Storage Initialization verb.

2. The key storage environment variables in the default user profile `/etc/profile.d/csulcca.sh` are changed at installation time to point to the `/opt/IBM/CCA/keys/` path. There is one set of environment variables for a profile. Users can override this by setting a local profile in their home `.profile` file that sets the environment variables differently.

3. Key storage ownership
The default key storage files are actually partially created (but not fully initialized) during the master key load process. This means the ownership and permissions of those files might have to be changed for them to be fully initialized by the user associated with the application that uses the key storage files.

Because of the mutually exclusive nature of the master key administrator groups, there can be some harmless access errors reported to the system log during master key load. The example users created previously in Master key load Step 7.3[a] avoids this. You do not need to fix key storage ownership because they were all created with the primary group set to cca_admin (the -g argument to useradd). By doing this, the first master key load creates the key storage files with group set to cca_admin and subsequent users all have membership in that group. You still might want to fix the owner of default key storage at the end to be the root user, but the group membership solves the access issue.

Typically, the root user needs to fix the ownership and permissions. We recommend that the owner of key storage should be the root user, and that the group be cca_admin (cca_admin group is created during the CCA package install process). We recommend that the permissions be set to 660, which is rw for owner (root), rw for group (cca_admin), and <none> for everyone else, for security. Then add the application user to the group cca_admin with the appropriate procedure detailed in Master key load Step 7a on page 1113.

To be able to use /opt/IBM/CCA/panel.exe, the user must be either root OR a member of the cca_admin group (the owner:group of /usr/11b64/11bsulccamk.so). The reasons for the separate cca_admin group are to allow one owner of /usr/11b64/11bsulccamk.so, and to allow use of panel.exe without allowing any of the master key processing calls.

4. Key storage initialization with panel.exe. This is the default.
   a. Ensure that the permissions to the default location (/opt/IBM/CCA/keys/) allow your user to perform key storage initialization.
   b. Initialize all types of key storage (DES is where DES key tokens are kept, AES is where AES key tokens are kept, and PKA is for all the RSA public/private internal key tokens).

   **panel.exe default syntax**
   
   ```
   /opt/IBM/CCA/bin/panel.exe --ks-type=AES --ks-init
   /opt/IBM/CCA/bin/panel.exe --ks-type=DES --ks-init
   /opt/IBM/CCA/bin/panel.exe --ks-type=PKA --ks-init
   ```

   **panel.exe legacy syntax**
   
   ```
   /opt/IBM/CCA/bin/panel.exe -t AES -i
   /opt/IBM/CCA/bin/panel.exe -t DES -i
   /opt/IBM/CCA/bin/panel.exe -t PKA -i
   ```

5. Key storage initialization with panel.exe (non-default)
   a. Ensure that you are using the account that uses the key storage. If you are not, you must fix its ownership and permissions later.
   b. Initialize all types of key storage (DES is where DES key tokens will be kept, AES is where AES key tokens will be kept, PKA is for all the RSA public/private internal key tokens). Use a different name for AES, DES, and PKA, because the second initialization would overwrite the first if different names are not used. Export new environment variables in the session where you will initialize the key storage (and where you sill use it), then initialize key storage again:
**panel.exe default syntax**

```bash
export CSUAESDS=<AES file name>
export CSUDESDS=<DES file name>
export CSUPKADS=<PKA file name>
/opt/IBM/CCA/bin/panel.exe --ks-type=AES --ks-init
/opt/IBM/CCA/bin/panel.exe --ks-type=DES --ks-init
/opt/IBM/CCA/bin/panel.exe --ks-type=PKA --ks-init
```

**panel.exe legacy syntax**

```bash
export CSUAESDS=<AES file name>
export CSUDESDS=<DES file name>
export CSUPKADS=<PKA file name>
/opt/IBM/CCA/bin/panel.exe -t AES -i
/opt/IBM/CCA/bin/panel.exe -t DES -i
/opt/IBM/CCA/bin/panel.exe -t PKA -i
```

For example, if you entered the following commands in default syntax:

```bash
export CSUAESDS=/tmp/a
export CSUDESDS=/tmp/d
export CSUPKADS=/tmp/p
/opt/IBM/CCA/bin/panel.exe --ks-type=AES --ks-init
/opt/IBM/CCA/bin/panel.exe --ks-type=DES --ks-init
/opt/IBM/CCA/bin/panel.exe --ks-type=PKA --ks-init
```

these files would be created:

```
/tmp/a
/tmp/a.NDX
/tmp/d
/tmp/d.NDX
/tmp/p
/tmp/p.NDX
```

### Using panel.exe for key storage re-encipher when changing the master key

Because all the key tokens are protected by the master key for the domain, a preexisting key storage must be re-enciphered when the master key is changed.

If the example group scheme is used, this is simple because the key storage files are owned by the group cca_admin and the user making the re-encipher call is also in group cca_admin. If this is not the case then, after changing the master key, the owner of key storage must log in and perform the re-enciphering. This can be done with the help of a program (using several verbs) or with /opt/IBM/CCA/panel.exe. Of course, as noted, the user of panel.exe must also be a member of cca_admin because of ownership of /usr/lib64/libcsulccamk.so.

Perform these steps for key storage re-encipher when changing the master key.

1. To re-encipher default key storage with panel.exe use:

   **panel.exe default syntax**

   ```bash
   /opt/IBM/CCA/bin/panel.exe --ks-type=AES --ks-reenc
   /opt/IBM/CCA/bin/panel.exe --ks-type=DES --ks-reenc
   /opt/IBM/CCA/bin/panel.exe --ks-type=PKA --ks-reenc
   ```

   **panel.exe legacy syntax**
2. To reencrypt non-default key storage with **panel.exe**:  
   - Export new versions of the environment variables specifying your key 
     storage file locations.  
   - Run the previously shown commands as you would for the default key 
     storage, but ensure to do so in the session with the new environment 
     variables.

### Using panel.exe to show the active role and ACPs

You can use the **panel.exe** utility to list the name of the role for the current 

To list the role for the domain:  

**panel.exe default syntax**  

```bash  
# panel.exe --list-roles  
```

**panel.exe legacy syntax**  

```bash  
# panel.exe -mrl  
```

The output is the same in both interface versions:

```
Showing returned list of ROLES:
[DFLT0019]  
---------------------------  
API CALL details:  
CSUAAACM [LSTROLES] card [DV73R354] number of items [1]  
---------------------------  
```

Data explained:
The only visible role to this domain has the 8 byte ASCII name DFLT0019, where 19 stands for domain 19. For more information on roles, see z/OS Cryptographic Services ICSF Trusted Key Entry Workstation User's Guide.

Note: With panel.exe, you can only see the DEFLTXX/DFLTXXXX roles for each domain. You cannot see further administrative roles. Also, on IBM Z, you cannot create or use multiple roles for use by different applications on one domain. If you require different roles for applications, these applications must use different domains, because all applications in a domain share the same permissions granted by the DEFLTXX/DFLTXXXX role.

To show the ACPs for that role:

panel.exe default syntax

# panel.exe --show-role --role "DFLT0019"
# panel.exe --show-role --role=DFLT0019

panel.exe legacy syntax

# panel.exe -mrp "DEFLT02"

The output is the same in both interface versions:

Showing returned ROLE DATA:

| API CALL details: |
|---|---|
| version: [0001] |
| comment: [System default role ] |
| authstr: [0000] |
| time range: [00:00] - [00:00] |
| DOW: [fc] |
| ACP Segments for role: 5 |
| ACP Segment [0] has [31] Bytes for bits [0×0008 - 0×00ff] |
| [ 03 f0 ] << ACP bits [0×0008 - 0×0017] |
| [ ff 70 ] << ACP bits [0×0018 - 0×0027] |

<...data continues...>

Output explanation:

The shown output is a parsed version of the role and access control point data structures described in Chapter 25, “Access control data structures,” on page 1073, to make it human-accessible. The size of the entire returned data structure is 208 bytes, for the role with ID DFLT0019.

The authstr, time range, and DOW fields are meaningless for a default role like this.

There are five ACP segments to be shown. The start of the data for the first or 0th segment is shown, it has 31 bytes of data that specify ACPs from offset 0x0008 to 0x00ff. Then each of those 31 bytes is shown in hexadecimal at left, with information at right to indicate the ACP bits that correspond to that hexadecimal byte. This information is useful to understand the role structure or to check your own program that might try to pull and process the same information.

Example:

[ 03 f0 ] << ACP bits [0×0008 - 0×0017]
Figure 48 shows how to interpret the output from the command `panel.exe --show-role --role=DFLT0019`. For example, you can see that ACP X'000E' is enabled for role DFLT0019.

To query a particular ACP value for that role

(ENABLED means 1, DISABLED means 0)

**panel.exe default syntax**

```
# panel.exe --query-acps --role=DFLT0019 --acps=0x0204
# panel.exe --query-acps --role "DFLT0019" --acps 0x0204
```

**panel.exe legacy syntax**

```
# panel.exe -mrq "DEFAUT02" 0x0204
```

The output is the same in both interface versions:

```
Role ID [DFLT0019]: ACP [0x0204] is [ENABLED]
```

Using `panel.exe` to control ACP tracking

The `panel.exe` utility allows you to query and control the tracking of ACP usage.

Note that usage tracking only indicates if a verb made a query to an ACP, it does not indicate if that query succeeded or failed. The notion of success or failure varies greatly depending on the perspective, and for some ACPs, a value of 1 (setting is ON, that is, enabled) implies a restriction, while for other ACPs, a value of 0 (setting is OFF, that is, disabled) also implies a restriction. Generally, users should compare the tracking information to their role configuration to determine what setting the verbs encountered when the ACPs were accessed during application run-time (if tracking was enabled).

The `panel.exe` utility allows a basic level of control and query functions for ACP tracking. You can also perform ACP tracking through the CSUAACT verb, if the ACP at offset 0x01CC is enabled through a TKE (see "Access Control Tracking (CSUAACT)" on page 93). This 0x01CC ACP is usually disabled by default. A typical user will access ACP tracking information through the Trusted Key Entry workstation (TKE). The `panel.exe` functionality is added as a convenience. If requested, the tracking data is returned in a data structure that matches the role data structure and ACP structure described in Chapter 25, "Access control data structures," on page 1073.
Using panel.exe for verifying CCA epoch certificates

Using the `--epoc-cert-validate` parameter, the panel.exe utility allows you to verify a CCA epoch certificate.

This command pulls the certificates from the adapter down to the root certificate created at the IBM factory, that validate the firmware levels, and then validates each certificate using the public key from the certificate below that one in the trust chain. The results are output to the command line. Here is an example of a successful run:

```
panel.exe --epoc-cert-validate

Epoch Cert Subject Name:
[07C8F43AD7854F599EB21EE9DC3651CAAC44DF745886CF3D83A6487F4616E455] 32
Signer Name:
[55EB692CE336BF3D0D1CCCA91158650851EDCF0CBEB21F69734E44438DF991A] 32
Named Cert Subject Name:
[55EB692CE336BF3D0D1CCCA91158650851EDCF0CBEB21F69734E44438DF991A] 32
Signer Name:
[94A2D4FEC9A07AD42DF550BC64E430E8273C1416A13CBDA0973D1B1ACB3F1C26] 32
Named Cert Subject Name:
[94A2D4FEC9A07AD42DF550BC64E430E8273C1416A13CBDA0973D1B1ACB3F1C26] 32
Signer Name:
[A4CE1BB3A923C0D07A4D20F50BC64E430E8273C1416A13CBDA0973D1B1ACB3F1C26] 32
Named Cert Subject Name:
[A4CE1BB3A923C0D07A4D20F50BC64E430E8273C1416A13CBDA0973D1B1ACB3F1C26] 32
Signer Name:
[94A2D4FEC9A07AD42DF550BC64E430E8273C1416A13CBDA0973D1B1ACB3F1C26] 32
Root cert has unused Cert section in do_DPK_work. (Expected).
Named Cert Subject Name:
[57006C46B69D16D63F3B0F2E85C8115AC75573B5C6823605FA963E0D1EFAAB1] 32
Signer Name:
[57006C46B69D16D63F3B0F2E85C8115AC75573B5C6823605FA963E0D1EFAAB1] 32
Root cert has unused Cert section in getCert_pubTok. (Expected).
./TKE_utils/panel.cpp:8518 pCertMeta->root_ca: [0x02]
./TKE_utils/panel.cpp:8521 pCertMeta->issuing_segment: [0x00]
./TKE_utils/panel.cpp:8522 pCertMeta->subject_segment: [0x00]
Calling verifySig with Prod key.
Trusting segments: [0x00] issuing_segment: [0x00] subject_segment: [0x00] root_ca: [2].
-----> SUCCESS!! Chain of 5 certs VERIFIED rc [00000000] -----
```

Using panel.exe to query the adapter compliance state

Using the `--qcomp --vsig` parameter, the panel.exe utility allows you to query the compliance state of a specified cryptographic coprocessor.

This command returns the status of a specified domain and parses all of the data fields to display the information on the console. Figure 49 on page 1133 and Figure 50 on page 1134 show a successful run - with added explanations - for a domain that is not in compliance mode, with the additional `--vsig` option which requests that the output be signed by the card, and that the signature be verified by panel.exe.
panel.exe --qcomp --vsig

---------------------------------------------------------------
-------------
-------------
-------------

for domain [42]...

Compliance Data for this card --[0x96 bytes]--
CARD PN [01KU719]  ----  CARD PART NUMBER
CARD EC [0N37513]  ----  CARD EC level (hardware sub-version)
CARD SN [YH10D0731308]  ----  CARD Serial Number
rsvd1/2 [0x00 0x00]  ----  EMPTY fields
CCA vs [6.0.8r]  ----  Segment 3 firmware level (CCA version)
UDX vs1 [NONE ]  ----  UDX version first field
UDX vs2 [NONE ]  ----  UDX version second field
CCA CARD TIME NOW [20180131185938]  ----  Current timestamp according to card internal clock
CCA BUILD TIME [20171108150441]  ----  Timestamp for build of Segment 3 firmware
CCA COMPLIANCE ]  ----  Detailed flags fields, see below

cardact[0x00000000] cmp iss[0x00000000]  
CARD ACTIONS on:
   <NONE>
slogmax[0x00000200] slgevsz[0x0002] kdf/rsvd [0x0000]
dom act[0x00000000]
   DOM ACTIONS:
   <NONE>
dom cmp[0x00000000]
   DOM COMPLIANCE FLAGS:
   <NONE>
slogcnt [0x00000000]

CARD Seg2 ownerID [0x0002]  ----  Owner ID code for Segment 2 firmware
CARD Seg3 ownerID [0x0002]  ----  Owner ID code for Segment 3 firmware
CARD AdapterID [0x00000008]  ----  Adapter ID code for internal work
CARD MiniBoot-0 Ver [0x00]  ----  Miniboot 0 version (unused)
CARD MiniBoot-0 Rel [0x00]  ----  Miniboot 0 release (unused)
CARD MiniBoot-1 Ver [0x40]  ----  Miniboot 1 version (unused)
CARD MiniBoot-1 Rel [0x00]  ----  Miniboot 1 release (unused)

--- Following output is added because of --vsig option ---

Getting Epoch Cert to verify Sig
Epoch Cert Subject Name:
[07C8F43A0754F5919EB21EE9DC3651CAAC4A4DF745886CF3DB3A6487F4616E455] 32
Signer Name:
[55EB0EB2C03368F3D01C6CA911586B50B51EDCF0B8C8B2F69734E44380F991A] 32
---------- Compliance Data for this card Signature Verified.---

Figure 49. Querying an adapter's compliance mode (part 1)
COMPLIANCE Fields detail:
Note: all un-used bits are reserved

cardact[0x00000000]  <--- (bitfield) Card state or actions in force now
   0x80000000, "CARDZEROOSTRT" -- the card is in the middle of a whole card zeroize
   0x40000000, "CARDCLOCKSET" -- the card has had the internal clock set

cmp iss[0x00000000]  <--- (bitfield) Bits representing reasons this domain cannot go into compliance mode, if any
   0x80000000, -- Card Firmware is a UDX, cannot enter compliance mode
   0x40000000, -- CCA responding is from a simulator or debug image, cannot enter compliance mode
   0x20000000, -- Outbound Authentication certificate chain is not valid, cannot enter compliance mode (contact IBM service if this occurs)

slogmax[0x00000200]  <--- Maximum count of secure audit log entries
slogsz[0x00000002]  <--- Size of each secure audit log entry
kdf/rsvd [0x00000000] <--- Current KDF for compliance level

dom act[0x00000000]  <--- (bitfield) Domain compliance / restart actions in force now
   0x80000000, "DOM ZEROOSTRT" -- the domain is in the middle of a domain-scope zeroize
   0x40000000, "DOM IMPRSTRT" -- the domain internal state is transitioning to imprint mode
   0x20000000, "DOM IMPR_ACT" -- the domain is now in imprint mode, a configuration state before compliance mode
   0x10000000, "DOM COMP_ACT" -- the domain is now in compliance mode
   0x08000000, "DOM COMP_MIG" -- the domain is now in migration mode, a sub-state of compliance mode
   0x00008000, "DOM SLOGENAB" -- the domain secure audit log is enabled (required for imprint and compliance modes)
   0x00004000, "DOM SLOGNOWR" -- the domain secure audit log is enabled for "no-wrap" mode: secure read/clear is required

dom cmp[0x00000000] <--- (bitfield) Domain compliance configuration (if in compliance mode)
   0x80000000, "CMP PCIH2016"

slogcnt [0x00000000] <=- Current count of secure audit log entries

Figure 50. Querying an adapter's compliance mode (part 2)
Chapter 32. Security API command and sub-command codes

View an alphabetical list of security API command and sub-command codes returned by the output rule-array for option STATDIAG of the Cryptographic Facility Query verb.

Elements 9, 11, 13, 15, and 17 contain these API command and sub-command codes to indicate the last five commands run. See Table 22 on page 106.

**Note:** The security API command is T2 padded on the right with 2 space characters, and the subcommand code is a 2-byte uppercase alphabetic value padded on the right with 2 space characters.

<table>
<thead>
<tr>
<th>Security API command code</th>
<th>Subcommand code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>AD</td>
<td>Symmetric Algorithm Decipher (CSNBSAD)</td>
</tr>
<tr>
<td>T2</td>
<td>AE</td>
<td>Symmetric Algorithm Encipher (CSNBSAE)</td>
</tr>
<tr>
<td>T2</td>
<td>AI</td>
<td>Access Control Initialization (CSUAAACI)</td>
</tr>
<tr>
<td>T2</td>
<td>AK</td>
<td>Diversified Key Generate2 (CSNBDKG2)</td>
</tr>
<tr>
<td>T2</td>
<td>AM</td>
<td>Access Control Maintenance (CSUAAACM)</td>
</tr>
<tr>
<td>T2</td>
<td>AP</td>
<td>Authentication Parameter Generate (CSNBAAPG)</td>
</tr>
<tr>
<td>T2</td>
<td>CA</td>
<td>Cryptographic Resource Allocate (CSUACRA)</td>
</tr>
<tr>
<td>T2</td>
<td>CD</td>
<td>Cryptographic Resource Deallocate (CSUACRD)</td>
</tr>
<tr>
<td>T2</td>
<td>CI</td>
<td>Data Key Import (CSNBCKI)</td>
</tr>
<tr>
<td>T2</td>
<td>CM</td>
<td>Multiple Clear Key Import (CSNBCKM)</td>
</tr>
<tr>
<td>T2</td>
<td>CT</td>
<td>Key Token Change (CSNBKTC)</td>
</tr>
<tr>
<td>T2</td>
<td>CV</td>
<td>MAC Generate2 (CSNBMGN2)</td>
</tr>
<tr>
<td>T2</td>
<td>CX</td>
<td>Control Vector Translate (CSNBCVT)</td>
</tr>
<tr>
<td>T2</td>
<td>CY</td>
<td>Key Token Change2 (CSNBKTC2)</td>
</tr>
<tr>
<td>T2</td>
<td>DA</td>
<td>DK Deterministic PIN Generate (CSNBDDPG)</td>
</tr>
<tr>
<td>T2</td>
<td>DB</td>
<td>DK PAN Translate (CSNBDPT)</td>
</tr>
<tr>
<td>T2</td>
<td>DC</td>
<td>Decipher (CSNBDEC)</td>
</tr>
<tr>
<td>T2</td>
<td>DE</td>
<td>Data Key Export (CSNBDKX)</td>
</tr>
<tr>
<td>T2</td>
<td>DH</td>
<td>Elliptic Curve Diffie-Hellman (CSNDEDHJ)</td>
</tr>
<tr>
<td>T2</td>
<td>DI</td>
<td>Data Key Import (CSNBDMK)</td>
</tr>
<tr>
<td>T2</td>
<td>DK</td>
<td>Retained Key Delete (CSNDRKD)</td>
</tr>
<tr>
<td>T2</td>
<td>DM</td>
<td>DK PRW CMAC Generate (CSNBDPCG)</td>
</tr>
<tr>
<td>T2</td>
<td>DN</td>
<td>DK PIN Change (CSNBDPC)</td>
</tr>
<tr>
<td>T2</td>
<td>DQ</td>
<td>DK Migrate PIN (CSNBDMP)</td>
</tr>
<tr>
<td>T2</td>
<td>DR</td>
<td>DK Random PIN Generate (CSNBDRPG)</td>
</tr>
</tbody>
</table>
### Table 317. Alphabetical list of security API command and subcommand codes returned by STATDIAG (continued).

<table>
<thead>
<tr>
<th>Security API command code</th>
<th>Subcommand code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>DS</td>
<td>Symmetric Key Export with Data (CSNDSXD)</td>
</tr>
<tr>
<td>T2</td>
<td>DT</td>
<td>DK PAN Modify in Transaction (CSNBDPMT)</td>
</tr>
<tr>
<td>T2</td>
<td>DU</td>
<td>DK PRW Card Number Update (CSNBDPNU)</td>
</tr>
<tr>
<td>T2</td>
<td>DV</td>
<td>DK PIN Verify (CSNBDPV)</td>
</tr>
<tr>
<td>T2</td>
<td>DW</td>
<td>DK Regenerate PRW (CSNBDRP)</td>
</tr>
<tr>
<td>T2</td>
<td>EC</td>
<td>Encipher (CSNBENC)</td>
</tr>
<tr>
<td>T2</td>
<td>EP</td>
<td>Encrypted PIN Generate (CSNBEPG)</td>
</tr>
<tr>
<td>T2</td>
<td>EX</td>
<td>Prohibit Export Extended (CSNBPEXX)</td>
</tr>
<tr>
<td>T2</td>
<td>EY</td>
<td>Restrict Key Attribute (CSNBKRA)</td>
</tr>
<tr>
<td>T2</td>
<td>FQ</td>
<td>Cryptographic Facility Query (CSUACFQ)</td>
</tr>
<tr>
<td>T2</td>
<td>GC</td>
<td>MAC Verify2 (CSNBMVR2)</td>
</tr>
<tr>
<td>T2</td>
<td>GD</td>
<td>Diversified Key Generate (CSNBDKG)</td>
</tr>
<tr>
<td>T2</td>
<td>GH</td>
<td>HMAC Generate (CSNBHMG) - SHA-1, SHA-224, SHA-256</td>
</tr>
<tr>
<td>T2</td>
<td>GK</td>
<td>Key Generate2 (CSNBKGN2)</td>
</tr>
<tr>
<td>T2</td>
<td>GL</td>
<td>HMAC Generate (CSNBHMG) - SHA-384, SHA-512</td>
</tr>
<tr>
<td>T2</td>
<td>GM</td>
<td>MDC Generate (CSNBMDG)</td>
</tr>
<tr>
<td>T2</td>
<td>GS</td>
<td>Symmetric Key Generate (CSNDSYG)</td>
</tr>
<tr>
<td>T2</td>
<td>IP</td>
<td>Key Part Import2 (CSNBKPI2)</td>
</tr>
<tr>
<td>T2</td>
<td>KC</td>
<td>PKA Public Key Register (CSNDPKR)</td>
</tr>
<tr>
<td>T2</td>
<td>KE</td>
<td>Key Export (CSNBKEX)</td>
</tr>
<tr>
<td>T2</td>
<td>KG</td>
<td>Key Generate (CSNBKGN)</td>
</tr>
<tr>
<td>T2</td>
<td>KI</td>
<td>Key Import (CSNBKIM)</td>
</tr>
<tr>
<td>T2</td>
<td>KN</td>
<td>Key Translate2 (CSNBBTR2)</td>
</tr>
<tr>
<td>T2</td>
<td>KP</td>
<td>Key Part Import (CSNBKPI)</td>
</tr>
<tr>
<td>T2</td>
<td>KR</td>
<td>Key Translate (CSNBKTR)</td>
</tr>
<tr>
<td>T2</td>
<td>KT</td>
<td>Key Test (CSNBKYT)</td>
</tr>
<tr>
<td>T2</td>
<td>KX</td>
<td>Key Test Extended (CSNBKYTX)</td>
</tr>
<tr>
<td>T2</td>
<td>KY</td>
<td>Key Test2 (CSNBKTY2)</td>
</tr>
<tr>
<td>T2</td>
<td>LK</td>
<td>Retained Key List (CSNDRKL)</td>
</tr>
<tr>
<td>T2</td>
<td>LQ</td>
<td>Log Query (CSUALGQ)</td>
</tr>
<tr>
<td>T2</td>
<td>MG</td>
<td>MAC Generate (CSNBMGN)</td>
</tr>
<tr>
<td>T2</td>
<td>MP</td>
<td>Master Key Process (CSNBMKP)</td>
</tr>
<tr>
<td>T2</td>
<td>MV</td>
<td>MAC Verify (CSNBMVR)</td>
</tr>
<tr>
<td>T2</td>
<td>OH</td>
<td>One-Way Hash (CSNBOWH)</td>
</tr>
<tr>
<td>T2</td>
<td>PA</td>
<td>Clear PIN Generate Alternate (CSNCPA)</td>
</tr>
<tr>
<td>T2</td>
<td>PC</td>
<td>Clear PIN Generate (CSNBPGN)</td>
</tr>
<tr>
<td>T2</td>
<td>PD</td>
<td>PKA Decrypt (CSNDPKD)</td>
</tr>
</tbody>
</table>
Table 317. Alphabetical list of security API command and subcommand codes returned by STATDIAG (continued).

<table>
<thead>
<tr>
<th>Security API command code</th>
<th>Subcommand code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>PE</td>
<td>Clear PIN Encrypt (CSNBCPE)</td>
</tr>
<tr>
<td>T2</td>
<td>PG</td>
<td>PKA Key Generate (CSNDPKG)</td>
</tr>
<tr>
<td>T2</td>
<td>PI</td>
<td>PKA Key Import (CSNDPKI)</td>
</tr>
<tr>
<td>T2</td>
<td>PK</td>
<td>PKA Encrypt (CSNDPKE)</td>
</tr>
<tr>
<td>T2</td>
<td>PO</td>
<td>Recover PIN from Offset (CSNPFO)</td>
</tr>
<tr>
<td>T2</td>
<td>PT</td>
<td>Encrypted PIN Translate (CSNBPTR)</td>
</tr>
<tr>
<td>T2</td>
<td>PU</td>
<td>PIN Change/Unblock (CSNBPUC)</td>
</tr>
<tr>
<td>T2</td>
<td>PV</td>
<td>Encrypted PIN Verify (CSNPV)</td>
</tr>
<tr>
<td>T2</td>
<td>PX</td>
<td>Data Key Export (CSNPXE)</td>
</tr>
<tr>
<td>T2</td>
<td>RE</td>
<td>Key Export to TR31 (CSNBT31X)</td>
</tr>
<tr>
<td>T2</td>
<td>RG</td>
<td>Random Number Generate (CSNBRNG)</td>
</tr>
<tr>
<td>T2</td>
<td>RI</td>
<td>TR31 Key Import (CSNBT31I)</td>
</tr>
<tr>
<td>T2</td>
<td>RK</td>
<td>Remote Key Export (CSNRKX)</td>
</tr>
<tr>
<td>T2</td>
<td>RL</td>
<td>Random Number Generate Long (CSNBRNL)</td>
</tr>
<tr>
<td>T2</td>
<td>RT</td>
<td>Random Number Tests (CSURNT)</td>
</tr>
<tr>
<td>T2</td>
<td>SC</td>
<td>SET Block Compose (CSNSBC)</td>
</tr>
<tr>
<td>T2</td>
<td>SD</td>
<td>SET Block Decompose (CSNSBD)</td>
</tr>
<tr>
<td>T2</td>
<td>SG</td>
<td>Digital Signature Generate (CSNDSG)</td>
</tr>
<tr>
<td>T2</td>
<td>SI</td>
<td>Symmetric Key Import (CSNSYI)</td>
</tr>
<tr>
<td>T2</td>
<td>SJ</td>
<td>Symmetric Key Import2 (CSNSYI2)</td>
</tr>
<tr>
<td>T2</td>
<td>SP</td>
<td>Secure Messaging for PINs (CSNBS)</td>
</tr>
<tr>
<td>T2</td>
<td>SV</td>
<td>Digital Signature Verify (CSNDSV)</td>
</tr>
<tr>
<td>T2</td>
<td>SX</td>
<td>Symmetric Key Export (CSNSXY)</td>
</tr>
<tr>
<td>T2</td>
<td>SY</td>
<td>Secure Messaging for Keys (CSNSKY)</td>
</tr>
<tr>
<td>T2</td>
<td>TB</td>
<td>Trusted Block Create (CSNTBC)</td>
</tr>
<tr>
<td>T2</td>
<td>TC</td>
<td>PKA Key Token Change (CSNKT)</td>
</tr>
<tr>
<td>T2</td>
<td>TK</td>
<td>PKA Key Translate (CSNPKT)</td>
</tr>
<tr>
<td>T2</td>
<td>TT</td>
<td>Cipher Text Translate2 (CSNBCTT2)</td>
</tr>
<tr>
<td>T2</td>
<td>TV</td>
<td>Transaction Validation (CSNTRV)</td>
</tr>
<tr>
<td>T2</td>
<td>UD</td>
<td>Unique Key Derive (CSNUKD)</td>
</tr>
<tr>
<td>T2</td>
<td>VC</td>
<td>CVV Key Combine (CSNVC)</td>
</tr>
<tr>
<td>T2</td>
<td>VE</td>
<td>Cryptographic Variable Encipher (CSNBCVE)</td>
</tr>
<tr>
<td>T2</td>
<td>VG</td>
<td>CVV Generate (CSNBCG)</td>
</tr>
<tr>
<td>T2</td>
<td>VH</td>
<td>HMAC Verify (CSNBHMV) - SHA-1, SHA-224, SHA-256</td>
</tr>
<tr>
<td>T2</td>
<td>VL</td>
<td>HMAC Verify (CSNBHMV) - SHA-256, SHA-384, SHA-512</td>
</tr>
<tr>
<td>T2</td>
<td>VV</td>
<td>CVV Verify (CSNBCV)</td>
</tr>
</tbody>
</table>
Chapter 33. openCryptoki support

openCryptoki is an open source implementation of the Cryptoki API as defined by the industry-wide PKCS #11 Cryptographic Token Interface Standard.

openCryptoki supports several cryptographic algorithms according to the PKCS #11 standards. The openCryptoki library loads plug-ins that provide hardware or software-specific support for cryptographic functions.

In PKCS #11 terminology, and hence in openCryptoki terminology, these plug-ins are called tokens. Do not confuse token in the context of PKCS #11 with token in the CCA context.

openCryptoki and PKCS #11
In the context of openCryptoki and PKCS #11, a token is a representation of a hardware or software component that implements cryptographic functions. For example, a PKCS #11 token can represent a cryptographic adapter, a smart card, or a cryptographic library.

The term CCA token is used to denote the openCryptoki plug-in for CCA. The CCA token represents a library that extends the openCryptoki token library and links to the CCA library (libcsulcca).

In the context of CCA, a token is a representation of a key and the attributes belonging to the key.

For more information about the openCryptoki services or about the interfaces between the openCryptoki main module and its tokens, obtain an openCryptoki package from openCryptoki github and see the documents in the doc subdirectory within this website.

The openCryptoki token directories
Each token has a separate token directory, which is used by openCryptoki to store token-specific information, like key objects, the user PIN, or the SO PIN. For most Linux distributions, the CCA token directory is /var/lib/opencryptoki/ccatok.

Note: The data in the token directory applies to all applications that use the token.

Prerequisites
- The libcsulcca library must be installed, see Chapter 29, “CCA installation instructions,” on page 1107.
- The cryptographic device driver must be loaded, see “Installing and loading the cryptographic device driver” on page 1103.
- A Crypto Express coprocessor (see CEX*C in “Terminology” on page xxi) must be accessible to Linux. For Linux on z/VM, this coprocessor must be dedicated to the guest.
- The master keys must be loaded and set in the coprocessor, see Chapter 31, “Utilities,” on page 1119.

Note: To properly configure the system, and to be authorized for the tasks described in this section, the pkcs11 group must be defined in /etc/group of the system. Every user of openCryptoki (including the users configuring openCryptoki
and its tokens) must be a member of this \textit{pkcs11} group. Use standard Linux management operations to create the \textit{pkcs11} group if needed, and to add users to this group as required.

You may also refer to the man page of openCryptoki \texttt{man openCryptoki} which has a \texttt{SECURITY NOTE} section that is important from the security perspective.

## Configuring openCryptoki for CCA support

After installing openCryptoki, you must configure tokens, start a daemon, and then initialize the tokens.

openCryptoki can handle several tokens with support for different hardware devices or software solutions. The CCA token interacts with the host part of the CCA library.

Perform the following tasks to configure the Linux on Z CCA enablement:

- "Confirming the openCryptoki configuration file"
- "Starting the openCryptoki slot daemon" on page 1141
- "Initializing the token" on page 1142

### Confirming the openCryptoki configuration file

For openCryptoki versions higher than 3.0, the global openCryptoki configuration file must define a slot entry for the CCA token.

As a minimum requirement, a slot entry must specify the library that implements the token. Most Linux distributions provide a default configuration file with a valid slot entry for the CCA token.

Table \ref{table:openCryptoki-libraries} lists the libraries that might be in place after you successfully installed openCryptoki. The actual list for your installation depends on your distribution and on the installed RPM packages.

<table>
<thead>
<tr>
<th>Library</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/lib64/opencryptoki/libopencryptoki.so</td>
<td>openCryptoki base library</td>
</tr>
<tr>
<td>/usr/lib64/opencryptoki/stdll/libpkcs11_icca.so</td>
<td>openCryptoki libica token library</td>
</tr>
<tr>
<td>/usr/lib64/opencryptoki/stdll/libpkcs11_sw.so</td>
<td>openCryptoki software token library</td>
</tr>
<tr>
<td>/usr/lib64/opencryptoki/stdll/libpkcs11_tpm.so</td>
<td>openCryptoki TPM token library (not supported by Linux on Z)</td>
</tr>
<tr>
<td>/usr/lib64/opencryptoki/stdll/libpkcs11_cca.so</td>
<td>openCryptoki CCA token library</td>
</tr>
<tr>
<td>/usr/lib64/opencryptoki/stdll/libpkcs11_ep11.so</td>
<td>openCryptoki EPII token library</td>
</tr>
<tr>
<td>/usr/lib64/opencryptoki/stdll/libpkcs11_icsf.so</td>
<td>openCryptoki ICSF token library</td>
</tr>
</tbody>
</table>

\textbf{Note:} The CCA token library is available only in 64-bit mode.

The \texttt{/etc/opencryptoki/opencryptoki.conf} file must exist and it must contain an entry for the CCA token to make the token available. By default, this entry is available after installing the CCA token. See the \texttt{slot 2} entry in the following sample configuration:
-------------- content of opencryptoki.conf --------------

version opencryptoki-3.4
#
# The following defaults are defined:
# hwversion = 0.0
# firmwareversion = 0.0
# description = Linux
# manufacturer = IBM
#
# The slot definitions below may be overridden and/or customized.
# For example:
# slot 0
# {
#     stdll = libpkcs11_ica.so
#     description = "OCK CCA Token"
#     manufacturer = "MyCompany Inc."
#     hwversion = 2.32
#     firmwareversion = 1.0
# }
#
# See man(5) opencryptoki.conf for further information.
#
# slot 0
# {
#     stdll = libpkcs11_ica.so
# }

# slot 1
# {
#     stdll = libpkcs11_cca.so
# }

# slot 2
# {
#     stdll = libpkcs11_sw.so
# }

# slot 3
# {
#     stdll = libpkcs11_ep11.so
#     confname = ep11tok.conf
# }

---------------------------- end -----------------------------

Note: The default path for slot token dynamic link libraries (STDLLs) is
/usr/lib64/opencryptoki/stdll/.

Normally, a token is available only if the token library is installed and the software
and hardware support that is required by the token is also installed. For example,
the CCA token is available only if all parts of the CCA library software are
installed and a CCA coprocessor is detected. After having started the slot daemon
as described in "Starting the openCryptoki slot daemon," you can use the
following command to display the list of available tokens:

$ pkcsconf -t

Starting the openCryptoki slot daemon

The openCryptoki slot daemon reads the configuration information and sets up the
tokens.

Before you begin

The openCryptoki configuration file must list all available tokens that are ready to
register to the openCryptoki slot daemon, see "Confirming the openCryptoki
configuration file" on page 1140.
Procedure

Start the slot daemon by using one of the following commands:

```
$ systemctl start pkcsslotd.service
$ systemctl start pkcsslotd
```

**Note:** For SLES11 and RHEL6, use the following commands to start and stop the slot daemon:

```
$ service pkcsslotd start
$ service pkcsslotd stop
```

See the Linux man page for the complete syntax of the `service` command. Use the following command to make the daemon start automatically after a reboot:

```
$ chkconfig pkcsslotd on
```

Initializing the token

After the openCryptoki library and the global configuration file are set up and the `pkcsslotd` daemon is started, the CCA token must be initialized.

PKCS #11 defines two users for each token: a security officer (SO) whose responsibility is the administration of the token, and a standard user (User) who wants to use the token to perform cryptographic operations. openCryptoki requires that for both the SO and the User a login PIN is defined as part of the token initialization.

The following command provides some useful slot information:

```
# pkcsconf -s
Slot #1 Info
   Description: OCK ICA Token
   Manufacturer: IBM
   Flags: 0x1 (TOKEN_PRESENT)
   Hardware Version: 0.0
   Firmware Version: 0.0
Slot #2 Info
   Description: OCK CCA Token
   Manufacturer: IBM
   Flags: 0x1 (TOKEN_PRESENT)
   Hardware Version: 0.0
   Firmware Version: 0.0
Slot #3 Info
   Description: Software Token
   Manufacturer: IBM
   Flags: 0x1 (TOKEN_PRESENT)
   Hardware Version: 0.0
   Firmware Version: 0.0
```

Find your preferred token in the details list and select the correct slot number. This number is used in the next initialization steps to identify your token:
pkcsconf -I

During token initialization, you are asked for a token label. Provide a meaningful name that helps you later to identify the token.

pkcsconf -P

openCryptoki security practices require that you change the default SO PIN (87654321) to a different value. Use the pkcsconf -P option to change the SO PIN.

pkcsconf -u

When you enter the user PIN initialization, you are asked for the newly set SO PIN. The length of the user PIN must be 4 - 8 characters.

pkcsconf -p

openCryptoki security practices require that you change the user PIN at least once with pkcsconf -p option. The length of the user PIN must be 4 - 8 characters. After you completed the PIN setup, the token is prepared and ready for use.

Note: Specify a user PIN that is different from 12345678 because this pattern is checked internally and marked as default PIN. A login attempt with this user PIN is recognized as not initialized.

How to recognize the CCA token

Use the pkcsconf -t command to display a list of all available tokens. You can check the slot and token information, and the PIN status at any time.

The following example shows information about the CCA token in the Token #2 Info section. This section provides information about the token that is plugged into slot number 2.

```
$ pkcsconf -t
...
Token #2 Info:
  Label: CCA Token
  Manufacturer: IBM Corp.
  Model: IBM CCA Token
  Serial Number: 123
  Flags: 0x44D (RNG|LOGIN_REQUIRED|USER_PIN_INITIALIZED|CLOCK_ON_TOKEN|TOKEN_INITIALIZED)
  Sessions: 0/-2
  R/W Sessions: -1/-2
  PIN Length: 4-8
  Public Memory: 0xFFFFFFFF/0xFFFFFFFF
  Private Memory: 0xFFFFFFFF/0xFFFFFFFF
  Hardware Version: 1.0
  Firmware Version: 1.0
  Time: 16:10:40
...
```

The output includes the following information:

**Label** The token label that was assigned when the token was initialized. You can change token labels with the pkcsconf -I command.

**Model** A unique designation for the token, “IBM CCA Token” for the CCA token.
Flags
A mask with information about the token initialization status, the PIN status, and features such as random number generator (RNG). For example, the mask for TOKEN_INITIALIZED is 0x00000400 and it is true if the token was initialized. “Login required” means that there is at least one mechanism that requires a session login to use that cryptographic function.

For more information about the flags, see the description of the TOKEN_INFO structure and the token information flags in the PKCS #11 Cryptographic Token Interface Standard.

PIN length
The PIN length range that was specified for the token.

Using the CCA token
You can use some CCA library functions through the openCryptoki standard interface (PKCS #11 standard C API).

The PKCS #11 Cryptographic Token Interface Standard describes the exact API.

Applications that are designed to work with openCryptoki can use the Linux on Z CCA enablement.

Supported mechanisms for the CCA token
Use the pkcsconf command to list the mechanisms (algorithms), that are supported by the CCA token.

Issue the pkcsconf command with the -m parameter to display mechanisms. Use the -c parameter to specify the slot number for the token of interest. The following example assumes that this slot number is 2:

```
# pkcsconf -m -c2
...
Mechanism #11
    Mechanism: 0x1080 (CKM_AES_KEY_GEN)
    Key Size: 16-32
    Flags: 0x8001 (CKF_HW|CKF_GENERATE)

Mechanism #12
    Mechanism: 0x1081 (CKM_AES_ECB)
    Key Size: 16-32
    Flags: 0x60301 (CKF_HW|CKF_ENCRYPT|CKF_DECRYPT|CKF_WRAP|CKF_UNWRAP)

Mechanism #13
    Mechanism: 0x1082 (CKM_AES_CBC)
    Key Size: 16-32
    Flags: 0x60301 (CKF_HW|CKF_ENCRYPT|CKF_DECRYPT|CKF_WRAP|CKF_UNWRAP)
...
```

The command output lists all mechanisms that are supported by the token in the specified slot. The mechanism ID and name correspond to the PKCS #11 specification.

Each entry includes the minimum and maximum supported key size and other properties, like hardware support and mechanism information flags. These flags provide information about the PKCS #11 functions that can use the mechanism. For some mechanisms, the flags show further attributes that describe the supported variants of the mechanism.

The supported Crypto Express coprocessors together with openCryptoki version 3.9 and CCA token support these PKCS #11 mechanisms:
Table 319. PKCS #11 mechanisms supported by the CCA token

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Key sizes</th>
<th>Properties</th>
<th>Support with OC version</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKM_AES_CBC</td>
<td>16-32</td>
<td>ENCRYPT, DECRYPT, WRAP, UNWRAP</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_AES_CBC_PAD</td>
<td>16-32</td>
<td>ENCRYPT, DECRYPT, WRAP, UNWRAP</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_AES_ECB</td>
<td>16-32</td>
<td>ENCRYPT, DECRYPT, WRAP, UNWRAP</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_AES_KEY_GEN</td>
<td>16-32</td>
<td>GENERATE</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_DES3_CBC</td>
<td>24-24</td>
<td>ENCRYPT, DECRYPT, WRAP, UNWRAP</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_DES3_CBC_PAD</td>
<td>24-24</td>
<td>ENCRYPT, DECRYPT, WRAP, UNWRAP</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_DES3_KEY_GEN</td>
<td>24-24</td>
<td>GENERATE</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_DES_CBC</td>
<td>8-8</td>
<td>ENCRYPT, DECRYPT, WRAP, UNWRAP</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_DES_CBC_PAD</td>
<td>8-8</td>
<td>ENCRYPT, DECRYPT, WRAP, UNWRAP</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_DES_KEY_GEN</td>
<td>8-8</td>
<td>GENERATE</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_ECDSA</td>
<td>160-521</td>
<td>SIGN, VERIFY, EC_F_P, EC_NAMEDCURVE</td>
<td>2.4.1</td>
</tr>
<tr>
<td>CKM_ECDSA_KEY_PAIR_GEN</td>
<td>160-521</td>
<td>GENERATE_KEYPAIR, EC_F_P, EC_NAMEDCURVE</td>
<td>2.4.1</td>
</tr>
<tr>
<td>CKM_ECDSA_SHA1</td>
<td>160-521</td>
<td>SIGN, VERIFY, EC_F_P, EC_NAMEDCURVE</td>
<td>3.0</td>
</tr>
<tr>
<td>CKM_ECDSA_SHA224</td>
<td>160-521</td>
<td>SIGN, VERIFY, EC_F_P, EC_NAMEDCURVE</td>
<td>3.9</td>
</tr>
<tr>
<td>CKM_ECDSA_SHA256</td>
<td>160-521</td>
<td>SIGN, VERIFY, EC_F_P, EC_NAMEDCURVE</td>
<td>3.7</td>
</tr>
<tr>
<td>CKM_ECDSA_SHA384</td>
<td>160-521</td>
<td>SIGN, VERIFY, EC_F_P, EC_NAMEDCURVE</td>
<td>3.7</td>
</tr>
<tr>
<td>CKM_ECDSA_SHA512</td>
<td>160-521</td>
<td>SIGN, VERIFY, EC_F_P, EC_NAMEDCURVE</td>
<td>3.7</td>
</tr>
<tr>
<td>CKM_GENERIC_SECRET_KEY_GEN</td>
<td>80-2048</td>
<td>GENERATE</td>
<td>3.4</td>
</tr>
<tr>
<td>CKM_MD5</td>
<td>0-0</td>
<td>DIGEST</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_MD5_HMAC</td>
<td>0-0</td>
<td>SIGN, VERIFY</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_MD5_HMAC_GENERAL</td>
<td>0-0</td>
<td>SIGN, VERIFY</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_MD5_RSA_PKCS</td>
<td>512-4096</td>
<td>SIGN, VERIFY</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_RSA_PKCS</td>
<td>512-4096</td>
<td>ENCRYPT, DECRYPT, SIGN, VERIFY</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_RSA_PKCS_KEY_PAIR_GEN</td>
<td>512-4096</td>
<td>GENERATE_KEYPAIR</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_SHA1_RSA_PKCS</td>
<td>512-4096</td>
<td>SIGN, VERIFY</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_SHA224</td>
<td>0-0</td>
<td>DIGEST</td>
<td>3.9</td>
</tr>
</tbody>
</table>
Table 319. PKCS #11 mechanisms supported by the CCA token (continued)

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Key sizes</th>
<th>Properties</th>
<th>Support with OC version</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKM_SHA224_HMAC</td>
<td>80-2048</td>
<td>SIGN, VERIFY</td>
<td>3.9</td>
</tr>
<tr>
<td>CKM_SHA224_HMAC_GENERAL</td>
<td>80-2048</td>
<td>SIGN, VERIFY</td>
<td>3.9</td>
</tr>
<tr>
<td>CKM_SHA256</td>
<td>0-0</td>
<td>DIGEST</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_SHA256_HMAC</td>
<td>80-2048</td>
<td>SIGN, VERIFY</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_SHA256_HMAC_GENERAL</td>
<td>80-2048</td>
<td>SIGN, VERIFY</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_SHA256_RSA_PKCS</td>
<td>512-4096</td>
<td>SIGN, VERIFY</td>
<td>2.4.2</td>
</tr>
<tr>
<td>CKM_SHA384</td>
<td>0-0</td>
<td>DIGEST</td>
<td>3.2</td>
</tr>
<tr>
<td>CKM_SHA384_HMAC</td>
<td>80-2048</td>
<td>SIGN, VERIFY</td>
<td>3.2</td>
</tr>
<tr>
<td>CKM_SHA384_HMAC_GENERAL</td>
<td>80-2048</td>
<td>SIGN, VERIFY</td>
<td>3.2</td>
</tr>
<tr>
<td>CKM_SHA512</td>
<td>0-0</td>
<td>DIGEST</td>
<td>3.2</td>
</tr>
<tr>
<td>CKM_SHA512_HMAC</td>
<td>80-2048</td>
<td>SIGN, VERIFY</td>
<td>3.2</td>
</tr>
<tr>
<td>CKM_SHA512_HMAC_GENERAL</td>
<td>80-2048</td>
<td>SIGN, VERIFY</td>
<td>3.2</td>
</tr>
<tr>
<td>CKM_SHA_1</td>
<td>0-0</td>
<td>DIGEST</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_SHA_1_HMAC</td>
<td>80-2048</td>
<td>SIGN, VERIFY</td>
<td>2.4</td>
</tr>
<tr>
<td>CKM_SHA_1_HMAC_GENERAL</td>
<td>80-2048</td>
<td>SIGN, VERIFY</td>
<td>2.4</td>
</tr>
</tbody>
</table>

For explanations of the key object properties, see the [PKCS #11 Cryptographic Token Interface Standard](#).

**Restrictions with using the CCA library functions**

The CCA library is subject to some limitations.

- Key imports through the `C_CreateObject` PKCS #11 function are supported for RSA private keys and for RSA public keys in CRT format only. Starting with openCryptoki version 3.4, the CCA token supports `C_CreateObject` function for AES, DES, DES3, and generic secret keys in addition to RSA keys.
- The default `CKA_SENSITIVE` setting for generating a key is `CK_FALSE` although the openCryptoki CCA token handles only secure keys, which correspond to sensitive keys in PKCS #11.
  
  Setting the value of `CKA_SENSITIVE` to `CK_FALSE` does not inhibit inspecting the value of `CKA_VALUE`. This setting does not compromise security because inspecting the value of `CKA_VALUE` does not reveal any sensitive information. The CCA secure key token (token as used in the CCA context) is stored in the `CKA_IBM_OPAQUE` attribute rather than in the `CKA_VALUE` attribute. Moreover, the key value is never stored in cleartext.
- The function `C_DigestKey` is not supported by the openCryptoki CCA token.

**Migrating openCryptoki version 2 tokens to version 3**

PKCS #11 token objects are persistent in the token. openCryptoki stores token objects in the token directory, `/var/lib/opencryptoki/ccatok/TOK_08J`, for the CCA token.
Private openCryptoki token objects that have been saved in openCryptoki version 2 format must be migrated before they can be used with openCryptoki version 3.

**Before you begin**

Public openCryptoki token objects are not encrypted and no migration is required.

**Note:** Throughout this section, *token object* is used to mean a PKCS #11 object, like a key, and the term CCA token is used to denote the openCryptoki plug-in for CCA.

**About this task**

In openCryptoki version 2, private token objects are encrypted and decrypted with a secure key in the cryptographic adapter. In version 3, this encryption and decryption is done with a clear key using cryptographic software functions. Therefore, openCryptoki version 3 cannot decrypt a version 2 private token object.

Migration decrypts the private key objects by using the CSNBDEC CCA verb and the openCryptoki key stored in MK_USER. The objects are then re-encrypted using the cryptographic software functions according to openCryptoki version 3. The key bits that are stored in MK_USER are subsequently used as a clear key.

You can find more detailed information about token migration in the documentation of the appropriate openCryptoki RPM.

**Procedure**

1. Back up the CCA token directory. For most Linux distributions, this directory is `/var/lib/opencryptoki/ccatok`. The content of the CCA token directory includes the following items:

   **MK_USER**
   The TDES key used for internal on-disk encryption, encrypted under the user's PIN by software routines.

   **MK_SO**
   The TDES key used for internal on-disk encryption, encrypted under the SO's PIN by software routines. This is the same key as in MK_USER, but with a different encryption.

   **NKTOK.DAT**
   Token information.

   **TOK_OBJ**
   The directory in which token objects are stored.

   **TOK_OBJ/OBJ.IDX**
   A list of current token objects.

2. Upgrade to openCryptoki version 3 or install version 3.
   - Upgrade to openCryptoki version 3.
     For most Linux distributions, upgrading from version 2 to version 3 preserves the contents of the CCA token directory.
   - Install openCryptoki version 3.
     For most Linux distributions, installing version 3 removes the contents of the CCA token directory. Restore your backup of this directory to migrate the version 2 tokens.

3. Ensure that no openCryptoki processes are running.
a. Stop all applications that use openCryptoki.

b. Find out whether the pkcs10td daemon is running by issuing the following command:

```bash
$ ps aux |grep pkcs10td
```

If the daemon is running, the command output shows a process for pkcs10td.

c. If applicable, stop the daemon by issuing a command of this form:

```bash
$ killall pkcs10td
```

4. Run pkcscca.

Example:

```bash
$ pkcscca -m v2objectsv3 -v
```

The `-v` option prints information about which objects did and did not get migrated. This command migrates CCA token objects in the default CCA token directory, `/var/lib/opencryptoki/ccatok`.

If your distribution uses a different token directory, use the `-d` option to specify this directory.

Example:

```bash
$ pkcscca -m v2objectsv3 -v -d /home/ccaadmin/ccatok
```

**Results**

The CCA private token objects are encrypted according to openCryptoki version 3 and ready to be accessed.

**What to do next**

If openCryptoki version 3 cannot find the newly migrated CCA private token objects, reboot or remove the shared memory file, `/dev/shm/var.lib.opencryptoki.ccatok`.

**Attention:** Ensure that no openCryptoki processes are running when removing the shared memory.

**Migrating to a new CCA master key**

If you need to migrate a CCA key to a new wrapping CCA master key (MK), use the pkcscca tool as described in this topic.

**Before you begin**

Prerequisite for using the migration key migration function described in this topic is that you have installed openCryptoki version 3.4 or higher.
About this task

There may be situations when CCA master keys must be changed. All CCA secret and private keys are enciphered (wrapped) with a master key (MK). After a CCA master key is changed, the keys wrapped with an old master key need to be re-enciphered with the new master key. Only keys which are marked as CKA_EXTRACTABLE=TRUE can be migrated. However, by default all keys are marked as CKA_EXTRACTABLE. So only those keys where the user explicitly chooses to mark them as non extractable, for example, by setting CKA_EXTRACTABLE=FALSE cannot be migrated.

Use the pkcscca tool to migrate wrapped CCA keys.

After a new master key is loaded and set, perform the following steps:

Procedure

1. Stop all processes that are currently using openCryptoki with the CCA token.
   a. Stop all applications that use openCryptoki.
   b. Find out whether the pkcsslotd daemon is running by issuing the following command:

   ```bash
   $ ps awx | grep pkcsslotd
   ```

   If the daemon is running, the command output shows a process for pkcsslotd.
   c. If applicable, stop the daemon by issuing a command of this form:

   ```bash
   $ killall pkcsslotd
   ```

2. Make sure pkcsslotd is running. Start or restart pkcsslotd if it was stopped in step 1.

3. Back up the token object repository of the CCA token. For example, you can use the following commands:

   ```bash
   cd /var/lib/opencryptoki/cca/
   tar -cvzf ~/cca/TOK_OBJ_backup.tgz TOK_OBJ
   ```

4. Migrate the keys of the CCA token object repository with the pkcscca migration tool.

   ```bash
   pkcscca -m keys -s <slotid> -k <aes|apka|asym|sym>
   ```

   The following parameters are mandatory:
   - `-s` slot number for the CCA token
   - `-k` master key type to be migrated: aes, apka, asym, or sym
   The following parameter is optional:
   - `-m keys` re-encipers private keys only with a new CCA master key.

All the specified token objects representing extractable keys that are found for the CCA token are re-encrypted and ready for use. Keys with an attribute CKA_EXTRACTABLE=FALSE are not eligible for migration. The keys that failed to migrate are displayed to the user.
Example:

$ pkcsca -m keys -s 2 -k sym

migrates all private keys wrapped with symmetric master keys found in the CCA plug-in for openCryptoki in PKCS slot 2.

5. Re-start the previously stopped openCryptoki processes.

Results

All specified keys, for example, all private and secret keys (for asymmetric and symmetric cryptography) are now re-encrypted with the new CCA master key and are ready for use in CCA verbs.

Tracing in openCryptoki

In openCryptoki version 3.3, tracing was introduced. There is no longer a need to compile openCryptoki with debug enabled via configure --enable-debug to debug problems.

Trace messages are enabled via the environment variable, OPENCRYPTOKI_TRACE_LEVEL=<level>. All trace output is logged into a trace.<pid> file in the /var/log/opencryptoki directory. A trace file is created per process.

Set the OPENCRYPTOKI_TRACE_LEVEL environment variable to one of the available levels:

0  None. No messages are locked.
1  ERROR: log error messages
2  WARNING: log warning messages
3  INFO: log informational messages
4  DEVELOPMENT: log development debug messages; these messages may help debug while developing pkcs#11 applications.
5  DEBUG: debug messages that are useful to openCryptoki developers. This level must be enabled via the configure --enable-debug option in the configure script.

Trace level 4 includes all the messages from trace levels 1, 2 and 3.
Chapter 34. List of abbreviations

A list of abbreviations used in this document.
ADB Actual Data Block
AES Advanced Encryption Standard
AESKW AES Key Wrap (ANS X9.102)
AIX® Advanced Interactive Executive operating system
ANS American National Standards
ANSI American National Standards Institute
API Application Programming Interface
APAR Authorized Problem Analysis Report
ASCII American National Standard Code for Information Interchange
ASN Abstract Syntax Notation
ATC Application Transaction Counter
ATM Automated Teller Machine
BC Block Contents
BDK Base Derivation Key
BER ASN.1 Basic Encoding Rules
CA Certification Authority
CBC Cipher block chaining
CCA Common Cryptographic Architecture
CEX3C Crypto Express3 Coprocessor
CEX4C Crypto Express4 Coprocessor
CEX5C Crypto Express5 Coprocessor
CEX6C Crypto Express6 Coprocessor
CKDS Cryptographic Key Data Set.
CKSN Current-Key Serial Number
CMAC Block Cipher-based Message Authentication Code Algorithm (analogous to HMAC), NIST SP 800-38B
CMK Current Master Key
CMS Cryptographic Message Syntax
CNI Coprocessor Node Initialization
CNM  Cryptographic Node Management (utility)
CPACF  Central Processor Assist for Cryptographic Functions
CRT  Chinese Remainder Theorem.
CSC  Card Security Code
CV  Control Vector
CVC  Card verification code used by MasterCard.
CVK  Card Verification Key
CVV  Card verification value used by VISA.
DEA  Data encryption algorithm
DES  Data Encryption Standard
DK  Deutsche Kreditwirtschaft (German Banking Industry Committee).
     Formerly known as ZKA.
DMA  Direct Memory Access
DOW  Day of the Week
DRBG  Deterministic Random Bit Generator
DRBGVS  Deterministic Random Bit Generator Validation System (NIST SP 800-90A)
DSA  Digital Signature Algorithm.
DSS  Digital Signature Standard.
DUKPT  Derived Unique Key Per Transaction
EBCDIC  Extended Binary Coded Decimal Interchange Code
EC  Elliptic Curve
ECB  Electronic codebook.
ECC  Elliptic Curve Cryptography
ECDH  Elliptic Curve Diffie-Hellman
ECDSA  Elliptic Curve Digital Signature Algorithm
ECI  Eurocheque International
EDE  Encipher-Decipher-Encipher
EEPROM  Electrically Erasable, Programmable Read-Only Memory
EID  Environment Identification.
EPP  Encrypting PIN PAD
FCV  function control vector
FIPS  Federal Information Processing Standards
FPE  Format Preserving Encryption
GBP  German Bank Pool.

HMAC  Keyed Hash MAC

HSM  Hardware Security Module

IBM  International Business Machines

ICSF  Integrated Cryptographic Service Facility.

ICV  Initial Chaining Value

IEC  International Electrotechnical Commission

IETF  Internet Engineering Task Force

I/O  Input/Output

IPEK  Initial PIN Encryption Key

IPL  initial program load

ISO  International Organization for Standardization.

ISO/DIS  International Organization for Standardization/Draft International Standard

ISO/FDIS  International Organization for Standardization/Final Draft International Standard

JNI  Java Native Interface

KC  Key Confirmation

KDF  Key Derivation Function

KEK  Key-Encrypting Key

KVP  key verification pattern

LRC  longitudinal redundancy check

LSB  least significant bit

MAC  Message Authentication Code

MD5  Message Digest-5 Hash Algorithm

MDC  Modification Detection Code

MDK  Master-Derivation Key

MFK  Master File Key

MK  Master Key

MKVP  Master-Key Verification Pattern

MSB  Most significant bit

NIST  US National Institute of Science and Technology.

NMK  New Master Key

OAEP  Optimal asymmetric encryption padding.

OCSP  Open Certificate Status Protocol
OEM  Original Equipment Manufacturer
OID  Object Identifier
OMK  Old Master Key
OPK  Object Protection Key
PAN  Personal Account Number
PBF  PIN Block Format
PCI  Peripheral Component Interconnect
PCIe  PCI Express
PCI-X  PCI Extended
PCICA  PCI Cryptographic Accelerator.
PCICC  PCI Cryptographic Coprocessor.
PCIXCC  PCI X Cryptographic Coprocessor.
PIN  Personal Identification Number
PKA  public key algorithm
PKCS  Public Key Cryptographic Standards (RSA Data Security, Inc.)
PKDS  Public key data set (PKA cryptographic key data set)
PKI  public key infrastructure
POST  Power-On Self Test
PRNG  Pseudo Random Number Generator
PROM  Programmable Read-Only Memory
PRW  PIN reference word/value
PVV  PIN Validation Value
RA  Registration Authority
RACF  Resource Access Control Facility
RAM  Random Access Memory
RFC  Request for Comments
RHEL  Red Hat Enterprise Linux
RNG  Random Number Generator
ROM  Read-Only Memory
RPM  RPM Package Manager (originally: Red Hat Package Manager)
RSA  Rivest, Shamir, and Adleman
SEC 2  Standards for Efficient Cryptography 2
SECG  Standards for Efficient Cryptography Group
SET  Secure Electronic Transaction.
SHA  Secure Hash Algorithm
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLES</td>
<td>SUSE Linux Enterprise Server</td>
</tr>
<tr>
<td>SNA</td>
<td>Systems Network Architecture</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer.</td>
</tr>
<tr>
<td>TDEA</td>
<td>Triple Data Encryption Algorithm (see also TDES).</td>
</tr>
<tr>
<td>TDES</td>
<td>Triple DES (Data Encryption Standard) or TDEA.</td>
</tr>
<tr>
<td>TKE</td>
<td>Trusted key entry.</td>
</tr>
<tr>
<td>TLV</td>
<td>Tag, Length, Value</td>
</tr>
<tr>
<td>TMK</td>
<td>Terminal Master Key</td>
</tr>
<tr>
<td>TVV</td>
<td>Token-validation value</td>
</tr>
<tr>
<td>UAT</td>
<td>UDX Authority Table.</td>
</tr>
<tr>
<td>UDF</td>
<td>User-defined function.</td>
</tr>
<tr>
<td>UDK</td>
<td>User-derived key.</td>
</tr>
<tr>
<td>UDP</td>
<td>User Developed Program.</td>
</tr>
<tr>
<td>UDX</td>
<td>User Defined Extension.</td>
</tr>
<tr>
<td>UKPT</td>
<td>Unique Key Per Transaction</td>
</tr>
<tr>
<td>UTC</td>
<td>Coordinated Universal Time</td>
</tr>
<tr>
<td>VIS</td>
<td>Visa Integrated Circuit Card Specification</td>
</tr>
<tr>
<td>XOR</td>
<td>Exclusive-OR</td>
</tr>
</tbody>
</table>
Part 4. Appendixes
Appendix. Accessibility

Accessibility features help users who have a disability, such as restricted mobility or limited vision, to use information technology products successfully.

Documentation accessibility

The Linux on Z and LinuxONE publications are in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when you use the PDF file and want to request a Web-based format for this publication, use the Readers' Comments form in the back of this publication, send an email to eservdoc@de.ibm.com, or write to:

IBM Deutschland Research & Development GmbH
Information Development
Department 3282
Schoenaicher Strasse 220
71032 Boeblingen
Germany

In the request, be sure to include the publication number and title.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility at

www.ibm.com/able

© Copyright IBM Corp. 2007, 2018
Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.
This information is for planning purposes only. The information herein is subject to change before the products described become available.

**Programming interface information**

This book documents intended Programming Interfaces that allow the customer to write programs to obtain the services of the Common Cryptographic Architecture.

**Trademarks**

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at:


Adobe is either a registered trademark or trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.
Glossary

This glossary includes some terms and definitions from the IBM Dictionary of Computing, New York: McGraw Hill, 1994. This glossary also includes some terms and definitions from:

- The IBM Glossary of Computing Terms. Definitions are identified by the symbol (D) after the definition.
- The IBM TotalStorage Enterprise Storage Server® documentation. Definitions of published parts of this vocabulary are identified by the symbol (E) after the definition.
- The Information Technology Vocabulary, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

access control
Ensuring that the resources of a computer system can be accessed only by authorized users in authorized ways.

access method
A technique for moving data between main storage and input/output devices.

adapter
A printed circuit card that modifies the system unit to allow it to operate in a particular way.

Advanced Encryption Standard (AES)
A data encryption technique that improved upon and officially replaced the Data Encryption Standard (DES). AES is sometimes referred to as Rijndael, which is the algorithm on which the standard is based.

Advanced Interactive Executive (AIX) operating system
IBM's implementation of the UNIX® operating system.

American National Standard Code for Information Interchange (ASCII)
The standard code using a coded character set consisting of 7-bit characters (8 bits including parity check) that is used for information exchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

American Institute of Standardization (ANSI)
An organization, consisting of producers, consumers, and general interest groups that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States. (A)

ANSI key-encrypting key (AKEK)
A 64- or 128-bit key used exclusively in ANSI X9.17 key management applications to protect data keys exchanged between systems.

ANSI X9.17
An ANSI standard that specifies algorithms and messages for DES key distribution.

ANSI X9.19
An ANSI standard that specifies an optional double-MAC procedure which requires a double-length MAC key.

application program interface (API)
A functional interface supplied by the operating system or by a separately deliverable licensed program that allows an application program written in a high-level language to use specific data or...
functions of the operating system or the licensed program. (D)

**Application System/400 system (AS/400)**

AS/400 was one of a family of general purpose mid-range systems with a single operating system, Operating System/400®, that provides application portability across all models. AS/400 is now referred to as IBM System i®.

**assembler language**

A source language that includes symbolic machine language statements in which there is a one-to-one correspondence between the instruction formats and the data formats of the computer.

**asymmetric cryptography**

Synonym for public key cryptography. (D)

**authentication**

A process used to verify the integrity of transmitted data, especially a message (I). In computer security, a process used to verify the user of an information system or protected resources.

**authentication pattern**

An 8-byte pattern that is calculated from the master key when initializing the cryptographic key data set. The value of the authentication pattern is placed in the header record of the cryptographic key data set.

**authorize**

To permit or give authority to a user to communicate with or make use of an object, resource, or function.

**authorization**

The right granted to a user to communicate with or make use of a computer system (I). The process of granting a user either complete or restricted access to an object, resource, or function.

**bus**

In a processor, a physical facility along which data is transferred.

**C**

**Card-Verification Code (CVC)**

See Card-Verification Value.

**Card-Verification Value (CVV)**

A cryptographic method, defined by VISA, for detecting forged magnetic-stripped cards. This method cryptographically checks the contents of a magnetic stripe. This process is functionally the same as MasterCard’s Card-Verification Code (CVC) process.

**Central Processor Assist for Cryptographic Functions (CPACF)**

Hardware that provides support for symmetric ciphers and secure hash algorithms (SHA) on every central processor. Hence the potential encryption/decryption throughput scales with the number of central processors in the system.

**channel**

A path along which signals can be sent, for example, a data channel or an output channel. (A)

**checksum**

The sum of a group of data associated with the group and used for checking purposes. (T)

**Chinese Remainder Theorem (CRT)**

A mathematical theorem that defines a format for the RSA private key that improves performance.

**Cipher Block Chaining (CBC)**

A mode of encryption that uses the data encryption algorithm and requires an initial chaining vector. For encipher, it exclusively ORs the initial block of data with the initial control vector and then enciphers it. This process results in the encryption both of the input block and of the initial control vector that it uses on the next input block as the process repeats. A comparable chaining process works for decipher.

**ciphertext**

Text that results from the encipherment of plaintext. Synonym for enciphered data. (D).

**clear data**

Data that is not enciphered.

**clear key**

Any type of encryption key not protected by encryption under another key.

**cleartext**

Text that has not been altered by a cryptographic process. Synonym for plaintext. See also ciphertext.
Common Cryptographic Architecture (CCA)
The CCA API is the programming interface described in this document.

Common Cryptographic Architecture:
Cryptographic Application Programming Interface
Defines a set of cryptographic functions, external interfaces, and a set of key management rules that provide a consistent, end-to-end cryptographic architecture across different IBM platforms.

concatenation
An operation that joins two characters or strings in the order specified, forming one string whose length is equal to the sum of the lengths of its parts.

canvas
A part of a computer used for communication between the operator or maintenance engineer and the computer.

control vector (CV)
In CCA, a 16-byte string that is exclusive-OR-ed with a master key or a key-encrypting key to create another key that is used to encipher and decipher data or data keys. A control vector determines the type of key and the restrictions on the use of that key.

cryptographic adapter
See cryptographic coprocessor.

cryptographic coprocessor
A microprocessor that adds cryptographic processing functions to specific processors. A cryptographic coprocessor on an IBM Z mainframe is a tamper-resistant chip built into the processor board.

cryptographic key data set (CKDS)
A data set that contains the encrypting keys used by an installation.

CUSP (Cryptographic Unit Support Program)
The IBM cryptographic offering, program product 5740-XY6, using the channel-attached 3848. CUSP is no longer in service.

data key or data-encrypting key
A key used to encipher, decipher, or authenticate data. Contrast with key-encrypting key.

Data Encryption Algorithm (DEA)
A 64-bit block cipher that uses a 64-bit key, of which 56 bits are used to control the cryptographic process and 8 bits are used for parity checking to ensure that the key is transmitted properly.

Data Encryption Standard (DES)
The National Institute of Standards and Technology Data Encryption Standard, adopted by the U.S. government as Federal Information Processing Standard (FIPS) Publication 46, which allows only hardware implementations of the data-encryption algorithm.

data translation key
A 64-bit key that protects data transmitted through intermediate systems when the originator and receiver do not share the same key.

decipher
To convert enciphered data in order to restore the original data.

In computer security, to convert ciphertext into plaintext by means of a cipher system.

To convert enciphered data into clear data. Synonym for decrypt. Contrast with encipher.

decrypt
To decipher or decode. Synonym for decipher. Contrast with encrypt.

device driver
A program that contains the code needed to attach and use a device.

directory server
A server that manages key records in key storage by using an Indexed Sequential Access Method.

digital signature
In public key cryptography, information created by using a private key and verified by using a public key. A digital signature provides data integrity and source nonrepudiation.

Digital Signature Algorithm (DSA)
A public key algorithm for digital signature generation and verification used with the Digital Signature Standard.

Digital Signature Standard (DSS)
A standard describing the use of
algorithms for digital signature purposes. One of the algorithms specified is DSA (Digital Signature Algorithm).

domain
That part of a network in which the data processing resources are under common control. (T)

double-length key
A key that is 128 bits long. A key can be either double- or single-length. A single-length key is 64 bits long.

Electronic Code Book (ECB)
A mode of operation used with block cipher cryptographic algorithms in which plaintext or ciphertext is placed in the input to the algorithm and the result is contained in the output of the algorithm.

Elliptic Curve Cryptography (ECC)
A public-key process discovered independently in 1985 by Victor Miller (IBM) and Neal Koblitz (University of Washington). ECC is based on discrete logarithms. The algebraic structure of elliptic curves over finite fields makes it much more difficult to challenge at equivalent RSA key lengths.

encipher
To scramble data or to convert data to a secret code that masks the meaning of the data to unauthorized recipients. Synonym for encrypt. Contrast with decipher. See also encode.

encrypt
Synonym for encipher. (I) To convert cleartext into ciphertext. Contrast with decrypt.

erasable programmable read-only memory (EPROM)
A type of memory chip that can retain its contents without electricity. Unlike the programmable read-only memory (PROM), which can be programmed only once, the EPROM can be erased by ultraviolet light and then reprogrammed. (E)

Eurocheque International S.C. (ECI)
A financial institution consortium that has defined three PIN-block formats.

exit routine
In the CCA products, a user-provided routine that acts as an extension to the processing provided with calls to the security API.

exit
To execute an instruction within a portion of a computer program in order to terminate the execution of that portion. Such portions of computer programs include loops, subroutines, modules, and so on. (T)

A user-written routine that receives control from the system during a certain point in processing - for example, after an operator issues the START command.

exportable form
A condition a key is in when enciphered under an exporter key-encrypting key. In this form, a key can be sent outside the system to another system. A key in exportable form cannot be used in a cryptographic function.

EXPOR TER key
In the CCA implementation, a type of DES key-encrypting key that can encipher a key at a sending node. Contrast with IMPORTER key.

exporter key-encrypting key
A 128-bit key used to protect keys sent to another system. A type of transport key.

feature
A part of an IBM product that can be ordered separately.

Federal Information Processing Standard (FIPS)
A standard published by the US National Institute of Science and Technology.

financial PIN
A Personal Identification Number used to identify an individual in some financial transactions. To maintain the security of the PIN, processes and data structures have been adopted for creating, communicating, and verifying PINs used in financial transactions. See also Personal Identification Number.

Flash-Erasable Programmable Read-Only Memory (flash EPROM)
A memory that has to be erased before new data can be saved into the memory.

Function control vector (FCV)
The CCA software that can be loaded into the coprocessor limits the functionality of the coprocessor based on the values in a
distributed function control vector (FCV). A CCA FCV is a digitally signed data structure (certificate) signed by IBM. The certificate is used to accommodate potential government export and import regulations under USA Government control. An IBM cryptographic coprocessor becomes a practical cryptographic engine when it validates and accepts digitally signed software. IBM exports its cryptographic coprocessor hardware as non-cryptographic products, and controls and reports the export of the cryptography-enabling software as required.

Galois/Counter Mode (GCM)
A mode of operation for symmetric key cryptographic block ciphers that uses universal hashing over a binary Galois field to provide authenticated encryption. The operation is an authenticated encryption algorithm designed to provide both data authenticity (integrity) and confidentiality. GCM is defined for block ciphers with a block size of 128 bits. It can be implemented in hardware to achieve high speeds with low cost and low latency. Software implementations can achieve excellent performance by using table-driven field operations. It uses mechanisms that are supported by a well-understood theoretical foundation, and its security follows from a single reasonable assumption about the security of the block cipher.

German Bank Pool (GBP)
A German financial institution consortium that defines specific methods of PIN calculation.

hashing
An operation that uses a one-way (irreversible) function on data, usually to reduce the length of the data and to provide a verifiable authentication value (checksum) for the hashed data.

header record
A record containing common, constant, or identifying information for a group of records that follows. (D)

host
In this publication, same as host computer or host processor. The machine in which the coprocessor resides. In a computer network, the computer that usually performs network-control functions and provides end-users with services such as computation and database access. (I)

importable form
A condition a key is in when it is enciphered under an importer key-encrypting key. A key is received from another system in this form. A key in importable form cannot be used in a cryptographic function.

IMPORTER key
In the CCA implementation, a type of DES key-encrypting key that can decipher a key at a receiving mode. Contrast with EXPORTER key.

importer key-encrypting key
A 128-bit key used to protect keys received from another system. A type of transport key.

Integrated Cryptographic Service Facility (ICSF)
An IBM licensed program that supports the cryptographic hardware feature for the high-end IBM Z machines running in a z/OS environment.

International Organization for Standardization (ISO)
An organization of national standards bodies established to promote the development of standards to facilitate the international exchange of goods and services, and develop cooperation in intellectual, scientific, technological, and economic activity.

initial chaining vector (ICV)
A 64-bit random or pseudo-random value used in the cipher block chaining mode of encryption with the data encryption algorithm.

input PIN-encrypting key
A 128-bit key used to protect a PIN block sent to another system or to translate a PIN block from one format to another.

installation exit
See exit.

jumper
A wire that joins two unconnected circuits on a printed circuit board.

key
In computer security, a sequence of symbols used with a cryptographic algorithm to encrypt or decrypt data.
key agreement
A key establishment procedure where the resultant secret keying material is a function of information contributed by two participants, so that no party can predetermine the value of the secret keying material independently from the contributions from the other parties.

key-encrypting key (KEK)
A key used for the encryption and decryption of other keys. Contrast with data-encrypting key. Also called a transport key.

key half
In CCA, one of the two DES keys that make up a double-length key.

key identifier
In CCA, a 64-byte variable which is either a key label or a key token.

key label
In CCA, an identifier of a key-record in key storage.

key storage
In CCA, a data file that contains cryptographic keys which are accessed by key label.

key token
In CCA, a data structure that can contain a cryptographic key, a control vector, and other information related to the key.

key output data set
A key generator utility program data set containing information about each key that the key generator utility program generates except an importer key for file encryption.

key part
A 32-digit hexadecimal value that you enter to be combined with other values to create a master key or clear key.

key part register
A register in the key storage unit that stores a key part while you enter the key part.

link
The logical connection between nodes including the end-to-end control procedures. The combination of physical media, protocols, and programming that connects devices on a network. In computer programming, the part of a program, in some cases a single instruction or an address, that passes control and parameters between separate portions of the computer program. (A) (I) To interconnect items of data or portions of one or more computer programs. (I) In SNA, the combination of the link connection and link stations joining network nodes.

LPAR mode
The central processor mode that enables the operator to allocate the hardware resources among several logical partitions.

MAC generation key
A 64-bit or 128-bit key used by a message originator to generate a message authentication code sent with the message to the message receiver.

MAC verification key
A 64-bit or 128-bit key used by a message receiver to verify a message authentication code received with a message.

make file
A composite file that contains either device configuration data or individual user profiles.

master key (MK)
In computer security, the top-level key in a hierarchy of key-encrypting keys.

master key register
A register in the cryptographic coprocessors that stores the master key that is active on the system.

master key variant
A key derived from the master key by use of a control vector. It is used to force separation by type of keys on the system.

MD5
Message Digest 5. A hash algorithm.

Message Authentication Code (MAC)
A number or value derived by processing data with an authentication algorithm, The cryptographic result of block cipher operations on text or data using a Cipher Block Chaining (CBC) mode of operation, A digital signature code.

Modification Detection Code (MDC)
In cryptography, a number or value that interrelates all bits of a data stream so that, when enciphered, modification of any bit in the data stream results in a new MDC.
multiple encipherment
The method of encrypting a key under a double-length key-encrypting key.

National Institute of Science and Technology (NIST)
The current name for the US National Bureau of Standards.

new master key (NMK) register
A register in the key storage unit that stores a master key before you make it active on the system.

NOCV processing
Process by which the key generator utility program or an application program encrypts a key under a transport key itself rather than a transport key variant.

nonce
A time-varying value that has at most a negligible chance of repeating, such as a random value that is generated anew for each use, a timestamp, a sequence number, or some combination of these.

nonrepudiation
A method of ensuring that a message was sent by the appropriate individual.

old master key (OMK) register
A register in the key storage unit that stores a master key that you replaced with a new master key.

operational form
The condition of a key when it is encrypted under the master key so that it is active on the system.

output PIN-encrypting key
A 128-bit key used to protect a PIN block received from another system or to translate a PIN block from one format to another.

PCI X Cryptographic Coprocessor
An asynchronous cryptographic coprocessor available on the IBM eServer zSeries 990 and IBM eServer zSeries 800.

Personal Account Number (PAN)
A Personal Account Number identifies an individual and relates that individual to an account at a financial institution. It consists of an issuer identification number, customer account number, and one check digit.

Personal Identification Number (PIN)
The 4-digit to 12-digit number entered at an automatic teller machine to identify and validate the requester of an automatic teller machine service. Personal identification numbers are always enciphered at the device where they are entered, and are manipulated in a secure fashion.

Personal Security card
An ISO-standard “smart card” with a microprocessor that enables it to perform a variety of functions such as identifying and verifying users, and determining which functions each user can perform.

PIN block
A 64-bit block of data in a certain PIN block format. A PIN block contains both a PIN and other data.

PIN generation key
A 128-bit key used to generate PINs or PIN offsets algorithmically.

PIN key
A 128-bit key used in cryptographic functions to generate, transform, and verify the personal identification numbers.

PIN offset
For 3624, the difference between a customer-selected PIN and an institution-assigned PIN. For German Bank Pool, the difference between an institution PIN (generated with an institution PIN key) and a pool PIN (generated with a pool PIN key).

PIN verification key
A 128-bit key used to verify PINs algorithmically.

plaintext
Data that has nor been altered by a cryptographic process. Synonym for cleartext. See also ciphertext.

Power-On Self Test (POST)
A series of diagnostic tests run automatically by a device when the power is turned on.

private key
In computer security, a key that is known only to the owner and used together with a public-key algorithm to decipher data. The data is enciphered using the related
public key. Contrast with public key. See also public-key algorithm.

profile
Data that describes the significant characteristics of a user, a group of users, or one-or-more computer resources.

profile ID
In the CCA implementation, the value used to access a profile within the CCA access-control system.

protocol
A set of semantic and syntactic rules that determines the behavior of functional units in achieving communication. (I) In SNA, the meanings of and the sequencing rules for requests and responses used to manage the network, transfer data, and synchronize the states of network components. A specification for the format and relative timing of information exchanged between communicating parties.

Programmed Cryptographic Facility (PCF)
An IBM licensed program that provides facilities for enciphering and deciphering data and for creating, maintaining, and managing cryptographic keys. (D)
The IBM cryptographic offering, program product 5740-XY5, using software only for encryption and decryption. This product is no longer in service.

public key
In computer security, a key that is widely known, and used with a public-key algorithm to encrypt data. The encrypted data can be decrypted only with the related private key. Contrast with private key. See also public-key algorithm.

Public Key Algorithm (PKA)
In computer security, an asymmetric cryptographic process that uses a public key to encrypt data and a related private key to decrypt data. Contrast with Data Encryption Algorithm and Data Encryption Standard algorithm. See also Rivest-Shamir-Adleman algorithm.

public key cryptography
In computer security, cryptography in which a public key is used for encryption and a private key is used for decryption. Synonymous with asymmetric cryptography.

Public-Key Cryptography Standards (PKCS)
Specifications produced by RSA Laboratories in cooperation with secure system developers worldwide, for the purpose of accelerating the deployment of public-key cryptography. First published in 1991.

Random access memory (RAM)
A storage device into which data are entered and from which data are retrieved in a non-sequential manner.

record chaining
When there are multiple cipher requests and the output chaining vector (OCV) from the previous encipher request is used as the input chaining vector (ICV) for the next encipher request.

Read-only memory (ROM)
Memory in which stored data cannot be modified by the user except under special conditions.

Resource Access Control Facility (RACF)
An IBM licensed program that enables access control by identifying and verifying the users to the system, authorizing access to protected resources, logging detected unauthorized attempts to enter the system, and logging detected accesses to protected resources.

retained key
A private key that is generated and retained within the secure boundary of the cryptographic coprocessor.

Rivest-Shamir-Adleman (RSA) algorithm
A process for public key cryptography that was developed by R. Rivest, A. Shamir, and L. Adleman.

RS-232
A specification that defines the interface between data terminal equipment and data circuit-terminating equipment, using serial binary data interchange.

RS-232C
A standard that defines the specific physical, electronic, and functional characteristics of an interface line that uses a 25-pin connector to connect a workstation to a communication device.

Secure Electronic Transaction
A standard created by Visa International
Secure Hash Algorithm (SHA), FIPS 180
A set of related cryptographic hash functions designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST). The first member of the family, published in 1993, is officially called SHA. However, today, it is often unofficially called SHA-0 to avoid confusion with its successors. Two years later, SHA-1, the first successor to SHA, was published. Four more variants have since been published with increased output ranges and a slightly different design: SHA-224, SHA-256, SHA-384, and SHA-512 (all are sometimes referred to as SHA-2).

secure key
A key that is encrypted under a master key. When using a secure key, it is passed to a cryptographic coprocessor where the coprocessor decrypts the key and performs the function. The secure key never appears in the clear outside of the cryptographic coprocessor.

Secure Sockets Layer
A security protocol that provides communications privacy over the Internet by allowing client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

security server
In the CCA implementation, the functions provided through calls made to the security API.

SHA-1 (Secure Hash Algorithm 1, FIPS 180)
A hash algorithm required for use with the Digital Signature Standard.

SHA-2 (Secure Hash Algorithm 2, FIPS 180)
Four additional variants to the SHA family, with increased output ranges and a slightly different design: SHA-224, SHA-256, SHA-384, and SHA-512 (all are sometimes referred to as SHA-2).

SHA-224
One of the SHA-2 algorithms.

SHA-256
One of the SHA-2 algorithms.

SHA-384
One of the SHA-2 algorithms.

SHA-512
One of the SHA-2 algorithms.

single-length key
A key that is 64 bits long. A key can be single- or double-length. A double-length key is 128 bits long.

smart card
A plastic card that has a microchip capable of storing data or process information.

supervisor state
A state during which a processing unit can execute input/output and other privileged instructions. (D)

Systems Network Architecture (SNA)
An architecture that describes logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks. Note: The layered structure of SNA allows the ultimate origins and destinations of information, that is, the end users, to be independent of and unaffected by the specific SNA network services and facilities used for information exchange.

TLV
A widely used construct, Tag, Length, Value, to render data self-identifying. For example, such constructs are used with EMV smart cards.

token
In a Local Area Network, the symbol of authority passed successively from one data station to another to indicate the station is temporarily in control of the transmission medium. (I) A string of characters treated as a single entity.

Transaction Security System (TSS)
An IBM product offering including both hardware and supporting software that provides access control and basic cryptographic key-management functions in a network environment. In the workstation environment, this includes the 4755 Cryptographic Adapter, the Personal Security Card, the 4754 Security Interface Unit, the Signature Verification feature, the Workstation Security Services Program, and the AIX Security Services Program/6000. In the host environment,
this includes the 4753 Network Security Processor and the 4753 Network Security Processor MVS™ Support Program.

**transport key**
A 128-bit key used to protect keys distributed from one system to another. A transport key can either be an exporter key-encrypting key, an importer key-encrypting key, or an ANSI key-encrypting key.

**transport key variant**
A key derived from a transport key by use of a control vector. It is used to force separation by type for keys sent between systems.

**Unique key per transaction (UKPT)**
A cryptographic process that can be used to decipher PIN blocks in a transaction.

**user-exit routine**
A user-written routine that receives control at predefined user-exit points.

**verb**
A function that has an entry-point-name and a fixed-length parameter list. The procedure call for a verb uses the standard syntax of a programming language.

**verification pattern**
An 8-byte pattern calculated from the key parts you enter when you enter a master key or clear key. You can use the verification pattern to verify that you have entered the key parts correctly and specified a certain type of key.

**virtual machine (VM)**
A functional simulation of a computer and its associated devices. Each virtual machine is controlled by a suitable operating system. VM controls concurrent execution of multiple virtual machines on one host computer.

**VISA**
A financial institution consortium that has defined four PIN block formats and a method for PIN verification.

**VISA PIN Verification Value (VISA PVV)**
An input to the VISA PIN verification process that, in practice, works similarly to a PIN offset.

**3621**
A model of an IBM Automatic Teller Machine that has a defined PIN block format.

**3624**
A model of an IBM Automatic Teller Machine that has a defined PIN block format and methods of PIN calculation.

**4753**
The Network Security processor. The IBM 4753 is a processor that uses the Data Encryption Algorithm and the RSA algorithm to provide cryptographic support for systems requiring secure transaction processing (and other cryptographic services) at the host computer. The NSP includes a 4755 cryptographic adapter in a workstation which is channel attached to a main frame computer.
adapter domain
  migration mode 1090
adapter ID
coprocessor 102
adapter serial number 102
ADAPTER1 102
ADD-PART 237, 238, 241
ADJUST 175, 246, 254
administration of CCA master key
  methods or tools 1119
AES 168, 212, 242, 250, 260, 302, 305,
  345, 354, 359, 365
Key Token Build2 265
AES CIPHER keys
definition 41
AES CIPHER variable-length
  symmetric 904
AES encrypted OPK section
  4096-bit Chinese Remainder Theorem
    format 869
RSA private key 869
AES encryption algorithm 392, 399
AES EXPORTER and IMPORTER
  variable-length symmetric 927
AES internal fixed-length key-token
  flag byte 848
AES key 16
  managing 165
  translation 16
  variable-length 304
AES Key Record Create
  (CSNBAKRC) 455
    format 456
    JNI version 457
    parameters 456
    related information 457
    required commands 457
    restrictions 457
AES Key Record Delete
  (CSNBAKRD) 457
    format 458
    JNI version 459
    parameters 458
    related information 459
    required commands 459
    restrictions 458
AES Key Record List (CSNBAKRL) 459
    format 460
    JNI version 461
    parameters 460
    related information 461
    required commands 461
AES Key Record Read
  (CSNBAKRR) 462
    format 462
    JNI version 463
    parameters 462
    related information 463
    required commands 463
    restrictions 463
AES Key Record Write
  (CSNBAKRW) 464
    format 464
    JNI version 466
    parameters 464
    related information 465
    required commands 465
    restrictions 465
AES key storage 150, 449
AES key storage files 51
AES key wrapping 37
AES key-encrypting key 221
AES MAC 430
AES MAC variable-length
  symmetric 912
AES master key 41, 221
AES PINPROT, PINCALC, and PINPRW
  variable-length symmetric 938
AES PKA Master Key (APKA-MK) 71
AES transport key 41
AES verb 61
AES-MK 41, 83, 158, 246, 254
AES, DES, and HMAC cryptography and
  verbs 33
AES, DES, and HMAC functions 33
AESDATA 41, 212
AESKW 345, 365
AESKW wrapping 40
AESKWCV 250, 365
AESTOKEN 41, 212
algorithm
  3624 PIN generation 1010
  3624 PIN verification 1013
  DES 33
  ECDSA 71
  GBP PIN generation 1011
  GBP PIN verification 1015
  GBP-PIN 556
  IBM-PIN 556
  IBM-PINO 556
  Interbank PIN generation 1019
  PIN offset generation 1012
  PIN, detailed 1010
  PIN, general 56
  PKA 71
  PVV generation 1018
  PVV verification 1019
  RSA 71
  VISA PIN 1017
  VISA-PVV 519, 556
  VISAPV4 556
algorithm parameter
  TR31 Key Token Parse verb 805
algorithms
  supported for the CCA token 1144
AMEX-CSC 41, 170, 260, 308
ANSI 9.9-1 algorithm 417
ANSI X3.106 1027
ANSI X3.106(CBC) 1028
ANSI X9.102 40
ANSI X9.19 426
C API 1139, 1144
C applications
   using CCA libraries 25
   c_variable_encrypting_key_identifier parameter
   Cryptographic Variable Encipher verb 178
calculation method
   MAC padding method 1033
   Message Authentication Code (MAC) 1033
   Modification Detection Code (MDC) 1024
   X9.19 method 1033
callable service 19
Cardholder Name 502
CBC processing rule 380, 382, 387, 392, 399
   Decipher 381
   Encrypt 385
   Symmetric Algorithm Decipher 390
   Symmetric Algorithm Encipher 397
CBC wrapping of DES keys
   enhanced CBC 37
CCA
API 3
   application programming 1
   Common Cryptographic Architecture 1
   functional overview 4
   functions 33
   programming 3
   service request 8
   software support 4
CCA access control 4
CCA API 9
CCA API build date 102
CCA API version 102
CCA application
   compile 24
   Java 28
   link 24
   CCA coprocessor mode xvii
   CCA DEB
   contained files 1109
   groups 1111
CCA DEB (continued)
   installation 1108
   samples 1110
   uninstall 1116
CCA DES-key verification 1023
CCA error log 102
CCA installation 1107
CCA JNI 26
calling 26
CCA libraries
   C applications 25
   CCA library 19
   functions 1144
   location 8
   restrictions 1146
CCA management 3
CCA master key 4
   administration 1119
   methods or tools 1119
CCA master key migration 1148
CCA node key management verbs 58
CCA nodes and resource control verb 89
CCA nodes and resource control verbs 59
CCA programming 19
CCA RPM
   contained files 1109
   groups 1111
   install and configure 1111
   installation 1108
   samples 1110
   uninstall 1116
CCA sample program 1093
   C 1093
   Java 1097
CCA security API 4
CCA services
   base 102
CCA system setup 1103
CCA token
   configuring 1140
   status information 1143
   supported mechanisms 1144
   using 1144
CCA verb 3, 61, 87
CCA verb description 4
Central Processor Assist for
   Cryptographic Functions (CPACF) xxii, 14, 1106
certificate section
   PKA public-key 885
   CEX/C xx, xxii, 3, 4, 11, 16, 18, 57, 71, 72,
   145, 304, 454, 1106, 1117
   data protection 33
   CEX/C feature coexistence 1117
   CEX/C xvii, 101
   CEX/C xxii, 16, 102, 495
   CEX/C xvii, xxi, 101
   CEX/C xvii, xxi, 101
   CEX/C information 1117
   CEX/C xvii, xxi, 101
chain_data parameter
   Symmetric Algorithm Decipher verb 392
   Symmetric Algorithm Encipher verb 399

B

base CCA services 102
basic structure of a role 1073
battery indicator 102
battery-backed RAM
   size 102
BCD 509, 639, 650, 653, 659, 663, 669
   becoming compliant-tagged
   key tokens 1090
binding
   PAN to the PIN 1085
   binding key halves 37
   binding PAN to the PIN 1085
block chaining 380
block_size parameter
   Symmetric Algorithm Decipher verb 392
   Symmetric Algorithm Encipher verb 399
Brainpool
   elliptic curve type 195
Byte * 26
chain_data_length parameter
Symmetric Algorithm Decipher verb 392
Symmetric Algorithm Encipher verb 399
chaining_vector parameter
Decipher verb 382
EC Diffie-Hellman verb 200
Encipher verb 387
HMAC Generate verb 420
HMAC Verify verb 423
MAC Generate verb 427
MAC Verify verb 434
MDC Generate verb 442
One-Way Hash verb 445
chaining_vector_length parameter
EC Diffie-Hellman verb 200
HMAC Generate verb 420
HMAC Verify verb 423
One-Way Hash verb 445
changing control vectors 1000
changing the master key panel.exe 1128
CHECK 464, 483
Chinese Remainder Theorem 327, 694, 699, 700, 716, 854, 855
chzcrypt, command 1105
CIPHER 170, 184, 190, 209, 212, 235, 260, 308
cipher block chaining 37
cipher block chaining (CBC) 379
Cipher Block Chaining (CBC) 178
CIPHER key type 41
Cipher Text Translate2
usage notes 433
Cipher Text Translate2 (CSNBCTT2) 404
format 406
JNI version 414
parameters 406
required commands 410
restrictions 410
usage notes 411
cipher_text parameter
Cryptographic Variable Encipher verb 178
Decipher verb 382
Encipher verb 387
ciphering methods 1026
ciphertext 178
deciphering 379
ciphertext parameter
Symmetric Algorithm Decipher verb 392
Symmetric Algorithm Encipher verb 399
ciphertext_length parameter
Symmetric Algorithm Decipher verb 392
Symmetric Algorithm Encipher verb 399
Clear Key Import (CSNBCKI) (continued)
format 166
JNI version 167
parameters 166
required commands 167
Clear PIN Encrypt (CSNBCPE) 512
format 512
JNI version 514
parameters 512
required commands 514
restrictions 514
Clear PIN Generate (CSNBPGN) 515
format 515
JNI version 518
parameters 515
related information 518
required commands 517
usage notes 518
Clear PIN Generate Alternate
(CSNBCPA) 518
extraction rules 1009
format 518
JNI version 522
parameters 519
required commands 522
clear_key parameter
Clear Key Import verb 166
Multiple Clear Key Import verb 168
clear_key_bit_length parameter
Key Generate2 verb 222
clear_key_length parameter
Multiple Clear Key Import verb 168
clear_key_value parameter
Key Token Build2 verb 265
clear_master_key parameter
Key Storage Initialization verb 150
clear_PIN parameter
Clear PIN Encrypt verb 512
clear_text parameter
Decipher verb 382
Encipher verb 387
Secure Messaging for Keys verb 596
Secure Messaging for PINs verb 599
CLEARPIN 599
cleartext parameter
Symmetric Algorithm Decipher verb 392
Symmetric Algorithm Encipher verb 399
cleartext_length parameter
Symmetric Algorithm Decipher verb 392
Symmetric Algorithm Encipher verb 399
CLR-A128 246
CLR-A192 246
CLR-A256 246
CLR-ENC 170, 184, 190, 260, 308
CMAC 430
CMK 452
current master key 454
CMK status 102
CMK status, AES 102
CMK status, ECC 102
Code Conversion parameters 817
Code Conversion (CSNBXEA) 817
format 817
required commands 819
restrictions 819
usage notes 819
Code Conversion (CSNBXEAJ)
JNI version 821
coefficients of CEX/C features 1117
combinations of verbs 57
command codes
security API 1135
Common Cryptographic Architecture
API 3
application programming 1
CCA 1
service request 8
Common Cryptographic Architecture
description xx
Common Cryptographic Architecture library
location 8
common parameters 22
COMPLETE 238
COMPLETE 237, 238, 241
compliance state of adapter
querying with panel.exe 1132
compliance warnings 32, 1086
compliant-tagged key tokens
PIN block translation 1085
concurrent installations 1117
configuration file
opencryptoki.conf 1140
contact IBM xxviii
contents of the DEFAULT role 1077
CONTINUE 406
continue processing rule 382, 387, 392, 399
collection information
for Cipher Text Translate2 406
for Clear PIN Encrypt 512
for Clear PIN Generate 515
for Control Vector Generate 170
for Control Vector Translate 175
for CVV Generate 523
for CVV Verify 532
do Decrypt 382
do Digital Signature Generate 676
do Digital Signature Verify 682
do Diversified Key Generate 184
for Diversified Key Generate2 190
for Encipher 387
for Encrypted PIN Generate 536
for Key Part Import 238
for Key Test 246
for Key Test Extended 256
for Key Token Change2 305
for MAC Generate 427
for MAC Verify 434
for MAC Verify2 438
for MDC Generate 442
for Multiple Clear Key Import 168
for One-Way Hash 445
for PIN Change/Unblock 586
for PKA Decrypt 327
for PFA Encrypt 331
<table>
<thead>
<tr>
<th>Control Information</th>
<th>CPACF Functions</th>
<th>CPACF (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>for PKA Key Import</td>
<td>170</td>
<td>AUTOSELECT option</td>
</tr>
<tr>
<td>for PKA Key Record</td>
<td></td>
<td>clear key 16</td>
</tr>
<tr>
<td>Delete</td>
<td></td>
<td>default card 19</td>
</tr>
<tr>
<td>for PKA Key Record</td>
<td></td>
<td>disabling for clear key 15</td>
</tr>
<tr>
<td>Write</td>
<td></td>
<td>disabling for protected key 15</td>
</tr>
<tr>
<td>for PKA Key Token</td>
<td></td>
<td>environment variable 15</td>
</tr>
<tr>
<td>Build</td>
<td></td>
<td>illustration 16</td>
</tr>
<tr>
<td>Token Build</td>
<td></td>
<td>preparation at startup 19</td>
</tr>
<tr>
<td>700</td>
<td></td>
<td>protected key 16, 19</td>
</tr>
<tr>
<td>for Secure Messaging</td>
<td></td>
<td>using protected key 18</td>
</tr>
<tr>
<td>for Keys</td>
<td></td>
<td>CPACF functions 18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPACF service action 16</td>
</tr>
<tr>
<td>for Secure Messaging</td>
<td></td>
<td>CPINENC 170, 260, 308</td>
</tr>
<tr>
<td>for PKI Import</td>
<td></td>
<td>CPINGEN 170, 260, 308</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPINGENA 170, 260, 308</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Crypto Express4 feature xvii</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Crypto Express5 feature xvii</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Crypto Express6 feature xvii</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cryptographic device driver installing 1103</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cryptographic engine 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cryptographic Facility Query</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(SUACFQ) xxi, 11, 100, 102, 117</td>
</tr>
<tr>
<td></td>
<td></td>
<td>format 101</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JNI version 142</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameters 102</td>
</tr>
<tr>
<td></td>
<td></td>
<td>required commands 142</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restrictions 142</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cryptographic Facility Version</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(SUACFV) 143</td>
</tr>
<tr>
<td></td>
<td></td>
<td>format 143</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JNI version 144</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameters 143</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restrictions 143</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cryptographic key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AES, DES, and HMAC functions 33</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cryptographic Resource Allocate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(SUACRA) 11, 15, 144</td>
</tr>
<tr>
<td></td>
<td></td>
<td>format 145</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JNI version 146</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameters 145</td>
</tr>
<tr>
<td></td>
<td></td>
<td>usage notes 146</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cryptographic Resource Deallocate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(SUACRD) 11, 15, 147</td>
</tr>
<tr>
<td></td>
<td></td>
<td>format 147</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JNI version 149</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameters 147</td>
</tr>
<tr>
<td></td>
<td></td>
<td>usage notes 149</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cryptographic services access layer 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cryptographic Unit Support Program</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(CUSP) Decipher 381</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Encipher 385</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cryptographic Variable Encipher</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(CSNBCVE) 178</td>
</tr>
<tr>
<td></td>
<td></td>
<td>format 178</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JNI version 179</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameters 178</td>
</tr>
<tr>
<td></td>
<td></td>
<td>required commands 179</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restrictions 179</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cryptography publications xxvi</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cryptoki 1139</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSC-3 604</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSC-345 604</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSC-4 604</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSC-5 604</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSC-V1 604</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSC-V2 604</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBKRC (AES Key Record Create) 455</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBKRCG 457</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBKRD (AES Key Record Delete) 457</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBKRDJ 459</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBKRL (AES Key Record List) 459</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBKRLJ 461</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBKRR (AES Key Record Read) 462</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBKRRJ 463</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBKRW (AES Key Record Write) 464</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBKRWJ 466</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBAPG (Authentication Parameter Generate) 508</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBPGJ 511</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBCKC (CVV Key Combine) 526</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBCKCJ 531</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBCKJ (Clear Key Import) 166</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBCKIJ 167</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBCKKM (Multiple Clear Key Import) 167</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBCKMJ 169</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNCBPPA (Clear PIN Generate Alternate) 518</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNCBPPAJ 522</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNCBCE (Clear PIN Encrypt) 512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNCBCEJ 514</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNCBG (CVV Generate) 523</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNCBGJ 526</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNCBSV (CVV Verify) 531</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNCBSVJ 534</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBCTT2 414</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBCTT2 (Cipher Text Translate2) 404</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBCVE (Cryptographic Variable Encipher) 178</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNCVEJ 179</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNCVGV (Control Vector Generate) 170</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNCVGJ 174</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNCVLT (Control Vector Translate) 174</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNCVJT 177</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBDDPG (DK Deterministic PIN Generate) 610</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBDDPGJ 616</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBDEC (Decipher) 381</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBDECJ 385</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBDDKG (Diversified Key Generate) 183</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBDDKG2 (Diversified Key Generate2) 189</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBDDKGGJ 194</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBDDKGJ 189</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBDDKM (Data Key Import) 182</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBDDKMJ 183</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBDDKX (Data Key Export) 180</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBDDXJ 181</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBDDMP (DK Migrate PIN) 617</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBDDMPJ 622</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBDDPC (DK PIN Change) 637</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBDDPCG (DK PRW CMAC Generate) 659</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSNBDDPCGJ 662</td>
</tr>
</tbody>
</table>

**Control Information:**
- PKA Key Import
- PKA Key Record Delete
- PKA Key Record Write
- PKA Key Token Build
- Secure Messaging for Keys
- Secure Messaging for PINs
- Symmetric Algorithm
- Decipher
- Symmetric Key Export
- Symmetric Key Generate
- Symmetric Key Import
- Transaction Validation
- Trusted Block Create

**Control Vector:**
- Control Vector Generate (CSNBVCV)
- Control Vector Translate (CSNBVCT)
- Control Vector Keyword Combinations
- Control Vector Length
- Control Vector Table
- Control Vector Parameter
- Control Vector Generate Verb
- Key Token Build Verb
- Key Token Parse Verb
- Control-vector-base bit maps

**Coprocessor:**
- Battery Indicator
- CCA Error Log
- Intrusion Latch
- Last Five Commands
- Number of 102
- Power-supply Voltage
- Radiation
- Tampering
- Temperature
- Coprocessor adapter ID
- Coprocessor Certification
- Coprocessor EC Level
- Coprocessor Mode
- Coprocessor Part Number
- Coprocessor POST2 Version
- Coprocessor Resource Selection
- Coprocessor Selection
- Coprocessor Serial Number
- CPACF

**Usage Notes:**
- CPACF functions
- CPACF service action
- CPINENC
- CPINGEN
- CPINGENA
- Crypto Express4 feature
- Crypto Express5 feature
- Crypto Express6 feature
- Cryptographic device driver
- Installing
- Cryptographic engine
- Cryptographic Facility Query
- Cryptographic Facility Version
- Cryptographic Resource Allocate
- Cryptographic Resource Deallocate
- Cryptographic Unit Support Program
- Cryptographic Variable Encipher
- Cryptography publications
CSNDPKE (PKA Encrypt) 330
CSNDPKJ 334
CSNDPKG
  parameters 691
CSNDPKG (PKA Key Generate) 689
CSNDPKJ 695
CSNDPKR (PKA Key Import) 696
CSNDPKR 699
CSNDPKT (PKA Key Translate) 713
CSNDPKTJ 719
CSNDPKX (PKA Public Key Extract) 720
CSNDPKXJ 721
CSNDRKD (Retained Key Delete) 485
CSNDRKDJ 487
CSNDRKLI (Retained Key List) 487
CSNDRKLIJ 489
CSNDRKXX (Remote Key Export) 722
CSNDRKXXJ 733
CSNDSXD (Symmetric Key Export with Data) 350
CSNDSXDJ 353
CSNDSYI (Symmetric Key Import) 358
CSNDSYI2 (Symmetric Key Import2) 362
CSNDSYI2J 369
CSNDSYIJ 362
CSNDSYXY (Symmetric Key Export) 344
CSNDSYXJ 349
CSNDTBC ( Trusted Block Create) 734
CSNDTBCJ 737
CSU_CMP_WARN_MODE 32, 1086
CSU_DEFAULT_ADAPTER 145, 147
CSU_DEFAULT_DOMAIN 10
CSU_EC_CHECKCURVE 680
CSU_HCPUTACLOR 15, 444
affected verbs 15
CSU_HCPUTAPRTJ 15
affected verbs 15
CSUAAAC
  required commands 92
usage notes 93
CSUAAAC (Access Control Maintenance) 89
CSUAAACMJ 93
CSUAAAC
  required commands 100
CSUAAACT (Access Control Tracking) 93
CSUAAACTJ 100
CSUAAACQ
CSUAAFCQ
  STATOAHL 102
  STATOAHL output data 133
CSUAAFCQ (Cryptographic Facility Query) 100
CSUAAFCQJ 142
CSUAAAFCV (Cryptographic Facility Version) 143
CSUAAFVJ 144
CSUAAACRA (Cryptographic Resource Allocate) 144
CSUAAACRAJ 146
CSUAAACRD ( Cryptographic Resource Deallocate) 147
CSUAAACRDJ 149
CSUAESDS 450, 455, 457, 459, 462, 464
CSUAESLD 460
CSUALGQ (Log Query) 152
CSUALGQJ 157
CSUARNT (Random Number Tests) 161
CSUARNTJ 163
CSUCACHE 9
CSUDEDS 450, 466, 467, 469, 471, 473
csulinc.h 24
CSUPKADS 450, 474, 476, 478, 481, 482
current master key
CMK 454
current_reference_PIN_data parameter
PIN Change/Unblock verb 586
current_reference_PIN_block parameter
PIN Change/Unblock verb 586
current_reference_PIN_key profile parameter
PIN Change/Unblock verb 586
current_reference_PIN_profile parameter
PIN Change/Unblock verb 586
current-key serial number
CSNBDKM 545
CSUPINE process ruling 382, 387
CV 260, 308
cv_source parameter
TR31 Key Import verb 785
CV-KEK 260
CVARDIC 41, 170, 212, 260, 308
CVAENC 41, 170, 178, 212, 260, 308
CVAENThe restrictions for .C 212
CVARPINE 41, 170, 212, 260, 308
CVARCVI 41, 170, 175, 212, 260, 308
CVARCVIR 41, 170, 175, 212, 260, 308
CVGEN (CSNBDG) 523
format 523
JNI version 526
parameters 523
required commands 525
CVV Key Combine (CSNBDKCC) 526
format 528
parameters 528
required commands 530
restrictions 530
CVV Key Combine (CSNBDKCCJ)
JNI version 531
CVV Verify (CSNBDCSV) 531
format 532
JNI version 534
parameters 532
required commands 534
CVV_key_A_Identifer parameter
CVV Generate verb 523
CVV Verify verb 532
CVV_key_B_Identifer parameter
CVV Generate verb 523
CVV Verify verb 532
CVV_value parameter
CVV Generate verb 523
CVV Verify verb 532
CVV-1 523, 532
CVV-2 523, 532
CVV-3 523, 532
CVV-4 523, 532
CVVKEY-A 41, 170, 260, 308
CVVKEY-B 41, 170, 260, 308

D
daemon
openCrypTokni 1141
DALL 170, 260, 308
data
deciphering 381
enciphering 385
protecting 379
DATA 41, 167, 170, 182, 184, 190, 209, 212, 234, 235, 260, 308, 322, 344, 354, 358, 382, 985
data confidentiality 3
data integrity 3
managing 55
verifying 417
data key
export 180
import 182
importing 166
re-encryption 180
Data Key Export (CSNBDX) 180
format 180
JNI version 181
parameters 180
required commands 181
restrictions 181
Data Key Import (CSNBDK) 182
format 182
JNI version 183
parameters 182
required commands 183
restrictions 182
usage notes 183
data parameter
Diversified Key Generate verb 184
Diversified Key Generate2 verb 190
data structure example of an ACP list 1078
data structures
  access control point list 1075
data_array parameter
Clear PIN Generate Alternate verb 519
Clear PIN Generate verb 515
Encrypted PIN Generate verb 536
Encrypted PIN Verify verb 536
data_length parameter
Diversified Key Generate verb 184
Diversified Key Generate2 verb 190
data structure parameter
PKA Decrypt verb 327
PKA Encrypt verb 331
data_structure_len parameter
PKA Decrypt verb 327
PKA Encrypt verb 331
data-encrypting key
definition 41
generating 985
length 41
DATAC 41, 170, 184, 190, 209, 235, 260, 308, 985
DATAM 41, 170, 184, 190, 209, 211, 212, 235, 260, 308, 985
DATAMV 41, 170, 184, 190, 209, 212, 235, 260, 308
dataset_name parameter
AES Key Record List verb 460
E

EBCDIC conversion 817
EC Diffie-Hellman
  key agreement models 52
EC Diffie-Hellman (CSNDEDH) 195
  format 200
  JNI version 208
  parameters 200

EC Diffie-Hellman (CSNDEDH)
  (continued)
    required commands 206
    restrictions 206
EC level
  coprocessor 102
  ECB processing rule 392, 399
  ECB wrapping of DES keys 37
  ECC 696, 710
  Elliptic Curve Cryptography 195
  ECC key 675, 680
  ECC key token 874, 881
  ECC-PAIR 700
  ECC-PUBL 700
  ECC-VER 700
  ECC-VER1 700
  ECDH protocol 195
  ECDSA 71, 675, 676, 680, 682
  ECDSA algorithm 71
  ECI-1 545
  ECI-2 PIN block format 498, 1008
  ECI-3 PIN block format 498, 1008
  ECI-4 545
  edition 2016 xix
  edition 2017
    second edition for CCA 5.2 xix
  edition 2018 xvii
  electronic code book (ECB) 380, 1036
  Electronic Code Book (ECB)
    Symmetric Algorithm Decipher 390
    Symmetric Algorithm Encipher 397
  Elliptic Curve Cryptography 195
  Elliptic Curve Cryptography (ECC) 23, 41, 151
  key token 71, 72, 73, 75, 150, 452,
    675, 676, 680, 689, 699, 700, 880
  Elliptic Curve Diffie-Hellman 195
  Elliptic Curve Digital Signature
    Algorithm (ECDSA) 71, 676, 682
  elliptic curve domain parameters 195
  elliptic curve type
    Brainpool 195
    Prime 195
  EMV 2000 184, 190
  EMV MAC smart card standard 1034
  EMVCRT 716
  EMVDDA 716
  EMVDDAE 716
  EMVMAC 427, 434
  EMVMACD 427, 434
  enable ACPs 1129
  ENC-ZERO 245, 246, 250, 254
  ENCRYPT 41, 170, 209, 212, 235, 260, 308
  Encipher (CSNBNENC) 385
    format 387
    JNI version 389
    parameters 387
    required commands 389
    restrictions 389
  Encipher processing rule 385
  enciphered_key parameter
    Symmetric Key Export verb 345
    Symmetric Key Import2 verb 365
  enciphered_key_length parameter
    Symmetric Key Export verb 345

Encrypted PIN Generate
  (CSNBEPG) 535
    format 535
    JNI version 539
    parameters 536
    required commands 538
    restrictions 538

Encrypted PIN Translate
  (CSNBPT) 495, 539
    extraction rules 1009
    format 540
    JNI version 545
    parameters 540
    required commands 543
    usage notes 545

Encrypted PIN Verify
  (CSNBPR) 555
    extraction rules 1008
    format 556
    JNI version 560
    parameters 556
    related information 560
    required commands 559
    encrypted_PIN_block parameter
      Clear PIN Encrypt verb 512
      Clear PIN Generate Alternate
        verb 519
      Encrypted PIN Generate verb 536
      Encrypted PIN Verify verb 556

encryption algorithm processing rule
  AES 392, 399
  DES 382, 387
  encryption_master_key parameter
    PIN Change/Unblock verb 586
  ENH-ONLY 168, 170, 184, 190, 260, 302,
    308, 323, 354, 359, 365
  enhanced PIN security mode 500
  entry point 20
  entry point name
    prefix 20
  entry-point names 19

Environment Identifier (EID) 102
  environment variable 1109

CSU_DEF 145, 147
CSU_DEFAULT_ADAPTER 10
CSU_ECHECKCURVE 680
CSU_HCPUIACLR 15, 444
CSU_HCPUIAPRT 15
CSUAESDS 450, 457, 459, 462, 464
CSUAESLD 460
CSUCACHE 9
CSUDDES 450, 455, 466, 467, 469,
  471, 473

Duhttom Adtheretogu Applotem Pogrammorer's Guide
environment variable (continued)
CSUPKADS 450, 474, 476, 478, 481, 482
key storage 1117
PATH 27
EPB 611, 618
EPINGEN 170, 260, 308
EPINGEN 41, 170, 260, 308
EPINVER 170, 260, 308
epoch certificate
panel.exe 1132
establishing master keys 84
Europay padding rule 426
EVEN 341, 343
even parity 237, 341, 343
EX 212, 354
EX key form 985
EXEX 170, 212, 260, 308
EXEX key form 987
exit_data parameter 22
exit_data_length parameter 22
expiration_date parameter
CVV Generate verb 523
CVV Verify verb 532
EXPORT 170, 260, 308
exportability parameter
TR31 Key Token Parse verb 805
Exportable (EX) key form 229
exportable key
generating 985
exportable key form 34, 212
EXPORTER 41, 170, 209, 212, 235, 260, 308, 321
exporter key encrypting key
any DES key 209
exporter_key-encrypting key 180
exporter_key_identifier parameter
Data Key Export verb 180
Key Export verb 209
exporting DES keys 722
EXTDWAK 716
EXTERNAL 260, 308
external key 254
external key token 23, 35, 74, 968
DES 850
PKA
RSA private 854
verbs 35
external RKX DES key tokens 851
extraction rules, PIN 1008

F

FCV
function control vector 102, 347, 1151, 1163
feature coexistence 1117
files
hikmNativeNumber.html 26
key storage 52
financial services support for DK 609
financial services verbs 491
FIPS-RNT 162
FIRST 237, 238, 241, 420, 423, 427, 434, 442, 445
fixed-length key token
AES, internal token 847
flash EPROM memory
size 102
form parameter
Random Number Generate verb 341
format control 500
formats
PIN 56
formatting hashes and keys 1044
FPE Decipher (CSNBFPED) 560
format 561
JNI version 567
parameters 561
required commands 567
usage notes 567
FPE Encipher (CSNBFPED) 568
format 568
JNI version 574
parameters 568
required commands 574
usage notes 574
FPE Translate (CSNBFPET) 575
format 576
JNI version 582
parameters 576
required commands 582
usage notes 582
function control vector
clear 1077
FCV 102, 347, 1151, 1163
load 1077
functional overview
CCA 4
functions
panel.exe 1124

G

Galois/Counter Mode
GCM 390
GBP-PIN 170, 260, 308, 536, 556
GBP-PIN algorithm 556
GBP-PINo 170, 260, 308
GCM
Galois/Counter Mode 390
general format variable-length
symmetric 889
GENERATE 246, 250, 254, 265, 604
generated_key_identifier parameter
Diversified Key Generate verb 184
Diversified Key Generate2 verb 190
PKA Key Generate verb 691
generated_key_identifier_1 parameter
Key Generate verb 212
Key Generate2 verb 222
generated_key_identifier_2 parameter
Key Generate2 verb 222
Key Generate2 verb 222
generated_key_identifier_2_length
parameter
Key Generate2 verb 222
generated_key_identifier_length
parameter
PKA Key Generate verb 691
generating keys 722

H

hardware requirements xx1
Hardware Security Module (HSM) 53
hash algorithm 16
hash formats
PKC #1 1045
hash formatting 1044
hash message authentication code
HMAC 430
hash parameter
Digital Signature Generate verb 676
Digital Signature Verify verb 682
One-Way Hash verb 445
hash pattern 123, 126, 128, 130
hash_length parameter
Digital Signature Generate verb 676
Digital Signature Verify verb 682
One-Way Hash verb 445
hashing 56
hashing functions 56
hashing method 746
hashing verb 61
HCPUAICLR 145, 147
HCPUPART 145, 147
HEX-8 427, 434
HEX-9 427, 434
HEXDIGIT 519
HEXDIGIT PIN extraction method
keyword 498
High-performance secure AES keys 16
High-performance secure DES keys 16
hikmNativeNumber 26
hikmNativeNumber.html file 26
HMACE 41, 242, 250, 265, 305, 337, 344, 345, 362, 365, 420, 423
hash message authentication
code 430
Key Token Build2 265
HMACE algorithm 37
HMACE Generate (CSNBHMG) 419
format 419
JNI version 422
parameters 420
related information 422
required commands 422
restrictions 422
HMACE key 221
managing 165
variable-length 304
HMACE keys
definition 41
HMACE MAC variable-length
symmetric 920
HMACE verb 61
HMACE Verify (CSNBHMV) 423
HMCAVER 41
host CPU acceleration 382, 386, 426, 434

I
I-CBC 406
I-CLUSP: 390-only 406
I-ECB 406
I-IFS: 390-only 406
I-X923 406
IBM
contacting xxviii
IBM 3624 515, 555
IBM 4700 Encrypting PINPAD 1007
IBM Crypto Express3 feature xxi
IBM Crypto Express4 feature xxi
IBM Crypto Express5 feature xxi
IBM Crypto Express6 feature xxi
IBM GBP 555
IBM-PIN 170, 260, 308, 536, 556
IBM-PIN algorithm 556
IBM-PINO 170, 260, 308, 519, 556
IBM-PINO algorithm 556
ICV
initial chaining value 390
ICV selection processing rule continue 382, 387, 392, 399
initial 382, 387, 392, 399
identifying key tokens for conversion 1089
IKEYXLAT 41
IKEK-AES 250, 696
IKEK-DES 250, 696
IKEK-PKA 250
IKEK-AES 406
IKEK-DES 406
IKEYXLAT 41, 170, 209, 212, 235, 260, 308, 321
IM 212, 354
IM key form 985
IMEX 170, 212, 260, 308
IMEX key form 987
IMIM 170, 212, 260, 308
IMP-PKA 41, 209
IMPORT 170, 260, 308
Importable (IM) key form 229
importable key generating 985
importable key form 34, 212
IMPORTER 41, 170, 182, 209, 212, 235, 260, 308, 321
importer key-encrypting key 41
importer_key_identifier
PKA Key Import verb 696
importer_key_identifier parameter
Data Key Import verb 182
Key Import verb 235
INACTIVE 735
INBK PIN 515

INBK-PIN 170, 260, 308, 536, 555, 556
Information Protection System (IPS)
Decipher 381
Encipher 385
INITIAL 406
initial chaining value ICV 390
initial processing rule 382, 387, 392, 399
initialization_vector parameter
Cryptographic Variable Encipher verb 178
Decipher verb 382
Encipher verb 387
Secure Messaging for Keys verb 596
Secure Messaging for PINs verb 599
Symmetric Algorithm Decipher verb 392
Symmetric Algorithm Encipher verb 399
initialization_vector_length parameter
Symmetric Algorithm Decipher verb 392
Symmetric Algorithm Encipher verb 399
initializing key storage 149
input_block_identifier parameter
Trusted Block Create verb 735
input_block_identifier_length parameter
Trusted Block Create verb 735
input_KEY_key_identifier parameter
Key Translate2 verb 323
input_KEY_key_identifier parameter
Key Translate verb 321
input_KEY_key_identifier_length
Key Translate2 verb 323
input_key_identifier parameter
Secure Messaging for Keys verb 596
input_key_token
Key Translate2 verb 323
input_key_token parameter
Key Translate verb 321
input_key_token_length
Key Translate2 verb 323
input_PAN_data parameter
Secure Messaging for PINs verb 599
input_PIN block parameter
Encrypted PIN Translate verb 540
Secure Messaging for PINs verb 599
input_PIN_encrypting_key_identifier parameter
Encrypted PIN Translate verb 540
Encrypted PIN Verify verb 556
input_PIN_profile parameter
Encrypted PIN Translate verb 540
Encrypted PIN Verify verb 556
Secure Messaging for PINs verb 599
installation directory 1108
installation instructions 1107
INTDWAKW 716
Interbank PIN 61, 494, 515, 555
intermediate PIN-block (IPB) 1006
INTERNAL 260, 265, 308
internal key token 23, 74
AES 847
AES, fixed-length 847
definition 35
DES 847, 849
internal key token (continued)
PKA
RSA private 855, 872, 873
intrusion latch 102
IPB (intermediate PIN-block) 1006
IPINENC 41, 170, 209, 212, 235, 260, 308
IPINENC key type 540
IPKCPAD 406
IPS processing rule 382, 387
ISO 16609 TDES MAC 1035
ISO 9796 682
ISO 9796-1 675
ISO format 0 497, 1005
ISO format 1 497, 1006
ISO format 2 1006
ISO format 3 1006
ISO format 4 497
ISO-O PIN block format 498, 1005
ISO-1 PIN block format 498, 1006
ISO-2 PIN block format 498, 1006
ISO-3 PIN block format 498, 1006
ISO-9796 676
ITER-38 691
ivp.e 1119

J
Java
data types 26
entry point names 26
environment 27
tested versions 27
Java byte code running 28
Java interaction 3
Java native interface 26
Java Native Interface 3
JNI 27
Java Native Interface (JNI) 28
byte code 28
JNI 3, 26
Java Native Interface sample code 27
sample modules 27

K
KAT 162
KDF in counter mode 37
kek_key_identifier parameter
Control Vector Translate verb 175
Key Test Extended verb 256
KEK_key_identifier parameter
Prohibit Export Extended verb 336
KEK_key_identifier_1 parameter
Key Generate verb 212
KEK_key_identifier_2 parameter
Key Generate verb 212
key
AES master key 41
AES transport 41
asymmetric master key 41
CIPHER 41
clear 16, 18
clear key 41
control vector 34, 40

1182 Common Cryptographic Architecture Application Programmer’s Guide
key (continued)
data key export 180  
importing 166  
re-enciphering 180  
DECIPHER 41  
DES exporter key-encrypting 41  
DES transport 41  
double length 37  
double-length 986, 987  
ENCIPHER 41  
generating encrypted 212  
HMAC 41  
key-encrypting 41  
MAC 41  
master 16  
multiple decipherment/encipherment 1036  
NOCV importers and exporters 41  
pair 986, 987  
parity 166  
PIN 41  
PIN-encrypting key 539  
protected 16, 19  
protecting data 379  
re-enciphering 234  
re-enciphering 209  
single-length 985, 986  
symmetric master key 41  
translated 16, 18  
triple length DES 37  
VISA PVV 518  

KEY 260, 308  
key agreement models  
EC Diffie-Hellman 52  
key bundling requirements 79  
key cache 9  
key cache, host side 9  
key completeness 140  
Key Derivation Function (KDF) 37  
key derivation wrapping 37  
key encrypting key 212, 234, 321, 323, 335, 722  
distribution 53  
exporter 209  
new 16  
key encrypting key variant  
definition 34  
key export 337  
Key Export (CSNBKEX) 209  
format 209  
JNI version 211  
parameters 209  
required commands 210  
restrictions 210  
usage notes 211  

Key Export to TR31 (CSNBT31X) 755  
format 757  
JNI version 781  
parameters 757  
required commands 778  
restrictions 778  
key form 212, 985  
combinations for a key pair 218  
combinations with key type 218  
exportable 34  

key form (continued)  
exportable 34  
key form bits 995  
key formats 847  
key formatting 1044  
key forms  
Exportable (EX) 229  
Importable (IM) 229  
Operational (OP) 229  
key functions 18  
Key Generate (CSNBKGN) 212  
format 212  
JNI version 220  
parameters 212  
required commands 218  
usage notes 218  
using 985  
Key Generate2  
key forms 229  
key types 229  

Key Generate2 (CSNBKGN2) 221  
format 222  
JNI version 234  
parameters 222  
required commands 228  
usage notes 229  
key generating key 183  
definition 41  
key identifier 18, 23  
definition 74  
PKA 74  
key identifier parameter  
Clear Key Import verb 166  
Key Import (CSNBKIM) 234  
format 235  
JNI version 237  
parameters 235  
required commands 236  
restrictions 236  
usage notes 237  
key label 8, 23, 74, 178, 209, 449, 450  
key length 132, 140  
key management 3, 53  
PKA 73  
key pair 218  
key pair generation 1035  
key part 157, 158, 254  

Key Part Import (CSNBKIPI) 237  
format 238  
JNI version 240  
parameters 240  
required commands 240  
restrictions 240  
Key Part Import2 (CSNBKPI2) 241  
format 242  
JNI version 245  
parameters 242  
required commands 244  
restrictions 244  
usage notes 244  
key part register 132, 140  
key part register hash 132, 140  
key record 52, 149  
caching 9  
key rule 246  
key separation 33  

key storage 8, 9, 157, 209, 237, 321, 454, 1117  
environment variables 450  
Linux on Z 452  
key storage file 454, 1121  
managing AES and DES 51  
Key Storage Initialization (CSNBKSI) 149  
format 149  
JNI version 151  
parameters 150  
required commands 151  
restrictions 151  
key storage mechanisms 449  
key storage re-encryptor  
pinless 1128  

key subtype  
list 41  
specified by rule_array 41  
Key Test (CSNBKTY) 245  
format 246  
JNI version 249  
parameters 246  
required commands 249  
usage notes 249  
Key Test Extended (CSNBKTYX) 254  
format 256  
JNI version 259  
parameters 256  
required commands 258  
restrictions 258  
usage notes 258  
Key Test2 (CSNBKTY2) 249  
format 250  
JNI version 254  
parameters 250  
required commands 253  
restrictions 253  
usage notes 253  
key token 18, 23, 37, 167, 180, 182, 183, 184, 190, 209, 234, 259, 307, 323, 335, 337, 344, 362, 423, 455, 734, 968  
AES 847  
AES CIPHER variable-length symmetric 904  
AES DESUSECV variable-length symmetric 947  
AES DRYGENKY variable-length symmetric 951  
AES EXPORTER and IMPORTER variable-length symmetric 927  
AES MAC variable-length symmetric 912  
AES SECMSG variable-length symmetric 960  
definition 35  
DES  
external 847, 850  
internal 847  
null 847, 852  
DES internal 849  
ECC 874  
Elliptic Curve Cryptography (ECC) 71, 73, 75, 150, 452, 675, 676, 680, 689, 696, 699, 700, 880  
external 23, 35  

Index 1183
Key Token (continued)

HMAC MAC variable-length symmetric 920
internal 23, 35, 75
null 35
null key token 76
operational 23
PKA 880
PINCALC variable-length symmetric 938
PINPROT variable-length symmetric 938
PINPRW variable-length symmetric 938
PKA 880
null 880
RSA 1024-bit modulus-exponent private 857
RSA 1024-bit private internal 872, 873
RSA 4096-bit modulus-exponent private 861
RSA private 854, 857
RSA private external 854
RSA private internal 855
RSA public 853
variable Modulus-Exponent 873
PKA external 75
symmetric 889
variable-length 889
variable-length symmetric general format 889
verbs 35
Key Token Build (CSNBKTB) 41, 259
format 259
JNI version 264
parameters 260
usage notes 264
Key Token Build2
AES 265
HMAC 265
Key Token Build2 (CSNBKTB2) 265
format 265
JNI version 301
keywords 269
parameters 265
restrictions 300
Key Token Change (CSNBKTC) 301
format 302
JNI version 304
parameters 302
required commands 304
Key Token Change2 (CSNBKTC2) 304
format 305
JNI version 307
parameters 305
required commands 307
restrictions 306
Key Token Parse (CSNBKTP) 307
format 308
JNI version 311
parameters 308
usage notes 311
Key Token Parse2 (CSNBKTP2) 311
format 313
JNI version 320
parameters 313
required commands 320
Key Token Parse2 (CSNBKTP2)
(continued)
usage notes 320
key tokens
becoming compliant-tagged 1090
compliant-tagged 1089
PCI-HSM 2016 compliant restrictions 1084
PKA
number representation 885
Key Translate (CSNBKTR) 321
format 321
JNI version 322
parameters 321
required commands 322
restrictions 322
Key Translate2 (CSNBKTR2) 323
format 323
JNI version 326
parameters 323
required commands 326
restrictions 326
key translation cache 16
key type 23, 34, 170, 212, 234, 985
list 41
key type 1 986
key type 2 986
key types
key usage field
must be equal, KUF-MBE 190
must be permitted, KUF-MBP 190
key verification pattern 157, 245, 249
KVP 250
key wrapping 323
AES 37
definition 37
DES 37
double length key 37
ECB wrapping of DES keys 37
electronic code book 37
enhanced CBC 37
wrapping key derivation 37
key_A_identifier parameter
CVV Key Combine verb 528
key_B_identifier parameter
CVV Key Combine verb 528
key_bit_length parameter
EC Diffie-Hellman verb 200
key_block_length parameter
TR31 Key Token Parse verb 805
key_block_version parameter
TR31 Key Token Parse verb 805
key_encrypting_key_identifier parameter
Key Test2 verb 250
Restrict Key Attribute verb 337
Secure Messaging for Keys verb 596
Symmetric Key Generate verb 354
key_encrypting_key_identifier_1 parameter
Key Generate2 verb 222
key_encrypting_key_identifier_1_length parameter
Key Generate2 verb 222
key_encrypting_key_identifier_2 parameter
Key Generate2 verb 222
key_encrypting_key_identifier_2_length parameter
Key Generate2 verb 222
key_encrypting_key_identifier_length parameter
Restrict Key Attribute verb 337
key_form parameter
Key Generate verb 212
key_generation_data parameter
PIN Change/Unblock verb 586
key_hash_algorithm parameter
Key Token Parse2 verb 313
key_identifier parameter
AES Key Record Delete verb 458
Clear Key Import verb 166
Decipher verb 382
Diversified Key Generate verb 184
Diversified Key Generate2 verb 190
Encipher verb 387
HMAC Generate verb 420
HMAC Verify verb 423
Key Part Import verb 238
Key Test Extended verb 256
Key Test verb 246
Key Test2 verb 250
Key Token Change verb 302
Key Token Change2 verb 305
MAC Generate verb 427
MAC Generate2 verb 430
MAC Verify verb 434
MAC Verify2 verb 438
PKA Key Token Change verb 710
Prohibit Export verb 334
Restrict Key Attribute verb 337
Symmetric Algorithm Decipher verb 392
Symmetric Algorithm Encipher verb 399
key_identifier_length parameter
HMAC Generate verb 420
HMAC Verify verb 423
PKA Key Token Change verb 710
Restrict Key Attribute verb 337
Symmetric Algorithm Decipher verb 392
Symmetric Algorithm Encipher verb 399
key_label parameter
AES Key Record Create verb 456
AES Key Record List verb 460
AES Key Record Read verb 462
AES Key Record Write verb 464
DES Key Record Create verb 466
DES Key Record Delete verb 468
DES Key Record List verb 470
DES Key Record Read verb 472
DES Key Record Write verb 473
PKA Key Record Create verb 475
PKA Key Record List verb 479
PKA Key Record Write verb 483
Retained Key Delete verb 485
key_label_mask parameter
Retained Key List verb 487
key_labels parameter
Retained Key List verb 487
key_labels_count parameter
Retained Key List verb 487
key_length parameter
  Key Generate verb 212
key_material_state parameter
  Key Token Parse2 verb 313
key_name parameter
  Key Token Build2 verb 265
  Key Token Parse2 verb 313
  Symmetric Key Import2 verb 365
key_name_1 parameter
  Key Generate2 verb 222
key_name_1_length parameter
  Key Generate2 verb 222
key_name_2 parameter
  Key Generate2 verb 222
key_name_2_length parameter
  Key Generate2 verb 222
key_offset parameter
  Secure Messaging for Keys verb 596
  Secure Messaging for Keys verb 596
key_parms parameter
  Symmetric Algorithm Decrypt verb 399
  Symmetric Algorithm Encrypt verb 399
key_parms_length parameter
  Symmetric Algorithm Decrypt verb 392
  Symmetric Algorithm Encrypt verb 399
key_part parameter
  Key Part Import verb 238
  Master Key Process verb 158
key_storage_description parameter
  Key Storage Initialization verb 150
key_storage_description_length parameter
  Key Storage Initialization verb 150
key_storage_file_name parameter
  Key Storage Initialization verb 150
key_storage_file_name_length parameter
  Key Storage Initialization verb 150
key_token parameter
  AES Key Record Create verb 456
  AES Key Record Read verb 462
  AES Key Record Write verb 464
  DES Key Record Read verb 472
  DES Key Record Write verb 473
  Key Token Build verb 260
  Key Token Parse verb 308
  Key Token Parse2 verb 313
  PKA Key Record Create verb 475
  PKA Key Record Create verb 475
  PKA Key Record Write verb 483
  PKA Key Token Build verb 700
key_token_length parameter
  AES Key Record Create verb 456
  AES Key Record Read verb 462
  AES Key Record Write verb 464
  PKA Key Record Create verb 475
  PKA Key Record Write verb 483
key_type parameter
  Control Vector Generate verb 170
  Key Export verb 209
  Key Import verb 235
  Key Token Build verb 260
  Key Token Parse verb 308
  Key Token Parse2 verb 313
  key_type_1 parameter
    Key Generate verb 212
    Key Generate2 verb 222
  key_type_2 parameter
    Key Generate verb 212
    Key Generate2 verb 222
key_usage parameter
  TR31 Key Token Parse verb 805
key_value parameter
  Key Token Build verb 260
  Key Token Parse verb 308
key_value_structure parameter
  PKA Key Token Build verb 700
key_value_structure_length parameter
  PKA Key Token Build verb 700
key_verification_pattern parameter
  Key Token Parse2 verb 313
key_verification_pattern_type parameter
  Key Token Parse2 verb 313
key_version_number parameter
  TR31 Key Token Parse verb 805
  key_wrapping_method parameter
  Key Token Parse2 verb 313
  KEY-CLR 246, 265, 392, 399
  KEY-CLRD 246
  KEY-ENC 246, 254
  KEY-ENCDC 246, 254
  key-encrypting key 41
  exporter 180
key-half processing 1002
  KEY-KM 246, 254
  KEY-MGMT 700
  KEY-NKM 246, 254
  KEY-OKM 246, 254
  KEY-PART 170, 260, 308
  key-storage initialization 149
  key-token verification patterns 1022
  key-verification 1021
Keyed-Hash Message Authentication Code (HMAC)
  generating 419
  verifying 423
  KEYGENKY 41, 170, 184, 190, 212, 260, 308
  KEYIDENT 392, 399
  KEYLN16 170, 212, 308, 354
  KEYLN24 212, 260, 354
  KEYLN32 212, 354
  KEYLN8 170, 212, 260, 308, 354
keys
  generating in PCI-HSM 2016
  compliance mode 1081
keyvalue parameter
  PKA Key Encrypt verb 331
  keyvalue_length parameter
  PKA Key Encrypt verb 331
  keyword combinations 41
  KM-ONLY 700
KUF
  key usage field 190
  KUF-MBE
  key usage field must be equal, 190
  KUF-MBP
  key usage field must be permitted 190
  KVP
  key verification pattern 250

L
label parameter
  PKA Key Record Read verb 481
LABEL-DL 458, 468, 477
LAST 237, 238, 241, 420, 423, 427, 434, 442, 445
legacy syntax
  panel.exe 1123
libcsulca.so 24
Linux
  distributions xx
LM-KEK 41, 170, 260, 308
load-balancing multi-domain 10
loading a master key 157
local_enciphered_key_identifier parameter
  Symmetric Key Generate verb 354
local_enciphered_key_identifier_length parameter
  Symmetric Key Generate verb 354
Log Query (CSUALGQ) 152
  format 152
  parameters 152
  required commands 156
  restrictions 156
Log Query (CSUALGQj)
  JNI version 157
  login PIN 1142
lszcrypt -b 1105
lszcrypt, command 1105

M
MAC 41, 55, 170, 184, 190, 209, 212, 235, 260, 265, 308, 985
  length keywords 427, 430, 434, 438
  managing 55
MAC Generate (CSNBKMGN) 426
  format 427
  JNI version 429
  parameters 427
  required commands 429
  restrictions 429
MAC Generate (CSNBKMGN2) restrictions 432
MAC Generate2 (CSNBKMGN2) 430
  format 430
  JNI version 433
  parameters 430
  related information 433
  required commands 432
MAC keys definition 41
mac parameter
  HMAC Generate verb 420
  HMAC Verify verb 423
MAC Generate verb 427
MAC Verify verb 434
MAC Verify (CSNBKMVR) 433
  format 434
  JNI version 437
  methods 433
  parameters 434
  related information 437
  required commands 437

Index 1185
MAC Verify (CSNBMVR) (continued)

restricted 436
usage notes 437
MAC Verify2 (CSNBMVR2) 438
format 438
methods 438
parameters 438
required commands 440
restrictions 440
usage notes 441
MAC Verify2 (CSNBMVR2J)
JNI version 441
mac_length parameter
HMAC Generate verb 420
HMAC Verify verb 423
MACD 211, 235
MACLEN4 427, 434
MACLEN6 427, 434
MACLEN8 427, 434
MACVER 41, 55, 170, 184, 190, 209, 212, 235, 260, 308
manage weak PIN tables 609
mask array preparation 1000
mask_array_left parameter
Control Vector Translate verb 175
mask_array_right parameter
Control Vector Translate verb 175
master key 4, 16, 234, 1121
administration 1119
APKA 1021
asymmetric 83
changing 454
possible effect on internal key tokens 35
enciphered key 234
establishing 4
methods or tools 1119
random generation of a new master-key 85
symmetric 83
master key coherence 11
master key loading 157
master key management 452
master key management verbs 58
master key migration 1148
Master Key Process (CSNBMKP) 25, 157
format 158
JNI version 161
parameters 158
Questionable DES keys 160
required commands 159
restrictions 159
master key register 157
master key variant 34
master key verification 246
master key verification pattern 849
master keys
establishing 84
managing 83
master-key loading 149
master-key verification 1021
MasterCard card-verification code (CVC) 492
MasterCard padding rule 426
masterkey_verification_pattern_vnn parameter
Key Token Parse verb 308

masterkey_verify_parm parameter
Key Token Build verb 260
MD5 55, 445
MDC Generate (CSNBMDDG) 441
format 442
JNI version 444
parameters 442
required commands 444
restrictions 444
MDC parameter
MDC Generate verb 442
MDC-2 442
MDC-4 246, 254, 442
mechanisms
supported for the CCA token 1144
message_text parameter
MAC Verify2 verb 438
message authenticating 417
message authentication definition 55
Message Authentication Code (MAC) 55
description 417
generating 417, 426
verifying 417, 433, 438
Message Authentication Code (MAC) calculation method 1033
message_text_length parameter
MAC Verify2 verb 438
microprocessor chip operating speed 102
MIDDLE 237, 238, 241, 420, 423, 427, 434, 442, 445
migration
CCA token master keys 1148
master key 1148
migration, openCryptoki CCA token 1147
migration to PCI-HSM 2016 compliance mode 1088
MINI1PART 242
MINI2PART 242
MINI3PART 242
miniboot firmware version 102
miscellaneous information 823
MIXED 170, 260, 308
MKVP 260
master key verification pattern 849
mode parameter
TR31 Key Token Parse verb 805
modes of operation 379
Modification Detection Code (MDC) 55, 417, 441, 1024
generate 418
verify 418
modular-exponentiation engine 9
Modulus-Exponent format 327, 700, 716, 718, 854, 857, 872, 873
MRP 331
multi-coprocessor selection functions 11
multi-domain load-balancing 10
multiple
decipherment 1036
encipherment 1036
Multiple Clear Key Import (CSNBECKM) 167
format 167
JNI version 169

Multiple Clear Key Import (CSNBECKM) (continued)
parameters 168
required commands 169
usage notes 169
multiprocessing 9

N

new_reference_PAN_data parameter
PIN Change/Unblock verb 586
new_reference_PIN_block parameter
PIN Change/Unblock verb 586
new_reference_PIN_key parameter
PIN Change/Unblock verb 586
new_reference_PIN_profile parameter
PIN Change/Unblock verb 586
NIST FIPS PUB 140-1 161
NIST standard SP 800-108 37
NMK 452
NMK status 102
NMK status, AES 102
NMK status, ECC 102
no key 16
NO-CV 260, 308
no-export bit 209
NO-KEY 260, 265, 308
NO-SPEC 170, 260, 308
NO-XLATE 700
NO-XPORT 170, 260, 308
NOADJUST 175, 246, 254
NOCV 34, 209, 235
NOCV importers and exporters 41
NOCV-KEK 260
node key management verbs 58
NOEFPB 611, 618
NOEX-SYM 265, 337
NOEXAASY 265, 337
NOEXPORT 337
NOEXUASY 265, 337
nonrepublication 3
NOOFFSET 170, 260, 308
NOT-KEK 41, 170, 260, 308
NOT3INPT 170
null key token 74, 76, 184, 190, 234
definition 35
format 852, 880
num_opt_blocks parameter
TR31 Key Import verb 785
TR31 Key Token Parse verb 805
TR31 Optional Data Build verb 809
TR31 Optional Data Read verb 812
NUM-DECT 102, 122
number of active coprocessors 102
number representation
PKA key tokens 885

O

O-CBC 406
O-CUSP 390-only 406
O-ECB 406
O-IPS 390-only 406
O-X923 406
OAEP 1045
object protection key (OPK) 859

1186 Common Cryptographic Architecture Application Programmer's Guide
object protection keys
OPK 83
OCV (output chaining value) 1028
ODD 341, 343
odd parity 212, 237, 254, 341, 343
OKEY-AES 691
OKEY-DES 691
OKEY-AES 406
OKEY-DES 406
OKEYXLAT 41, 170, 209, 212, 235, 260, 308, 321
OMK 452
OMK status 102
OMK status, AES 102
OMK status, ECC 102
One-Way Hash (CSNBOWH) 445
format 445
JNI version 447
parameters 445
required commands 447
usage notes 447
ONLY 425, 427, 434, 442, 445
OP 212, 354
OP key form 985
openCryptoki 1139
configuration file 1140
configuring 1140
migration, CCA token 1147
shared library (C API) 1139
slot daemon 1141
SO PIN 1142
standard PIN 1142
status information 1143
token library 1140
tracing 1150
openCryptoki.conf 1140
operating speed
microprocessor chip 102
operating system firmware name 102
operating system firmware version 102
Operational (OP) key form 229
operational key 167, 334, 722
distribution 53
generating 985
operational key form 34, 212
operational key token 23
operational private key 72
OPEX 170, 212, 260, 308
OPEX key form 986
OPIM 170, 212, 260, 308
OPIM key form 986
OPINENC 41, 170, 209, 212, 235, 260, 308
OPINENC key type 540
OPK
object protection keys 83
OPK, object protection key 859
OPKCSPAD 406
OPP 212
OPOP 212
OPOP key form 986
opt_block_data parameter
TR31 Optional Data Build verb 809
TR31 Optional Data Read verb 812
opt_block_data_length parameter
TR31 Optional Data Build verb 809
TR31 Optional Data Read verb 812
opt_block_id parameter
TR31 Optional Data Build verb 809
TR31 Optional Data Read verb 812
opt_block_ids parameter
TR31 Optional Data Build verb 809
TR31 Optional Data Read verb 812
opt_block_lengths parameter
TR31 Optional Data Build verb 809
TR31 Optional Data Read verb 812
opt_blocks parameter
TR31 Optional Data Build verb 809
TR31 Optional Data Read verb 812
opt_blocks_bfr_length parameter
TR31 Optional Data Build verb 809
TR31 Optional Data Read verb 812
opt_blocks_length parameter
TR31 Optional Data Build verb 809
TR31 Optional Data Read verb 812
opt_parameter1 parameter
Restrict Key Attribute verb 337
opt_parameter1_length parameter
Restrict Key Attribute verb 337
opt_parameter2 parameter
Restrict Key Attribute verb 337
opt_parameter2_length parameter
Restrict Key Attribute verb 337
opt/IBM/CCA 1108
opt/IBM/CCA
optional_data parameter
Symmetric Algorithm Decipher verb 392
Symmetric Algorithm Encipher verb 399
optional_data_length parameter
Symmetric Algorithm Decipher verb 392
Symmetric Algorithm Encipher verb 399
other documentation xxv
outbound_PIN_encrypting_key_identifier parameter
Encrypted PIN Generate verb 536
output chaining value (OCV) 1028
output chaining vector (OCV) description 380
output_chaining_vector parameter
Secure Messaging for Keys verb 596
Secure Messaging for PINs verb 599
output_KEY_encrypting_key_identifier parameter
Key Translate2 verb 323
output_KEY_encrypting_key_identifier parameter
EC Diffie-Hellman verb 200
Key Translate verb 321
output_KEY_encrypting_key_identifier_length parameter
Key Translate2 verb 323
output_KEY_encrypting_key_identifier_length parameter
EC Diffie-Hellman verb 200
output_key_identifier parameter
CVV Key Combine verb 528
EC Diffie-Hellman verb 200
TR31 Key Import verb 785
output_key_identifier_length parameter
EC Diffie-Hellman verb 200
TR31 Key Import verb 785
output_key_token parameter
Key Translate2 verb 323
output_key_token parameter
Key Translate verb 321
output_key_token_length parameter
Key Translate2 verb 323
output_PAN_data parameter
Encrypted PIN Translate verb 540
output_PAN_data parameter (continued)
Secure Messaging for PINs verb 599
output_PIN_block parameter
Encrypted PIN Translate verb 540
output_PIN_data parameter
PIN Change/Unblock verb 586
output_PIN_encrypting_key_identifier parameter
Encrypted PIN Translate verb 540
output_PIN_message parameter
PIN Change/Unblock verb 586
output_PIN_profile parameter
Encrypted PIN Translate verb 540
PIN Change/Unblock verb 586
Secure Messaging for PINs verb 599
overlapped processing restrictions 9
OVERLAY 464, 483

P

pad digit 501
format 500
pad_character parameter
Encipher verb 387
PADDIGIT 519
PADDIGIT PIN extraction method
keyword 498
padding method 385
PADEXIST 519
PADEXIST PIN extraction method
keyword 498
PADMDC-2 442
PADMDC-4 442
pair of keys 986, 987
PAN 502
PAN_data parameter
Clear PIN Encrypt verb 512
Clear PIN Generate Alternate verb 519
CVV Generate verb 523
CVV Verify verb 532
Encrypted PIN Generate verb 536
Encrypted PIN Verify verb 556
PAN_data_in parameter
Encrypted PIN Translate verb 540
PAN-13 523, 532
PAN-14 523, 532
PAN-15 523, 532
PAN-16 523, 532
PAN-17 523, 532
PAN-18 523, 532
PAN-19 523, 532
panel.exe 18, 1119
ACP tracking 1131
authorization 1121
changing the master key 1128
functions 1121, 1124
key storage re-encrypt 1128
legacy syntax 1123
query adapter compliance state 1132
show access control points 1129
show active role 1129
utilities 1124
utility 41, 101, 454, 1109, 1121, 1126
verifying CCA epoch certificate 1132
parity of key 166

Index 1187
parity of key (continued) EVEN form parameter 341 ODD form parameter 341 part number coprocessor 102 party_info parameter EC Diffie-Hellman verb 200 party_info_length parameter EC Diffie-Hellman verb 200 PATH 27 payload parameter Key Token Parse2 verb 313 Payment Card Industry PCI 31 PCI Payment Card Industry 31 PCI-HSM 2016 compliance mode 31 converting key tokens 1090 generating keys 1081 identifying key tokens for conversion 1088 identifying key tokens to be converted 1088 impact on verbs 1082 migrating applications 1081, 1088 non-compliant verbs 1081 warning events 1088 warnings 32, 1086 PCI-HSM 2016 compliance moden migrating key tokens 1090 migration mode 1090 PCI-HSM 2016 compliant key tokens restrictions 1084 pending change stored in adapter 102 pending change user ID 102 personal account number (PAN) for Encrypted PIN Translate 540 for Encrypted PIN Verify 556 personal identification number (PIN) 3624 PIN generation algorithm 1010 3624 PIN verification algorithm 1013 algorithm format 519, 556 algorithms 56, 494, 515 block format 494, 539 Clear PIN Generate Alternate verb 518 Clear PIN Generate verb 515 definition 56 description 491 detailed algorithms 1010 encrypting 494 encrypting key 495, 539 extraction rules 1008 formats 56 GBP PIN verification algorithm 1015 generating 493, 515 from encrypted PIN block 494 German Banking Pool PIN algorithm 1011 Interbank PIN generation algorithm 1019 keys 41 managing 56 PIN offset generation algorithm 1012 PVV generation algorithm 1018 personal identification number (PIN) (continued) PVV verification algorithm 1019 translating 494 translation of, in networks 492 translation verb 539 using 491 verification verb 555 verifying 494, 555 VISA PIN algorithm 1017 PIN 41, 170, 260, 308, 1142 PIN block 494 PIN block format 3621 1008 3624 1007 additional names 545 ANSI X9.8 1005 detail 1005 ECI-2 1008 ECI-3 1008 IBM 4700 Encrypting PINPAD 1007 ISO-1 1086 ISO-2 1086 PIN extraction method keywords 498 values 498 Visa-2 1007 Visa-3 1007 PIN block translation compliant-tagged key tokens 1085 PIN Change/Unblock PIN extraction rules 1009 PIN Change/Unblock (CSNBPCU) 583, 591 format 586 JNI version 591 parameters 586 required commands 590 usage notes 591 PIN decimalization table identifier 123 PIN extraction rulesCSNBPCU PIN Change/Unblock () 1009 PIN keys 41 PIN notation 1005 PIN profile 497 description 540, 556 PIN reference word/value (PRW) 610 PIN security mode 500 PIN Transaction Security PTS 31 PIN validation value (PVV) 494, 515 PIN verb 493 PIN_check_length parameter Clear PIN Generate Alternate verb 519 Clear PIN Generate verb 515 Encrypted PIN Verify verb 556 PIN_encrypting_key_identifier parameter Clear PIN Encrypt verb 512 Secure Messaging for PINs verb 599 PIN_encryption_key_identifier parameter Clear PIN Generate Alternate verb 519 PIN_generating_key_identifier parameter Clear PIN Generate verb 515 Encrypted PIN Generate verb 536 PIN_generation_key_identifier parameter Clear PIN Generate Alternate verb 519 PIN_length parameter Clear PIN Generate verb 515 Encrypted PIN Generate verb 536 PIN_offset parameter Secure Messaging for PINs verb 599 PIN_offset_field_length parameter Secure Messaging for PINs verb 599 PIN_profile parameter Clear PIN Encrypt verb 512 Clear PIN Generate Alternate verb 519 Encrypted PIN Generate verb 536 PIN_verifying_key_identifier parameter Encrypted PIN Verify verb 556 PIN-encrypting key 539 PINBLOCK 519 PINBLOCK PIN extraction method keyword 498 PINGEN 41, 170, 209, 212, 235, 260, 308, 985 PINGEN key 985 PINLEN04 PIN extraction method keyword 498 PINLEN12 PIN extraction method keyword 498 PINLENn 519 PINVER 41, 170, 209, 212, 235, 260, 308 PINVER key 985 PKA CMK 102 PKA cryptographic key 689 PKA cryptography 71 PKA Decrypt (CSNDPKD) 327 format 327 JNI version 330 parameters 327 required commands 329 restrictions 329 usage notes 330 PKA Encrypt (CSNDPKE) 330 format 331 JNI version 334 parameters 331 required commands 333 restrictions 333 usage notes 333 PKA external key token 74, 75 PKA internal key token 75 PKA key 327, 330 PKA key algorithm 71 PKA Key Generate parameters 691 PKA Key Generate (CSNDPKG) 689 format 691 JNI version 695 required commands 695 restrictions 694 PKA key identifier 74 PKA Key Import (CSNDPKI) 696 format 696 JNI version 699 parameters 696 required commands 698 restrictions 698 usage notes 699
PKA key label 74
PKA key management 73
PKA key management verb 74
PKA Key Record Create
(CSNDKRC) 474
format 475
JNI version 476
parameters 475
related information 476
required commands 476
PKA Key Record Delete
(CSNDKRD) 476
format 476
JNI version 478
parameters 477
related information 478
required commands 477
PKA Key Record List (CSNDKRL) 478
format 479
JNI version 480
parameters 479
related information 480
required commands 480
PKA Key Record Read (CSNDKRR) 481
format 481
JNI version 482
parameters 481
related information 482
required commands 482
PKA Key Record Write
(CSNDKRW) 482
format 483
JNI version 484
parameters 483
related information 484
required commands 484
PKA key storage 150, 449
PKA key storage file 52
PKA key token 74, 880
ECC 881
external 75
record format 854
RSA 1024-bit modulus-exponent private 857
RSA 1024-bit private internal 872, 873
RSA 4096-bit modulus-exponent private 861
RSA private 854, 857
RSA private external 854
RSA private internal 855
RSA public 853
variable Modulus-Exponent 873
RSA 881
PKA Key Token Build (CSNDPKB) 699
format 700
JNI version 709
parameters 700
required commands 709
PKA Key Token Change
(CSNDKTC) 710
format 710
JNI version 712
parameters 710
required commands 712
PKA key token identifier 880
PKA key token sections 880, 881
PKA key tokens
number representation 885
PKA Key Translate (CSNDPKT) 713
format 715
JNI version 719
parameters 716
required commands 718
restrictions 718
usage notes 719
PKA master key 71
PKA NMK 102
PKA null key token 74
PKA OKM 102
PKA private key
integrity 884
PKA Public Key Extract
(CSNDPKX) 720
format 720
JNI version 721
parameters 720
usage notes 721
PKA public-key certificate section 885
PKA verb summary 76
PKA verbs 72
PKA_enciphered_keyvalue parameter
PKA Decrypt verb 327
PKA Encrypt verb 331
PKA_enciphered_keyvalue_length parameter
PKA Decrypt verb 327
PKA Encrypt verb 331
PKA_key_identifier parameter
PKA Decrypt verb 327
PKA Encrypt verb 331
PKA_key_identifier_length parameter
PKA Decrypt verb 327
PKA Encrypt verb 331
PKA_private_key_identifier parameter
Digital Signature Generate verb 676
PKA_private_key_identifier_length parameter
Digital Signature Generate verb 676
PKA_public_key_identifier parameter
Digital Signature Verify verb 682
PKA_public_key_identifier_length parameter
Digital Signature Verify verb 682
PKA92 354, 359
PKA92 key format and encryption process 1042
PKCS #1 hash formats 1045
PKCS #11 1139
PKCS #11 standard C API 1144
PKCS 1.0 675, 676, 682
PKCS 1.1 675, 676, 682
PKCS-1.2 327, 331, 345, 354, 359
PKCS-PAD
Symmetric Algorithm Decipher 390
Symmetric Algorithm Encipher 397
PKCS-PAD processing rule 392, 399
pkcscc tool
master key migration 1148
pkcsconf 1142
pkcsconf -i 1143
pkcsconf command 1140
PKCSOAEP 345, 354, 359
pkcs11t 1141
PKI
public key infrastructure 745
PKOAEP2 345, 365
plain_text parameter
Cryptographic Variable Encipher verb 178
plaintext
encipher 178
enciphering 379
encrypt 178
POST firmware version 102
POST2 version
coprocessor 102
power-supply voltage 102
Prime
elliptic curve type 195
private external key token
RSA 854
private internal key token
RSA 855, 872, 873
private key
integrity 884
OPK, object protection key 859
private key token
RSA 854, 857
private_KEY_key_identifier parameter
EC Diffie-Hellman verb 200
private_KEY_key_identifier_length parameter
EC Diffie-Hellman verb 200
private_key_identifier parameter
EC Diffie-Hellman verb 200
private_key_identifier_length parameter
EC Diffie-Hellman verb 200
private_key_name parameter
PKA Key Token Build verb 700
private_key_name_length parameter
PKA Key Token Build verb 700
problems, reporting xxviii
procedure call 19
processing a master key 157
processing overlap 9
processing rule
ANS X9.23 380, 382, 387
CBC 380, 382, 387, 392, 399
CUSP 380, 382, 387
Decipher 381, 382
description 380
ECB 380, 392, 399
Encipher 385, 387
GPB-PIN 515
IBM-PIN 515
IBM-PINO 515
INBK-PIN 515
IPS 380, 382, 387
PKCS-PAD 380, 392, 399
recommendations for Encipher 387
Symmetric Algorithm Decipher 390, 392
Symmetric Algorithm Encipher 397, 399
VISA-PVV 515
profile 4
Prohibit Export (CSNBPEX) 334
format 334
JNI version 335
Index 1189
Prohibit Export (CSNBPEX) (continued)
parameters 334
required commands 335
Prohibit Export Extended
(CSNBPExx) 335
format 336
JNI version 336
parameters 336
required commands 336
restrictions 336
protected key 16, 19
protected key CPACF 19
protecting data 379
protection_method parameter
TR31 Key Import verb 785
PRW (PIN reference word/value) 610
pseudonym 20
PTS
PIN Transaction Security 31
Public Infrastructure Certificate
(CSNDPIC) 738
format 738
parameters 738
required commands 744
Public Infrastructure Certificate
(CSNDPICJ)
JNI version 744
Public Infrastructure Manage
(CSNPIM) 745
format 746
parameters 746
required commands 752
Public Infrastructure Manage
(CSNPIMJ)
JNI version 753
public key cryptography 71
public key infrastructure
PKI 745
public key token
RSA 853
public_key_identifier parameter
EC Diffie-Hellman verb 200
public_key_identifier_length parameter
EC Diffie-Hellman verb 200
publications about cryptography xxvi
Random Number Generate Long
(CSNBRNGL) 342
format 342
JNI version 344
parameters 343
Random Number Tests (CSUARNT) 161
format 162
JNI version 163
parameters 162
random_number parameter
Key Test Extended verb 256
Random Number Generate Long
verb 343
Random Number Generate verb 341
random_number_length parameter
Random Number Generate Long
verb 343
raw Z value 52
reason code 826
reason codes
with return code 0 826
with return code 12 844
with return code 16 845
with return code 4 827
with return code 8 827
reason_code parameter 22
recommendations for Encipher processing
rule 387
record chaining 380
Recover PIN from Offset (CSNBPOFO)
format 592
JNI version 595
parameters 592
required commands 594
usage notes 595
REFORMAT 170, 260, 302, 308, 323, 540
regeneration_data parameter
PKA Key Generate verb 691
regeneration_data_length parameter
PKA Key Generate verb 691
related publications xxv
remote key distribution 53
Remote Key Export (CSNDRKX) 722
format 724
JNI version 733
parameters 724
required commands 731
restrictions 731
remote key loading
ACP 53
definition 53
new example 53
old example 53
reserved parameter
Control Vector Generate verb 170
Key Test2 verb 250
reserved_1_length parameter
EC Diffie-Hellman verb 200
reserved_2_length parameter
EC Diffie-Hellman verb 200
reserved_3_length parameter
EC Diffie-Hellman verb 200
reserved_4_length parameter
EC Diffie-Hellman verb 200
reserved_5_length parameter
EC Diffie-Hellman verb 200
resource_name parameter
Cryptographic Resource Allocate
verb 145
Cryptographic Resource Deallocate
verb 147
resource_name_length parameter
Cryptographic Resource Allocate
verb 145
Cryptographic Resource Deallocate
verb 147
Restrict Key Attribute (CSNBRKA) 336
format 337
JNI version 340
parameters 337
required commands 340
restrictions 340
usage notes 340
restrictions of CCA library 1146
retained key 485
Retained Key Delete (CSNDRKD) 485
format 485
JNI version 487
parameters 485
related information 486
required commands 486
Retained Key List (CSNDRKL) 487
format 487
JNI version 489
parameters 487
related information 489
required commands 488
retained_keys_count parameter
Retained Key List verb 487
RETRKPR 238, 242
return code 825
return_code parameter 22
returned_PVV parameter
Clear PIN Generate Alternate
verb 519
returned_result parameter
Clear PIN Generate verb 515
revision history xvii
RIPEMD-160 55
RKX DES key tokens
external 851
RKX key token 326
role
DEFAULT 4
default role 1129
role data structure example 1079
role Identifier 102
role structures 1073
basic structure of a role 1073
RPM
contained files 1109
install and configure 1111
samples 1110
RPM installation package 1108
RPMD-160 445, 676
RSA 676, 682, 696, 710
RSA 1024-bit private internal key
token 872, 873
RSA algorithm 71
RSA hardware version 102
RSA key 327, 330, 354, 358, 362, 675,
680, 689
RSA key generation 1035
Q
QPENDING 102
query adapter compliance state
panel.exe 1132
questionable DES key 160
R
radiation 102
RANDOM 341, 343
random generation of a new
master-key 85
random number 341, 342
Random Number Generate
(CSNBRNG) 341
format 341
JNI version 342
parameters 341
required commands 342
random_number_length parameter
Remote Key Create (CSNDRKC)
format 373
JNI version 375
parameters 375
required commands 375
restrictions 375
remote key loading
ACP 53
definition 53
new example 53
old example 53
reserved parameter
Control Vector Generate verb 170
Key Test2 verb 250
reserved_1_length parameter
EC Diffie-Hellman verb 200
reserved_2_length parameter
EC Diffie-Hellman verb 200
reserved_3_length parameter
EC Diffie-Hellman verb 200
reserved_4_length parameter
EC Diffie-Hellman verb 200
reserved_5_length parameter
EC Diffie-Hellman verb 200
resource_name parameter
Cryptographic Resource Allocate
verb 145
Cryptographic Resource Deallocate
verb 147
resource_name_length parameter
Cryptographic Resource Allocate
verb 145
Cryptographic Resource Deallocate
verb 147
Restrict Key Attribute (CSNBRKA) 336
format 337
JNI version 340
parameters 337
required commands 340
restrictions 340
usage notes 340
restrictions of CCA library 1146
retained key 485
Retained Key Delete (CSNDRKD) 485
format 485
JNI version 487
parameters 485
related information 486
required commands 486
Retained Key List (CSNDRKL) 487
format 487
JNI version 489
parameters 487
related information 489
required commands 488
retained_keys_count parameter
Retained Key List verb 487
RETRKPR 238, 242
return code 825
return_code parameter 22
returned_PVV parameter
Clear PIN Generate Alternate
verb 519
returned_result parameter
Clear PIN Generate verb 515
revision history xvii
RIPEMD-160 55
RKX DES key tokens
external 851
RKX key token 326
role
DEFAULT 4
default role 1129
role data structure example 1079
role Identifier 102
role structures 1073
basic structure of a role 1073
RPM
contained files 1109
install and configure 1111
samples 1110
RPM installation package 1108
RPMD-160 445, 676
RSA 676, 682, 696, 710
RSA 1024-bit private internal key
token 872, 873
RSA algorithm 71
RSA hardware version 102
RSA key 327, 330, 354, 358, 362, 675,
680, 689
RSA key generation 1035
rule_array_count (continued)
Retained Key Delete verb 485
Retained Key List verb 487
Secure Messaging for Keys verb 596
Secure Messaging for PINs verb 599
Symmetric Algorithm Decrypt verb 392
Symmetric Algorithm Encrypt verb 399
Symmetric Key Export verb 345
Symmetric Key Generate verb 354
Symmetric Key Import verb 359
Symmetric Key Import2 verb 365
TR31 Key Import verb 785
TR31 Key Token Parse verb 805
TR31 Optional Data Build verb 809
TR31 Optional Data Read verb 812
Transaction Validation verb 604
Trusted Block Create verb 735

seed_length parameter
Random Number Generate Long
verb 343
segmenting
control keywords 427, 434, 438
selecting a coprocessor resource 144, 147
SELFCENC 599
sequence_number parameter
Clear PIN Encrypt verb 512
Encrypted PIN Generate verb 536
Encrypted PIN Translate verb 540
sequences of verbs 57
serial number
adapter 102
coprocessor 102
service_code parameter
CVV Generate verb 523
CVV Verify verb 532
SESS-XOR 184, 190
SET command 453
SHA-1 55, 246, 254, 265, 345, 420, 423, 445, 676
SHA-1 engine 9
SHA-224 246, 420, 423, 445
SHA-256 246, 250, 254, 265, 345, 420, 423, 445, 676
SHA-384 265, 345, 420, 423, 445, 676
SHA-512 265, 345, 420, 423, 445, 676
SHA2VPI 250
short blocks 385
SIG-ONLY 700
signature_bit_length parameter
Digital Signature Generate verb 676
signature_field parameter
Digital Signature Generate verb 676
Digital Signature Verify verb 682
signature_field_length parameter
Digital Signature Generate verb 676
Digital Signature Verify verb 682
SIGSEGV error 16
SINGLE 170, 212, 260, 308, 354
single domain configuration 10
single-length key
multiple decipherment 1037
multiple encipherment 1037
purpose 985, 986
using 41
SINGLE-R 212, 354
size of battery-backed RAM 102
size of dynamic RAM (DRAM) memory 102
size of flash EPROM memory 102
SIZEWPIN 123
skeleton token 140
skeleton token length 140
skeleton_key_identifier parameter
PKA Key Generate verb 691
skeleton_key_identifier_length parameter
PKA Key Generate verb 691
slot daemon, openCryptoKi 1141
slot entry 1140
slot manager 1140
starting 1140
SMKEY 170, 184, 190, 260, 308
SMPIN 170, 184, 190, 260, 308
SNA-SLE 1029
SO
login PIN 1142
source_key parameter
PKA Key Translate verb 716
source_key_identifier
PKA Key Import verb 696
source_key_identifier parameter
Data Key Export verb 180
Key Export verb 209
Key Import verb 235
PKA Public Key Extract verb 720
Symmetric Key Export verb 345
source_key_identifier_length parameter
PKA Public Key Extract verb 720
Symmetric Key Export verb 345
source_key_token parameter
Control Vector Translate verb 175
Data Key Import verb 182
Prohibit Export Extended verb 336
source_transport_key parameter
PKA Key Translate verb 716
SSL support 55
standard user (User)
login PIN 1142
STATAES 102
STATAPKA 102
STATCARD 102
STATCCA 102, 143
STATSCA 102, 143
STATCRD2 102
STATDECT 123
STATDIAG 102
STATEID 102
STATEXPT 102
static keys 545
static TDES keys 545
STATICSA 102, 123
adapter serial number 102
serial number
adapter 102
verb_data field 102
STATICSA operational key parts
output data format 123
STATICSB 126
adapter serial number 102
serial number
adapter 102
verb_data field 102
STATICSB operational key parts
output data format 126
STATICSE 102, 128
adapter serial number 102
serial number
adapter 102
verb_data field 102
STATICSE operational key parts
output data format 128
STATICSX 102, 130
STATICSX operational key parts
output data format 130
STATKP 102, 132
output data 132
STATKP operational key parts
output data format 132
STATKPR 102, 133
input data 132
STATKPR 102
Symmetric Key Import (CSNDSI) (continued)

- restrictions 361
- usage notes 362

Symmetric Key Generate

- (CSNDSYI2) 362
- format 365
- JNI version 369
- parameters 365
- required commands 367
- restrictions 367
- usage notes 368

Symmetric key management

- TR-31 79
- symmetric key token 889
- symmetric keys master key 41
- symmetric master key 83

- panel.exe 1123
- sysfs interface

T

T31XPTOK 170
tampering 102

- target_key_identifier

  - PKA Key Import verb 696
  - target_key_identifier parameter
    - Data Key Export verb 180
    - Data Key Import verb 182
    - Key Export verb 209
    - Key Import verb 235
    - Multiple Clear Key Import verb 168
    - Symmetric Key Import verb 359
    - Symmetric Key Import2 verb 365

- target_key_token parameter

  - Control Vector Translate verb 175
  - Key Token Build2 verb 265
  - PKA Key Translate verb 716
  - target_keyvalue parameter

  - PKA Decrypt verb 327
  - target_keyvalue_length parameter

  - PKA Decrypt verb 327
  - target_public_key_identifier parameter

  - PKA Public Key Extract verb 720
  - target_public_key_identifier_length parameter

  - PKA Public Key Extract verb 720
  - target_transport_key parameter

  - PKA Key Translate verb 716

TDES 1030
TDES encryption 102
TDES-CBC 596, 599
TDES-DEC 184, 190
TDES-ECB 596, 599
TDES-ENC 184, 190
TDES-MAC 427, 434
TDES-XOR 184, 190, 586
TDESEMV2 184, 190, 586
TDESEMV4 586
temperature 102

termology

text parameter

  - HMAC Generate verb 420
  - HMAC Verify verb 423
  - MAC Generate verb 427
  - MAC Generate2 verb 430

- text parameter (continued)

  - MAC Verify verb 434
  - MDC Generate verb 442
  - One-Way Hash verb 445

- text_length parameter

  - Cryptographic Variable Encipher verb 178
  - Decrypt verb 382
  - Encipher verb 387
  - HMAC Generate verb 420
  - HMAC Verify verb 423
  - MAC Generate verb 427
  - MAC Generate2 verb 430
  - MAC Verify verb 434
  - MDC Generate verb 442
  - One-Way Hash verb 445
  - Secure Messaging for Keys verb 596
  - Secure Messaging for PINs verb 599

- time of day 102

TIMEHOLD 102
TKE access 102
TKE workstation 4
  - for administering weak PIN tables 609
TKESTATE 102
TLV_data parameter

  - Key Token Parse2 verb 313
token

  - openCryptoki 1139

TOKEN 41, 209, 211, 212, 234, 235, 246, 254
token agreement scheme 52
token parameter

  - PKA Key Record Read verb 481
  - Token validation value

  - TVV 847

  - token validation value (TVV) 849
token_data parameter

  - Key Token Build2 verb 265
token_length parameter

  - PKA Key Record Read verb 481
  - TOKEN-DL 458, 468, 477
tokens

  - wrapping method 102

TPK-ONLY 682
TR-31 250
symmetric key management 79

TR31 Key Import (CSNBT31I) 781
examples 784
format 784
JNI version 803
parameters 785
required commands 801
restrictions 801
special notes 783

TR31 Key Token Parse (CSNBT31P) 804
format 805
JNI version 807
parameters 805

TR31 Optional Data Build (CSNBT31O) 808
format 809
JNI version 811
parameters 809
restrictions 810

TR31 Optional Data Read (CSNBT31R) 811
TR31 Optional Data Read (CSNBT31R) (continued)
format 812
JNI version 814
parameters 812
tr31_key parameter
TR31 Key Token Parse verb 805
TR31 Optional Data Read verb 812
tr31_key_block parameter
TR31 Key Import verb 785
tr31_key_block_length parameter
TR31 Key Import verb 785
tr31_key_length parameter
TR31 Key Token Parse verb 805
TR31 Optional Data Read verb 812
tracing in openCryptoki 1150
Track 1 Discretionary Data 502
Track 2 Discretionary Data 502
trademarks 1162
trailing short blocks 385
Transaction Sequence Number (TSN) 90
Transaction Validation (CSNBTTRV) 603
format 604
JNI version 606
parameters 604
required commands 606
usage notes 606
transaction_info parameter
Transaction Validation verb 604
transaction_key parameter
Transaction Validation verb 604
TRANSLAT 170, 260, 308, 540
translated key 16, 18
transport key 34, 182, 722, 968
transport key variant
definition 34
transport_key_identifier parameter
PKA Key Generate verb 691
Symmetric Key Import2 verb 365
Trusted Block Create verb 735
transporter_key_identifier parameter
Symmetric Key Export verb 345
transporter_key_identifier_length parameter
Symmetric Key Export verb 345
Triple DES encryption 380
triple-DES 1030
Triple-DES encryption 102
triple-length keys
multiple encipherment 1040
multiple decipherment 1041
trust parent 745
trusted block 53, 734, 968
number representation 971
section format 971
Trusted Block Create (CSNDTBC) 734
format 735
JNI version 737
parameters 735
required commands 737
restrictions 737
trusted block integrity 970
trusted block section X’11’ 972
trusted block section X’12’ 972
subsections 974
trusted block section X’13’ 980
trusted block sections 969

TR31 Optional Data Read (CSNBT31R)

Trusted Key Entry (TKE) 1071
Trusted Key Entry workstation (TKE)
overview 57
trusted_block_identifier parameter
Trusted Block Create verb 735
trusted_block_identifier_length parameter
Trusted Block Create verb 735
TSN
See Transaction Sequence Number
Token validation value 847
token validation value (TVV) 849

U
Ubuntu installation 1107
UKPT 170, 260, 308
format 501
UKPTBOTH 540
UKPTPIN 540, 556
UKPTOPIN 540
unbinding
PAN to the PIN 1085
unbinding PAN to the PIN 1085
Unique Key Derive
parameters 370
Unique Key Derive (CSNBUKD) 369
format 370
required commands 376
restrictions 375
usage notes 376
Unique Key Derive (CSNBUKD) JNI version 377
unwrap working keys 83
unwrap_kek_identifier parameter
TR31 Key Import verb 785
unwrap_kek_identifier_length parameter
TR31 Key Import verb 785
USE-CV 260
USECONFIG 168, 184, 190, 238, 302, 323, 354, 359, 365
User
login PIN 1142
user ID
pending change 102
user_associated_data parameter
Key Token Build2 verb 265
Key Token Parse2 verb 313
user_associated_data_1 parameter
Key Generate2 verb 222
user_associated_data_1_length parameter
Key Generate2 verb 222
user_associated_data_2 parameter
Key Generate2 verb 222
user_associated_data_2_length parameter
Key Generate2 verb 222
user_definable_associated_data parameter
PKA Key Token Build verb 700
user_definable_associated_data_length parameter
PKA Key Token Build verb 700
usr/lib64 1108
UTC time of day 102
utilities
ivp.exe 1119
panel.exe 18, 41, 101, 454, 1109, 1119,
1121, 1124, 1126, 1128, 1129

utilities (continued)
ACP tracking 1131
query adapter compliance state 1132
verifying CCA epoch certificate 1132
PKA Key Token Build 699
utility verbs 817

V
VALIDATE 710
validating CCA epoch certificate
panel.exe 1132
validation_values parameter
Transaction Validation verb 604
value_1 parameter
Key Test verb 246
value_2 parameter
Key Test verb 246
variable length token 40
variable Modulus-Exponent token
RSA 873
variable types 20
variable-length AES key 304
variable-length HMAC key 304
variable-length key token 889
VDSP standard
Visa Data Secure Platform 545 verb
access control 59
Access Control Maintenance (CSUAACM) 89
Access Control Tracking (CSUAAC) 93
AES 61
AES Key Record Create (CSNBAKRC) 455
AES Key Record Delete (CSNBAKRD) 457
AES Key Record List (CSNBAKRL) 459
AES Key Record Read (CSNBAKR) 462
AES Key Record Write (CSNBAKRW) 464
Authentication Parameter Generate (CSNBAPEG) 508
CCA 61, 87
CCA node 59
CCA nodes and resource control 89
Cipher Text Translate2 (CSNBT2) 404
Clear Key Import (CSNBIK) 166
Clear PIN Encrypt (CSNBPCE) 512
Clear PIN Generate (CSNBPGN) 515
Clear PIN Generate Alternate (CSNBPCEPA) 518
Code Conversion (CSNBPXEA) 817
common parameters 22
Control Vector Generate (CSNBVC) 170
Control Vector Translate (CSNVCVT) 174
Cryptographic Facility Query (CSUAFQ) 100, 102
VFPE
  Visa Format Preserving Encryption 502, 545
Visa (EMV) padding rule 426
Visa card-verification value (CVV) 492
Visa Data Secure Platform
  VDSP standard 545
  VSDP 568
Visa Format Preserving Encryption
  VFPE 502, 545
Visa payment card data formats
  Cardholder Name 502
  PAN 502
  Track 1 Discretionary Data 502
  Track 2 Discretionary Data 502
VISA PVV 515
VISA PVV key 518
VISA-1 545
Visa-2 PIN block format 1007
Visa-2 PIN block format 498
Visa-3 PIN block format 1007
Visa-3 PIN block format 498
Visa-4 PIN block format 498
VISA-PVV 170, 260, 308, 519, 556
VISA-PVV algorithm 519, 556
VISAPCU1 586
VISAPCU2 586
VISAPVV4 556
VISAPVV4 algorithm 556
VSDP
  Visa Data Secure Platform 568

W
  warning events
    identifying key tokens for conversion 1088
    PCI-HSM 2016 compliance mode 32, 1086
weak PIN table
  TKE workstation 609
who should use this document xx
wrap working keys 83
wrap_kek_identifier parameter
  TR31 Key Import verb 785
wrap_kek_identifier_length parameter
  TR31 Key Import verb 785
WRAP-ECB 168, 184, 190, 238, 260, 302, 308, 323, 354, 359, 365
WRAP-ENH 168, 184, 190, 238, 260, 302, 308, 323, 354, 359, 365
WRAPMTHD 102
wrapping key 16

X
  X.509 certificate 745
  X3.106 (CBC) method 1028
  X9.19OPT 427, 434
  X9.31 676, 682
  X9.31 hash format 1044
  X9.9-1 427, 434
  X9.9-1 keyword 427, 434, 438
  XLA-TE 170, 260, 308
  XLA-TE-OK 700
  XPORT-OK 170, 260, 308
XPRT-SYM 265
XPRTAASY 265
XPRTUASY 265

Z
  z/VM xxi
  z/VM guest 1106
  zcrypt installing 1103
  ZERO-PAD 327, 331, 345, 354, 359, 676, 682
  Zone Encryption Keys 545
Readers’ Comments — We'd Like to Hear from You

Linux on Z and LinuxONE
Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide
Version 6.0

Publication No. SC33-8294-08

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.
Submit your comments using one of these channels:
• Send your comments to the address on the reverse side of this form.
• Send your comments via email to: eservdoc@de.ibm.com

If you would like a response from IBM, please fill in the following information:

Name

Address

Company or Organization

Phone No.

Email address
Readers’ Comments — We’d Like to Hear from You

IBM Deutschland Research & Development GmbH
Information Development
Department 3282
Schoenaicher Strasse 220
71032 Boeblingen
Germany