z/OSCryptographic Services
Integrated Cryptographic Service Facility

PKCS #11 Enhancements for IPsec and Large Keys – APAR OA34403

(May 16, 2011)

Contents

Chapter 1. Overview

This document update describes PKCS #11 enhancements that provide PKCS #11 derived key and RSA signature verify capabilities. This document also describes support for larger DSA and DH keys, and contains alterations to information previously presented in the following books:

- *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*, SA23-2231-02
- *z/OS Cryptographic Services ICSF Application Programmer's Guide*, SA22-7522-12
- *z/OS Cryptographic Services ICSF System Programmer's Guide*, SA22-7520-15

The preceding books document capabilities provided by FMID HCR7780, and support z/OS Version 1 Release 12.

Technical changes or additions related to the PKCS #11 enhancements in this document update are indicated by a vertical line to the left of the change.

These updates relate to the enhancements made to the ICSF product by the application of the PTF for APAR OA34403.

Systems running with ICSF FMID HCR7770 that will share a TKDS with an HCR7780 system that has the PTF for APAR OA34403 applied and is using the new support for larger DSA and DH keys, need to have coexistence support enabled. This coexistence support can be enabled with the PTF for APAR OA34404.

Chapter 2. Update of z/OS Cryptographic Services ICSF Writing PKCS #11 Applications, SA23-2231-03, information

This chapter contains updates to the document *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*, SA23-2231-03, for the PKCS #11 enhancements provided by the PTF for APAR OA34403. Refer to this source document if background information is needed.

Key types and mechanisms supported

ICSF supports the following PKCS #11 key types (CK_KEY_TYPE). All of these key types are supported in software. Whether they are also supported in hardware will depend on the limitations of your cryptographic hardware configuration.

- CKK_AES - key lengths 128, 192, and 256 bits
- CKK_BLOWFISH - key lengths 8 up to 448 bits (in increments of 8 bits)
- CKK_DES
- CKK_DES2
- CKK_DES3
- CKK_DH - key lengths 512 up to 2048 bits (in increments of 64 bits)
- CKK_DSA - key lengths 512 up to 2048 bit prime lengths (in increments of 64 bits)
- CKK_EC (CKK_ECDSA) - key lengths 160 up to 521 bits
- CKK_GENERIC_SECRET - key lengths 8 up to 2048 bits, unless further restricted by the generation mechanism:
  - CKM_DH_PKCS_DERIVE - key lengths 512 up to 2048 bits
  - CKM_SSL3_MASTER_KEY_DERIVE - 384-bit key lengths
  - CKM_SSL3_MASTER_KEY_DERIVE_DH - 384-bit key lengths
  - CKM_SSL3_PRE_MASTER_KEY_GEN - 384-bit key lengths
  - CKM_TLS_MASTER_KEY_DERIVE - 384-bit key lengths
  - CKM_TLS_MASTER_KEY_DERIVE_DH - 384-bit key lengths
  - CKM_TLS_PRE_MASTER_KEY_GEN - 384-bit key lengths
- CKK_RC4 - key lengths 8 up to 2048 bits
- CKK_RSA - key lengths 512 up to 4096 bits

The following table shows the mechanisms supported by different hardware configurations. All the mechanisms are supported in software, and some may be available in hardware. If the mechanism is available in hardware, ICSF will use the hardware mechanism. If the mechanism is not available in hardware, ICSF will use the software mechanism. The following table also shows the flags returned by the C_GetMechanismInfo function in the CK_MECHANISM_INFO structure. Whether or not the CKF_HW flag is returned in the CK_MECHANISM_INFO structure indicates whether or not the mechanism is supported in the hardware.

Table 1. Mechanism information as returned by C_GetMechanismInfo (CK_MECHANISM_INFO)

Type (CK_MECHANISM_TYPE)	Size factor	Flags
CKM_RSA_PKCS_KEY_PAIR_GEN	Bits	[CKF_HW] CKF_GENERATE_KEY_PAIR
CKM_DES_KEY_GEN	not applicable	[CKF_HW] CKF_GENERATE

Table 1. Mechanism information as returned by C_GetMechanismInfo (CK_MECHANISM_INFO) (continued)

Type (CK_MECHANISM_TYPE)	Size factor	Flags
CKM_DES2_KEY_GEN	not applicable	[CKF_HW] CKF_GENERATE
CKM_DES3_KEY_GEN	not applicable	[CKF_HW] CKF_GENERATE
CKM_RSA_PKCS[6]	Bits	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_WRAP CKF_UNWRAP CKF_SIGN CKF_VERIFY CKF_SIGN_RECOVER CKF_VERIFY_RECOVER
CKM_RSA_X_509[6, 7]	Bits	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_SIGN CKF_VERIFY CKF_SIGN_RECOVER CKF_VERIFY_RECOVER
CKM_MD2_RSA_PKCS[6, 7]	Bits	CKF_SIGN CKF_VERIFY
CKM_MD5_RSA_PKCS[6, 7]	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA1_RSA_PKCS[6, 7]	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA224_RSA_PKCS[6, 7]	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA256_RSA_PKCS[6, 7]	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA384_RSA_PKCS[6, 7]	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA512_RSA_PKCS[6, 7]	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_DES_ECB[3]	not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_DES_CBC	not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_DES_CBC_PAD	not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_WRAP CKF_UNWRAP
CKM_DES3_ECB[3, 4]	not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_DES3_CBC[4]	not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_DES3_CBC_PAD[4]	not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_WRAP CKF_UNWRAP
CKM_SHA_1	not applicable	[CKF_HW] CKF_DIGEST
CKM_SHA224	not applicable	[CKF_HW] CKF_DIGEST
CKM_SHA256	not applicable	[CKF_HW] CKF_DIGEST
CKM_SHA384	not applicable	[CKF_HW] CKF_DIGEST
CKM_SHA512	not applicable	[CKF_HW] CKF_DIGEST
CKM_RIPEMD160	not applicable	CKF_DIGEST
CKM_MD2	not applicable	CKF_DIGEST
CKM_MD5	not applicable	[CKF_HW] CKF_DIGEST
CKM_AES_KEY_GEN	Bytes	[CKF_HW] CKF_GENERATE
CKM_AES_ECB[4]	Bytes	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_AES_CBC[4]	Bytes	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_AES_CBC_PAD[4]	Bytes	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_WRAP CKF_UNWRAP
CKM_AES_GCM[4]	Bytes	CKF_ENCRYPT CKF_DECRYPT
CKM_DSA_KEY_PAIR_GEN	Bits	CKF_GENERATE_KEY_PAIR
CKM_DH_PKCS_KEY_PAIR_GEN	Bits	CKF_GENERATE_KEY_PAIR

Table 1. Mechanism information as returned by C_GetMechanismInfo (CK_MECHANISM_INFO) (continued)

Type (CK_MECHANISM_TYPE)	Size factor	Flags
CKM_EC_KEY_PAIR_GEN	Bits	CKF_GENERATE_KEY_PAIR CKF_EC_F_P[1] CKF_EC_NAMEDCURVE[2] CKF_EC_UNCOMPRESS
CKM_DSA_PARAMETER_GEN	Bits	CKF_GENERATE
CKM_DH_PKCS_PARAMETER_GEN	Bits	CKF_GENERATE
CKM_BLOWFISH_KEY_GEN	Bytes	[CKF_HW] CKF_GENERATE
CKM_RC4_KEY_GEN	Bits	[CKF_HW] CKF_GENERATE
CKM_SSL3_PRE_MASTER_KEY_GEN	Bytes	[CKF_HW] CKF_GENERATE
CKM_TLS_PRE_MASTER_KEY_GEN	Bytes	[CKF_HW] CKF_GENERATE
CKM_GENERIC_SECRET_KEY_GEN	Bits	[CKF_HW] CKF_GENERATE
CKM_BLOWFISH_CBC[5]	Bytes	CKF_ENCRYPT CKF_DECRYPT
CKM_RC4[5]	Bits	CKF_ENCRYPT CKF_DECRYPT
CKM_DSA_SHA1	Bits	CKF_SIGN CKF_VERIFY
CKM_DSA	Bits	CKF_SIGN CKF_VERIFY
CKM_ECDSA_SHA1	Bits	CKF_SIGN CKF_VERIFY CKF_EC_F_P[1] CKF_EC_NAMEDCURVE[2] CKF_EC_UNCOMPRESS
CKM_ECDSA	Bits	CKF_SIGN CKF_VERIFY CKF_EC_F_P[1] CKF_EC_NAMEDCURVE[2] CKF_EC_UNCOMPRESS
CKM_MD5_HMAC	not applicable	CKF_SIGN CKF_VERIFY
CKM_SHA_1_HMAC	not applicable	CKF_SIGN CKF_VERIFY
CKM_SHA224_HMAC	not applicable	CKF_SIGN CKF_VERIFY
CKM_SHA256_HMAC	not applicable	CKF_SIGN CKF_VERIFY
CKM_SHA384_HMAC	not applicable	CKF_SIGN CKF_VERIFY
CKM_SHA512_HMAC	not applicable	CKF_SIGN CKF_VERIFY
CKM_SSL3_MD5_MAC	Bits	CKF_SIGN CKF_VERIFY
CKM_SSL3_SHA1_MAC	Bits	CKF_SIGN CKF_VERIFY
CKM_DH_PKCS_DERIVE	Bits	CKF_DERIVE
CKM_ECDH1_DERIVE	Bits	CKF_DERIVE CKF_EC_F_P[1] CKF_EC_NAMEDCURVE[2] CKF_EC_UNCOMPRESS
CKM_SSL3_MASTER_KEY_DERIVE	Bytes	CKF_DERIVE
CKM_SSL3_MASTER_KEY_DERIVE_DH	Bytes	CKF_DERIVE
CKM_SSL3_KEY_AND_MAC_DERIVE	not applicable	CKF_DERIVE
CKM_TLS_MASTER_KEY_DERIVE	Bytes	CKF_DERIVE
CKM_TLS_MASTER_KEY_DERIVE_DH	Bytes	CKF_DERIVE
CKM_TLS_KEY_AND_MAC_DERIVE	not applicable	CKF_DERIVE
CKM_TLS_PRF	not applicable	CKF_DERIVE

Footnotes for table Table 1 on page 3.

[1] The PKCS11 standard designates two ways of implementing Elliptic Curve Cryptography, nicknamed F_p and F_{2^m}. z/OS PKCS11 supports the F_p variety only.

[2] ANSI X9.62 has the following ASN.1 definition for Elliptic Curve domain parameters:

```
Parameters ::= CHOICE {
  ecParameters  ECParameters,
  namedCurve    OBJECT IDENTIFIER,
  implicitlyCA  NULL }
```

z/OS PKCS11 supports the specification of CKA_EC_PARAMS attribute using the namedCurved CHOICE. The following NIST-recommended named curves are supported:

- secp192r1 – { 1 2 840 10045 3 1 1 }
- secp224r1 – { 1 3 132 0 33 }
- secp256r1 – { 1 2 840 10045 3 1 7 }
- secp384r1 – { 1 3 132 0 34 }
- secp521r1 – { 1 3 132 0 35 }

The following Brainpool-defined named curves are supported:

- brainpoolP160r1 – { 1 3 36 3 3 2 8 1 1 1 }
- brainpoolP192r1 – { 1 3 36 3 3 2 8 1 1 3 }
- brainpoolP224r1 – { 1 3 36 3 3 2 8 1 1 5 }
- brainpoolP256r1 – { 1 3 36 3 3 2 8 1 1 7 }
- brainpoolP320r1 – { 1 3 36 3 3 2 8 1 1 9 }
- brainpoolP384r1 – { 1 3 36 3 3 2 8 1 1 11 }
- brainpoolP512r1 – { 1 3 36 3 3 2 8 1 1 13 }

In addition, z/OS PKCS11 has limited support for the ecParameters CHOICE. When specified, the DER encoding must contain the optional cofactor field and must not contain the optional Curve.seed field. Also, calls to C_GetAttributeValue to retrieve the CKA_EC_PARAMS attribute will always return the value in the namedCurve form regardless of how the attribute was specified when the object was created. Due to these limitations, the CKF_EC_ECPARAMETERS flag is not turned on for the applicable mechanisms.

[3] Mechanism not present on a CCF system.

[4] Mechanism not present on a system that is export controlled.

[5] Mechanism limited to 56-bit on a system that is export controlled.

[6] In general, z/OS PKCS #11 expects RSA private keys to be in Chinese Remainder Theorem (CRT) format. However, for Decrypt, Sign, or UnwrapKey (z890, z990 or higher only) where one of the following is true, the shorter Modulus Exponent (ME) is permitted:

- There is an accelerator present and the key is less than or equal to 2048 bits in length.
- There is a coprocessor present and the key is less than or equal to 1024 bits in length and FIPS restrictions don’t apply.

[7] RSA public or private keys that have a public exponent greater than 8 bytes in length, or a modulus that has an odd number of bits, can only be used when an

accelerator is present or a coprocessor is present and FIPS restrictions don't apply. If only an accelerator is present, the key must be less than or equal to 2048 bits in length.

The following table lists the mechanisms supported by specific cryptographic hardware. When a particular mechanism is not available in hardware, ICSF will use the software implementation of the mechanism.

Table 2. Mechanisms supported by specific cryptographic hardware

Machine type and cryptographic hardware	**Mechanisms supported**	**Notes**
z800, z900 - CCF	CKM_DES_KEY_GEN CKM_DES2_KEY_GEN CKM_DES3_KEY_GEN CKM_RSA_PKCS CKM_RSA_X_509 CKM_MD5_RSA_PKCS CKM_SHA1_RSA_PKCS CKM_DES_CBC CKM_DES_CBC_PAD CKM_DES3_CBC CKM_DES3_CBC_PAD CKM_SHA_1 CKM_BLOWFISH_KEY_GEN CKM_RC4_KEY_GEN CKM_AES_KEY_GEN CKM_SSL3_PRE_MASTER_KEY_GEN CKM_TLS_PRE_MASTER_KEY_GEN CKM_GENERIC_SECRET_KEY_GEN	This is the base set. RSA private key operations limited to 1024 bits in length (maximum) and no key pair generation capability.
z800, z900 - PCICC	Base set plus: CKM_RSA_PKCS_KEY_PAIR_GEN	RSA private key operations limited to 2048 bits in length (maximum).
z890, z990 - PCIXCC	Base set plus: CKM_RSA_PKCS_KEY_PAIR_GEN CKM_DES_ECB CKM_DES3_ECB	RSA private key operations limited to 2048 bits in length (maximum).
z890, z990 - CEX2C	Base set plus: CKM_RSA_PKCS_KEY_PAIR_GEN CKM_DES_ECB CKM_DES3_ECB	RSA private key operations limited to 2048 bits in length (maximum).
z9® - CEX2C	Base set plus: CKM_RSA_PKCS_KEY_PAIR_GEN CKM_DES_ECB CKM_DES3_ECB CKM_SHA224_RSA_PKCS CKM_SHA256_RSA_PKCS CKM_SHA224 CKM_SHA256 CKM_AES_CBC CKM_AES_CBC_PAD CKM_AES_ECB	AES key operations limited to 128 bits in length (maximum). RSA private key operations limited to 4096 bits in length (maximum).
z10 - CEX2C or CEX3C	z9 CEX2C set plus: CKM_SHA384_RSA_PKCS CKM_SHA512_RSA_PKCS CKM_SHA384 CKM_SHA512	AES key operations limited to 256 bits in length (maximum). RSA private key operations limited to 4096 bits in length (maximum).

Table 2. Mechanisms supported by specific cryptographic hardware (continued)

Machine type and cryptographic hardware	Mechanisms supported	Notes
z196 - CEX3C	z9 CEX2C set plus: CKM_SHA384_RSA_PKCS CKM_SHA512_RSA_PKCS CKM_SHA384 CKM_SHA512	AES key operations limited to 256 bits in length (maximum). RSA private key operations limited to 4096 bits in length (maximum).

The following table lists the algorithms and uses (by mechanism) that are not allowed when operating in compliance with FIPS 140-2.

Table 3. Restricted algorithms and uses when running in compliance with FIPS 140-2

Algorithm	Mechanisms	Usage disallowed
RIPEMD	CKM_RIPEMD160	All
MD2	CKM_MD2, CKM_MD2_RSA_PKCS	All
MD5	CKM_MD5, CKM_MD5_RSA_PKCS, CKM_MD5_HMAC	All
SSL3	CKM_SSL3_MD5_MAC, CKM_SSL3_SHA1_MAC, CKM_SSL3_MASTER_KEY_DERIVE, CKM_SSL3_MASTER_KEY_DERIVE_DH, CKM_SSL3_KEY_AND_MAC_DERIVE	All
TLS	CKM_TLS_MASTER_KEY_DERIVE, CKM_TLS_MASTER_KEY_DERIVE_DH, CKM_TLS_KEY_AND_MAC_DERIVE	Base key sizes less than 10 bytes
Diffie Hellman	CKM_DH_PKCS_DERIVE	Prime size less than 1024 bits
	CKM_DH_PKCS_PARAMETER_GEN	Prime sizes other than 1024 or 2048 bits
DSA	CKM_DSA_SHA1, CKM_DSA	Prime sizes less than 1024 bits
	CKM_DSA_PARAMETER_GEN, CKM_DSA_KEY_PAIR_GEN or Sign	Combinations other than the following: • Prime size = 1024 bits, subprime size = 160 bits • Prime size = 2048 bits, subprime size = 224 bits, or 256 bits
Single DES	CKM_DES_ECB, CKM_DES_CBC, CKM_DES_CBC_PAD	All
Triple DES	CKM_DES3_ECB, CKM_DES3_CBC, CKM_DES3_CBC_PAD	Two key Triple DES
Blowfish	CKM_BLOWFISH_KEY_GEN, CKM_BLOWFISH_CBC	All
RC4	CKM_RC4	All
RSA	CKM_RSA_X_509	All
	CKM_RSA_PKCS	Key sizes less than 1024 bits
	CKM_RSA_PKCS_KEY_PAIR_GEN or Sign without an active accelerator	Key sizes that are less than 1024 bits or not a multiple of 256 bits or public key exponents less than 0x010001

Table 3. Restricted algorithms and uses when running in compliance with FIPS 140-2 (continued)

Algorithm	Mechanisms	Usage disallowed
ECC	CKM_ECDSA, CKM_ECDSA_SHA1, CKM_ECDH1_DERIVE	Brainpool curves
HMAC	CKM_SHA_1, CKM_SHA224, CKM_SHA256, CKM_SHA384, CKM_SHA512	Base key sizes less than one half the output size
AES GCM	CKM_AES_GCM	GCM encryption or GMAC generation with externally generated initialization vectors. Initialization vector lengths other than 12 bytes. Tag byte sizes 4 and 8

Objects and attributes supported

ICSF supports the following PKCS #11 object types (CK_OBJECT_CLASS):

- CKO_DATA
- CKO_CERTIFICATE - CKC_X_509 only
- CKO_DOMAIN_PARAMETERS - CKK_DSA and CKK_DH only
- CKO_PUBLIC_KEY - CKK_RSA, CKK_EC (CKK_ECDSA), CKK_DSA, and CKK_DH only
- CKO_PRIVATE_KEY - CKK_RSA, CKK_EC (CKK_ECDSA), CKK_DSA, and CKK_DH only
- CKO_SECRET_KEY - CKK_DES, CKK_DES2, CKK_DES3, CKK_AES, CKK_BLOWFISH, and CKK_RC4, CKK_GENERIC_SECRET only

The footnotes described in Table 4 are taken from the PKCS #11 specification and apply to the attribute tables that follow.

Table 4. Common footnotes for object attribute tables

Footnote number	Footnote meaning
1	Must be specified when object is created with **C_CreateObject**.
2	Must *not* be specified when object is created with **C_CreateObject**.
3	Must be specified when object is generated with **C_GenerateKey** or **C_GenerateKeyPair**.
4	Must *not* be specified when object is generated with **C_GenerateKey** or **C_GenerateKeyPair**.
5	Must be specified when object is unwrapped with **C_UnwrapKey**.
6	Must *not* be specified when object is unwrapped with **C_UnwrapKey**.
7	Cannot be revealed if object has its **CKA_SENSITIVE** attribute set to TRUE or its **CKA_EXTRACTABLE** attribute set to FALSE.
8	May be modified after object is created with a **C_SetAttributeValue** call, or in the process of copying object with a **C_CopyObject** call. However, it is possible that a particular token may not permit modification of the attribute, or may not permit modification of the attribute during the course of a **C_CopyObject** call.
9	Default value is token-specific, and may depend on the values of other attributes.
10	Can only be set to TRUE by the SO user.
11	May be changed during a **C_CopyObject** call but not on a **C_SetAttributeValue** call

Table 5. Data object attributes that ICSF supports. For the meanings of the footnotes, see Table 4 on page 9.

Attribute	**Data type**	**Notes**
CKA_CLASS[1]	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.
CKA_TOKEN[11]	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE. An application can specify the value when the object is created (or generated) only.
CKA_PRIVATE[11]	CK_BBOOL	Default value on create is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_MODIFIABLE[11]	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_LABEL	Printable EBCDIC string	Application-specific nickname. Limited to 32 characters. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
CKA_ID	Byte array	Key or other identifier. Default is empty. An application can set or change the value at any time.
CKA_VALUE	Byte array	Any value. Default is empty. An application can set or change the value at any time.
CKA_APPLICATION	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
CKA_OBJECT_ID	Byte array	DER-encoded OID. Default is empty. An application can set or change the value at any time.

Table 6. X.509 certificate object attributes that ICSF supports. For the meanings of the footnotes, see Table 4 on page 9.

Attribute	**Data type**	**Notes**
CKA_CLASS[1]	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.
CKA_TOKEN[11]	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE. An application can specify the value when the object is created (or generated) only.

Table 6. X.509 certificate object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_PRIVATE[11]	CK_BBOOL	Default value on create is FALSE. An application can specify the value when the object is created (or generated) only.
CKA_MODIFIABLE[11]	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_LABEL	Printable EBCDIC string	Application-specific nickname. Limited to 32 characters. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
CKA_CERTIFICATE_TYPE	CK_CERTIFICATE_TYPE	Always CKC_X_509. An application can specify the value when the object is created (or generated) only.
CKA_TRUSTED	CK_BBOOL	Always set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_SUBJECT	Byte array	DER-encoding as found in certificate. If not specified, ICSF sets it from the certificate. If specified, ICSF enforces that it matches the subject in the certificate. An application can specify the value when the object is created (or generated) only.
CKA_ID	Byte array	Key identifier. Default is empty. An application can set or change the value at any time.
CKA_ISSUER	Byte array	DER-encoding as found in certificate. If not specified, ICSF sets from the certificate. If specified, ICSF enforces that it matches the issuer in the certificate An application can specify the value when the object is created (or generated) only.
CKA_SERIAL_NUMBER	Byte array	DER-encoding as found in certificate. If not specified, ICSF sets from the certificate. If specified, ICSF enforces that it matches the serial number in the certificate. An application can specify the value when the object is created (or generated) only.
CKA_VALUE	Byte array	This is the DER-encoding of the certificate. (Required.) An application can specify the value when the object is created (or generated) only.

Table 6. X.509 certificate object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_CERTIFICATE_CATEGORY	CK_ULONG	Categorization of the certificate: **1** Token user **2** Certificate authority **3** Other entity If not specified, ICSF sets it to 2 if the certificate has the BasicConstraints CA flag on. Otherwise it is not set. **Note:** If specified (or defaulted) to 2, the certificate is considered a CA certificate. The user must have appropriate authority. An application can set or change the value at any time.
CKA_APPLICATION	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page. An application can specify the value when the object is created (or generated) only.
CKA_IBM_DEFAULT (vendor specific attribute - 0x80000002)	CK_BBOOL	Default flag. Default is FALSE. An application can set or change the value at any time.

Table 7. Secret key object attributes that ICSF supports. For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_CLASS[1]	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.
CKA_TOKEN[11]	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE. An application can specify the value when the object is created (or generated) only.
CKA_PRIVATE[11]	CK_BBOOL	Default value on create is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_MODIFIABLE[11]	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_LABEL	Printable EBCDIC string	Application-specific nickname. Limited to 32 characters. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
CKA_ID	Byte array	Default is empty. An application can set or change the value at any time.

Table 7. Secret key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_KEY_TYPE[1, 5]	CK_KEY_TYPE	Type of key: CKK_DES, CKK_DES2, CKK_DES3, CKK_BLOWFISH, CKK_RC4, CKK_GENERIC_SECRET, or CKK_AES. An application can specify the value when the object is created (or generated) only.
CKA_START_DATE[8]	CK_DATE	Start date for the key. Default is empty. An application can set or change the value at any time.
CKA_END_DATE[8]	CK_DATE	End date for the key. Default is empty. An application can set or change the value at any time.
CKA_DERIVE[8]	CK_BBOOL	TRUE if key supports key derivation (other keys can be derived from this one). Default is TRUE. An application can set or change the value at any time.
CKA_LOCAL[2, 4, 6]	CK_BBOOL	TRUE only if key was generated locally. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_GEN_MECHANISM[2, 4, 6]	CK_MECHANISM_TYPE	Identifier of the mechanism used to generate the key. Always CK_UNAVAILABLE_INFORMATION. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_ENCRYPT[8]	CK_BBOOL	TRUE if key supports encryption[9]. Default is TRUE. An application can set or change the value at any time.
CKA_VERIFY[8]	CK_BBOOL	TRUE if key supports verification where the signature is an appendix to the data. Default is TRUE. An application can set or change the value at any time.
CKA_WRAP[8]	CK_BBOOL	TRUE if key supports wrapping (can be used to wrap other keys).[9] Default is TRUE. An application can set or change the value at any time.
CKA_DECRYPT[8]	CK_BBOOL	TRUE if key supports decryption.[9] Default is TRUE. An application can set or change the value at any time.
CKA_SIGN[8]	CK_BBOOL	TRUE if key supports signatures where the signature is an appendix to the data.[9] Default is TRUE. An application can set or change the value at any time.

Table 7. Secret key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_UNWRAP[8]	CK_BBOOL	TRUE if key supports unwrapping (can be used to unwrap other keys)[9]. Default is TRUE. An application can set or change the value at any time.
CKA_EXTRACTABLE[8]	CK_BBOOL	TRUE if key is extractable. Caller can change from TRUE to FALSE only. Default is TRUE. An application can set or change the value, as per PKCS #11 restrictions.
CKA_SENSITIVE[8]	CK_BBOOL	TRUE if key is sensitive. Caller can change from FALSE to TRUE only. Default is FALSE. An application can set or change the value, as per PKCS #11 restrictions.
CKA_ALWAYS_SENSITIVE[2, 4, 6]	CK_BBOOL	TRUE if key has always had the CKA_SENSITIVE attribute set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_NEVER_EXTRACTABLE[2, 4, 6]	CK_BBOOL	TRUE if key has never had the CKA_EXTRACTABLE attribute set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_VALUE[1, 4, 6, 7]	Byte array	The key. Sensitive key part. An application can specify the value when the object is created (or generated) only.
CKA_VALUE_LEN[2, 3]	CK_ULONG	Length of the key in bytes (AES, Blowfish, RC4, and Generic secret keys only). An application can specify the value when the object is generated only.
CKA_APPLICATION	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page. An application can specify the value when the object is created (or generated) only.
CKA_IBM_FIPS140 (vendor specific attribute 0x80000005)	CK_BBOOL	TRUE if the key must only be used in a FIPS 140-2 compliant fashion. The default value is FALSE. An application can specify the value when the object is created (or generated) only.

Table 8. Public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_CLASS[1]	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.
CKA_TOKEN[11]	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE. An application can specify the value when the object is created (or generated) only.
CKA_PRIVATE[11]	CK_BBOOL	Default value on create is FALSE. An application can specify the value when the object is created (or generated) only.
CKA_MODIFIABLE[11]	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_LABEL	Printable EBCDIC string	Application-specific nickname. Limited to 32 characters. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
CKA_TRUSTED	CK_BBOOL	Always set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_SUBJECT	Byte array	DER-encoding. Default empty. An application can set or change the value at any time.
CKA_ID	Byte array	Key identifier. Default empty. An application can set or change the value at any time.
CKA_KEY_TYPE[1, 5]	CK_KEY_TYPE	Type of key. CKK_RSA, CKK_EC, CKK_DSA, and CKK_DH only. An application can specify the value when the object is created (or generated) only.
CKA_START_DATE[8]	CK_DATE	Start date for the key. Default empty. An application can set or change the value at any time.
CKA_END_DATE[8]	CK_DATE	End date for the key. Default empty. An application can set or change the value at any time.
CKA_DERIVE[8]	CK_BBOOL	TRUE if key supports key derivation (if other keys can be derived from this one). Default is TRUE. An application can set or change the value at any time.

Table 8. Public key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_LOCAL[2, 4, 6]	CK_BBOOL	TRUE only if key was generated locally. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_KEY_GEN_MECHANISM[2, 4, 6]	CK_MECHANISM_TYPE	Identifier of the mechanism used to generate the key. Always CK_UNAVAILABLE_INFORMATION. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_ENCRYPT[8]	CK_BBOOL	TRUE if key supports encryption.[9] Default is TRUE. An application can set or change the value at any time.
CKA_VERIFY[8]	CK_BBOOL	TRUE if key supports verification where the signature is an appendix to the data. Default is TRUE. An application can set or change the value at any time.
CKA_VERIFY_RECOVER[8]	CK_BBOOL	TRUE if key supports verification where the data is recovered from the signature.[9] Default is TRUE. An application can set or change the value at any time.
CKA_WRAP[8]	CK_BBOOL	TRUE if key supports wrapping (can be used to wrap other keys).[9] Default is TRUE. An application can set or change the value at any time.
CKA_APPLICATION	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page. An application can specify the value when the object is created (or generated) only.
CKA_IBM_FIPS140 (vendor specific attribute 0x80000005)	CK_BBOOL	TRUE if the key must only be used in a FIPS 140-2 compliant fashion. The default value is FALSE. An application can specify the value when the object is created (or generated) only.

Table 9. RSA public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_MODULUS[1, 4]	Big integer	Modulus *n* An application can specify the value when the object is created (or generated) only.
CKA_MODULUS_BITS[2, 3]	CK_ULONG	Length in bits of modulus *n* An application can specify the value when the object is created (or generated) only.

Table 9. RSA public key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_PUBLIC_EXPONENT[1]	Big integer	Public exponent e An application can specify the value when the object is created (or generated) only.

Table 10. DSA public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_PRIME[1,3]	Big integer	Prime p (512 to 2048 bits in steps of 64 bits)
CKA_SUBPRIME[1,3]	Big integer	Subprime q (160 bits for p <= 1024 bits, 224 bits or 256 bits for p > 1024 bits)
CKA_BASE[1,3]	Big integer	Base g
CKA_VALUE[1,4]	Big integer	Public value y

Table 11. Diffie-Hellman public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_PRIME[1,3]	Big integer	Prime p (512 to 2048 bits in steps of 64 bits)
CKA_BASE[1,3]	Big integer	Base g
CKA_VALUE[1,4]	Big integer	Public value y

Table 12. Elliptic Curve public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_EC_PARAMS[1,3] (CKA_ECDSA_PARAMS)	Byte Array	DER-encoding of an ANSI X9.62 Parameters value
CKA_EC_POINT[1,4]	Byte Array	DER-encoding of an ANSI X9.62 ECPoint value Q

Table 13. Private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_CLASS[1]	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.
CKA_TOKEN[11]	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE. An application can specify the value when the object is created (or generated) only.
CKA_PRIVATE[11]	CK_BBOOL	Default value on create is TRUE. An application can specify the value when the object is created (or generated) only.

Table 13. Private key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_MODIFIABLE[11]	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_LABEL	Printable EBCDIC string	Application-specific nickname. Limited to 32 characters. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
CKA_SUBJECT	Byte array	DER-encoding. An application can set or change the value at any time.
CKA_ID	Byte array	Default is empty. An application can set or change the value at any time.
CKA_KEY_TYPE[1, 5]	CK_KEY_TYPE	Type of key. CKK_EC, CKK_RSA, CKK_DSA, and CKK_DH only. An application can specify the value when the object is created (or generated) only.
CKA_START_DATE[8]	CK_DATE	Start date for the key. Default empty. An application can set or change the value at any time.
CKA_END_DATE[8]	CK_DATE	End date for the key. Default empty. An application can set or change the value at any time.
CKA_DERIVE[8]	CK_BBOOL	TRUE if key supports key derivation (if other keys can be derived from this one). Default is TRUE. An application can set or change the value at any time.
CKA_LOCAL[2, 4 ,6]	CK_BBOOL	TRUE only if key was generated locally. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_KEY_GEN_ MECHANISM[2, 4, 6]	CK_MECHANISM_TYPE	Identifier of the mechanism used to generate the key material. Always CK_UNAVAILABLE_INFORMATION. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_DECRYPT[8]	CK_BBOOL	TRUE if key supports decryption.[9] Default is TRUE. An application can set or change the value at any time.

Table 13. Private key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_SIGN[8]	CK_BBOOL	TRUE if key supports signatures where the signature is an appendix to the data.[9] Default is TRUE. An application can set or change the value at any time.
CKA_SIGN_RECOVER[8]	CK_BBOOL	TRUE if key supports signatures where the data can be recovered from the signature.[9] Default is TRUE. An application can set or change the value at any time.
CKA_UNWRAP[8]	CK_BBOOL	TRUE if key supports unwrapping (can be used to unwrap other keys).[9] Default is TRUE. An application can set or change the value at any time.
CKA_EXTRACTABLE[8]	CK_BBOOL	TRUE if key is extractable. Default is TRUE. An application can set or change the value, as per PKCS #11 restrictions. Caller can change from TRUE to FALSE only.
CKA_SENSITIVE[8]	CK_BBOOL	TRUE if key is sensitive. Default is FALSE. An application can set or change the value, as per PKCS #11 restrictions. Caller can change from FALSE to TRUE only.
CKA_ALWAYS_SENSITIVE[2,4, 6]	CK_BBOOL	TRUE if key has always had the CKA_SENSITIVE attribute set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_NEVER_EXTRACTABLE[2,4, 6]	CK_BBOOL	TRUE if key has never had the CKA_EXTRACTABLE attribute set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_APPLICATION	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page. An application can specify the value when the object is created (or generated) only.
CKA_IBM_FIPS140 (vendor specific attribute 0x80000005)	CK_BBOOL	TRUE if the key must only be used in a FIPS 140-2 compliant fashion. The default value is FALSE. An application can specify the value when the object is created (or generated) only.

Table 14. RSA private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_MODULUS[1, 4, 6]	Big integer	Modulus n An application can specify the value when the object is created (or generated) only.
CKA_PUBLIC_EXPONENT[4, 6]	Big integer	Public exponent e An application can specify the value when the object is created (or generated) only.
CKA_PRIVATE_EXPONENT[1, 4,6 ,7]	Big integer	Private exponent d Sensitive key part. An application can specify the value when the object is created (or generated) only.
CKA_PRIME_1[4, 6, 7]	Big integer	Prime p Sensitive key part. An application can specify the value when the object is created (or generated) only.
CKA_PRIME_2[4, 6, 7]	Big integer	Prime q Sensitive key part. An application can specify the value when the object is created (or generated) only.
CKA_EXPONENT_1[4, 6, 7]	Big integer	Private exponent d modulo p-1 Sensitive key part. An application can specify the value when the object is created (or generated) only.
CKA_EXPONENT_2[4, 6, 7]	Big integer	Private exponent d modulo q-1 Sensitive key part. An application can specify the value when the object is created (or generated) only.
CKA_COEFFICIENT[4, 6, 7]	Big integer	CRT coefficient q^{-1} mod p Sensitive key part. An application can specify the value when the object is created (or generated) only.

Table 15. DSA private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_PRIME[1,4,6]	Big integer	Prime p (512 to 2048 bits in steps of 64 bits)
CKA_SUBPRIME[1,4,6]	Big integer	Subprime q (160 bits for p <= 1024 bits, 224 bits or 256 bits for p > 1024 bits)
CKA_BASE[1,4,6]	Big integer	Base g
CKA_VALUE[1,4,6,7]	Big integer	Private value x

Table 16. Diffie-Hellman private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_PRIME[1,4,6]	Big integer	Prime p (512 to 2048 bits in steps of 64 bits)
CKA_BASE[1,4,6]	Big integer	Base g
CKA_VALUE[1,4,6,7]	Big integer	Private value x
CKA_VALUE_BITS[2,6]	CK_ULONG	Length in bits of private value x. For non-FIPS or when prime bit size = 1024, the default is 160. For FIPS prime bit size = 2048, the default is 256

Table 17. Elliptic Curve private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_EC_PARAMS[1,4,6] (CKA_ECDSA_PARAMS)	Byte Array	DER-encoding of an ANSI X9.62 Parameters value
CKA_VALUE[1,4,6,7]	Big integer	ANSI X9.62 private value d

Table 18. Domain parameter object attributes that ICSF supports. For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_CLASS[1]	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.
CKA_TOKEN[11]	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE.
CKA_PRIVATE[11]	CK_BBOOL	Default value on create is FALSE
CKA_MODIFIABLE[11]	CK_BBOOL	Default value is TRUE
CKA_LABEL	Printable EBCDIC string	Application specific nickname. Limit to 32 chars. Default is empty. The string is assumed to come from the IBM1047 code page.
CKA_KEY_TYPE[1]	CK_KEY_TYPE	Type of key the domain parameters can be used to generate. CKK_DSA and CKK_DH only in this release
CKA_LOCAL[2,4]	CK_BBOOL	TRUE only if the parameters were generated locally
CKA_APPLICATION	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page.

Table 19. DSA domain parameter object attributes that ICSF supports. For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_PRIME[1,4]	Big integer	Prime p (512 to 2048 bits in steps of 64 bits)
CKA_SUBPRIME[1,4]	Big integer	Subprime q (160 bits for p <= 1024 bits, 224 bits or 256 bits for p > 1024 bits)
CKA_BASE[1,4]	Big integer	Base g

Table 19. DSA domain parameter object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_PRIME_BITS[2,3]	CK_ULONG	Length of the prime value

Table 20. Diffie-Hellman domain parameter object attributes that ICSF supports. For the meanings of the footnotes, see Table 4 on page 9.

Attribute	Data type	Notes
CKA_PRIME[1,4]	Big integer	Prime p (512 to 2048 bits in steps of 64 bits)
CKA_BASE[1,4]	Big integer	Base g
CKA_PRIME_BITS[2,3]	CK_ULONG	Length of the prime value

Chapter 3. Update of z/OS Cryptographic Services ICSF Application Programmer's Guide, SA22-7522-14, information

This chapter contains updates to the document *z/OS Cryptographic Services ICSF Application Programmer's Guide*, SA22-7522-14, for the PKCS #11 enhancements provided by the PTF for APAR OA34403. Refer to this source document if background information is needed.

PKCS #11 Derive key (CSFPDVK)

Use the PKCS #11 Derive Key callable service to generate a new secret key object from an existing key object. This service does not support any recovery methods.

The deriving key handle must be a handle of an existing PKCS #11 key object. The CKA_DERIVE attribute for this object must be true. The mechanism keyword specified in the rule array indicates what derivation protocol to use. The derive parms list provides additional input data. The format of this list is dependent on the protocol being used.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPDVK6.

Format

```
CALL CSFPDVK(
            return_code,
            reason_code,
            exit_data_length,
            exit_data,
            rule_array_count,
            rule_array,
            attribute_list_length,
            attribute_list,
            base_key_handle,
            parms_list_length,
            parms_list,
            target_key_handle)
```

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 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-1). The data is defined 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 supplied in the *rule_array* parameter. This value must be 1.

rule array

Direction: Input Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 21. Keywords for derive key

Keyword	Meaning
Mechanism (required)	
PKCS-DH	Use the Diffie-Hellman PKCS derivation protocol as defined in the PKCS #11 standard as mechanism CKM_DH_PKCS_DERIVE.
SSL-MS	Use the SSL 3.0 Master Secret derivation protocol as defined in the PKCS #11 standard as mechanism CKM_SSL3_MASTER_KEY_DERIVE. The SSL protocol version is also returned. The base key must have been generated according to the rules for SSL 3.0
SSL-MSDH	Use the SSL 3.0 Master Secret for Diffie-Hellman derivation protocol as defined in the PKCS #11 standard as mechanism CKM_SSL3_MASTER_KEY_DERIVE_DH.
TLS-MS	Use the TLS Master Secret derivation protocol as defined in the PKCS #11 standard as mechanism CKM_TLS_MASTER_KEY_DERIVE. The base key must have been generated according to the rules for TLS 1.0 or TLS 1.1
TLS-MSDH	Use the TLS Master Secret for Diffie-Hellman derivation protocol as defined in the PKCS #11 standard as mechanism CKM_TLS_MASTER_KEY_DERIVE_DH.
EC-DH	Use the Elliptic Curve Diffie-Hellman derivation protocol as defined in the PKCS #11 standard as mechanism CKM_ECDH1_DERIVE

Table 21. Keywords for derive key (continued)

Keyword	Meaning
IKESEED	Use the IKEv1 or IKEv2 initial seeding protocol to derive a seed key using a previously derived secret key as the base key. Using IKE terminology, this mechanism performs either *SKEYID* = *prf*(*Ni_b* | *Nr_b*, *g^xy*) for IKEv1 or *SKEYSEED* = *prf*(*Ni* | *Nr*, *g^ir*) for IKEv2. Where: • *Ni_b* | *Nr_b* or *Ni* | *Nr* - is the concatenated initiator/responder nonce string • *g^xy* or *g^ir* - is the base key
IKESHARE	Use the IKEv1 initial seeding protocol to derive a seed key using a pre-shared secret key as the base key. Using IKE terminology, this mechanism performs *SKEYID* = *prf*(*pre-shared-key*, *Ni_b* | *Nr_b*). Where: • *Ni_b* | *Nr_b* - is the concatenated initiator/responder nonce string • *pre-shared-key* - is the base key
IKEREKEY	Use the IKEv2 rekeying protocol to derive a new seed key using a previously derived IKE derivation key as the base key and a previously derived secret key as an additional key. Using IKE terminology, this mechanism performs *SKEYSEED* = *prf*(*SK_d*, *g^ir* | *Ni* | *Nr*). Where: • *Ni* | *Nr* - is the concatenated initiator/responder nonce string • *SK_d* - is the base key • *g^ir* - is the additional key

attribute_list_length

Direction: Input Type: Integer

The length of the attributes supplied in the *attribute_list* parameter in bytes. The maximum value for this field is 32750.

attribute_list

Direction: Input Type: String

List of attributes for the derived secret key object.

base_key_handle

Direction: Input Type: String

The 44-byte handle of the source key object.

parms_list_length

Direction: Input Type: Integer

The length of the parameters supplied in the *parms_list* parameter in bytes.

parms_list

Direction: Input/Output Type: String

The protocol specific parameters. This field has a varying format depending on the mechanism specified:

Table 22. parms_list parameter format for PKCS-DH mechanism

Offset	Length in bytes	Direction	Description
0	4	Input	length in bytes of the other party's public value, where 64 <= length <= 256
4	<=256	Input	binary value representing the other party's public value.

Table 23. parms_list parameter format for SSL-MS, SSL-MSDH, TLS-MS, and TLS-MSDH mechanisms

Offset	Length in bytes	Direction	Description
0	2	Output	SSL protocol version returned for SSL-MS and TLS-MS only. For the other protocols, this field is left unchanged.
2	2	not applicable	reserved
4	4	Input	length in bytes of the client's random data (x), where 1 <= length <= 32
8	4	Input	length in bytes of the server's random data (y)), where 1 <= length <= 32
12	x	Input	client's random data
12+x	y	Input	server's random data

Table 24. parms_list parameter format for EC-DH mechanism

Offset	Length in bytes	Direction	Description
0	1	Input	KDF function code, x'01' = NULL; x'02' = SHA1. x'05' = SHA224, x'06' = SHA256, x'07' = SHA384, and x'08' = SHA512
1	3	not applicable	reserved
4	4	Input	length in bytes of the optional data shared between the two parties. A zero length means no shared data. For the NULL KDF the length must be zero. Otherwise, the maximum shared data length 2147483647.
8	8	Input	64-bit address of the data shared between the two parties. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers. This field is ignored if the length is zero.

Table 24. parms_list parameter format for EC-DH mechanism (continued)

Offset	Length in bytes	Direction	Description
16	4	Input	length in bytes of the other party's public value (x). This length is dependent on the curve type/size of the base key and on whether the value is DER encoded or not: secp192r1 – 49 (51 w/DER) secp224r1 – 57 (59 w/DER) secp256r1 – 65 (67 w/DER) secp384r1 – 97 (99 w/DER) secp521r1 – 133 (136 w/DER) brainpoolP160r1 – 41 (43 w/DER) brainpoolP192r1 – 49 (51 w/DER) brainpoolP224r1 – 57 (59 w/DER) brainpoolP256r1 – 65 (67 w/DER) brainpoolP320r1 – 81 (83 w/DER) brainpoolP384r1 – 97 (99 w/DER) brainpoolP512r1 – 129 (132 w/DER)
20	x<=136	Input	binary value representing the other party's public value with or without DER encoding.

Table 25. parms_list parameter format for IKESEED, IKESHARE, and IKEREKEY mechanisms

Offset	Length in bytes	Direction	Description
0	1	Input	IKE version code. Must be x'01' for IKESHARE, x'02' for IKEREKEY, x'01' or x'02' for IKESEED
1	1	Input	PRF function code x'01' = HMAC_MD5, x'02' = HMAC_SHA1, x'04' = HMAC_SHA256, x'05' = SHA384, and x'06' = SHA512
2	2	Input	Length of concatenated initiator/responder nonce string (n), where 16 <= n <= 512
4	44	Input	Key handle of additional key - required for IKEREKEY. Ignored for the other mechanisms.
48	n	Input	Concatenated initiator/responder nonce string

`target_key_handle`

Direction: Output   Type: String

Upon successful completion, the 44-byte handle of the secret key object that was derived.

Authorization

There are multiple keys involved in this service — one or two base keys and the target key (the new key created from the base key) .

- To use a base key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).
- To use a base key that is a private object, the caller must have USER (READ) authority (user access).
- To derive a target key that is a public object, the caller must have SO (READ) authority or USER (UPDATE) authority.

- To derive a target key that is a private object, the caller must have SO (CONTROL) authority or USER (UPDATE) authority.

Usage Notes

Key derivation operations are performed in software.

For the IKESEED, IKESHARE, and IKEREKEY mechanisms, the following attribute rules apply to the derived key:

- The key will have the following attributes which may not be overridden by other values in the attribute list:
  - CKA_CLASS=CKO_SECRET_KEY
  - CKA_KEY_TYPE=CKK_GENERIC_SECRET
  - CKA_DERIVE=TRUE
  - CKA_VALUE_LEN=l*ength of the output of the PRF function*
- Other applicable secret key attributes may be specified in the attribute list. However, an attribute list is not required. Any attribute not specified will be assigned the default value normally assigned to a newly created secret key. In particular, CKA_SENSITIVE defaults to FALSE and CKA_EXTRACTABLE defaults to TRUE.
- CKA_ALWAYS_SENSITIVE is set to FALSE if the CKA_ALWAYS_SENSITIVE attribute from the base key is FALSE. Otherwise it is set equal to the value of the CKA_SENSITIVE attribute assigned to the derived key.
- CKA_NEVER_EXTRACTABLE is set to FALSE if the CKA_NEVER_EXTRACTABLE attribute from the base key is FALSE. Otherwise it is set opposite to the value of the CKA_EXTRACTABLE attribute assigned to the derived key.

For the IKEREKEY mechanism, the additional key must be a secret key (CKA_CLASS=CKO_SECRET_KEY) capable of performing key derivation (CKA_DERIVE=TRUE). It must also be contained in the same PKCS #11 token as the base key.

For the IKESEED, IKESHARE, and IKEREKEY mechanisms, the MD5 PRF may not be specified if the operation is FIPS 140 restricted.

For the IKESHARE and IKEREKEY mechanisms, the length of the base key must be at least half the length of the output of the PRF function if the operation is FIPS 140 restricted.

For the IKESEED mechanism, the length of the concatenated initiator/responder nonce value must be at least half the length of the output of the PRF function if the operation is FIPS 140 restricted.

PKCS #11 Derive multiple keys (CSFPDMK)

Use the PKCS #11 Derive Multiple Keys callable service to generate multiple secret key objects and protocol dependent keying material from an existing secret key object. This service does not support any recovery methods.

The key handle must be a handle of a PKCS #11 secret key object. The CKA_DERIVE attribute for the secret key object must be true. The mechanism keyword specified in the rule array indicates what derivation protocol to use. The derive parms list provides additional input/output data. The format of this list is dependent on the protocol being used.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPDMK6.

Format

```
CALL CSFPDMK(
            return_code,
            reason_code,
            exit_data_length,
            exit_data,
            rule_array_count,
            rule_array,
            attribute_list_length,
            attribute_list,
            base_key_handle,
            parms_list_length,
            parms_list)
```

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 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-1). The data is defined 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 supplied in the *rule_array* parameter. This value must be 1.

rule array

Direction: Input   Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 26. Keywords for derive multiple keys

Keyword	**Meaning**
Mechanism (required)	
SSL-KM	Use the SSL 3.0 Key and MAC derivation protocol as defined in the PKCS #11 standard as mechanism CKM_SSL3_KEY_AND_MAC_DERIVE.
TLS-KM	Use the TLS 1.0/1.1 Key and MAC derivation protocol as defined in the PKCS #11 standard as mechanism CKM_TLS_KEY_AND_MAC_DERIVE.
IKE1PHA1	Use the IKEv1 phase 1 protocol to derive multiple keys using a previously derived IKE seed key as the base key and a previously derived secret key as an additional key. 3 keys are derived (one derivation, one authentication, and one encryption key). Using IKE terminology, this mechanism performs {*SKEYID_d* | *SKEYID_a* | *SKEYID_e*} = *prf*(*SKEYID*, *g^xy* | *CKY-I* | *CKY-R*) with key expansion for *SKEYID_e*, if required. (*SKEYID_d,a* are always the size of the prf output.) Where: • *CKY-I* | *CKY-R* - is the concatenated initiator/responder cookie string • *SKEYID* - is the base key • *g^xy* - is the additional key • *SKEYID_d,a,e* - are the to-be-derived derivation, authentication and encryption keys
IKE2PHA1	Use the IKEv2 phase 1 (SA) protocol to derive multiple keys using a previously derived IKE seed key as the base key. 7 keys are derived (one derivation, two authentication, two encryption, and two peer authentication keys). Using IKE terminology, this mechanism performs {*SK_d* | *SK_ai* | *SK_ar* | *SK_ei* | *SK_er* | *SK_pi* | *SK_pr* } = *prf+*(*SKEYSEED*, *Ni* | *Nr* | *SPIi* | *SPIr*). Where: • *Ni* | *Nr* | *SPIi* | *SPIr* - is the concatenated initiator/responder nonce and Security Parameter Index string • *SKEYSEED* - is the base key • *SK_d,ai,ar,ei,er,pi,pr* - are the to-be-derived derivation, initiator authentication, responder authentication, initiator encryption, responder encryption, initiator peer authentication, and responder peer authentication keys

Table 26. Keywords for derive multiple keys (continued)

Keyword	Meaning
IKE1PHA2	Use the IKEv1 phase 2 (CHILD SA) protocol to derive multiple keys and salt values using a previously derived IKE derivation key as the base key and a previously derived secret key as an additional key (optional). The derivation produces one of the following key sets: • One authentication key • One GMAC key plus salt value • One authentication key plus one encryption key • One GCM key plus a salt value Up to two such sets are produced, one for the sender and one for the receiver. Using IKE terminology, this mechanism performs *KEYMAT* = *prf*(*SKEYID_d*, [*g^xy* |] *protocol* | *SPI* | *Ni_b* | *Nr_b*), done in two passes – once for the sender and once for the receiver. Where: • *protocol* | *SPI* | *Ni_b* | *Nr_b* - is the concatenated Protocol, Security Parameter Index, and initiator/responder nonce string • *SKEYID_d* - is the base key • *g^xy* - is the optional additional key • *KEYMAT* - is the generated key material which is partitioned into the key set
IKE2PHA2	Use the IKEv2 phase 2 protocol to derive multiple keys and salt values using a previously derived IKE derivation key as the base key and a previously derived secret key as an additional key (optional). The derivation produces one of the following key sets: • One authentication key • One GMAC key plus salt value • One authentication key plus one encryption key • One GCM key plus a salt value Two such sets are produced, one for the initiator and one for the responder. Using IKE terminology, this mechanism performs *KEYMAT* = *prf+*(*SK_d*, [*g^ir* |] *Ni* | *Nr*). Where: • *Ni* | *Nr* - is the concatenated initiator/responder nonce string • *SK_d* - is the base key • *g^ir* - is the optional additional key • *KEYMAT* - is the generated key material which is partitioned into the key set

attribute_list_length

Direction: Input Type: Integer

The length of the attributes supplied in the *attribute_list* parameter in bytes. The maximum value for this field is 32752.

attribute_list

Direction: Input Type: String

List of attributes for the derived secret key object(s).

base_key_handle

Direction: Input Type: String

The 44-byte handle of the base key object.

parms_list_length

Direction: Input Type: Integer

The length of the parameters supplied in the *parms_list* parameter in bytes.

parms_list

Direction: Input/Output Type: String

The protocol specific parameters. This field has a varying format depending on the mechanism specified:

Table 27. parms_list parameter format for SSL-KM and TLS-KM mechanisms

Offset	Length in bytes	Direction	Description
0	1	Input	Boolean indicating if "export" processing is required. Any value other than x'00' means yes
1	3	Not applicable	reserved
4	4	Input	length in bytes of the client's random data (x)), where 1 <= length <= 32
8	4	Input	length in bytes of the server's random data (y)), where 1 <= length <= 32
12	4	Input	size of MAC to be generated in bits, where 8 <= size <= 384, in multiples of 8
16	4	Input	size of key to be generated in bits, Must match a supported size for the key type specified in the attribute list. Zero if no encryption keys are to be generated.
20	4	Input	size of IV to be generated in bits (v), where 0<= size <= 128, in multiples of 8. Must be zero if no encryption keys are to be generated.
24	44	Output	handle of client MAC secret object created
68	44	Output	handle of server MAC secret object created
112	44	Output	handle of client key object created
156	44	Output	handle of server key object created
200	x	Input	client's random data
200+x	y	Input	server's random data
200+x+y	v/8	Output	client's IV
200+x+y+v/8	v/8	Output	server's IV

Table 28. parms_list parameter format for IKE1PHA1 mechanism

Offset	Length in bytes	Direction	Description
0	1	Input	IKE version code. Must be x'01'

Table 28. parms_list parameter format for IKE1PHA1 mechanism (continued)

Offset	Length in bytes	Direction	Description
1	1	Input	PRF function code x'01' = HMAC_MD5, x’02’ = HMAC_SHA1, x’04’ = HMAC_SHA256, x’05’ = SHA384, and x’06’ = SHA512
2	4	Input	reserved
6	2	Input	length of to-be-derived encryption key, SKEYID_e
8	44	Input	Key handle of additional key
52	16	Input	Concatenated cookie string
68	44	Output	SKEYID_d key handle
112	44	Output	SKEYID_a key handle
156	44	Output	SKEYID_e key handle

Table 29. parms_list parameter format for IKE2PHA1 mechanism

Offset	Length in bytes	Direction	Description
0	1	Input	IKE version code. Must be x’02’
1	1	Input	PRF function code x'01' = HMAC_MD5, x’02’ = HMAC_SHA1, x’04’ = HMAC_SHA256, x’05’ = SHA384, and x’06’ = SHA512
2	2	Input	length of to-be-derived derivation key, SK_d
4	2	Input	length of a single to-be-derived authentication key, SK_a
6	2	Input	length of a single to-be-derived encryption key, SK_e
8	2	Input	length of a single to-be-derived peer authentication key, SK_p
10	2	Input	Concatenated nonce, SPI string length (n), where 24 <= n <= 520
12	44	Output	SKEYID_d key handle
56	44	Output	Initiator SKEYID_a key handle
100	44	Output	Responder SKEYID_a key handle
144	44	Output	Initiator SKEYID_e key handle
188	44	Output	Responder SKEYID_e key handle
232	44	Output	Initiator SKEYID_p key handle
276	44	Output	Responder SKEYID_p key handle
320	n	Input	Concatenated nonce, SPI string

Table 30. parms_list parameter format for IKE1PHA2 and IKE2PHA2 mechanisms

Offset	Length in bytes	Direction	Description
0	1	Input	IKE version code. Must be x’01’ for IKE1PHA2, x’02’ for IKE2PHA2
1	1	Input	PRF function code x'01' = HMAC_MD5, x’02’ = HMAC_SHA1, x’04’ = HMAC_SHA256, x’05’ = SHA384, and x’06’ = SHA512
2	2	Input	length of to-be-derived salts (s), where 0 <= s <= 4. Zero if salts are not to be derived

Table 30. parms_list parameter format for IKE1PHA2 and IKE2PHA2 mechanisms (continued)

Offset	Length in bytes	Direction	Description
4	2	Input	length of to-be-derived authentication keys. Zero if authentication keys are not to be derived
6	2	Input	length of to-be-derived encryption, GMAC, or GCM keys. Zero if no such keys are to be derived
8	2	Input	First pass parameter string length (n) • For IKE1PHA2 – Receiver concatenated Protocol, Security Parameter Index, and initiator/responder nonce string length, where 25 <= n <= 525 • For IKE2PHA2 – Concatenated initiator/responder nonce string length, where 16 <= n <= 512.
10	2	Input	Second pass parameter string length (m) • For IKE1PHA2 – Sender concatenated Protocol, Security Parameter Index, and initiator/responder nonce string length, where 25 <= m <= 525. Zero if second pass is to be skipped • For IKE2PHA2 – Not used. Must be zero
12	44	Input	Key handle of additional key. Fill with binary zeros if n/a
56	44	Output	Initiator (sender) authentication key handle
100	44	Output	Responder (receiver) authentication key handle
144	44	Output	Initiator (sender) encryption, GMAC, or GCM key handle
188	44	Output	Responder (receiver) encryption, GMAC, or GCM key handle
232	n	Input	First pass parameter string
232+n	m	Input	Second pass parameter string
232+n+m	s	Output	Initiator (sender) salt
232+n+m+s	s	Output	Responder (receiver) salt

Authorization

There are multiple keys involved in this service — one or two base keys and the target keys (the new keys created from the base key).

- To use a base key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).
- To use a base key that is a private object, the caller must have USER (READ) authority (user access).
- To derive a target key that is a public object, the caller must have SO (READ) authority or USER (UPDATE) authority.
- To derive a target key that is a private object, the caller must have SO (CONTROL) authority or USER (UPDATE) authority.

Usage Notes

Key derivation operations are performed in software.

For the SSL-KM and TLS-KM mechanisms, an attribute list is required if encryptions keys are to be generated.

For the IKE1PHA1, IKE2PHA1, IKE1PHA2, and IKE2PHA2 mechanisms, the following attribute rules apply to the derived keys:

- Derivation keys will have the following attributes which may not be overridden by other values in the attribute list:
  - CKA_CLASS=CKO_SECRET_KEY
  - CKA_KEY_TYPE=CKK_GENERIC_SECRET
  - CKA_DERIVE=TRUE
  - CKA_VALUE_LEN=*as specified in the parms list*
- Authentication keys will have the following attributes which may not be overridden by other values in the attribute list:
  - CKA_CLASS=CKO_SECRET_KEY
  - CKA_KEY_TYPE=CKK_GENERIC_SECRET
  - CKA_SIGN=TRUE=TRUE
  - CKA_VERIFY=TRUE=TRUE
  - CKA_VALUE_LEN= *as specified in the parms list*
- Encryption, GMAC, and GCM keys will be typed according to information found in the attribute list. However, they will have the following attributes which may not be overridden by other values in the attribute list:
  - CKA_CLASS=CKO_SECRET_KEY
  - For key types other than CKK_DES, CKK_DES2, and CKK_DES3, CKA_VALUE_LEN= *as specified in the parms list*
- All key types will inherit the values of the CKA_SENSITIVE, CKA_ALWAYS_SENSITIVE, CKA_EXTRACTABLE, and CKA_NEVER_EXTRACTABLE attributes from the base key. These may not be overridden by other values in the attribute list. If an additional key is specified, its values will be applied after setting the base key values as follows:
  - If the additional key has CKA_SENSITIVE=TRUE, so will the derived key(s)
  - If the additional key has CKA_EXTRACTABLE=FALSE, so will the derived keys(s)
  - If the additional key has CKA_ALWAYS_SENSITIVE=FALSE, so will the derived keys(s)
  - If the additional key has CKA_NEVER_EXTRACTABLE=FALSE, so will the derived keys(s)
- If encryption, GMAC, or GCM keys are to be derived, an attribute list is required for the key typing information. Otherwise, it is optional. For all keys, other applicable secret key attributes may be specified in the attribute list. Any attribute not specified will be assigned the default value normally assigned to a newly created secret key.

For the IKE1PHA1, IKE1PHA2, and IKE2PHA2 mechanisms, the additional key must be a secret key (CKA_CLASS=CKO_SECRET_KEY) capable of performing key derivation (CKA_DERIVE=TRUE). It must also be contained in the same PKCS #11 token as the base key.

The IKE1PHA1, IKE2PHA1, IKE1PHA2, and IKE2PHA2 mechanisms have the following limitations if the operation is FIPS 140 restricted:

- The MD5 PRF may not be specified.
- The length of the base key must be at least half the length of the output of the PRF function.

PKCS #11 One-way hash, sign, or verify (CSFPOWH)

Use the one-way hash, sign, or verify callable service to generate a one-way hash on specified text, sign specified text, or verify a signature on specified text. For one-way hash, this service supports the following methods:

- MD2 - software only
- MD5 - software only
- SHA-1
- RIPEMD-160 - software only
- SHA-224
- SHA-256
- SHA-384
- SHA-512

For sign and verify, the following methods are supported:

- MD2 with RSA-PKCS 1.5
- MD5 with RSA-PKCS 1.5
- SHA1 with RSA-PKCS 1.5, DSA, or ECDSA
- SHA-224 with RSA-PKCS 1.5, DSA, or ECDSA
- SHA-256 with RSA-PKCS 1.5, DSA, or ECDSA
- SHA-384 with RSA-PKCS 1.5, DSA, or ECDSA
- SHA-512 with RSA-PKCS 1.5, DSA, or ECDSA

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPOWH6.

Format

```
CALL CSFPOWH(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             text_length,
             text,
             text_id,
             chain_data_length,
             chain_data,
             handle,
             hash_length,
             hash )
```

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 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-1). The data is defined 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 supplied in the *rule_array* parameter. This value must be 1 or 2.

rule array

Direction: Input Type: String

Keywords that provide control information to the callable service. Each keyword is left-justisfied in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 31. Keywords for one-way hash generate

Keyword	Meaning
Hash Method (required)	
MD2	Hash algorithm is MD2 algorithm. Length of hash generated is 16 bytes.
MD5	Hash algorithm is MD5 algorithm. Length of hash generated is 16 bytes.
RPMD-160	Hash algorithm is RIPEMD-160. Length of hash generated is 20 bytes.
SHA-1	Hash algorithm is SHA-1. Length of hash generated is 20 bytes.
SHA-224	Hash algorithm is SHA-224. Length of hash generated is 28 bytes.
SHA-256	Hash algorithm is SHA-256. Length of hash generated is 32 bytes.
SHA-384	Hash algorithm is SHA-384. Length of hash generated is 48 bytes.
SHA-512	Hash algorithm is SHA-512. Length of hash generated is 64 bytes.
DETERMIN	For use with non-chained RSA signature verifies only. Hash algorithm is to be determined from the input signature.
Chaining Flag (optional)	
FIRST	Specifies this is the first call in a series of chained calls. Intermediate results are stored in the *hash* and *chain_data* fields. Cannot be specified with hash method DETERMIN.

Table 31. Keywords for one-way hash generate (continued)

Keyword	Meaning
MIDDLE	Specifies this is a middle call in a series of chained calls. Intermediate results are stored in the *hash* and *chain_data* fields. Cannot be specified with hash method DETERMIN.
LAST	Specifies this is the last call in a series of chained calls. Cannot be specified with hash method DETERMIN.
ONLY	Specifies this is the only call and the call is not chained. This is the default.
Requested Operation (optional)	
HASH	The specified text is to be hashed only. This is the default. Cannot be specified (either explicitly or by default) with hash method DETERMIN.
SIGN-RSA	The data is to be hashed then signed using RSA-PKCS 1.5 formatting. Any hash method is acceptable except RPMD-160 and DETERMIN.
SIGN-DSA	The data is to be hashed then signed using DSA. The hash method must be SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.
SIGN-EC	The data is to be hashed then signed using ECDSA. The hash method must be SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.
VER-RSA	The data is to be hashed then signature verified using RSA-PKCS 1.5 formatting. Any hash method is acceptable except RPMD-160. This operation is required for hash method DETERMIN.
VER-DSA	The data is to be hashed then signature verified using DSA. The hash method must be SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.
VER-EC	The data is to be hashed then signature verified using ECDSA. The hash method must be SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.

text_length

Direction: Input Type: Integer

The length of the text parameter in bytes.

If you specify the FIRST or MIDDLE keyword, then the text length must be a multiple of the block size of the hash method. For MD2, this is a multiple of 16 bytes. For MD5, RPMD-160, SHA-1, SHA-224, and SHA-256, this is a multiple of 64 bytes. For SHA-384 and SHA-512, this is a multiple of 128 bytes. For ONLY and LAST, this service performs the required padding according to the algorithm specified. The length can be from 0 to 2147483647.

text

Direction: Input Type: String

Value to be hashed

text_id

Direction: Input Type: Integer

The ALET identifying the space where the text resides.

chain_data_length

Direction: Input/Output Type: Integer

The byte length of the *chain_data* parameter. This must be 128 bytes.

chain_data

Direction: Input/Output Type: String

This field is a 128-byte work area. The chain data permits chaining data from one call to another. ICSF initializes the chain data on a FIRST call and may change it on subsequent MIDDLE calls. Your application must not change the data in this field between the sequence of FIRST, MIDDLE, and LAST calls for a specific message. The chain data has the following format:

Table 32. chain_data parameter format

Offset	Length	Description
0	4	Flag word **Bit** **Meaning when set on** **0** Cryptographic state object has been allocated **1-31** Reserved for IBM’s use
4	44	Cryptographic state object handle
48	80	Reserved for IBM’s use

handle

Direction: Input Type: String

For hash requests, this is the 44-byte name of the token to which this hash operation is related. The first 32 bytes of the handle are meaningful. The remaining 12 bytes are reserved.

For sign and verify requests, this is the 44-byte handle to the key object that is to be used. For FIRST and MIDDLE chaining requests, only the first 32 bytes of the handle are meaningful, to identify the token.

hash_length

Direction: Input/Output Type: Integer

The length of the supplied hash field in bytes.

For hash requests, this field is input only. For SHA-1 and RPMD-160 this must be at least 20 bytes; for MD2 and MD5 this must be at least 16 bytes. For SHA-224 and SHA-256, this must be at least 32 bytes. Even though the length of the SHA-224 hash is less than SHA-256, the extra bytes are used as a work area during the generation of the hash value. The SHA-224 value is left-justified and padded with 4 bytes of binary zeroes. For SHA-384 and SHA-512, thus must be at least 64 bytes. Even though the length of the SHA-384 hash is less than SHA-512, the extra bytes are used as a work area during the generation of the hash value. The SHA-384 value is left-justified and padded with 16 bytes of binary zeroes.

For FIRST and MIDDLE sign and verify requests, this field is ignored.

For LAST and ONLY sign requests, this field is input/output. If the signature generation is successful, ICSF will update this field with the length of the generated signature. If the signature generation is unsuccessful because the supplied hash field is too small, ICSF will update this field with the required length.

For LAST and ONLY verify requests, this field is input only.

hash

Direction: Input/Output Type: String

This field contains the hash or signature, left-justified. The processing of the rest of the field depends on the implementation.

For hash requests, this field is the generated hash. 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.

For FIRST and MIDDLE sign and verify requests, this field is ignored.

For LAST and ONLY sign requests, this field is the generated signature.

For LAST and ONLY verify requests, this field is input signature to be verified.

Authorization

To use this service to sign or verify with a public object, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

To use this service to sign or verify with a private object, the caller must have at least USER (READ) authority (user access).

Usage Notes

If the FIRST rule is used to start a series of chained calls, the application must not change the Hash Method or Requested Operation rules between the calls. The behavior of the service is undefined if the rules are changed.

If the FIRST rule is used to start a series of chained calls, the application should make a LAST call to free ICSF resources allocated. If processing is to be aborted without making a LAST call and the *chain_data* parameter indicates that a cryptographic state object has been allocated, the caller must free the object by calling CSFPTRD (or CSFPTRD6 for 64-bit callers) passing the state object’s handle.

The CSFSERV resource name that protects this service is CSFOWH, the same resource name used to protect the non-PKCS #11 One Way Hash service.

For hash method DETERMIN, ICSF determines the hashing method by RSA decrypting the input signature using the specified public key and examining the result. ICSF will return the “signature did not verify” error (return code 4, reason code X'2AF8') if this process is unsuccessful for any of the following reasons:

1. ICSF cannot successfully perform the decryption because the public key is the wrong size.
2. The resulting clear text block is not properly RSA-PKCS 1.5 formatted.
3. The resulting clear text block indicates a hashing algorithm not supported by this service was used.

PKCS #11 Private key sign (CSFPPKS)

Use the PKCS #11 private key sign callable service to:

- Decrypt or sign data using an RSA private key using zero-pad or PKCS #1 v1.5 formatting
- Sign data using a DSA private key
- Sign data using an Elliptic Curve private key in combination with DSA

The key handle must be a handle of a PKCS #11 private key object. When the request type keyword DECRYPT is specified in the rule array, CKA_DECRYPT attribute must be true. When no request type is specified, the CKA_SIGN attribute must be true.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPPKS6.

Format

```
CALL CSFPPKS(
            return_code,
            reason_code,
            exit_data_length,
            exit_data,
            rule_array_count,
            rule_array,
            cipher_value_length,
            cipher_value,
            key_handle,
            clear_value_length,
            clear_value )
```

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 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-1). The data is defined 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 supplied in the *rule_array_parameter*. This value may be 1 or 2.

rule array

Direction: Input Type: String

Keywords that provide control information to the callable service.

Table 33. Keywords for private key sign

Keyword	Meaning
Mechanism (One of the following must be specified)	
RSA-ZERO	Mechanism is RSA decryption or signature generation using zero-pad formatting
RSA-PKCS	Mechanism is RSA decryption or signature generation using PKCS #1 v1.5 formatting
DSA	Mechanism is DSA signature generation
ECDSA	Mechanism is Elliptic Curve with DSA signature generation
Request type (optional)	
DECRYPT	The request is to decrypt data. This type of request requires the CKA_DECRYPT attribute to be true. If DECRYPT is not specified, the CKA_SIGN attribute must be true. Valid with RSA only.

cipher_value_length

Direction: Input Type: Integer

Length of the *cipher_value* parameter in bytes.

cipher_value

Direction: Input Type: String

For decrypt, this is the value to be decrypted. Otherwise this is the value to be signed. For RSA-PKCS signature requests, the data to be signed is expected to be a DER encoded DigestInfo structure. For DSA and ECDSA signature requests, the data to be signed is expected to be a SHA1, SHA224, SHA256, SHA384 or SHA512 digest.

key_handle

Direction: Input Type: String

The 44-byte handle of a private key object.

clear_value_length

Direction: Input/Output Type: Integer

Length of the *clear_value* parameter in bytes. On output, this is updated to be the actual length of the decrypted value or the generated signature.

clear_value

Direction: Output Type: String

For decrypt, this field will contain the decrypted value. Otherwise this field will contain the generated signature.

Authorization

To use this service with a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have USER (READ) authority (user access).

Usage Notes

DSA operations are performed in software. RSA operations may be done in hardware or software.

Request type DECRYPT is not supported for an Elliptic Curve or DSA private key.

PKCS #11 Public key verify (CSFPPKV)

Use the PKCS #11 public key verify callable service to:

- Encrypt or verify data using an RSA public key using zero-pad or PKCS #1 v1.5 formatting. For encryption, the encrypted data is returned
- Verify a signature using a DSA public key. No data is returned
- Verify a signature using an Elliptic Curve public key in combination with DSA. No data is returned

The key handle must be a handle of a PKCS #11 public key object. When the request type keyword ENCRYPT is specified in the rule array, CKA_ENCRYPT attribute must be true. When no request type is specified, the CKA_VERIFY attribute must be true.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPPKV6.

Format

```
CALL CSFPPKV(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             clear_value_length,
             clear_value,
             key_handle,
             cipher_value_length,
             cipher_value )
```

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 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-1). The data is defined 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 supplied in the *rule_array* parameter. This value must be 1 or 2.

rule array

Direction: Input Type: String

Keywords that provide control information to the callable service.

Table 34. Keywords for public key verify

Keyword	Meaning
Mechanism (One of the following must be specified)	
RSA-ZERO	Mechanism is RSA encryption or signature verification using zero-pad formatting
RSA-PKCS	Mechanism is RSA encryption or signature verification using PKCS #1 v1.5 formatting
DSA	Mechanism is DSA signature verification
ECDSA	Mechanism is Elliptic Curve with DSA signature verification
Request type (optional)	
ENCRYPT	The request is to encrypt data. This type of request requires the CKA_ENCRYPT attribute to be true. If ENCRYPT is not specified, the CKA_VERIFY attribute must be true. Valid with RSA only.

clear_value_length

Direction: Input Type: Integer

The length of the *clear_value* parameter

clear_value

Direction: Input Type: String

For encrypt, this is the value to be encrypted. Otherwise this is the signature to be verified.

key_handle

Direction: Input Type: String

The 44-byte handle of public key object.

cipher_value_length

Direction: Input/Output Type: Integer

For encrypt, on input, this is the length of the *cipher_value* parameter in bytes. On output, this is updated to be the actual length of the text encrypted into the *cipher_value* parameter. For signature verification, this is the length of the data to be verified (input only).

cipher_value

Direction: Input/Output Type: String

For encrypt, this is the encrypted value (output only). For signature verification, this is the data to be verified (input only). For RSA-PKCS signature verification requests, the data to be verified is expected to be a DER encoded DigestInfo structure. For DSA and ECDSA signature verification requests, the data to be verified is expected to be a SHA1, SHA224, SHA256, SHA384 or SHA512 digest.

Authorization

To use this service with a public object, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have at least USER (READ) authority (user access).

Usage Notes

DSA operations are performed in software. RSA and ECDSA operations may be done in hardware or software.

Request type ENCRYPT is not supported for an Elliptic Curve or DSA public key.

Return Codes and Reason Codes

The following reason code for Return Code 8 (the call to the service was unsuccessful) has been modified.

Table 35. Reason Code for Return Code 8 (8)

Reason Code Hex (Decimal)	Description
BFF (3071)	An application using a z/OS PKCS #11 token that is marked 'Write Protected' is attempting to do one of the following: • Store a persistent object in the token. • Delete the token. • Reinitialize the token. ICSF always marks the session object only omnipresent token as 'Write Protected.' ICSF will also mark an ordinary token 'Write Protected' if it contains objects not supported by this release of ICSF. **User action:** Use a z/OS PKCS #11 token that is not marked 'Read Only' or, if this is an ordinary token (not the omnipresent token), attempt the delete or reinitialization from a different member of the sysplex.

Chapter 4. Update of z/OS Cryptographic Services ICSF System Programmer’s Guide, SA22-7520-15, information

This chapter contains updates to the document *z/OS Cryptographic Services ICSF System Programmer’s Guide*, SA22-7520-15, for the PKCS #11 enhancements provided by the PTF for APAR OA34403. Refer to this source document if background information is needed.

Format of the token and object records

Each z/OS PKCS #11 token record and token object record begins with the same 188 bytes of data. The remainder of the record is specific to the token or object.

Common section of the token and object records

Every record in the token data set, with the exception of the header record, begins with these 188 bytes of data.

Table 36. Format of the common section of the token and object records

Offset (decimal)	Length of field (bytes)	Description
0	72	Handle of token or object **Bytes 0-31:** Token name **Bytes 32-39:** Sequence number **Byte 40:** Character “T” for token object **Bytes 41-43** Blank characters **Bytes 44-71:** Binary zeros
72	8	Reserved for IBM's use
80	8	The date that this record was created, in the format *yyyymmdd*
88	8	The time that this record was created, in the format *hhmmssth*
96	8	The most recent date that this record was updated, in the format *yyyymmdd*
104	8	The most recent time that this record was updated, in the format *hhmmssth*
112	4	Length of the entire TKDS record entry
116	20	Reserved for IBM's use
136	52	User data
188	variable	The TKDS token or object (see mappings)

Format of the token-specific section of the token record

Each z/OS PKCS #11 token record begins with the 188 bytes. The remainder of the record contains the contents of the token. The mapping of the record shows the data beginning at offset 0, which is its offset into the token-specific portion of the record; however, that portion of the record is at an offset of 188 into the entire record.

Table 37. Format of the unique section of the token record

Offset (decimal) 188 +	Length of field (bytes)	Description
0	4	Eye catcher for token: “TOKN”
4	2	Version number of structure: EBCDIC '00'
6	2	Length of structure in bytes
8	4	Reserved for IBM's use. Must be zeros.
12	8	Last assigned sequence number
20	32	Manufacturer identification
52	16	Model
68	16	Serial number
84	8	Date of the most recent update to this token, expressed as Coordinated Universal Time (UTC) in the format *yyyymmdd*. This includes any update to token information or to a token object.
92	8	Time of the most recent update to this token, expressed as Coordinated Universal Time (UTC) in the format *hhmmssth*. This includes any update to token information or to a token object.
100	44	Reserved for IBM's use
144		End of token

Format of the object-specific sections of the token object records

The following classes of objects can be associated with a z/OS PKCS #11 token:

- Certificate
- Public key
- Private key
- Secret key
- Data objects
- Domain parameters

The token object record for each begins with the common section described “Common section of the token and object records” on page 47, followed by a section specific to the class of object. Each of the object-specific sections begins with a 12-byte header record, followed by a variable-length section. Each 12-byte header contains a 4-byte flag field that has the same mapping for all classes of objects.

Table 38. Format of the token object flags. This 4-byte flag field occurs in the object header section of each token object record.

Offset (decimal)	Field name	Description
Flag byte 1		
Bit 0	OBJ_IS_TOKOBJ	When on, the object is a token object. When off, the object is a session object.
Bit 1	OBJ_IS_PRVOBJ	When on, the object is a private object. When off, the object is a public object.
Bit 2	OBJ_IS_MODOBJ	When on, the object is modifiable.

Table 38. Format of the token object flags (continued). This 4-byte flag field occurs in the object header section of each token object record.

Offset (decimal)	Field name	Description
Bit 3	KEY_DERIVE	When on, the key supports key derivation.
Bit 4	KEY_LOCAL	When on, the key was generated locally.
Bit 5	KEY_ENCRYPT	When on, the key supports encryption.
Bit 6	KEY_DECRYPT	When on, the key supports decryption.
Bit 7	KEY_VERIFYA	When on, the key supports verification where the signature is an appendix to the data.
Flag byte 2		
Bit 0	KEY_VERIFYR	When on, the key supports verification where the data is recovered from the signature
Bit 1	KEY_SIGA	When on, the key supports signatures where the signature is an appendix to the data.
Bit 2	KEY_SIGR	When on, the key supports signatures where the data is recovered from the signature.
Bit 3	KEY_WRAP	When on, the key supports wrapping.
Bit 4	KEY_UNWRAP	When on, the key supports unwrapping.
Bit 5	KEY_EXTRACT	When on, the key is extractable.
Bit 6	KEY_IS_SENSITIVE	When on, the key is sensitive.
Bit 7	KEY_IS_ALWAYS_SENSITIVE	When on, the SENSITIVE attribute (KEY_IS_SENSITIVE) is always true.
Flag byte 3		
Bit 0	KEY_NEVER_EXTRACT	When on, the EXTRACTABLE attribute (KEY_EXTRACT) is never true. When off, the EXTRACTABLE attribute (KEY_EXTRACT) can be true.
Bit 1	OBJ_IS_TRUSTED	When on, the certificate can be trusted for the application for which it was created.
Bit 2	CERT_IS_DEFAULT	When on, this is the default certificate.
Bit 3	FIPS140	When on, key is only to be used in a FIPS-compliant manner.
Bits 4-7		Reserved for IBM's use
Flag byte 4		
Bits 0-7		Reserved for IBM's use

Table 39. Format of the token certificate object

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for certificate object: “CERT”
4	2	Version: EBCDIC '00'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 38 on page 48)
Object type-specific section		
12	4	TYPE attribute: X'00000000': CKC_X_509
16	4	Certificate category **0** Undefined **1** Token user **2** Certificate authority **3** Other entity
20	8	Reserved for IBM's use
28	32	Reserved for IBM's use
60	2	Length of SUBJECT attribute in bytes (*aa*)
62	2	Length of ID attribute in bytes (*bb*)
64	2	Length of ISSUER attribute in bytes (*cc*)
66	2	Length of SERIAL_NUMBER attribute in bytes (*dd*)
68	2	Length of VALUE attribute in bytes (*ee*)
70	2	Length of LABEL attribute in bytes (*ff*)
72	2	Length of APPLICATION attribute in bytes (*gg*)
74	22	Reserved for IBM's use
96	4	Offset of SUBJECT attribute in bytes
100	4	Offset of ID attribute in bytes
104	4	Offset of ISSUER attribute in bytes
108	4	Offset of SERIAL_NUMBER attribute in bytes
112	4	Offset of VALUE attribute in bytes
116	4	Offset of LABEL attribute in bytes
120	4	Offset of APPLICATION attribute in bytes
124	44	Reserved for IBM's use
168	*aa* + *bb* + *cc* + *dd* + *ee* + *ff* + *gg*	Certificate attributes (variable length)
168 + *aa* + *bb* + *cc* + *dd* + *ee* + *ff* + *gg*		End of certificate object

Table 40. Format of the token public key object (Version 0)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		

Table 40. Format of the token public key object (Version 0) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
0	4	Eye catcher for public key object: “PUBK”
4	2	Version: EBCDIC '00'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 38 on page 48)
Object type-specific section		
12	4	TYPE attribute: CKK_RSA
16	8	Start date for the key, in the format *yyyymmdd*
24	8	End date for the key, in the format *yyyymmdd*
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
72	4	Length in bits of modulus n
76	256	Modulus n
332	256	Reserved
588	256	Public exponent e
844	256	Reserved
1100	2	Length of SUBJECT attribute in bytes (*aa*)
1102	2	Length of ID attribute in bytes (*bb*)
1104	2	Length of LABEL attribute in bytes (*cc*)
1106	2	Length of APPLICATION attribute in bytes (*dd*)
1108	20	Reserved
1128	4	Offset of SUBJECT attribute in bytes
1132	4	Offset of ID attribute in bytes
1136	4	Offset of LABEL attribute in bytes
1140	4	Offset of APPLICATION attribute in bytes
1144	40	Reserved
1184	*aa+bb+cc+dd*	Public key attributes (variable length)
1184+*aa+bb+cc+dd*		End of public key object

Table 41. Format of the token public key object (Version 1)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for public key object: “PUBK”
4	2	Version: EBCDIC '01'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 38 on page 48)
Object type-specific section		

Table 41. Format of the token public key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
12	4	TYPE attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key, in the format *yyyymmdd*
24	8	End date for the key, in the format *yyyymmdd*
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus *n*
76	512	Modulus *n*
588	512	Public exponent *e*
Algorithm-specific section (DSA)		
72	4	Length in bits of prime *p*
76	128	Reserved
204	128	Prime *p*
332	128	Reserved
460	128	Base *g*
588	128	Reserved
716	128	Value *y*
844	20	Reserved
864	20	Subprime *q*
884	216	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime *p*
76	256	Prime *p*
332	256	Base *g*
588	256	Value *y*
844	256	Reserved
Algorithm-specific section (EC)		

Table 41. Format of the token public key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	128	Reserved
204	136	EC point Q (DER encoded)
340	760	Reserved
Variable length attribute section		
1100	2	Length of SUBJECT attribute in bytes (*aa*)
1102	2	Length of ID attribute in bytes (*bb*)
1104	2	Length of LABEL attribute in bytes (*cc*)
1106	2	Length of APPLICATION attribute in bytes (*dd*)
1108	20	Reserved
1128	4	Offset of SUBJECT attribute in bytes
1132	4	Offset of ID attribute in bytes
1136	4	Offset of LABEL attribute in bytes
1140	4	Offset of APPLICATION attribute in bytes
1144	40	Reserved
1184	*aa+bb+cc+dd*	Public key attributes (variable length)
1184+*aa+bb+cc+dd*		End of public key object

Table 42. Format of the token public key object (Version 2)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		

Table 42. Format of the token public key object (Version 2) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
0	4	Eye catcher for public key object: “PUBK”
4	2	Version: EBCDIC '02'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 38 on page 48)
Object type-specific section		
12	4	TYPE attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key, in the format *yyyymmdd*
24	8	End date for the key, in the format *yyyymmdd*
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus *n*
76	512	Modulus *n*
588	512	Public exponent *e*
Algorithm-specific section (DSA)		
72	4	Length in bits of prime *p*
76	256	Prime *p*
332	256	Base *g*
588	256	Value *y*
844	8	Reserved
852	32	Subprime *q*
884	216	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime *p*
76	256	Prime *p*
332	256	Base *g*
588	256	Value *y*
844	256	Reserved
Algorithm-specific section (EC)		

Table 42. Format of the token public key object (Version 2) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	128	Reserved
204	136	EC point Q (DER encoded)
340	760	Reserved
Variable length attribute section		
1100	2	Length of SUBJECT attribute in bytes (*aa*)
1102	2	Length of ID attribute in bytes (*bb*)
1104	2	Length of LABEL attribute in bytes (*cc*)
1106	2	Length of APPLICATION attribute in bytes (*dd*)
1108	20	Reserved
1128	4	Offset of SUBJECT attribute in bytes
1132	4	Offset of ID attribute in bytes
1136	4	Offset of LABEL attribute in bytes
1140	4	Offset of APPLICATION attribute in bytes
1144	40	Reserved
1184	*aa+bb+cc+dd*	Public key attributes (variable length)
1184+*aa+bb+cc+dd*		End of public key object

Table 43. Format of the token private key object (Version 0)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		

Table 43. Format of the token private key object (Version 0) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
0	4	Eye catcher for private key object: “PRIV”
4	2	Version: EBCDIC '00'
6	2	Length of object (in bytes)
8	4	Flags (see Table 38 on page 48)
Object type-specific section		
12	4	Type attribute: CKK_RSA
16	8	Start date for the key (in the format *yyyymmdd*)
24	8	End date for the key (in the format *yyyymmdd*)
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
72	4	Length in bits of modulus n
76	256	Modulus: modulus n
332	256	Reserved
588	256	Public exponent e
844	256	Reserved
1100	32	Reserved
1132	256	Private exponent d
1388	256	Reserved
1644	136	Prime p
1780	128	Reserved
1908	128	Prime q
2036	128	Reserved
2172	136	Private exponent d modulo p-1
2300	128	Reserved
2428	128	Private exponent d modulo q-1
2556	128	Reserved
2684	136	CRT coefficient q-1 mod p
2820	128	Reserved
2948	2	Length of SUBJECT attribute in bytes (*xx*)
2950	2	Length of ID attribute in bytes (*yy*)
2952	2	Length of LABEL attribute in bytes (*zz*)
2954	2	Length of APPLICATION attribute in bytes (*ww*)
2956	20	Reserved
2976	4	Offset of SUBJECT attribute in bytes
2980	4	Offset of ID attribute in bytes
2984	4	Offset of LABEL attribute in bytes
2988	4	Offset of APPLICATION attribute in bytes
2992	40	Reserved

Table 43. Format of the token private key object (Version 0) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
3032	*xx*+*yy*+*zz*+*ww*	Private key attributes (variable length)
3032+*xx*+*yy*+*zz*+*ww*		End of private key object

Table 44. Format of the token private key object (Version 1)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for private key object: "PRIV"
4	2	Version: EBCDIC '01'
6	2	Length of object (in bytes)
8	4	Flags (see Table 38 on page 48)
Object type-specific section		
12	4	Type attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key (in the format *yyyymmdd*)
24	8	End date for the key (in the format *yyyymmdd*)
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus *n*
76	512	Modulus: modulus *n*
588	512	Public exponent *e*
1100	32	Reserved
1132	512	Private exponent *d*
1644	264	Prime *p*
1908	256	Prime *q*
2164	264	Private exponent *d* modulo *p*-1
2428	256	Private exponent *d* modulo *q*-1
2684	264	CRT coefficient *q*-1 mod *p*
Algorithm-specific section (DSA)		
72	4	Length in bits of prime *p*
76	128	Reserved
204	128	Prime *p*
332	128	Reserved
460	128	Base *g*
588	236	Reserved
824	20	Value *x*
844	20	Reserved
864	20	Subprime *q*

Table 44. Format of the token private key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
884	2064	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime *p*
76	256	Prime *p*
332	256	Base *g*
588	236	Reserved
824	20	Value *x*
844	2104	Reserved
Algorithm-specific section (EC)		
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	64	Reserved
140	66	Value *d*
206	2742	Reserved
Variable length attribute section		
2948	2	Length of SUBJECT attribute in bytes (*xx*)
2950	2	Length of ID attribute in bytes (*yy*)
2952	2	Length of LABEL attribute in bytes (*zz*)
2954	2	Length of APPLICATION attribute in bytes (*ww*)
2956	20	Reserved
2976	4	Offset of SUBJECT attribute in bytes
2980	4	Offset of ID attribute in bytes
2984	4	Offset of LABEL attribute in bytes

Table 44. Format of the token private key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
2988	4	Offset of APPLICATION attribute in bytes
2992	40	Reserved
3032	*xx*+*yy*+*zz*+*ww*	Private key attributes (variable length)
3032+*xx*+*yy*+*zz*+*ww*		End of private key object

Table 45. Format of the token private key object (Version 2)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for private key object: “PRIV”
4	2	Version: EBCDIC '02'
6	2	Length of object (in bytes)
8	4	Flags (see Table 38 on page 48)
Object type-specific section		
12	4	Type attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key (in the format *yyyymmdd*)
24	8	End date for the key (in the format *yyyymmdd*)
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus *n*
76	512	Modulus: modulus *n*
588	512	Public exponent *e*
1100	32	Reserved
1132	512	Private exponent *d*
1644	264	Prime *p*
1908	256	Prime *q*
2164	264	Private exponent *d* modulo *p*-1
2428	256	Private exponent *d* modulo *q*-1
2684	264	CRT coefficient *q*-1 mod *p*
Algorithm-specific section (DSA)		
72	4	Length in bits of prime *p*
76	256	Prime *p*
332	256	Base *g*
588	224	Reserved
812	32	Value *x*
844	8	Reserved
852	32	Subprime *q*

Table 45. Format of the token private key object (Version 2) (continued)

Offset (decimal) 188 +	**Length of field (bytes)**	**Description**
884	2064	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime *p*
76	256	Prime *p*
332	256	Base *g*
588	256	Value *x*
844	4	Length in bits of value *x*
848	2100	Reserved
Algorithm-specific section (EC)		
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	64	Reserved
140	66	Value *d*
206	2742	Reserved
Variable length attribute section		
2948	2	Length of SUBJECT attribute in bytes (*xx*)
2950	2	Length of ID attribute in bytes (*yy*)
2952	2	Length of LABEL attribute in bytes (*zz*)
2954	2	Length of APPLICATION attribute in bytes (*ww*)
2956	20	Reserved
2976	4	Offset of SUBJECT attribute in bytes
2980	4	Offset of ID attribute in bytes
2984	4	Offset of LABEL attribute in bytes

Table 45. Format of the token private key object (Version 2) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
2988	4	Offset of APPLICATION attribute in bytes
2992	40	Reserved
3032	*xx*+*yy*+*zz*+*ww*	Private key attributes (variable length)
3032+*xx*+*yy*+*zz*+*ww*		End of private key object

Table 46. Format of the token secret key object (Version 0)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for secret key object: “SECK”
4	2	Version: EBCDIC '00'
6	2	Length of the object in bytes
8	4	Flags (see Table 38 on page 48)
Object type-specific section		
12	4	Type of key: CKK_DES, CKK_DES2, CKK_DES3, CKK_AES
16	8	Start date for the key (in the format *yyyymmdd*)
24	8	End date for the key (in the format *yyyymmdd*)
32	4	Key generate mechanism CK_UNAVAILABLE_INFORMATION
36	2	Length of the key in bytes
38	32	Reserved
70	64	VALUE: value of the key
134	538	Reserved
672	4	Usage counter field
676	2	Reserved
678	2	Length of LABEL attribute in bytes (*xx*)
680	2	Length of APPLICATION attribute in bytes (*yy*)
682	2	Length of the ID attribute in bytes (*zz*)
684	20	Reserved
704	4	Offset of LABEL attribute in bytes
708	4	Offset of APPLICATION attribute in bytes
712	4	Offset of the ID attribute in bytes
716	40	Reserved
756	*xx*+*yy*+*zz*	Secret key attributes (variable length)
756+*xx*+*yy*+*zz*		End of secret key object

Table 47. Format of the token secret key object (Version 1)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for secret key object: “SECK”
4	2	Version: EBCDIC '01'
6	2	Length of the object in bytes
8	4	Flags (see Table 38 on page 48)
Object type-specific section		
12	4	Type of key: CKK_DES, CKK_DES2, CKK_DES3, CKK_BLOWFISH, CKK_RC4, CKK_GENERIC_SECRET, and CKK_AES.
16	8	Start date for the key (in the format *yyyymmdd*)
24	8	End date for the key (in the format *yyyymmdd*)
32	4	Key generate mechanism CK_UNAVAILABLE_INFORMATION
36	2	Length of the key in bytes
38	32	Reserved
70	256	VALUE: value of the key
326	346	Reserved
672	4	Usage counter field
676	2	Reserved
678	2	Length of LABEL attribute in bytes (*xx*)
680	2	Length of APPLICATION attribute in bytes (*yy*)
682	2	Length of the ID attribute in bytes (*zz*)
684	20	Reserved
704	4	Offset of LABEL attribute in bytes
708	4	Offset of APPLICATION attribute in bytes
712	4	Offset of the ID attribute in bytes
716	40	Reserved
756	*xx*+*yy*+*zz*	Secret key attributes (variable length)
756+*xx*+*yy*+*zz*		End of secret key object

Table 48. Format of the token domain parameters object (Version 1)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for token domain object: “DOMP”
4	2	Version: EBCDIC '01'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 38 on page 48)
Object type-specific section		

Table 48. Format of the token domain parameters object (Version 1) (continued)

Offset (decimal) 188 +	**Length of field (bytes)**	**Description**
12	4	TYPE attribute: CKK_DSA or CKK_DH
16	28	Reserved
Algorithm-specific section (DSA)		
44	4	Length in bits of prime *p*
48	128	Reserved
176	128	Prime *p*
304	128	Reserved
432	128	Base *g*
560	20	Reserved
580	20	Subprime *q*
600	636	Reserved
Algorithm-specific section (DH)		
44	4	Length in bits of prime *p*
48	4	Reserved
52	256	Prime *p*
308	256	Reserved
564	256	Base *g*
820	416	Reserved
Variable length attribute section		
1236	2	Length of LABEL attribute in bytes (*aa*)
1238	2	Length of APPLICATION attribute in bytes (*bb*)
1240	20	Reserved
1260	4	Offset of LABEL attribute in bytes
1264	4	Offset of APPLICATION attribute in bytes
1268	40	Reserved
1308	*aa+bb*	Domain parameters attributes (variable length)
1308+*aa+bb*		End of domain parameters object

Table 49. Format of the token domain parameters object (Version 2)

Offset (decimal) 188 +	**Length of field (bytes)**	**Description**
Object header		
0	4	Eye catcher for token domain object: “DOMP”
4	2	Version: EBCDIC '02'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 38 on page 48)
Object type-specific section		
12	4	TYPE attribute: CKK_DSA or CKK_DH
16	28	Reserved
Algorithm-specific section (DSA)		

Table 49. Format of the token domain parameters object (Version 2) (continued)

Offset (decimal) 188 +	**Length of field (bytes)**	**Description**
44	4	Length in bits of prime *p*
48	256	Prime *p*
304	256	Base *g*
560	8	Reserved
568	32	Subprime *q*
600	636	Reserved
Algorithm-specific section (DH)		
44	4	Length in bits of prime *p*
48	4	Reserved
52	256	Prime *p*
308	256	Reserved
564	256	Base *g*
820	416	Reserved
Variable length attribute section		
1236	2	Length of LABEL attribute in bytes (*aa*)
1238	2	Length of APPLICATION attribute in bytes (*bb*)
1240	20	Reserved
1260	4	Offset of LABEL attribute in bytes
1264	4	Offset of APPLICATION attribute in bytes
1268	40	Reserved
1308	*aa*+*bb*	Domain parameters attributes (variable length)
1308+*aa*+*bb*		End of domain parameters object

Table 50. Format of the token data object

Offset (decimal) 188 +	**Length of field (bytes)**	**Description**
Object header		
0	4	Eye catcher for data object: “DATA”
4	2	Version: EBCDIC '00'
6	2	Length of object, in bytes
8	4	Flags (see Table 38 on page 48)
Object type-specific section		
12	4	Reserved for IBM's use
16	28	Reserved for IBM's use
44	2	Length of VALUE attribute in bytes (*aa*)
46	2	Length of OBJECT_ID attribute in bytes (*bb*)
48	2	Length of LABEL attribute in bytes (*cc*)
50	2	Length of APPLICATION attribute in bytes (*dd*)
52	2	Length of ID attribute in bytes (*ee*)
54	22	Reserved for IBM's use

Table 50. Format of the token data object (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
76	4	Offset of VALUE attribute in bytes
80	4	Offset of OBJECT_ID attribute in bytes
84	4	Offset of LABEL attribute in bytes
88	4	Offset of APPLICATION attribute in bytes
92	4	Offset of ID attribute in bytes
96	44	Reserved for IBM's use
140	*aa* + *bb* + *cc* + *dd* + *ee*	Data attributes (variable length)
140 + *aa* + *bb* + *cc* + *dd* + *ee*		End of data object