z/OS Cryptographic Services
Integrated Cryptographic Service Facility

TR-31 Optional Data Read Update for CCA Compliance – APAR OA38616

(March 1, 2012)

Contents

Chapter 1. Overview

The Integrated Cryptographic Service Facility (ICSF) implements the IBM Common Cryptographic Architecture (CCA) API, which is the lowest available API for invoking Crypto Express3 (CEX3C) services. ICSF's implementation of the TR-31 Optional Data Read callable service in the Cryptographic Support for z/OS V1R11-R13 web deliverable (FMID HCR7790) is not fully compatible with the CCA API. Specifically, the *opt_block_length* parameter is returned as an array of 31-bit integers, while the CCA API defines that field as an array of 16-bit integers.

This document describes changes to the ICSF product to make its implementation of the TR-31 Optional Data Read callable service compatible with the CCA API. In addition, the description of the *cv_source* and *protection_method* parameters of the TR-31 Import callable service were modified to show the full integer values.

These changes are available through the application of the PTF for APAR OA38616. This document contains alterations to information previously presented in *z/OS Cryptographic Services ICSF Application Programmer's Guide*, SA22-7522-15.

The technical changes made to the ICSF product by the application of the PTF for APAR OA38616 are indicated in this document by a vertical line to the left of the change.

Chapter 2. Update of z/OS Cryptographic Services ICSF Application Programmer’s Guide, SA22-7522-15, information

This chapter contains updates to the document *z/OS Cryptographic Services ICSF Application Programmer’s Guide*, SA22-7522-15, for the callable service changes implemented by the PTF for APAR OA38616. Refer to this source document if background information is needed.

TR-31 Import (CSNBT31I and CSNET31I)

Use the TR-31 Import callable service to convert a TR-31 key block to a CCA token. Since there is not always a one-to-one mapping between the key attributes defined by TR-31 and those defined by CCA, the caller may need to specify the attributes to attach to the imported key through the rule array.

The callable service name for AMODE(64) is CSNET31I.

Format

```
CALL 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_blks,
        cv_source,
        protection_method )
```

Parameters

return_code

Direction: Output   Type: Integer

The return code specifies the general result of the callable service.

reason_code

Direction: Output   Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems.

exit_data_length

Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit_data* parameter.

exit_data

Direction: Input/Output Type: String

The data that is passed to the installation exit.

rule_array_count

Direction: Input Type: Integer

The number of keywords you are supplying in the *rule_array* parameter. The *rule_array_count* parameter must be 1, 2, 3, 4, or 5.

rule_array

Direction: Input Type: String

The *rule_array* contains keywords that provide control information to the callable service. 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 callable service are shown in the following table. One keyword from one CCA output key usage subgroup shown in the following table is required based on TR-31 input key usage, unless the CV is included in the TR-31 key block as an optional block. If the CV is included in the TR-31 key block as an optional block, the included CV will be used in the output key block as long as it does not conflict with the TR-31 header data.

See Table 3 on page 10 for valid combinations of Usage and Mode

Table 1. Keywords for TR-31 Import Rule Array Control Information

Keyword	Meaning
Key Wrapping Method (One Required)	
INTERNAL	Desired *output_key_identifier* is a CCA internal key token, wrapped using the card master key.
EXTERNAL	Desired *output_key_identifier* is a CCA external key token, wrapped using the key represented by the *unwrap_kek_identifier*.
CCA Output Key Usage Subgroups (One keyword from one CCA output key usage subgroup shown in the following table is required based on TR-31 input key usage, unless the CV is included in the TR-31 key block as an optional block. If the CV is included in the TR-31 key block as an optional block, the included CV will be used in the output key block as long as it does not conflict with the TR-31 header data.)	
C0 Subgroup (One Required for this TR-31 key usage)	
CVK-CVV	Convert TR-31 CVK to a CCA key for use with CVV/CVC. The CCA key will be a MAC key with subtype CVVKEY-A.
CVK-CSC	Convert TR-31 CVK to a CCA key for use with CSC. The CCA key will be a MAC key with subtype AMEX CSC.
K0 Subgroup (One Required for this TR-31 key usage)	

Table 1. Keywords for TR-31 Import Rule Array Control Information (continued)

Keyword	Meaning
EXPORTER	For TR-31 K0-E or K0-B usage+mode keys. Convert TR-31 KEK to a CCA wrapping key. The key will convert to a CCA EXPORTER key. Note that the K0-B key import has a unique ACP.
OKEYXLAT	For TR-31 K0-E or K0-B usage+mode keys. Convert TR-31 KEK to a CCA wrapping key. The key will convert to a CCA OKEYXLAT key. Note that the K0-B key import has a unique ACP.
IMPORTER	For TR-31 K0-D or K0-B usage+mode keys. Convert TR-31 KEK to a CCA unwrapping key. The key will convert to a CCA IMPORTER key. Note that the K0-B key import has a unique ACP.
IKEYXLAT	For TR-31 K0-D or K0-B usage+mode keys. Convert TR-31 KEK to a CCA unwrapping key. The key will convert to a CCA IKEYXLAT key. Note that the K0-B key import has a unique ACP.
V0/V1/V2 Subgroup (One Required for these TR-31 key usages)	
PINGEN	Convert a TR-31 PIN verification key to a CCA PINGEN key.
PINVER	Convert a TR-31 PIN verification key to a CCA PINVER key.
E0/E2,F0/F2 Subgroup (One Required for these TR-31 key usages)	
DMAC	Convert TR-31 EMV master key (chip card or issuer) for Application Cryptograms or Secure Messaging for Integrity to CCA DKYGENKY type DMAC
DMV	Convert TR-31 EMV master key (chip card or issuer) for Application Cryptograms or Secure Messaging for Integrity to CCA DKYGENKY type DMV
E1,F1 Subgroup (One Required for these TR-31 key usages)	
DMPIN	Convert TR-31 EMV master key (chip card or issuer) for Secure Messaging for Confidentiality to CCA DKYGENKY type DMPIN
DDATA	Convert TR-31 EMV master key (chip card or issuer) for Secure Messaging for Confidentiality to CCA DKYGENKY type DDATA
E5 Subgroup (One Required for this TR-31 key usage)	
DMAC	Convert TR-31 EMV master key (issuer) for Card Personalization to CCA DKYGENKY type DMAC.
DMV	Convert TR-31 EMV master key (issuer) for Card Personalization to CCA DKYGENKY type DMV.
DEXP	Convert TR-31 EMV master key (issuer) for Card Personalization to CCA DKYGENKY type DEXP.
Key Derivation Level (One Required with E0, E1, E2 TR-31 key usages unless the CV is included in the TR-31 key block as an optional block. If the CV is included in the TR-31 key block, the included CV will be used in the output key block as long as it does not conflict with the TR-31 header data.)	
DKYL0	Convert TR-31 EMV master key (chip card or issuer) to CCA DKYGENKY at derivation level DKYL0.
DKYL1	Convert TR-31 EMV master key (chip card or issuer) to CCA DKYGENKY at derivation level DKYL1.

Table 1. Keywords for TR-31 Import Rule Array Control Information (continued)

Keyword	Meaning
Key Type Modifier (Optional)	
NOOFFSET	Valid only for V0/V1 TR-31 key usage values. Import the PINGEN or PINVER key into a key token that cannot participate in the generation or verification of a PIN when an offset or the Visa PVV process is requested.
Key Wrapping Method (Optional) **Note:** Conflicts between wrapping keywords used and a CV passed in an optional data block of the TR-31 token will result in errors being returned. The main example of this is a CV that indicates 'enhanced-only' in bit 56 when the user or configured default specifies ECB for key wrapping.	
USECONFG	Specifies that the configuration setting for the default wrapping method is to be used to wrap the key. This is the default.
WRAP-ENH	Specifies that the new enhanced wrapping method is to be used to wrap the key.
WRAP-ECB	Specifies that the original wrapping method is to be used.
Translation Control (One Optional)	
ENH-ONLY	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. If the keyword is not specified translation to the original method will be allowed. This turns on bit 56 in the control vector. This keyword is not valid if processing a zero CV data key. **Notes:** 1. If the TR-31 block contains a CV in the optional data block that does not have bit 56 turned on, bit 56 will be turned on in the output token, since with this keyword the user is asking for this behavior. The exception to this is for CVs of all 0x00 bytes, for this case no error will be generated but the CV will remain all 0x00 bytes. 2. Conflicts between wrapping keywords used and a CV passed in an optional data block of the TR-31 token will result in errors being returned. The main example of this is a CV that indicates 'enhanced-only' in bit 56 when the user or configured default specifies ECB for key wrapping. If the default wrapping method is ECB mode, but the enhanced mode and the ENH-ONLY restriction are desired for a particular key token, combine the ENH-ONLY keyword with the WRAP-ENH keyword.

TR31_key_block_length

Direction: Input　　Type: Integer

This parameter specifies the length of the TR31_key_block parameter, in bytes. The length field in the TR-31 block is a 4-digit decimal number, so the maximum acceptable length is 9992 bytes.

TR31_key_block

Direction: Input　　Type: String

This parameter contains the TR-31 key block that is to be imported. The key block is protected with the key passed in parameter *unwrap_kek_identifier*.

unwrap_kek_identifier_length

Direction: Input Type: Integer

This parameter specifies the length of the *unwrap_kek_identifier* parameter, in bytes. The value in this parameter must currently be 64, since only CCA internal key tokens are supported for the *unwrap_kek_identifier* parameter.

unwrap_kek_identifier

Direction: Input/Output Type: String

This parameter contains either the label or the key token for the key that is used to unwrap and check integrity of the imported key passed in the TR31_key_block parameter. The key must be a CCA internal token for a KEK IMPORTER or IKEYXLAT type. If a key token is passed which is wrapped under the old master key, it will be updated on output so that it is wrapped under the current master key.

Note: ECB-mode wrapped DES keys (CCA legacy wrap mode) cannot be used to wrap/unwrap TR-31 version 'B'/'C' key blocks that have, or will have ,'E' exportability. This is because ECB-mode does not comply with ANSI X9.24 Part 1.

wrap_kek_identifier_length

Direction: Input Type: Integer

This parameter specifies the length of the *wrap_kek_identifier* parameter, in bytes. If the *unwrap_kek_identifier* is also to be used to wrap the output CCA token, specify 0 for this parameter. Otherwise, this parameter must be 64.

wrap_kek_identifier

Direction: Input/Output Type: String

When *wrap_kek_identifier_length* is 0, this parameter is ignored and the *unwrap_kek_identifier* is also to be used to wrap the output CCA token. Otherwise, this parameter contains either the label or the key token for the KEK to use for wrapping the output CCA token. It must be a CCA internal token for a KEK EXPORTER or OKEYXLAT type and must have the same clear key as the *unwrap_kek_identifier*. If a key token is passed which is wrapped under the old master key, it will be updated on output so that it is wrapped under the current master key.

Note: ECB-mode wrapped DES keys (CCA legacy wrap mode) cannot be used to wrap/unwrap TR-31 version 'B'/'C' key blocks that have/will have 'E' exportability. This is because ECB-mode does not comply with ANSI X9.24 Part 1.

output_key_identifier_length

Direction: Input/Output Type: Integer

This parameter specifies the length of the *output_key_identifier* parameter, in bytes. On input, it specifies the length of the buffer represented by the *output_key_identifier* parameter and must be at least 64 bytes long. On output, it contains the length of the token returned in the *output_key_identifier* parameter.

output_key_identifier

Direction: Output Type: String

This parameter contains the key token that is to receive the imported key. The output token will be a CCA internal or external key token containing the key received in the TR-31 key block.

num_opt_blocks

Direction: Output Type: Integer

This parameter contains the number of optional blocks that are present in the TR-31 key block.

cv_source

Direction: Output Type: Integer

This parameter contains information about how the control vector in the output key token was created. It can be one of the following three values:

X'00000000'
No CV was present in an optional block, and the output CV was created by the callable service based on input parameters and on the attributes in the TR-31 key block header.

X'00000001'
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 callable service verified compatibility of the header values with the CV and then used that CV in the output key token.

X'00000002'
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.

Any value other than these are reserved for future use and are currently invalid.

protection_method

Direction: Output Type: Integer

This parameter contains information about what method was used to protect the input TR-31 key block. It can have one of the following values:

X'00000000'
The TR-31 key block was protected using the variant method as identified by a Key Block Version ID value of "A" (0x41).

X'00000001'
The TR-31 key block was protected using the derived key method as identified by a Key Block Version ID value of "B" (0x42).

X'00000002'
The TR-31 key block was protected using the variant method as identified by a Key Block Version ID value of "C" (0x43). Functionally this method is the same as 'A', but to maintain consistency a different value will be returned here for 'C'.

Any value other than these are reserved for future use and are currently invalid.

Restrictions

This callable service only imports DES and TDES keys.

Proprietary values for the TR-31 header fields are not supported by this callable service with the exception of the proprietary values used by IBM CCA when carrying a control vector in an optional block in the header.

Usage Notes

Unless otherwise noted, all String parameters that are either written to, or read from, a TR-31 key block will be in EBCDIC format. Input parameters are converted to ASCII before being written to the TR-31 key block and output parameters are converted to EBCDIC before being returned . TR-31 key blocks themselves are always in printable ASCII format as required by the ANSI TR-31 specification.

If the TR-31 key block is marked as a key component, the resulting CCA key will have the Key Part bit (bit 44) in the control vector set to 1.

The exportability attributes of the imported CCA token are set based on attributes in the TR-31 key block as described in the following table.

Table 2. Export attributes of an imported CCA token

TR-31 export attribute value	CCA action on import
Non-exportable ("N")	CCA imports the key to an internal CCA key token. CV bit 17 (export) is set to zero to indicate that the key is not exportable. CV bit 57 (TR-31 export) is set to one to indicate that the key is not exportable to TR-31.
Exportable under trusted key ("E")	If the TR-31 token is wrapped with a CCA KEK in the old ECB format, the request is rejected because that KEK is not a trusted key. If the CCA KEK is in a newer X9.24 compliant CCA key block, then the TR-31 key is imported to CCA in exactly the same way as described below for keys that are exportable under any key.
Exportable under any key ("S")	CCA imports the key to an internal CCA key token. CV bit 17 (export) is set to one to indicate that the key is exportable. CV bit 57 (TR-31 export) is set to zero to indicate that the key is also exportable to TR-31.

If necessary, use the Prohibit Export, Prohibit Exported Extended, or Restrict Key Attribute callable service to alter the export attributes of the CCA token after import.

If the TR-31 key block contains an optional block with a CCA CV of '00007D00030000000000000000000000' for a single length key or '00007D0003410000000000000000000000007D00032100000000000000000000' for a double length key, the resulting CCA token will be a zero CV DATA token.

The TR-31 key block can contain a CCA control vector in an optional data field in the header. If the CV is present, the service will check that CV for compatibility with the TR-31 key attributes to ensure the CV is valid for the key and if there are no problems it will use that CV in the CCA key token that is output by the service.

TR-31 Import

If a CV is received, the import operation is not subject to any ACP controlling the importation of specific key types. The CV may be present in the TR-31 key block in two different ways, depending on options used when creating that block.

- If the TR-31 Export callable service was called with option INCL-CV, the control vector is included in the TR-31 key block and the TR-31 key usage and mode of use fields contain attributes from the set defined in the TR-31 standard. The TR-31 Import callable service checks that those TR-31 attributes are compatible with the CV included in the block. It also verifies that no rule array keywords conflict with the CV contained in the TR-31 block.
- If the TR-31 Export callable service was called with option ATTR-CV, the control vector is included in the TR-31 key block and the TR-31 key usage and mode of use fields contain proprietary values (ASCII “10” and “1”, respectively) to indicate that the usage and mode information is contained in the included control vector. In this case, the TR-31 Import service uses the included CV as the control vector for the CCA key token it produces. It also verifies that the CV does not conflict with rule array keywords passed

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

The access control points in the ICSF role that control the general function of this service are:

- TR31 Import – Permit version A TR-31 key blocks
- TR31 Import – Permit version B TR-31 key blocks
- TR31 Import – Permit version C TR-31 key blocks
- TR31 Import – Permit override of default wrapping method

The following table lists the valid attribute translations for import of TR-31 key blocks to CCA keys along with the access control points which govern those translations. Any translation not listed here will result in an error. If an individual cell is blank, it represents the value of the cell immediately above it.

Note: In order to import a TR-31 key block to a CCA key, the appropriate key block version ACP needs to be enabled in addition to any required translation specific ACPs from below.

Table 3. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs)

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg’m	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
DUKPT Base Derivation Keys							
B0	A	N	T	(none)	KEYGENKY	UKPT	(none)
B0	B,C	X	T	(none)	KEYGENKY	UKPT	
B1	B,C	(none)	(none)	(none)	(none)	(none)	
Note: These are the base keys from which DUKPT initial keys are derived for individual devices such as PIN pads.							
Card Verification Keys							
C0	A,B,C	G, C	D	CVK-CSC	MAC	AMEX-CSC	Permit C0 to MAC/MACVER:AMEX-CSC
	A,B,C		T	CVK-CSC	MAC	AMEX-CSC	
	A,B,C	V	D	CVK-CSC	MACVER	AMEX-CSC	
	A,B,C		T	CVK-CSC	MACVER	AMEX-CSC	

Table 3. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs) (continued)

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
	A,B,C	G, C	T	CVK-CVV	MAC	CVVKEY-A	Permit C0 to MAC/MACVER:CVVKEY-A
	A,B,C	V	T	CVK-CVV	MACVER	CVVKEY-A	

The card verification keys are keys for computing or verifying (against supplied value) a card verification code with the CVV, CVC, CVC2 and CVV2 algorithms.

Notes:

1. In CCA, this corresponds to keys used with two different APIs.
   - Visa CVV and MasterCard CVC codes are computed with CVV_Generate and verified with CVV_Verify. Keys must be DATA or MAC with sub-type (in bits 0-3) “ANY-MAC” , “CVVKEY-A” or “CVVKEY-B”. The GEN bit (20) or VER bit (21) must be set appropriately.
   - American Express CSC codes are generated and verified with the Transaction_Validate verb. The key must be a MAC or MACVER key with sub-type “ANY-MAC” or “AMEX-CSC”. The GEN bit (20) or VER bit (21) must be set appropriately.
2. CCA and TR-31 represent CVV keys incompatibly. CCA represents the “A” and “B” keys as two 8 B keys, while TR-31 represents these as one 16 B key. The CVV generate and verify verbs now accept a 16 B CVV key, using left and right parts as A and B. Current Visa standards require this.
3. Import and export of the 8 B CVVKEY-A and CVVKEY-B types will only be allowed using the proprietary TR-31 usage+mode values to indicate encapsulation of the IBM CV in an optional block, since the 8 B CVVKEY-A is meaningless / useless as a TR-31 C0 usage key of any mode.
4. Import of a TR-31 key of usage C0 to CCA key type ‘ANY-MAC’ will not be allowed, although the ANY-MAC key is also usable for card verification purposes.
5. It is possible to convert a CCA CVV key into a CSC key or vice-versa, since the translation from TR-31 usage “C0” is controlled by rule array keywords on the import verb. This can be restricted by using ACPs, but if both of translation types are required they cannot be disabled and control is up to the development, deployment, and execution of the applications themselves.

   CCA does not have a ‘MAC GEN ONLY’ key type, so TR-31 usage of G will translate to a full MAC key.

Data Encryption Keys

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
D0	A,B,C	E	D, T	(none)	ENCIPHER	(none)	(none)
	A,B,C	D	D, T	(none)	DECIPHER	(none)	
	A,B,C	B	D, T	(none)	CIPHER	(none)	

Notes:

1. There is asymmetry in the TR-31 to CCA and CCA to TR-31 translation. CCA keys can be exported to TR-31 ‘D0’ keys from CCA type ENCIPHER, DECIPHER, or CIPHER, or type DATA with proper Encipher and Decipher CV bits on. A TR-31 ‘D0’ key can only be imported to CCA types ENCIPHER, DECIPHER, or CIPHER, not the lower security DATA key type. This eliminates conversion to the lower security DATA type by export / re-import.
2. There are no ACPs controlling import since the intent of the TR-31 key's control is not interpreted, just directly translated to CCA control.

Key Encrypting Keys

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
K0	A,B,C	E	T	OKEYXLAT	OKEYXLAT	(none)	Permit K0:E to EXPORTER/OKEYXLAT
	A,B,C			EXPORTER	EXPORTER	(none)	
	A,B,C	D	T	IKEYXLAT	IKEYXLAT	(none)	Permit K0:D to IMPORTER/IKEYXLAT
	A,B,C			IMPORTER	IMPORTER	(none)	
	A,B,C	B	T	OKEYXLAT	OKEYXLAT	(none)	Permit K0:B to EXPORTER/OKEYXLAT
	A,B,C			EXPORTER	EXPORTER	(none)	

Table 3. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs) (continued)

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
	A,B,C			IKEYXLAT	IKEYXLAT	(none)	Permit K0:B to IMPORTER/IKEYXLAT
	A,B,C			IMPORTER	IMPORTER	(none)	
K1	B,C	E	T	OKEYXLAT	OKEYXLAT	(none)	Permit K1:E to EXPORTER/OKEYXLAT
	B,C			EXPORTER	EXPORTER	(none)	
	B,C	D	T	IKEYXLAT	IKEYXLAT	(none)	Permit K1:D to IMPORTER/IKEYXLAT
	B,C			IMPORTER	IMPORTER	(none)	
	B,C	B	T	OKEYXLAT	OKEYXLAT	(none)	Permit K1:B to EXPORTER/OKEYXLAT
	B,C			EXPORTER	EXPORTER	(none)	
	B,C			IKEYXLAT	IKEYXLAT	(none)	Permit K1:B to IMPORTER/IKEYXLAT
	B,C			IMPORTER	IMPORTER	(none)	

Notes:

1. K1' keys are not distinguished from 'K0' keys within CCA. The 'K1' key is a particular KEK for deriving keys used in the 'B' or 'C' version wrapping of TR-31 key blocks. 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.
2. It is possible to convert a CCA EXPORTER key to an OKEYXLAT, or to convert an IMPORTER to an IKEYXLAT by export / re-import. This can be restricted by using ACPs, but if both translations are required they cannot be disabled and control is up to the development, deployment, and execution of the applications themselves.
3. It will not be possible to export a CCA key to TR-31 type K0-B, in order to avoid the ability to translate a CCA EXPORTER to a CCA IMPORTER via export/import to the TR-31 token type. When a TR-31 key block does not have an included CV as an optional block, the default CV will be used to construct the output token. For IMPORTER / EXPORTER keys this means that the Key Generate bits will also be on in the KEK.

MAC Keys

M0	A,B,C	G,C	T	(none)	MAC	ANY-MAC	Permit M0/M1/M3 to MAC/MACVER:ANY-MAC
	A,B,C	V	T	(none)	MACVER	ANY-MAC	
M1	A,B,C	G,C	D, T	(none)	MAC	ANY-MAC	
	A,B,C	V	D, T	(none)	MACVER	ANY-MAC	
M3	A,B,C	G,C	D, T	(none)	MAC	ANY-MAC	
	A,B,C	V	D, T	(none)	MACVER	ANY-MAC	

Notes:

1. M0 and M1 are identical (ISO 16609 based on ISO 9797) normal DES/TDES (CBC) MAC computation, except M1 allows 8 byte and 16 byte keys while M0 allows only 16 byte keys. Mode M3 is the X9.19 style triple-DES MAC.
2. CCA does not support M2, M4, or M5.
3. Although export of DATAM/DATAMV keys to TR-31 M0/M1/M3 key types is allowed, import to DATAM/DATAMV CCA types is not allowed since they are obsolete types

PIN Keys

P0	A,B,C	E	T	(none)	OPINENC	(none)	Permit P0:E to OPINENC
	A,B,C	D		(none)	IPINENC	(none)	Permit P0:D to IPINENC
	A,B,C	B – not supp		(none)	(none)	(none)	(none)

Table 3. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs) (continued)

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
V0	A	N	T	PINGEN [NOOFFSET]	PINGEN	NO-SPEC [+NOOFFSET]	Permit V0 to PINGEN:NO-SPEC, Permit V0/V1/V2:N to PINGEN/PINVER
	A,B,C	G,C		[NOOFFSET]	PINGEN	NO-SPEC [+NOOFFSET]	Permit V0 to PINGEN:NO-SPEC
	A	N		PINVER [NOOFFSET]	PINVER	NO-SPEC [+NOOFFSET]	Permit V0 to PINVER:NO-SPEC, Permit V0/V1/V2:N to PINGEN/PINVER
	A,B,C	V		[NOOFFSET]	PINVER	NO-SPEC [+NOOFFSET]	Permit V0 to PINVER:NO-SPEC
V1	A	N	T	PINGEN [NOOFFSET]	PINGEN	IBM-PIN /IBM-PINO	Permit V1 to PINGEN:IBM-PIN/IBM-PINO, Permit V0/V1/V2:N to PINGEN/PINVER
	A,B,C	G,C		[NOOFFSET]	PINGEN	IBM-PIN /IBM-PINO	Permit V1 to PINGEN:IBM-PIN/IBM-PINO
	A	N		PINVER [NOOFFSET]	PINVER	IBM-PIN /IBM-PINO	Permit V1 to PINVER:IBM-PIN/IBM-PINO, Permit V0/V1/V2:N to PINGEN/PINVER
	A,B,C	V		[NOOFFSET]	PINVER	IBM-PIN /IBM-PINO	Permit V1 to PINVER:IBM-PIN/IBM-PINO
V2	A	N	T	PINGEN	PINGEN	VISA-PVV	Permit V2 to PINGEN:VISA-PVV, Permit V0/V1/V2:N to PINGEN/PINVER
	A,B,C	G,C			PINGEN	VISA-PVV	Permit V2 to PINGEN:VISA-PVV
	A	N		PINVER	PINVER	VISA-PVV	Permit V2 to PINVER:VISA-PVV, Permit V0/V1/V2:N to PINGEN/PINVER
	A,B,C	V			PINVER	VISA-PVV	Permit V2 to PINVER:VISA-PVV

Notes:

1. NOOFFSET keyword may be passed to specify resultant CCA key to have NOOFFSET bit (bit 37) on in CV. However this will be automatic if CV is included and has NOOFFSET bit set.
2. NOOFFSET keyword is not supported for V2 usage since VISA-PVV algorithm does not support that concept.
3. There is a subtle difference between TR-31 V0 mode and CCA 'NO-SPEC' subtype. V0 mode restricts keys from 3224 or PVV methods, while CCA 'NO-SPEC' allows any method.
4. Turning on the ACP(s) controlling export of PINVER to usage:mode V*:N and import of V*:N to PINGEN at the same time will allow changing PINVER keys to PINGEN keys. This is not recommended. This is possible because legacy (TR-31 2005-based) implementations used the same mode 'N' for PINGEN as well as PINVER keys.

EMV Chip / Issuer Master Keys

Table 3. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs) (continued)

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
E0	A	N	T	DKYL0 +DMAC	DKYGENKY	DKYL0 +DMAC	Permit E0 to DKYGENKY:DKYL0+DMAC
	B,C	X		DKYL0 +DMAC		DKYL0 +DMAC	
	A	N		DKYL0 +DMV		DKYL0 +DMV	Permit E0 to DKYGENKY:DKYL0+DMV
	B,C	X		DKYL0 +DMV		DKYL0 +DMV	
	A	N		DKYL1 +DMAC		DKYL1 +DMAC	Permit E0 to DKYGENKY:DKYL1+DMAC
	B,C	X		DKYL1 +DMAC		DKYL1 +DMAC	
	A	N		DKYL1 +DMV		DKYL1 +DMV	Permit E0 to DKYGENKY:DKYL1+DMV
	B,C	X		DKYL1 +DMV		DKYL1 +DMV	
E1	A	N, E, D, B	T	DKYL0 +DMPIN	DKYGENKY	DKYL0 +DMPIN	Permit E1 to DKYGENKY:DKYL0+DMPIN
	B,C	X		DKYL0 +DMPIN		DKYL0 +DMPIN	
	A	N, E, D, B		DKYL0 +DDATA		DKYL0 +DDATA	Permit E1 to DKYGENKY:DKYL0+DDATA
	B,C	X		DKYL0 +DDATA		DKYL0 +DDATA	
	A	N, E, D, B		DKYL1 +DMPIN		DKYL1 +DMPIN	Permit E1 to DKYGENKY:DKYL1+DMPIN
	B,C	X		DKYL1 +DMPIN		DKYL1 +DMPIN	
	A	N, E, D, B		DKYL1 +DDATA		DKYL1 +DDATA	Permit E1 to DKYGENKY:DKYL1+DDATA
	B,C	X		DKYL1 +DDATA		DKYL1 +DDATA	
E2	A	N	T	DKYL0 +DMAC	DKYGENKY	DKYL0 +DMAC	Permit E2 to DKYGENKY:DKYL0+DMAC
	B,C	X		DKYL0 +DMAC		DKYL0 +DMAC	
	A	N		DKYL1 +DMAC		DKYL1 +DMAC	Permit E2 to DKYGENKY:DKYL1+DMAC
	B,C	X		DKYL1 +DMAC		DKYL1 +DMAC	
E3	A	N, E, D, B, G	T	(none)	ENCIPHER	(none)	Permit E3 to ENCIPHER
	B,C	X		(none)		(none)	

Table 3. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs) (continued)

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
E4	A	N, B	T	(none)	DKYGENKY	DKYL0 +DDATA	Permit E4 to DKYGENKY:DKYL0+DDATA
	B,C	X		(none)		DKYL0 +DDATA	
E5	A	G, C, V, E, D, B, N	T	DKYL0 +DMAC	DKYGENKY	DKYL0 +DMAC	Permit E5 to DKYGENKY:DKYL0+DMAC
	B,C	X		DKYL0 +DMAC		DKYL0 +DMAC	
	A	G, C, V, E, D, B, N		DKYL0 +DDATA		DKYL0 +DDATA	Permit E5 to DKYGENKY:DKYL0+DDATA
	B,C	X		DKYL0 +DDATA		DKYL0 +DDATA	
	A	G, C, V, E, D, B, N		DKYL0 +DEXP		DKYL0 +DEXP	Permit E5 to DKYGENKY:DKYL0+DEXP
	B,C	X		DKYL0 +DEXP		DKYL0 +DEXP	

Note: EMV Chip Card Master 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 are:

- Key Gen Keys of level DKYL0 or DKYL1 allowing derivation of operational keys, or
- operational keys.

EMV support in CCA is significantly different. CCA key types do not match TR-31 types.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 4. TR-31 export required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900		This service is not supported.
IBM @server zSeries 990 IBM @server zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.

Table 4. TR-31 export required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC IBM System z10 BC		This service is not supported.
z196	Crypto Express3 Coprocessor	TR-31 key support requires the Sept. 2011 or later LIC.

TR-31 Optional Data Read (CSNBT31R and CSNET31R)

A TR-31 key block can hold optional fields which are securely bound to the key block using the integrated MAC. The optional blocks may either contain information defined in the TR-31 standard, or they may contain proprietary data. A separate range of optional block identifiers is reserved for use with proprietary blocks.

Note that some of the parameters are only used with keyword INFO and others are only used with keyword DATA.

The callable service name for AMODE(64) is CSNET31R.

Format

```
CSNBT31R(
        return_code,
        reason_code,
        exit_data_length,
        exit_data,
        rule_array_count,
        rule_array,
        TR31_key_block_length,
        TR31_key_block,
        opt_block_id,
        num_opt_blocks,
        opt_block_ids,
        opt_block_lengths,
        opt_block_data_length,
        opt_block_data )
```

Parameters

return_code

Direction: Output　　Type: Integer

The return code specifies the general result of the callable service.

reason_code

Direction: Output　　Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems.

exit_data_length

Direction: Ignored Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

exit_data

Direction: Ignored Type: String

This field is ignored.

rule_array_count

Direction: Input Type: Integer

The number of keywords you are supplying in the *rule_array* parameter. The *rule_array_count* parameter must be 1

rule_array

Direction: Input Type: String

The *rule_array* contains keywords that provide control information to the callable service. 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 callable service are shown in the following table.

Table 5. Keywords for TR-31 Optional Data Read Rule Array Control Information

Keyword	Meaning
Operation – one required	
INFO	Return information about the optional blocks in the TR-31 key block.
DATA	Return the data contained in a specified optional block in the TR-31 key block.

TR31_key_block_length

Direction: Input Type: Integer

This parameter specifies the length of the *TR31_key_block* parameter, in bytes. The parameter may specify a length that is greater than the size of the key block however it can never be greater than the size of the buffer where the key block resides. This value must be between 16 and 9992 inclusive.

TR31_key_block

Direction: Input Type: String

This parameter contains the TR-31 key block that is to be parsed. The length of the TR-31 block is specified using parameter *TR31_key_block_length*.

opt_block_id

Direction: Input Type: String

This parameter is only used with option DATA. It is ignored for others. It specifies a 2-byte string which contains the identifier of the block from which the application is requesting data. The callable service will locate this optional block within the TR-31 structure and copy the data from that optional block into the returned *opt_block_data* buffer. If the specified optional block is not found in the TR-31 key block, an error will occur.

num_opt_blocks

Direction: Input Type: Integer

This parameter specifies the number of optional blocks in the TR-31 key block. The value is compared to the corresponding value in the TR-31 block header and if they do not match the callable service fails with an error. This parameter is only used for option INFO and is not examined for any other options.

opt_block_ids

Direction: Output Type: String Array

This parameter contains an array of two-byte string values. Each of these values is the identifier (ID) of one of the optional blocks contained in the TR-31 key block. The callable service returns a list containing the ID of each optional block that is in the TR-31 block, and the list is in the order that the optional blocks appear in the TR-31 header. The total length of the returned list will be two times the number of optional blocks, and the caller must supply a buffer with a length at least twice the value it passes in parameter n*um_opt_blocks*. This parameter is only used for option INFO and is not examined for any other options.

opt_block_lengths

Direction: Output Type: Array

This parameter contains an array of 16-bit integer values. Each of these values is the length in bytes of one of the optional blocks contained in the TR-31 key block. The callable service returns a list containing the length of each optional block that is in the TR-31 block, and the list is in the order that the optional blocks appear in the TR-31 header. The total length of the returned list will be two times the number of optional blocks and the application program must supply a buffer with a length at least two times the value it passes in parameter *num_opt_blocks*. This parameter is only used for option INFO and is not examined or altered for any other options.

opt_block_data_length

Direction: Input/Output Type: Integer

This parameter specifies the length for parameter opt_block_data. On input it must be set to the length of the buffer provided by the application program, and on output it is updated to contain the length of the returned optional block data, in bytes. It is only used for option DATA.

opt_block_data

Direction: Output Type: String

This parameter contains a buffer where the callable service stores the data it reads from the specified optional block. The buffer must have enough space for the data, as indicated by the input value of parameter opt_block_data_length. If not an error occurs and no changes are made to the contents of the buffer. If the size of the buffer is sufficient, the data is copied to the buffer and its length is stored in parameter opt_block_data_length. It is only used for option DATA and is not examined or altered for any other options.

Restrictions

None

Usage Notes

Unless otherwise noted, all String parameters that are either written to, or read from, a TR-31 key block will be in EBCDIC format. Input parameters are converted to ASCII before being written to the TR-31 key block and output parameters are converted to EBCDIC before being returned . TR-31 key blocks themselves are always in printable ASCII format as required by the ANSI TR-31 specification.

The TR-31 Optional Data Read callable service (CSNBT31R and CSNET31R) can be used in conjunction with the TR-31 Parse callable service (CSNBT31P and CSNET31P) to obtain both the standard header fields and any optional data blocks from the key block. This is generally a three-step process.

1. Use the TR-31 Parse callable service to determine how many optional blocks are in the TR-31 token. This is returned in the *num_opt_blocks* parameter.
2. Use keyword INFO with the TR-31 Optional Data Read callable service to obtain lists of the optional block identifiers and optional block lengths. Your buffers must be large enough to hold the returned data, but the required size can be determined from the number of blocks obtained in the step above.
3. Use keyword DATA with the TR-31 Optional Data Read callable service to obtain the data for a particular optional block, specified by the block identifier.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 6. TR-31 Optional Data Read required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None	
IBM @server zSeries 990 IBM @server zSeries 890	None	
IBM System z9 EC IBM System z9 BC	None	
IBM System z10 EC IBM System z10 BC	None	
z196	None	