z/OSCryptographic Services
Integrated Cryptographic Service Facility

PKCS #11 Enhancements for FIPS 140-2 – APAR OA32012

(May 31, 2010)

Contents

Chapter 1. Overview

This document update describes PKCS #11 Enhancements for FIPS 140-2, 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*, SA22-7520-14

The preceding books document capabilities provided by FMID HCR7770, and support z/OS Version 1 Release 11.

Technical changes or additions related to PKCS #11 enhancements for FIPS 140-2 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 APAR OA32012.

Chapter 2. Update of z/OS Cryptographic Services ICSF Writing PKCS #11 Applications, SA23-2231-02, information

This chapter contains updates to the document *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*, SA23-2231-02, for the PKCS #11 enhancements provided by this APAR for FIPS 140-2. Refer to this source document if background information is needed.

Operating in compliance with FIPS 140-2

The National Institute of Standards and Technology (NIST) is the US federal technology agency that works with industry to develop and apply technology, measurements, and standards. One of the standards published by NIST is the Federal Information Processing Standard Security Requirements for Cryptographic Modules, referred to as FIPS 140-2. FIPS 140-2 provides a standard that can be required by organizations who specify that cryptographic-based security systems are to be used to provide protection for sensitive or valuable data.

z/OS PKCS #11 cryptography is designed to meet FIPS 140-2 Level 1 criteria, and can be configured to operate in compliance with FIPS 140-2 specifications. Applications that need to comply with the FIPS 140-2 standard can therefore use the z/OS PKCS #11 services in a way that allows only the cryptographic algorithms (including key sizes) approved by the standard and restricts access to the algorithms that are not approved. There are two modes of FIPS operation:

- The services can be configured so that all z/OS PKCS #11 applications are forced to comply with the FIPS 140-2 standard. This is called *FIPS standard mode*.
- For installations where only certain z/OS PKCS #11 applications need to comply with the FIPS 140-2 standard, the services can be configured so that only the necessary applications are restricted from using the non-approved algorithms and key sizes, while other applications are not. This is called *FIPS compatibility mode*.

You can also use FIPS compatibility mode to test individual applications to ensure FIPS compliance before switching to FIPS standard mode.

ICSF installation options are described in the *z/OS Cryptographic Services ICSF System Programmer's Guide*. The installation option FIPSMODE indicates one of the following:

- the z/OS PKCS #11 services will operate in FIPS standard mode. The installation option to specify this is FIPSMODE(YES, FAIL(*fail-option*)) and is described in more detail in "Requiring FIPS 140-2 compliance from all z/OS PKCS #11 applications" on page 5.
- the z/OS PKCS #11 services will operate in FIPS compatibility mode. The installation option to specify this is FIPSMODE(COMPAT, FAIL(*fail-option*)). When operating in FIPS compatibility mode, it is expected that further specifications will be made to identify which applications must comply with the FIPS 140-2 standard, and which applications do not need to comply. These further specifications can be made:
  - at the PKCS #11 token and application level, using FIPSEXEMPT.*token-name* resource profiles in the CRYPTOZ class.

– within applications themselves for individual keys. When an application creates a key, the application can specify that the key must be used in a FIPS 140-2 compliant fashion. The application can specify this by setting the Boolean key attribute CKA_IBM_FIPS140 to TRUE.

The FIPSMODE(COMPAT, FAIL(*fail-option*)) installation option, FIPSEXEMPT.*token-name* resource profiles, and the CKA_IBM_FIPS140 key attribute, are described in more detail in "Requiring FIPS 140-2 compliance from select z/OS PKCS #11 applications" on page 6.

- no FIPS 140-2 compliance is required by any application. This is the default behavior if the FIPSMODE installation option is not used, but can be set explicitly using the FIPSMODE(NO, FAIL(fail-option)) installation option.

If any z/OS PKCS #11 application intends to use the services in compliance with the FIPS 140-2 standard, then, in accordance with that standard, the integrity of the load module containing the z/OS PKCS #11 services must be checked when ICSF is started. This load module is digitally signed, and, in order for applications using its services to be FIPS 140-2 compliant, the signature must be verified when ICSF is started. For more information, refer to "Requiring signature verification for ICSF module CSFINPV2."

If any application will use PKCS #11 objects for AES Galois/Counter Mode (GCM) encryption or GMAC generation, and will have ICSF generate the initialization vectors, then you need to set ECVTSPLX or CVTSNAME to a unique value. Refer to "Steps to customize SYS1.PARMLIB" on page 28 for more information.

Requiring signature verification for ICSF module CSFINPV2

If your installation needs to operate z/OS PKCS #11 in compliance with the FIPS 140-2 standard, then the integrity of the cryptographic functions shipped by IBM must be verified at your installation during ICSF startup. The load module that contains the software cryptographic functions is SYS1.SIEALNKE(CSFINPV2), and this load module is digitally signed when it is shipped from IBM. Using RACF, you can verify that the module has remained unchanged from the time it was built and installed on your system. To do this, you create a profile in the PROGRAM class for the CSFINPV2 module, and use this profile to indicate that signature verification is required before the module can be loaded.

To requiring signature verification for ICSF module CSFINPV2:

1. Make sure that RACF has been prepared to verify signed programs. As described in *z/OS Security Server RACF Security Administrator's Guide*, a security administrator prepares RACF to verify signed programs by creating a key ring for signature verification, and adding the code-signing CA certificate that is supplied with RACF to the key ring. If RACF has been prepared to verify signed programs, there will be a key ring dedicated to signature verification, the code-signing CA certificate will be attached to the key ring, and the PROGRAM class will be active.
   a. If RACF has been prepared to verify signed programs, the discrete profile IRR.PROGRAM.SIGNATURE.VERIFICATION in the FACILITY class will specify the name the name of the signature-verification key ring. To determine if a signature key ring is already active, enter the command:

      ```
      RLIST FACILITY IRR.PROGRAM.SIGNATURE.VERIFICATION
      ```

      If there is no discrete profile with this name, have your security administrator prepare RACF to verify signed programs using the information in *z/OS Security Server RACF Security Administrator's Guide*.

b. If the signature verification key ring exists, the RLIST command will display information for the discrete profile IRR.PROGRAM.SIGNATURE.VERIFICATION in the FACILITY class. The name of the signature verification key ring and the name of the key ring owner will be included in the APPLICATION DATA field of the RLIST command output. Using this information, enter the RACDCERT LISTRING command to make sure the code-signing CA certificate is attached to the key ring:

```
RACDCERT ID(key-ring-owner) LISTRING(key-ring-name)
```

The label of the code-signing CA certificate is 'STG Code Signing CA'. If this label is not shown in the RACDCERT LISTRING command output, have your security administrator prepare RACF to verify signed programs using the information in *z/OS Security Server RACF Security Administrator's Guide*.

c. Program control must be active in order for RACF to perform signature verification processing. To make sure the PROGRAM class is active, enter the SETROPTS LIST command.

```
SETROPTS LIST
```

The ACTIVE CLASSES field of the command output should include the PROGRAM class. If it does not, have your security administrator prepare RACF to verify signed programs using the information in *z/OS Security Server RACF Security Administrator's Guide*.

2. Create a profile for the CSFINPV2 program module in the PROGRAM class, indicating that the program must be signed. The following command specifies that the program should fail to load if the signature cannot be verified for any reason. This command also specifies that all signature verification failures should be logged.

Note: Due to space constraints, this command example appears on two lines. However, the RDEFINE command should be entered completely on one line.

```
RDEFINE PROGRAM CSFINPV2 ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
 SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))
```

You will need to activate your profile changes in the PROGRAM class.

```
SETROPTS WHEN(PROGRAM) REFRESH
```

Requiring FIPS 140-2 compliance from all z/OS PKCS #11 applications

If all z/OS PKCS #11 applications running on your system must comply with the FIPS 140-2 standard, your installation's system programmer should configure ICSF so that z/OS PKCS #11 operates in FIPS standard mode. To do this:

1. Make sure the integrity of the cryptographic functions shipped by IBM in the ICSF module CSFINPV2 will be verified by RACF before the module is loaded. This is done by following the instructions in "Requiring signature verification for ICSF module CSFINPV2" on page 4. If the these steps are not followed to verify the digital signature of the module, no application calling the z/OS PKCS #11 services can be considered FIPS 140-2 compliant.
2. To specify FIPS standard mode, have you installation's system programmer include the installation option FIPSMODE(YES, FAIL(*fail-option*)) in the ICSF installation options data set.

When this option is used, ICSF will operate in FIPS standard mode. In this mode, ICSF initialization will test that it is running on an IBM System z model type, and a version and release of z/OS, that supports FIPS. If so, then ICSF will perform a series of cryptographic known answer tests as required by the FIPS 140-2 standard. If the tests succeed, then all applications calling z/OS PKCS services will be restricted from using the PKCS #11 algorithms and key sizes that are prohibited by the FIPS 140-2 standard (as outlined in Table 3 on page 13).

If any of the installation tests should fail, the action ICSF initialization takes depends on the *fail-option* specified. The *fail-option* within the FIPSMODE(YES, FAIL(*fail-option*)) installation option can be either:

- YES (which indicates that ICSF should terminate abnormally if there is a failure in any of the tests that are performed).
- NO (which indicates that ICSF initialization processing should continue even if there is a failure in one or more of the tests that are performed). If an initialization test does fail, however, PKCS #11 support will be limited or nonexistent depending on the test that failed.
  - If ICSF is running on an IBM system z model type or with a version of z/OS that does not support FIPS, most FIPS processing is bypassed. PKCS #11 callable services will be available, but ICSF will not adhere to FIPS 140 restrictions. Requests to generate or use a key with CKA_IBM_FIPS140=TRUE will result in a failure return code.
  - If a known answer test failed, all ICSF PKCS #11 callable services will be unavailable.

For more information on this on other ICSF installation options, refer to *z/OS Cryptographic Services ICSF System Programmer's Guide*.

Requiring FIPS 140-2 compliance from select z/OS PKCS #11 applications

If only certain z/OS PKCS #11 applications running on your system must comply with the FIPS 140-2 standard, while other z/OS PKCS #11 applications do not, your system programmer should configure ICSF so that z/OS PKCS #11 operates in FIPS compatibility mode. In FIPS compatibility mode, you can use resource profiles in the CRYPTOZ class to specify, at a token level, the applications that are exempt from FIPS 140-2 compliance and, for that reason, should not be subject to FIPS restrictions. To configure the z/OS PKCS #11 services to operate in FIPS compatibility mode:

1. Make sure the integrity of the cryptographic functions shipped by IBM in the module ICSF module CSFINPV2 will be verified by RACF before the module is loaded. This is done by following the instructions in "Requiring signature verification for ICSF module CSFINPV2" on page 4. If the these steps are not followed to verify the digital signature of the module, no application calling the z/OS PKCS #11 services can be considered FIPS 140-2 compliant.
2. To specify FIPS compatibility mode, have you installation's system programmer include the installation option FIPSMODE(COMPAT, FAIL(*fail-option*)) in the ICSF installation options data set.

   When this option is used, ICSF will operate in FIPS compatibility mode. In this mode, ICSF initialization will test that it is running on a IBM System z model type, and a version and release of z/OS, that supports FIPS. If so, then ICSF will perform a series of cryptographic known answer tests as required by the FIPS 140-2 standard. If the tests are successful, then, by default, all applications calling z/OS PKCS services will be restricted from using the PKCS #11 algorithms and key sizes that are prohibited by the FIPS 140-2 standard (as

outlined in Table 3 on page 13). Using profiles in the CRYPTOZ class, however, you can identify applications that are exempt from FIPS 140-2 compliance (as described in the next step).

If any of the installation tests should fail, the action ICSF initialization takes depends on the *fail-option* specified. The *fail-option* within the FIPSMODE(COMPAT, FAIL(*fail-option*)) installation option can be either:

- YES (which indicates that ICSF should terminate abnormally if there is a failure in any of the tests that are performed).
- NO (which indicates that ICSF initialization processing should continue even if there is a failure in one or more of the tests that are performed). If an initialization test does fail, however, PKCS #11 support will be limited or nonexistent depending on the test that failed.
  - If ICSF is running on an IBM system z model type or with a version of z/OS that does not support FIPS, most FIPS processing is bypassed. PKCS #11 callable services will be available, but ICSF will not adhere to FIPS 140 restrictions. Requests to generate or use a key with CKA_IBM_FIPS140=TRUE will result in a failure return code.
  - If a known answer test failed, all ICSF PKCS #11 callable services will be unavailable.

For more information on this on other ICSF installation options, refer to *z/OS Cryptographic Services ICSF System Programmer's Guide*.

3. To specify which applications must comply with FIPS 140-2 restrictions and which applications do not need to comply, create FIPSEXEMPT.*token-label* resource profiles in the CRYPTOZ class. If no FIPSEXEMPT.*token-label* resource profiles are created, then all z/OS PKCS #11 applications will be subject to FIPS restrictions. By creating a FIPSEXEMPT.*token-label* resource profile for a particular token, however, you can specify whether or not a particular user ID should be considered exempt from FIPS restrictions.
   - If a user ID has access authority NONE to the FIPSEXEMPT.*token-label* resource, ICSF will enforce FIPS 140-2 compliance for that user ID.
   - If a user ID has access authority READ to the FIPSEXEMPT.*token-label* resource, that user ID is exempt from FIPS 140-2 restrictions.

   To specify which applications must comply with the FIPS 140-2 restrictions, and which do not, the security administrator must:

   a. If it is not already activated, activate the CRYPTOZ class with generics and RACLIST it:

      ```
      SETROPTS CLASSACT(CRYPTOZ) GENERIC(CRYPTOZ) RACLIST(CRYPTOZ)
      ```

   b. Create the FIPSEXEMPT.*token-label* resource profile for each z/OS PKCS #11 token. The following command creates the profile for the omnipresent session-object token SYSTOK-SESSION-ONLY.

      ```
      RDEF CRYPTOZ FIPSEXEMPT.SYSTOK-SESSION-ONLY UACC(NONE)
      ```

      Although the use of generic profiles in the CRYPTOZ class is permitted, we recommend you begin the profile name with "FIPSEXEMPT". Failure to do this could result in generic characters unintentionally matching the SO.*token-label* or USER.*token-label* resources for token access, and so could have unintended consequences.

   c. Using the PERMIT command, specify READ access authority for user IDs that are exempt from FIPS 140-2 restrictions, and NONE access authority for user IDs that must comply with FIPS 140-2. The following command indicates that all user IDs are exempt, except for the daemon user ID BOGD.

```
PERMIT FIPSEXEMPT.SYSTOK-SESSION-ONLY CLASS(CRYPTOZ) ID(*) ACC(READ)
PERMIT FIPSEXEMPT.SYSTOK-SESSION-ONLY CLASS(CRYPTOZ) ID(BOGD) ACC(NONE)
```

d. Refresh the CRYPTOZ class in common storage:

```
SETROPTS RACLIST(CRYPTOZ) REFRESH
```

Specifying FIPS 140-2 compliance from within a z/OS PKCS #11 application

When running in FIPS compatability mode, a PKCS #11 application can, when creating a key, specify that generation and subsequent use of the key must adhere to FIPS 140-2 restrictions. An application specifies this by setting the Boolean attribute CKA_IBM_FIPS140 to TRUE when creating the key. If an application does this, the FIPS 140-2 restrictions (as outlined in Table 3 on page 13) will be enforced for the key regardless of any specifications made at the token level using FIPSEXEMPT.*token-label* resource profiles.

An application controls FIPS 140-2 compliance for a key when in FIPS compatibility mode as specified by the FIPSMODE(COMPAT, FAIL(*fail-option*)) installation option. If the installation option FIPSMODE(NO, FAIL(*fail-option*)), which indicates no FIPS 140-2 compliance for any application, is specified (or defaulted to), an application that sets the Boolean attribute CKA_IBM_FIPS140 to TRUE will fail with return/reason code 8/3069. If the FIPSMODE(YES, FAIL(*fail-option*)) installation option is specified, indicating FIPS 140-2 compliance is required by all applications, setting the Boolean attribute CKA_IBM_FIPS140 to TRUE is merely redundant and does not result in an error.

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 1024 bits (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
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

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

Type (CK_MECHANISM_TYPE)	Size factor	Flags
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
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

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

Type (CK_MECHANISM_TYPE)	Size factor	Flags
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 9.

[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).

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

Machine type and cryptographic hardware	Mechanisms supported	Notes
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).

The following table lists the algorithms and uses (by mechanism) that are not allowed when operating in compliance with FIPS 140-2. For more information about how the z/OS PKCS #11 services can be configured to operate in compliance with the FIPS 140-2 standard, refer to "Operating in compliance with FIPS 140-2" on page 3.

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	Key sizes less than 1024 bits
DSA	CKM_DSA_SHA1, CKM_DSA	Key sizes less than 1024 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_CBC	All
RC4	CKM_RC4	All
RSA	CKM_RSA_X_509	All
	CKM_RSA_PKCS	Key sizes less than 1024 bits
ECC	CKM_ECDSA, CKM_ECDSA_SHA1, CKM_ECDH1_DERIVE	Brainpool curves

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

Algorithm	Mechanisms	Usage disallowed
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

Chapter 3. Update of z/OS Cryptographic Services ICSF Application Programmer's Guide, SA22-7522-13, information

This chapter contains updates to the document *z/OS Cryptographic Services ICSF Application Programmer's Guide*, SA22-7522-13, for the PKCS #11 enhancements provided by this APAR for FIPS 140-2. Refer to this source document if background information is needed.

PKCS #11 Secret key encrypt (CSFPSKE)

Use the PKCS #11 secret key encrypt callable service to encipher data using a clear symmetric key. AES, DES, BLOWFISH, and RC4 are supported. This service supports CBC, ECB, Galois/Counter, and stream modes and PKCS #7 padding. The key handle must be a handle of a PKCS #11 secret key object. The CKA_ENCRYPT attribute must be true.

If the length of output field is too short to hold the output, the service will fail and return the required length of the output field in the *cipher_text_length* parameter.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPSKE6.

Format

```
CALL CSFPSKE(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             rule_array_count,
             rule_array,
             key_handle,
             initialization_vector_length,
             initialization_vector,
             chain_data_length,
             chain_data,
             clear_text_length,
             clear_text,
             clear_text_id,
             cipher_text_length,
             cipher_text,
             cipher_text_id )
```

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 0, 1, 2, or 3.

rule array

Direction: Input Type: String

Keywords that provide control information to the callable service.

Table 4. Keywords for secret key encrypt

Keyword	Meaning
Encryption Mechanism (Optional. No default. If not specified, mechanism will be taken from key type of secret key. If specified , must match key type)	
AES	AES algorithm will be used.
DES	DES algorithm will be used. This is only single-key encryption.
DES3	DES3 algorithm will be used, This includes double- and triple-key encryption.
BLOWFISH	BLOWFISH algorithm will be used.
RC4	RC4 algorithm will be used. This is a stream cipher.
Processing Rule (optional)	
CBC	Performs cipher block chaining. The text length must be a multiple of the block size for the specified algorithm (8 bytes for DES, DES3, and BLOWFISH, 16 bytes for AES). CBC is the default value for DES, DES3, AES, and BLOWFISH. CBC cannot be specified for RC4.
CBC-PAD	Performs cipher block chaining. Except for FINAL and ONLY chaining calls, the clear text length must be a multiple of the block size for the specified algorithm. For FINAL and ONLY calls: • The clear text length may be shorter than the block size and may even be zero. • PKCS #7 padding is performed. Thus, the cipher text will always be longer than the clear text. CBC-PAD cannot be specified for BLOWFISH or RC4.

Table 4. Keywords for secret key encrypt (continued)

Keyword	Meaning
ECB	Performs electronic code book encryption. The text length must be a multiple of the block size for the specified algorithm. ECB cannot be specified for BLOWFISH or RC4.
GCM	Performs Galois/Counter mode encryption. The clear text length may be shorter than the block size and may even be zero. The authentication tag is returned appended to the cipher text. GCM may only be specified with AES. GMAC is a specialized form of GCM where no plain text is specified.
GCMIVGEN	Performs similarly to the GCM processing rule except that ICSF will generate part of the initialization vector and return it in the *initialization_vector* parameter. Having ICSF generate the initialization vector ensures that initialization vectors are never repeated for a given key object.
STREAM	Performs a stream cipher. STREAM cannot be specified for BLOWFISH, DES, DES3, or AES. STREAM is the default value for RC4.
Chaining Selection (optional)	
INITIAL	Specifies this is the first call in a series of chained calls. For cipher block chaining, the initialization vector is taken from the *initialization_vector* parameter. Intermediate results are stored in the chain_data field. Cannot be specified with processing rule ECB, GCM, or GCMIVGEN.
CONTINUE	Specifies this is a middle call in a series of chained calls. Intermediate results are read from and stored in the *chain_data* field. Cannot be specified with processing rule ECB, GCM, or GCMIVGEN.
FINAL	Specifies this is the last call in a series of chained calls. Intermediate results are read from the *chain_data* field. Cannot be specified with processing rule ECB, GCM, or GCMIVGEN.
ONLY	Specifies this is the only call and the call is not chained. For cipher block chaining, the initialization vector is taken from the *initialization_vector* parameter. For Galois Counter mode, the initialization parameters are taken from the *initialization_vector* parameter. ONLY is the default chaining.

key_handle

Direction: Input Type: String

The 44-byte handle of secret key object.

Initialization_vector_length

Direction: Input Type: Integer

Length of the *initialization_vector* in bytes. For CBC and CBC-PAD, this must be 8 bytes for DES and BLOWFISH and 16 bytes for AES. For GCM and GCMVGEN, this must be the size of the *initialization_vector* field (28 bytes).

initialization_vector

Direction: Input Type: String

This field has a varying format depending on the mechanism specified. For CBC and CBC-PAD this is the 8 or 16 byte initial chaining value. The format

for GCM and GCMIVGEN are shown in the following tables.

Table 5. initialization_vector parameter format for GCM mechanism

Offset	Length in bytes	Direction	Description
0	4	Input	length in bytes of the initialization vector area. The minimum value is 1. The maximum value is 128. 12 is recommended.
4	8	Input	64-bit address of the initialization vector area. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers.
12	4	Input	length in bytes of the additional authentication data. The minimum value is 0. The maximum value is 1048576.
16	8	Input	64-bit address of the additional authentication data. 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 of the additional authentication data is zero.
24	4	Input	Length in bytes of the desired authentication tag. This value must be one of 4, 8, 12, 13, 14, 15, or 16.

Table 6. initialization_vector parameter format for GCMIVGEN mechanism

Offset	Length in bytes	Direction	Description
0	4	Input	Nonce value which ICSF is to use as the first 4 bytes of the initialization vector. The remaining 8 bytes will be generated and returned to the caller in the initialization vector area.
4	8	Input	64-bit address of the initialization vector area into which ICSF will store the 8 bytes it generates. The area must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers. The complete initialization vector to be used for decryption is the 4-byte nonce concatenated with the 8 bytes stored in the area
12	4	Input	length in bytes of the additional authentication data. The minimum value is 0. The maximum value is 1048576.
16	8	Input	64-bit address of the additional authentication data. 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 of the additional authentication data is zero.
24	4	Input	Length in bytes of the desired authentication tag. This value must be one of 4, 8, 12, 13, 14, 15, or 16.

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 an INITIAL call, and may change it on subsequent CONTINUE calls. Your application must not change the data in this field between the sequence of INITIAL, CONTINUE, and FINAL calls for a specific message. The chain data has the following format:

Table 7. 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

clear_text_length

Direction: Input Type: Integer

Length of the *clear_text* parameter in bytes. Except for processing rules GCM and GCMIVGEN, the length can be up to 2147483647. For processing rules GCM and GCMIVGEN, the length cannot exceed 1048576.

clear_text

Direction: Input Type: String

Text to be encrypted

clear_text_id

Direction: Input Type: Integer

The ALET identifying the space where the clear text resides.

cipher_text_length

Direction: Input/Output Type: Integer

On input, the length in bytes of the *cipher_text* parameter. On output, the length of the text encrypted into the *cipher_text* parameter.

cipher_text

Direction: Output Type: String

Encrypted text

cipher_text_id

Direction: Output Type: Integer

The ALET identifying the space where the cipher text resides.

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

If the INITIAL rule is used to start a series of chained calls:

- The key used to initiate the chained calls must not be deleted until the chained calls are complete.
- The application should make a FINAL call to free ICSF resources allocated. If processing is to be aborted without making a FINAL 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.

GCM encryption may be used to produce a GMAC on some authentication data. To do this, request AES encryption with processing rule GCM or GCMVGEN. The *clear_text_length* field must be set to zero. The authentication tag (the GMAC) is returned in the *cipher_text* field.

For Processing Rule GCMIVGEN, the total number of initialization vector generations for a token key object is limited to 4294967295. Once this number is exceeded, the key object will no longer be eligible for Processing Rule GCMIVGEN and is considered "retired". This usage counter is maintained in the TKDS as part of the key object. For keys that are copied using CSFPTRC (C_CopyObject), the existing counter value is copied to the new key object, but not synchronized after that.

For Processing Rule GCMIVGEN, session key objects have no maximum lifetime. They may be retired at any time. Once retired, the key object will no longer be eligible for Processing Rule GCMIVGEN.

For Processing Rule GCMIVGEN, the nonce value portion of the initialization vector is predetermined by the caller. It is used to ensure that initialization vector values are not repeated for any given key value. The caller should provide a random value and change the value as often as practical. It must be changed whenever:

- a given key value is replicated as a new persistent key object
- a given persistent key object is replicated as a new session key object
- a given session key value is re-instantiated after system IPL
- a given key value is re-instantiated after ICSF indicates it has been retired

Use of Processing Rule GCMIVGEN with token key objects requires that the first 4 bytes of ECVTSPLX or CVTSNAME be set to a unique value with respect to other systems. See *z/OS Cryptographic Services ICSF System Programmer's Guide*, SA22-7520 for information on how to set these fields.

A session key object should never be used for Processing Rule GCMIVGEN if the key value is distributed to multiple systems outside the current sysplex where new initialization vectors may be generated. Use only token key objects in such cases. If session key objects are used, the other systems must use different nonces.

For Processing Rule GCMIVGEN, the 8 bytes of generated initialization vector are stored back into the initialization vector area before the GCM operation is performed. This allows the generated initialization vector to be part of the additional authentication data, if desired.

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 or ECDSA
- SHA-256 with RSA-PKCS 1.5 or ECDSA
- SHA-384 with RSA-PKCS 1.5 or ECDSA
- SHA-512 with RSA-PKCS 1.5 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 8. 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.
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* field.
MIDDLE	Specifies this is a middle call in a series of chained calls. Intermediate results are stored in the *hash* and *chain_data* field.
LAST	Specifies this is the last call in a series of chained calls.

Table 8. Keywords for one-way hash generate (continued)

Keyword	Meaning
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.
SIGN-RSA	The data is to be hashed then signed using RSA-PKCS 1.5 formatting. Any hash method is acceptable except RPMD-160.
SIGN-DSA	The data is to be hashed then signed using DSA. The hash method must be SHA-1.
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.
VER-DSA	The data is to be hashed then signature verified using DSA. The hash method must be SHA-1.
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 9. 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.

Reason Codes for Return Code 8 (8)

Table 10 lists reason codes reason codes added by the PKCS #11 enahancements for FIPS 140-2. These reason codes are returned from certain callable services that give return code 8.

Table 10. Reason Codes for Return Code 8 (8)

Reason Code Hex (Decimal)	**Description**
BFC (3068)	A cryptographic operation using a specific PKCS #11 key object is being requested. The key object has exceeded its useful life for the operation requested. The request is not processed. **User action**: Use a different key.

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

Chapter 4. Update of z/OS Cryptographic Services ICSF System Programmer's Guide, SA22-7520-14, information

This chapter contains updates to the document *z/OS Cryptographic Services ICSF System Programmer's Guide*, SA22-7520-14, for the PKCS #11 enhancements provided by this APAR for FIPS 140-2. Refer to this source document if background information is needed.

Table 11. 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
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 12. 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
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

Steps to customize SYS1.PARMLIB

The installation options data set you will create is generally stored in SYS1.PARMLIB. If your administrator does not have access to SYS1.PARMLIB, you need to use another data set instead.

Update the data set you are using as follows:

1. Add CEE.SCEERUN and CSF.SCSFMOD0 to the LNKLST concatenation. This adds the ICSF library to the z/OS library search. This is an example of an ICSF entry to the LNKLST concatenation.

   ```
   CSF.SCSFMOD0
   ```

2. APF authorize CSF.SCSFMOD0, if LNKAUTH=APFTAB. This is an example of an ICSF entry for APF authorization.

```
APF ADD DSNAME(CSF.SCSFMOD0)  VOLUME(******)
```

3. In the IKJTSOxx parameter, add CSFDAUTH and CSFDPKDS as a value in the AUTHPGM parameter list and in the AUTHTSF parameter list. This is an example of an ICSF entry in the IKJTSOxx member.

```
AUTHPGM NAMES(        /* AUTHORIZED PROGRAMS             */  +
 ....
 ....
CSFDAUTH              /* ICSF                            */  +
CSFDPKDS              /* ICSF                            */  +

 ....

AUTHTSF NAMES(        /* PROGRAMS TO BE AUTHORIZED WHEN  */  +
                      /* WHEN CALLED THROUGH THE TSO     */  +
                      /* SERVICE FACILITY                */  +
 ....
 ....
CSFDAUTH              /* ICSF                            */  +
CSFDPKDS              /* ICSF                            */
```

4. If your application programmers intend to use PKCS #11 token key objects for AES Galois/Counter Mode (GCM) encryption or GMAC generation, and have ICSF generate the initialization vectors, then you need to set ECVTSPLX or CVTSNAME to a unique value.

   This needs to be done, because, for AES GCM encryption or GMAC generation, the security of the algorithm is dependent on never repeating a key, initialization vector combination for two or more distinct sets of data. In PKCS #11, applications can request that ICSF generate a new (unique) initialization vectors each time AES GCM or GMAC is initiated. In fact, this is the only permitted way to perform AES GCM or GMAC when PKCS #11 is operating in FIPS mode. When ICSF generates initialization vectors, it uses the ECVTSPLX (sysplex mode) or CVTSNAME (non-sysplex mode) field as the cryptographic module name. The name ensures uniqueness if such keys are distributed to multiple systems, but only if each system is set with a unique name.

   When setting ECVTSPLX or CVTSNAME to unique values, be aware that ICSF uses only the first (left most) 4 characters of these fields. For this reason, these 4 characters must be set to uniquely identify the system.

   For example, suppose AES key value 123 is created on the current single-image system (known as System A) and is distributed to another system residing in a Sysplex (known as Sysplex B). Both systems will be performing GCM encryption where ICSF generates the initialization vectors. To ensure that unique initialization vectors are generated, set CVTSNAME=SYSA on System A and ECVTSPLX=PLXB on Sysplex B.

   CVTSNAME is normally set from the SYSNAME=value statement in the IEASYSxx member of "SYS1.PARMLIB". For more information, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592.

   ECVTSPLX is normally set from the COUPLE SYSPLEX(value) in the COUPLExx member of "SYS1.PARMLIB". For more information, see *z/OS MVS Setting Up a Sysplex*, SA22-7625.

Notes:

1. If you will be using TKE V3.0 or higher on this host, you should also add CSFTTKE as a value in the AUTHCMD parameter list.
2. If you will only be using ICSF for SMP/E electronic delivery, this step does not need to be performed. TKE is not needed for SMP/E electronic delivery.

3. To change the active IKJTSOxx member of SYS.PARMLIB without an IPL, use the PARMLIB UPDATE command.

z/OS MVS Initialization and Tuning Guide and *z/OS MVS Initialization and Tuning Reference* provide more information.

Parameters in the installation options data set

The installation options data set is an intended programming interface.

When specifying parameter values within parentheses, leading and trailing blanks are ignored. Embedded blanks may cause unpredictable results.

Support is provided for the use of system symbols in the installation options data set. System symbols can be used as values for any of the parameters. System symbols are specified in the IEASYMxx member of SYS1.PARMLIB; the IEASYM statement of the LOADxx member of SYS1.PARMLIB specifies the IEASYMxx member(s) to be used for the resolution of system symbols. This example shows the use of a system symbol for specifying the domain to be used for the start of ICSF:

```
DOMAIN(&PARDOM.)
```

When the Installation Options Data Set is processed during the start of ICSF, the value of the system symbol PARDOM will be substituted as the value of the DOMAIN parameter.

For the first start, you specified an empty VSAM data set name for the CKDS in the CKDSN option, an empty VSAM data set name for the PKDS in the PKDSN option, and SSM(YES). You may want to change these and other options for subsequent starts. Here is a complete list of installation options:

CHECKAUTH(YES or NO)
Indicates whether ICSF performs security access control checking of Supervisor State and System Key callers. If you specify CHECKAUTH(YES), ICSF issues RACROUTE calls to perform the security access control checking and the results are logged in RACF SMF records that are cut by RACF. If you specify CHECKAUTH(NO), the authorization checks against resources in the CSFSERV class are not performed resulting in a significant performance enhancement for supervisor state and system key callers. However, the authorization checks are not logged in the RACF SMF records.

If you do not specify the CHECKAUTH option, the default is CHECKAUTH(NO).

If you configure CHECKAUTH(YES) in the ICSF options dataset, the Health Checker address space user identity must be authorized to the CSFRKL profile in class CSFSERV for the ICSFMIG7731_ICSF_RETAINED_RSAKEY migration check to successfully execute. However, you have no action to take if you choose not to run the migration check. If you configure CHECKAUTH(NO), there is no requirement to authorize the Health Checker user identity for any ICSF profiles or classes, since the check routine executes in supervisor state. This is not an implementation consideration, but rather a check deployment or activation time customer administration consideration.

CKDSN(data-set-name)
Specifies the CKDS name the system uses to start ICSF. Whenever you

restart ICSF, the CKDS named in the CKDSN option becomes the active in-storage CKDS. (At first-time startup, you should specify the name of an empty VSAM data set you created to use as the CKDS.)

If you do not specify this keyword, ICSF does not become active. There is no default for this option, so you must specify a value.

CKTAUTH(YES or NO)
Decides if authentication will be performed for every CKDS record read from DASD.

Note: If the active CKDS does not use record level authentication, the CKTAUTH option will be ignored. It will be displayed as DISABLED on the Installation Options Display panel.

YES Indicates authentication will be performed.

NO Indicates no authentication will be performed.

COMPAT(YES, NO, or COEXIST)
Indicates whether ICSF runs in compatibility mode, non-compatibility mode, or coexistence mode with PCF.

YES Indicates **compatibility mode**.

In compatibility mode, you can run a PCF application on ICSF, because ICSF supports the PCF macros. You do not have to reassemble PCF applications to do this. You cannot start PCF at the same time as ICSF on the same operating system.

NO Indicates **non-compatibility mode**. In noncompatibility mode, you run PCF applications on PCF and ICSF applications on ICSF. You cannot run PCF applications on ICSF, because ICSF does not support the PCF macros in this mode. PCF can be started at the same time as ICSF on the same operating system. You can start ICSF and then start PCF, or you can start PCF and then start ICSF.

You should use noncompatibility mode unless you are migrating from PCF to ICSF.

COEXIST Indicates **coexistence mode**.

In coexistence mode, you can run a PCF application on PCF, or you can reassemble the PCF application to run on ICSF. To do this, you reassemble the application against coexistence macros that are shipped with ICSF. You can start PCF at the same time as ICSF on the same operating system.

If you do not specify the COMPAT option, the default value is COMPAT(NO).

When you initialize ICSF for the first time, noncompatibility mode must be active. Therefore, at first-time startup, you must specify COMPAT(NO) or allow the default to be used.

COMPENC(DES or CDMF)
This keyword is no longer supported but is tolerated.

DOMAIN(n)

Specifies the number of the domain that you want to use for this start of ICSF. You can specify only one domain in the options data set.

DOMAIN is an optional parameter. The DOMAIN parameter is only required if more than one domain is specified as the usage domain on the PR/SM panels or if running in native mode. If specified in the options data set, it will be used and it must be one of the usage domains for the LPAR.

If DOMAIN is not specified in the options data set, ICSF determines which domains are available in this LPAR. If only one domain is defined for the LPAR, ICSF will use it. If more than one is available, ICSF will issue error message CSFM409E.

The cryptographic processors support multiple sets of master key registers, which the specific domain values identify.

- The Cryptographic Coprocessor Feature has a master key register for the DES master key, the auxiliary DES master key, the signature master key and the key management master key. The auxiliary DES master key register may contain either the new or old DES master key. On the PCI Cryptographic Coprocessor, each domain has a master key register for the current, new, and old SYM-MK and ASYM-MK.
- The PCIXCC/CEX2C/CEX3C has master key registers for the DES-MK, AES-MK and ASYM-MK master keys. Each domain has a master key register for the current, new, and old DES-MK, AES-MK and ASYM-MK.

For more information about partitions and running different configurations, see *z/OS Cryptographic Services ICSF Overview*.

If you run ICSF in compatibility or coexistence mode, you cannot change the domain number without re-IPLing the system. A re-IPL ensures that a program does not access a cryptographic service with a key that is encrypted under a different master key. If you are certain that no cryptographic applications are still running, you can:

1. Stop CSF
2. Start CSF in COMPAT(NO) mode with a different domain number
3. Stop CSF
4. Start CSF in compatibility or coexistence mode with a different domain number.

EXIT(ICSF-name,load-module-name,FAIL(fail-option))

Indicates information about an installation exit.

The *ICSF-name* is the identifier for each exit. Table 13 on page 33 lists all the ICSF exit names and explains when ICSF calls each exit. The load module name is the name of the module that contains the exit. The name can be any valid name your installation chooses.

Using the FAIL keyword of the EXIT statement, you specify the action ICSF, the KGUP, or the PCF conversion program takes if the exit ends abnormally. The fail action that you specify applies to subsequent calls of the exit. If an exit ends abnormally, ICSF takes a system dump. The exit is protected with an ESTAE or the ICSF service functional recovery routine (FRR).

In general, you can specify one of these values for a fail option:

NONE
No action is taken. The exit can be called again and will end abnormally again.

EXIT The exit is no longer available to be called again.

SERVICE
The service or program that called the exit is no longer available to be called again.

ICSF ICSF or the key generator utility program or the PCF conversion program is ended, depending on the exit.

Some fail options are not valid for a specific exit. If you specify a fail option that is not valid, ICSF uses the next valid fail option. For example, if SERVICE is not a valid fail option for an exit, ICSF uses the EXIT option. EXIT is responsible for logging in SMF the results of any authorization checks that are made.

Table 13. Exit Identifiers and Exit Invocations

Exit Identifiers	Exit Invocations
CSFEXIT1	Gets control after the operator issues the START command, but before processing takes place. **Note:** You must not specify an EXIT statement for the first mainline exit, CSFEXIT1.
CSFEXIT2	Gets control after ICSF reads and interprets the installation options data set.
CSFEXIT3	Gets control before ICSF completes initialization.
CSFEXIT4	Gets control after the operator issues the STOP command to stop ICSF.
CSFEXIT5	Gets control when the operator issues the MODIFY command to modify ICSF.
CSFEMK	Gets control during the compatibility service for the PCF EMK macro.
CSFGKC	Gets control during the compatibility service for the PCF GENKEY macro.
CSFRTC	Gets control during the compatibility service for the PCF RETKEY macro.
CSFEDC	Gets control during the compatibility service for the PCF CIPHER macro.
CSFCKDS	Gets control when a callable service retrieves an entry from the CKDS.
CSFKGUP	Gets control during key generator utility program initialization, processing, and termination.
CSFCONVX	Gets control when you run the PCF CKDS conversion program.
CSFSRRW	Gets control when an access to a single record in the CKDS is made using the key entry hardware.
CSFAEGN	Gets control during the ANSI X9.17 EDC generate callable service.
CSFAKEX	Gets control during the ANSI X9.17 key export callable service.
CSFAKIM	Gets control during the ANSI X9.17 key import callable service.
CSFAKTR	Gets control during the ANSI X9.17 key translate callable service.
CSFATKN	Gets control during the ANSI X9.17 transport key partial notarize callable service.
CSFCKI	Gets control during the clear key import callable service.
CSFCPE	Gets control during the clear PIN encrypt callable service.
CSFCPA	Gets control during the clear PIN generate alternate callable service.
CSFCTT	Gets control during the ciphertext translate callable service.
CSFCTT1	Gets control during the ciphertext translate (with ALET) callable service.
CSFPGN	Gets control during the Clear PIN generate callable service.
CSFCVT	Gets control during the control vector translate callable service.

Table 13. Exit Identifiers and Exit Invocations (continued)

Exit Identifiers	Exit Invocations
CSFCVE	Gets control during the cryptographic variable encipher callable service.
CSFDKX	Gets control during the data key export callable service.
CSFDKM	Gets control during the data key import callable service.
CSFDEC	Gets control during the decipher callable service.
CSFDEC1	Gets control during the decipher (with ALET) callable service.
CSFDCO	Gets control during the decode callable service.
CSFDSG	Gets control during the digital signature generate service.
CSFDSV	Gets control during the digital signature verify callable service.
CSFDKG	Gets control during the diversified key generate callable service.
CSFENC	Gets control during the encipher callable service.
CSFENC1	Gets control during the encipher (with ALET) callable service.
CSFECO	Gets control during the encode callable service.
CSFEPG	Gets control during the encrypted PIN generate callable service.
CSFPTR	Gets control during the encrypted PIN translate callable service.
CSFPVR	Gets control during the encrypted PIN verify callable service.
CSFKEX	Gets control during the key export callable service.
CSFKGN	Gets control during the key generate callable service.
CSFKIM	Gets control during the key import callable service.
CSFKPI	Gets control during the key part import callable service.
CSFKRC	Gets control during the key record create callable service.
CSFKRD	Gets control during the key record delete callable service.
CSFKRR	Gets control during the key record read callable service.
CSFKRW	Gets control during the key record write callable service.
CSFKYT	Gets control during the key test callable service.
CSFKYTX	Gets control during the key test extended callable service.
CSFMDG	Gets control during the MDC generate callable service.
CSFKTR	Gets control during the key translate callable service.
CSFMGN1	Gets control during the MAC generate (with ALET) callable service.
CSFMVR	Gets control during the MAC verify callable service.
CSFMVR1	Gets control during the MAC verify (with ALET) callable service.
CSFMDG1	Gets control during the MDC generate (with ALET) callable service.
CSFMGN	Gets control during the MAC generate callable service.
CSFCKM	Gets control during the multiple clear key import callable service.
CSFSKM	Gets control during the multiple secure key import callable service.
CSFOWH	Gets control during the one-way hash generate callable service.
CSFOWH1	Gets control during the one-way hash generate (with ALET) callable service.
CSFPCI	Gets control during the PCI interface callable service.
CSFPCU	Gets contol during the PIN Change/Unblock callable service
CSFPEX	Gets control during the prohibit export callable service.
CSFPEXX	Gets control during the prohibit export extended callable service.

Table 13. Exit Identifiers and Exit Invocations (continued)

Exit Identifiers	Exit Invocations
CSFPKD	Gets control during the PKA decrypt callable service.
CSFPKE	Gets control during the PKA encrypt callable service.
CSFPKG	Gets control during the PKA key generate callable service.
CSFPKI	Gets control during the PKA key import callable service.
CSFPKT	Gets control during the PKA key translate callable service.
CSFPKTC	Gets control during the PKA key token change callable service.
CSFPKX	Gets control during the PKA Public Key Extract callable service.
CSFPKRC	Gets control during the PKDS record create callable service.
CSFPKRD	Gets control during the PKDS record delete callable service.
CSFPKRR	Gets control during the PKDS record read callable service.
CSFPKRW	Gets control during the PKDS record write callable service.
CSFPKSC	Gets control during the PKSC interface callable service.
CSFRNG	Gets control during the random number generate callable service.
CSFRNGL	Gets control during the random number generate long callable service.
CSFRKD	Gets control during the retained key delete callable service.
CSFRKL	Gets control during the retained key list callable service.
CSFRKX	Gets control during the remote key export callable service.
CSFSKI	Gets control during the secure key import callable service.
CSFSKY	Gets control during the secure messaging for keys callable service.
CSFSMG	Gets control during the symmetric MAC generate callable service.
CSFSMG1	Gets control during the symmetric MAC generate (with ALET) callable service.
CSFSMV	Gets control during the symmetric MAC verify callable service.
CSFSMV1	Gets control during the symmetric MAC verify (with ALET) callable service.
CSFSPN	Gets control during the secure messaging for PINs callable service.
CSFSBC	Gets control during the SET block compose callable service.
CSFSBD	Gets control during the SET block decompose callable service.
CSFSYX	Gets control during the symmetric key export callable service.
CSFSYG	Gets control during the symmetric key generate callable service.
CSFSYI	Gets control during the symmetric key import callable service.
CSFTBC	Gets control during the trusted block create callable service.
CSFTCK	Gets control during the transform CDMF key callable service.
CSFTRV	Gets control during the transaction validation callable service
CSFUDK	Gets control during the user derived key callable service.
CSFCSG	Gets control during the VISA CVV service generate callable service.
CSFCSV	Gets control during the VISA CVV service verify callable service.

Note: z/OS no longer ships IBM-supplied security exit routines; the security exit points remain. Users of z/OS should use the Security Server (RACF) or an equivalent product to obtain access checking of services and keys. ICSF no longer needs these exit routines.

FIPSMODE(YES or COMPAT or NO,FAIL(fail-option))

Indicates whether z/OS PKCS #11 services must run in compliance with the Federal Information Processing Standard Security Requirements for Cryptographic Modules, referred to as FIPS 140-2. FIPS 140-2, published by the National Institute of Standards and Technology (NIST), is a standard that defines rules and restrictions for how cryptographic modules should protect sensitive or valuable information. The standard is available at http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf.

By configuring z/OS PKCS #11 services to operate in compliance with FIPS 140-2 specifications, installations or individual applications can use the z/OS PKCS #11 services in a way that allows only the cryptographic algorithms (including key sizes) approved by the standard, and restricts access to the algorithms that are not approved. For more information, refer to *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

YES Indicates that the z/OS PKCS #11 services will operate in *FIPS standard mode*. Any application using the PKCS #11 services will be forced to use those services in a FIPS-compliant manner. Applications will not have access to the algorithms or key sizes not approved by FIPS 140-2. In addition, ICSF initialization will test that it is running on an IBM System z model type, and a version and release of z/OS, that supports FIPS. If so, then ICSF will perform a series of cryptographic known answer tests as required by the FIPS 140-2 standard. If any of these initialization tests should fail, the action the ICSF initialization process takes will depend on the *fail-option* specified.

COMPAT

Indicates that the z/OS PKCS #11 services will operate in *FIPS compatibility mode*. This mode is intended for installations where only certain z/OS PKCS #11 applications must comply with the FIPS 140-2 standard, while other applications do not. In this mode, the PKCS #11 services can be further configured so that the applications that do not need to comply with the FIPS 140-2 standard are not restricted from using any of the PKCS #11 algorithms, while applications that must comply with the standard are restricted from using the non-approved algorithms. By default, the COMPAT option will have the same effect as the YES option, and all applications using the PKCS #11 services will be forced to use those services in a FIPS-compliant manner. However, additional specifications can be made:

- at the PKCS #11 token and application level, by creating FIPSEXEMPT.*token-name* resource profiles in the CRYPTOZ class. A FIPSEXEMPT.*token-name* resource exists for each token. User IDs with READ access authority to a FIPSEXEMPT.*token-name* are exempt from FIPS compliance, while user IDs with access authority NONE can only use the PKCS #11 services in a FIPS-compliant manner.
- within applications themselves for individual keys. When an application creates a key, the application can specify that the key must be used in a FIPS 140-2 compliant fashion. The application can specify this by setting the Boolean key attribute CKA_IBM_FIPS140 to TRUE.

When the COMPAT option is specified, ICSF initialization will test that it is running on an IBM System z model type, and a version

and release of z/OS, that supports FIPS. If so, then ICSF will perform a series of cryptographic known answer tests as required by the FIPS 140-2 standard. If any of these initialization tests should fail, the action the ICSF initialization process takes will depend on the *fail-option* specified.

NO Indicates that no z/OS PKCS #11 applications at the installation need to comply with the FIPS 140-2 standard, and ICSF will bypass the extra processing that is required to ensure FIPS compliance. FIPSEXEMPT.*token-name* profiles, if they exist, will not be examined. Requests to generate or use a key with CKA_IBM_FIPS140=TRUE will result in a failure return code.

The *fail-option* is either YES or NO, and indicates what action the ICSF initialization process should take if any of the initialization tests (performed when **FIPSMODE** is **YES** or **COMPAT**) should fail.

YES indicates that ICSF should end abnormally if any of the tests fail.

NO Specifies that ICSF initialization process should continue even if one or more of the tests fail. However, z/OS PKCS #11 support will be limited or nonexistent depending on the test that failed.

- If ICSF is running on an IBM system z model type or with a version of z/OS that does not support FIPS, most FIPS processing is bypassed. PKCS #11 callable services will be available, but ICSF will not adhere to FIPS 140 restrictions. Requests to generate or use a key with CKA_IBM_FIPS140=TRUE will result in a failure return code.
- If a known answer test failed, all ICSF PKCS #11 callable services will be unavailable.

KEYAUTH(YES or NO)

Indicates whether or not ICSF authenticates a key entry after ICSF retrieves one from the in-storage CKDS. If you specify KEYAUTH(YES), ICSF authenticates the key. ICSF generates a message authentication code (MAC) for each key entry in the CKDS when you create or update the entry. If you specify KEYAUTH(YES), ICSF performs a MAC verification to ensure that the entry has not changed. If you specify KEYAUTH(NO), ICSF does not perform this authentication and gains a small performance enhancement. If you do not specify the KEYAUTH option, the default value is KEYAUTH(NO).

Note: If the active CKDS does not use record level authentication, the KEYAUTH option will be ignored. It will be displayed as DISABLED on the Installation Options Display panel.

MAXLEN(n)

Defines the maximum length of characters in a text string, including any necessary padding, for some callable service requests. For example, this option defines the maximum length of the text the encipher service encrypts for each call. Specify *n* as a decimal value from 1024 through 2147483647. If you do not specify the MAXLEN option, the default value is MAXLEN(65535).

The MAXLEN parameter may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647). If a value greater than this is specified, an error will result and ICSF will not start.

Note: MAXLEN is no longer displayed on the Installation Option Display panel.

PKDSCACHE
This keyword is no longer supported but is tolerated.

PKDSN(data-set-name)
Specifies the PKDS name the system uses to start ICSF. Whenever you restart ICSF, the PKDS named in the PKDSN option becomes the active PKDS. (At first-time startup, you should specify the name of an empty VSAM data set you created to use as the PKDS.)

If you do not specify this keyword, ICSF does not become active. There is no default for this option, so you must specify a value.

REASONCODES(ICSF or TSS)
Specifies which set of reason codes are to be returned from callable services. If you do not specify the REASONCODES option, the default of REASONCODES(ICSF) is used. If you specify REASONCODES(TSS), TSS reason codes will be returned. If there is a 1-to-1 mapping, the codes will be converted.

If you specified REASONCODES(ICSF) and your service was processed on a PCICC, PCIXCC, CEX2C, or CEX3C, a TSS reason code may be returned if there is no 1–1 corresponding ICSF reason code.

SERVICE(service-number,load-module-name,FAIL(fail-option))
Indicates information about an installation-defined service.

ICSF allows an installation to define its own service similar to an ICSF callable service. The *service-number* specifies a number that identifies the service to ICSF. The valid service numbers are 1 through 32767, inclusive. This set of service numbers is valid for both installation-defined services and UDX services. The service number of an installation-defined service must not be the same as the service number of a UDX service. The *load-module-name* is the name of the module that contains the service. During ICSF startup, ICSF loads this module and binds it to the *service-number* you specified.

The *fail-option* is YES or NO, indicating the action ICSF should take if loading the service ends abnormally.

YES Specifies that ICSF ends abnormally if your service cannot be loaded.

NO Specifies that ICSF continues to start if your service cannot be loaded.

If the service itself ends abnormally, ICSF does not end, but takes a system dump instead. The ICSF service functional recovery routine (FRR) protects the service.

SSM(YES or NO)
Specifies whether or not an installation can enable special secure mode (SSM) while running ICSF. SSM lowers the security of your system to let you enter clear keys and generate clear PINs. You must enable SSM for KGUP to permit generation or entry of clear keys and to enable the secure key import or clear pin generate callable services.

YES Indicates that you can enable the SSM.

NO Indicates that you cannot enable the SSM.

If you do not specify the SSM option, the default value is SSM(NO).

Note: **CCF Systems only**: When you initialize ICSF for the first time, SSM must be active. Therefore, at first-time startup, you must specify SSM(YES).

If you are running with the IBM @server zSeries 900, IBM @server zSeries 800, S/390 Enterprise Servers and S/390 Multiprise servers, you must perform these tasks to make SSM active:

- Specify SSM(YES) in the installation options data set
- Enable SSM in the cryptographic hardware
- When running under a logical partition (LPAR), enable SSM for each partition.

SSM must be enabled or disabled in ALL places or errors may be logged and functions will not work as expected.

Note: The setting of the Environment Control Mask (ECM) enables SSM. Without TKE, the supplied ECM enables SSM. With TKE, you can set the ECM directly; the supplied ECM enables SSM, but you have the ability to disable it. For details, refer to *Support Element Operations Guide* and *z/OS Cryptographic Services ICSF TKE PCIX Workstation User's Guide*.

SYSPLEXCKDS(YES or NO,FAIL(*fail-option***))**

SYSPLEXCKDS(YES,FAIL(*fail-option***))**
ICSF will join the ICSF sysplex group SYSICSF and this system will participate in sysplex-wide consistency for CKDS data.

SYSPLEXCKDS(YES,FAIL(YES))
Indicates ICSF initialization will end abnormally if the ICSF cross-system services environment cannot be established during ICSF initialization due to a failure creating the CKDS latch set or a failure to join the ICSF sysplex group.

SYSPLEXCKDS(YES,FAIL(NO))
Indicates ICSF initialization processing will continue even if the request to create a CKDS latch set fails or the request to join the ICSF sysplex group fails. The system will not be notified of updates to the CKDS by other members of the ICSF sysplex group. A value of either FAIL(YES) or FAIL(NO) will be ignored with SYSPLEXCKDS(NO,...).

SYSPLEXCKDS(NO,FAIL(*fail-option***))**
CKDS update processing proceeds as it does today (i.e. no Cross-System Services task will be initialized, nor will any XCF signalling be performed when an update to a CKDS record occurs).

If you do not specify the SYSPLEXCKDS option, the default value is SYSPEXCKDS(NO,FAIL(NO)).

SYSPLEXPKDS(YES or NO,FAIL(*fail-option***))**
ICSF will join the ICSF sysplex group SYSICSF and this system will participate in sysplex-wide consistency for PKDS data.

SYSPLEXPKDS(YES,FAIL(*fail-option***))**
ICSF will join the ICSF sysplex group SYSICSFP and this system will participate in sysplex-wide consistency for PKDS data.

SYSPLEXPKDS(YES,FAIL(YES))
Indicates ICSF initialization will fail to join the sysplex if the ICSF cross-system services environment cannot be established during ICSF initialization due to a failure creating the PKDS latch set or a failure to join the ICSF sysplex group.

SYSPLEXPKDS(YES,FAIL(NO))
Indicates ICSF initialization processing will continue even if the request to create a PKDS latch set fails or the request to join the ICSF sysplex group fails. The system will not be notified of updates to the PKDS by other members of the ICSF sysplex group. A value of either FAIL(YES) or FAIL(NO) will be ignored with SYSPLEXPKDS(NO,...).

SYSPLEXPKDS(NO,FAIL(*fail-option***))**
PKDS update processing proceeds without trying to join the ICSF sysplex group.

If you do not specify the **SYSPLEXPKDS** option, the default value is **SYSPEXPKDS(NO,FAIL(NO))**.

SYSPLEXTKDS(YES or NO,FAIL(*fail-option***))**

ICSF will join the ICSF sysplex group SYSICSF and this system will participate in sysplex-wide consistency for TKDS data.

Note: TKDSN needs to be specified for this to work. See on page 40.

SYSPLEXTKDS(NO,FAIL(*fail-option***))**
Indicates no XCF signalling will be performed when an update to a TKDS record occurs.

SYSPLEXTKDS(YES,FAIL(*fail-option***))**
Indicates the system will be notified of updates made to the TKDS by other members of the sysplex who have also specified SYSPLEXTKDS(YES,FAIL(*fail-option*)).

SYSPLEXTKDS(YES,FAIL(YES))
Indicates ICSF will terminate abnormally if there is a failure creating the TKDS latch set.

SYSPLEXTKDS(YES,FAIL(NO))
Indicates ICSF initialization processing will continue even if the request to create a TKDS latch set fails with an environment failure. This system will not be notified of updates to the TKDS by other members of the ICSF sysplex group.

If you do not specify the SYSPLEXTKDS option, the default value is SYSPLEXTKDS(NO,FAIL(NO)) is the default.

TKDSN(data-set-name)
The name of an existing TKDS or an empty VSAM data set to be used as

the TKDS. To enable applications to create and use persistent PKCS #11 tokens and objects using the PKCS #11 services, this option must be specified.

TRACEENTRY(n)
Specifies the number, *n*, of trace buffers to allocate for ICSF tracing. Specify *n* as a decimal value from 10000 through 500000, inclusive. The default is 10000.

You should set this parameter to the maximum in case you ever need this trace material.

UDX(UDX-id,service-number,load-module-name,'comment_text',FAIL(fail-option))
ICSF allows the development of User Defined Extensions for the PCICC, PCIXCC, CEX2C, or CEX3C. The *UDX-id* is supplied to the installation when the UDX is developed. The *service-number* specifies a number that identifies the service to ICSF. The valid service numbers are 1 to 32767, inclusive. This set of service numbers is valid for both installation-defined services and UDX services. The service number of a UDX service must not be the same as the service number of an installation-defined service. The *load-module-name* is the name of the module that contains this service. During ICSF startup, ICSF loads this module and binds it to the service-number that was specified. A *comment* may be specified. The positional parameter is required. The comment consists of up to 40 EBCDIC characters, and may include imbedded blank characters. The comment text is enclosed by single quotes. If no comment text is desired, two contiguous single quotes should be specified.

The *fail-option* is YES or NO, indicating the action ICSF should take if loading the service ends abnormally. If the service itself ends abnormally, ICSF does not end, but takes a system dump instead.

YES Specifies that ICSF ends abnormally if your service cannot be loaded.

NO Specifies that ICSF continues to start if your service cannot be loaded.

The User Defined Extension (UDX) is responsible for logging in SMF the results of any authorization checks that were made.

USERPARM(value)
Specifies an 8-byte field for installation use. The Installation Option Display panel displays this value, which is stored in the Cryptographic Communication Vector Table (CCVT) in the *CCVT_USERPARM* field. An application program or installation exit can examine this field and use it to set system environment information. The default is eight blanks.

WAITLIST(data_set_name)
This optional parameter can be used if you have ICSF with CICS (CICS 4.1 or higher) installed. It specifies a customer modifiable data set will be used to determine names of the services to be placed into the ICSF CICS Wait List. A sample data set is provided by ICSF via member CSFWTL00 of SYS1.SAMPLIB with CCFs/PCICCs and CSFWTL01 for systems with PCIXCCs, CEX2Cs, or CEX3Cs. The sample data set contains the same entries as the default ICSF CICS Wait List (i.e., the data set contains the names of all ICSF callable services which, by default, will be driven through the CICS TRUE). The WAITLIST option should be added to the Installation Options data set under these conditions.

- Non-CICS customers will not specify a WAITLIST keyword. You must ensure, however, that if you have any existing CICS applications which invoke any of the ICSF services in the Wait List and if these applications were linked with ICSF stubs at a pre-OS/390 V2R10 level, then these applications must be re-linked with the current ICSF stubs.

  If running on a z990, z890, z9 EC or z9 BC however, you must also ensure that any existing CICS applications which invoke any of these services are re-linked to ensure that the correct version of the stub is used: CSNBCKI, CSNBCKM, CSNBDEC, CSNBENC, CSNBKYTX, CSNBMGN, CSNBMVR, CSNBPEXX, CSNBRNG
- CICS customers who do not want to make use of CICS TRUE must either not enable the TRUE or must specify a WAITLIST keyword and point to an empty wait list data set (or specify WAITLIST(DUMMY)) in the Installation Options data set.
- CICS customers who wish to modify the ICSF default CICS Wait List should modify the sample Wait List data set supplied in member CSFWTL00 or CSFWTL01 of SYS1.SAMPLIB. The WAITLIST keyword in the Installation Options Data Set should be set to point to this modified data set. If you have any existing CICS applications which invoke any of the ICSF services in the Wait List and if these applications were linked with ICSF stubs at a pre-OS/390 V2R10 level, then these applications must be re-linked with the current ICSF stubs.

  If running on a z990, z890, z9 EC or z9 BC any existing CICS applications which invoke any of these services are re-linked to ensure that the correct version of the stub is used: CSNBCKI, CSNBCKM, CSNBDEC, CSNBENC, CSNBKYTX, CSNBMGN, CSNBMVR, CSNBPEXX, CSNBRNG.

CICS Attachment Facility

If you have the CICS Attachment Facility (CICS 4.1 or higher) installed and you specify your own CICS wait list data set, you need to modify the wait list data set to include the new callable services.

On a z900 or z800 machine, modify and include:

- HCR7770: CSF1DMK, CSF1DVK, CSF1GKP, CSF1GSK, CSF1PKS, CSF1PKV, CSF1SAV, CSF1SKE, CSF1TRC, CSF1TRD, CSF1UWK, CSF1WPK, CSFTBC, CSFRKX
- HCR7751, HCR7750, HCR7740: CSF1GKP, CSF1GSK, CSF1PKS, CSF1PKV, CSF1SAV, CSF1TRC, CSF1TRD, CSF1UWK, CSF1WPK, CSFTBC, CSFRKX
- HCR7731: CSFTBC, CSFRKX

Note: If no Wait List is specified on a z900 or z800, the default wait list will be used. (See sample CSFWTL00 for the contents of the default wait list for z900 or z800.)

On a z990 or z890 (or later) class machine, modify and include:

- HCR7770: CSNBSYD, CSNBSYD1, CSNBSYE, CSNBSYE1, CSFPKT, CSF1DMK, CSF1DVK, CSF1SKD, CSF1SKE, CSF1HMG, CSF1HMV, CSF1OWH, CSF1PRF, CSNBSAD, CSNBSAD1, CSNBSAE, CSNBSAE1, CSFRNGL, CSF1GKP, CSF1GSK, CSF1PKS, CSF1PKV, CSF1SAV, CSF1TRC, CSF1TRD, CSF1UWK, CSF1WPK, CSFTBC, CSFRKX

- HCR7751: CSNBSAD, CSNBSAD1, CSNBSAE, CSNBSAE1, CSFRNGL, CSF1GKP, CSF1GSK, CSF1PKS, CSF1PKV, CSF1SAV, CSF1TRC, CSF1TRD, CSF1UWK, CSF1WPK, CSFTBC, CSFRKX
- HCR7750: CSFRNGL, CSF1GKP, CSF1GSK, CSF1PKS, CSF1PKV, CSF1SAV, CSF1TRC, CSF1TRD, CSF1UWK, CSF1WPK, CSFTBC, CSFRKX
- HCR7740: CSF1GKP, CSF1GSK, CSF1PKS, CSF1PKV, CSF1SAV, CSF1TRC, CSF1TRD, CSF1UWK, CSF1WPK, CSFTBC, CSFRKX
- HCR7731: CSFTBC, CSFRKX

Note: If no Wait List is specified on a z990, z890, z9 EC, z9 BC, z10 EC and z10 BC the default wait list will be used. (See sample CSFWTL01 for the contents of the default wait list for z990, z890, z9 EC, z9 BC, z10 EC and z10 BC.)

Implementing the CICS wait list

The CICS Wait List can be implemented by means of a customer modifiable data set, pointed to by the Installation Options Data Set (WAITLIST parameter). The default WAITLIST includes all services which can complete asynchronously (for example, those services which perform I/O to a cryptographic key data set and those services which are routed to a PCICC or PCIXCC). If the option is not specified, the default CICS Wait List will be utilized by ICSF when a CICS application invokes an ICSF callable service. If WAITLIST is specified, the data set specified by this parameter will be used to determine the names of the services to be placed on the CICS Wait List. A sample data set is provided by ICSF via member CSFWTL00 (for CCF systems with PCICCs) and CSFWTL01 (for systems with PCIXCCs) of SYS1.SAMPLIB. The sample data set contains the same entries as the default ICSF CICS Wait List -- for example, the data set contains the names of all ICSF callable services which, by default, will be driven through the CICS TRUE.

The WAITLIST option should be added to the Installation Options data set under these conditions.

- Non-CICS customers will not specify a WAITLIST keyword.
- CICS customers who want to use the default CICS Wait List shipped with ICSF will not specify a WAITLIST keyword. If you have any existing CICS applications which invoke any of the ICSF services in the Wait List, then these applications must be re-linked with the current ICSF stubs.
- CICS customers who do not want to make use of CICS TRUE must either not enable the TRUE or specify a WAITLIST keyword and point to an empty wait list data set or you can specify WAITLIST(DUMMY) in the Installation Options data set.
- CICS customers who wish to modify the ICSF default CICS Wait List should modify the sample Wait List data set supplied in member CSFWTL00 (for CCF systems with PCICCs) or member CSFWTL01 (for systems with PCIXCCs, CEX2Cs, or CEX3Cs) of SYS1.SAMPLIB. The WAITLIST keyword in the Installation Options Data Set should be set to point to this data set. If you have any existing CICS applications which invoke any of the ICSF services in the Wait List, then these applications must be re-linked with the current ICSF stubs.

If you already have the CICS-ICSF Attachment facility installed, there are a number of callable services which may potentially be routed to the PCICC, PCIXCC, CEX2C, or CEX3C or may perform other asynchronous processing. If you have a modified CICS Wait List, you should ensure that the wait list data set includes all such services, and any CICS applications which invoke any of these services are

re-linked with the current ICSF stubs. As a model, you can use the default CICS Wait List that is shipped with ICSF which includes all services which have an asynchronous interface to ICSF or you can use a sample Wait List data set that is also shipped with ICSF. The sample CICS Wait List data set is contained in member CSFWTL00 (for CCF systems with PCICCs) or in member CSFWTL01 (for systems with PCIXCC, CEX2C, or CEX3C) of SYS1.SAMPLIB. The sample data set contains the same entries as the default ICSF CICS Wait List. You can modify the sample data set to add and/or delete items from the Wait List. Here are some examples of why you might want to modify the sample data set.

For CCF Systems:

- If you do not have a PCI Cryptographic Coprocessor installed, you can delete all of the services identified with an "*" that are in the sample wait list.
- If you have a PCI Cryptographic Coprocessor installed, you can examine the services your applications invoke in a CICS environment and determine, based upon the routing information provided for each service in *z/OS Cryptographic Services ICSF Application Programmer's Guide*, SA22-7522, that the service will never be routed to a PCI Cryptographic Coprocessor. In this case (except for the CKDS/PKDS access services) the service can be deleted from the list.

For CCF systems with a PCICC or z990/z890 systems with a PCIXCC/CEX2C:

- If you have an application which invokes a UDX while running under CICS, then the name of the UDX generic service should be added to the CICS Wait List.

If you use a CICS Wait List data set, you need to identify the data set to ICSF through the WAITLIST(data_set_name) option in the ICSF Installation Options data set. The data set can be a member of a PARMLIB, a member of a partitioned data set, or a sequential data set. The data set should be allocated on a permanently resident volume and should adhere to:

- The format of each record in the data set must be fixed length or fixed block length.
- A physical line in the data set must be a LRECL of 80 characters long. The system ignores any characters in positions 73 to 80 of the line.
- You can delimit comments by "/*" and "*/" and include them anywhere in the text. A comment cannot span physical records.
- Only one service may be specified on a logical line.

The Cryptographic Communication Vector Table Extension (CCVE)

The CCVE is an extension of the CCVT that contains fields that can exist. The CCVE exists in ICSF extended private. It should contain any ICSF base control block fields that are not needed by other address spaces.

CCVE

ONLY these fields are part of the programming interface:

- CCVEINPP
- CCVEINPL
- CCVESECC

Table 14 on page 45 describes the contents of the Cryptographic Communication Vector Table Extension. Any bits that are not described in the table are reserved.

Table 14. Cryptographic Communication Vector Table Extension

Offset (Dec)	Number of Bytes	Field Name	Description
0	4	CCVEID	Cryptographic Communication Vector Table Extension ID. This field must contain the character string `CCVE`.
4	2	CCVEVER	Version. The version number of the CCVE. This field must contain the character string 04.
8	8		Reserved.

Table 14. Cryptographic Communication Vector Table Extension (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
16	4	CCVESTAT	Status word First status byte – CCVESTA1 **Bit** — **Meaning When Set On** **0** Special secure mode allowed. **1** Special secure mode enabled. **3** Authentication required for key retrieval. **4** The hardware has gone from active to inactive. **5** First start of ICSF during this IPL. **6** Security Server (RACF) checking required for authorized callers. **7** PCF coexistence. Second status byte – CCVESTA2 **0** Dynamic CKDS updates disallowed. **1** Refresh needed. **2** Dynamic CKDS creates disallowed. **3** Linear CKDS 80% full. **4** 80% message already sent. **5** CDMF used (rather than DES). This indicates setting of COMPENC keyword. **6** PKA callable services disallowed. **7** Authenticate the CKT when bit is one Third status byte – CCVESTA3 **1** PKDS write, create, and delete not permitted. **2** SYSPLEXCKDS(YES) was specified in Install Options Data Set. **3** SYSPLEXCKDS(YES,FAIL(YES)) was specified in Install Options Data Set. **4** SYSPLEXTKDS(YES) was specified in Install Options Data Set. **5** SYSPLEXTKDS(YES,FAIL(YES)) was specified in Install Options Data Set. **6** TKDS refresh requested. **7** TKDS empty at initialization Fourth status byte – CCVESTA4 **Bit** — **Meaning When Set On** **0** PKDS dataspace needs refresh. **1** PKDS dataspace can't be updated. **2** PKDS dataspace is 80% full. **3** 80% message already sent. **4** SYSPLEXPKDS(YES) **5** SYSPLEXPKDS(YES,FAIL(YES)) **6** CKDS MAC record authentication **7** Sysplex running in sysplex mode (not XCF-local mode)
20	4	CCVECAMQ	Pointer to MCAMQ.
24	4	CCVEEXIT	Pointer to the installation exit router (CSFEXIT).

Table 14. Cryptographic Communication Vector Table Extension (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
28	4	CCVECLIC	Software Crypto control block
32	4	CCVE_ENQ_TIMEOUT	XCF Failure detection interval in 0.01 seconds used for Sysplex ENQ timeout interval.
36	4	CCVETRCB	Pointer to the current trace buffer. **Bit** **Meaning When Set On** **0** Trace is active.
40	4	CCVECPRM	Address of CPRM.
44	4	CCVEMGST	Address of the generic service table.
48	4	CCVEENT	Address of the exit name table.
52	4	CCVETSKT	Address of task table.
56	4	CCVEMKVN	Master key version numbers. Byte 1: Current master key version number. Bytes 2 and 3: Reserved. Byte 4: Cryptographic domain index.
60	54	CCVEWLDS	Dataset name of WaitList dataset.
114	1	CCVEIBMR	IBM reserved byte.
115	1	CCVEHFL2	Hardware flags **Bit** **Meaning When Set On** **0** CCA level 3.41 detected **1** CA level 4.00 detected **2** STFLE.15 was on **3** AP-special-command facility available
116	4		Reserve
120	4	CCVE_NOPKA_MSGID	WTO message ID saved when PKA callable services are not available at startup
124	12	CCVEDCTLARR	DCTL address array.
136	4	CCVESERBCPID	SERB cell pool ID
140	4	CCVEFIXS	Address of the fixed area storage used as dynamic storage for the RISGNL routines.
144	4	CCVEFIXL	Length of the fixed area storage.
148	4	CCVECPUF	CPUF routine — used to manipulate the control register.
152	4	CCVERFMK	RFOMK routine — used to RFOMK keys on specific CPs.
156	4	CCVERMKV	MKV RISGNL routine — used by MKV to validate a CP.
160	4	CCVESTHW	STHW routine — used to obtain the current status of the hardware.
164	4	CCVEKEYM	KEYM routine — used to manipulate keys from the key entry hardware.

Table 14. Cryptographic Communication Vector Table Extension (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
168	4	CCVEDKEF	DKEF routine — used to manipulate keys for clear key entry.
172	16	CCVE_PKA_KMMK_HP	KMMK hash pattern
188	16	CCVE_PKA_SMK_HP	SMK hash pattern
204	4	CCVELFDD	ECB for look for disabled Cryptographic Coprocessor Feature task termination (LFD Done).
208	4	CCVELFDT	Pointer to TCB for CSFMLFDT.
212	4	CCVEENFS	ECB for *Issue ENF SIGNAL*.
216	4	CVESMCA	Address of SMCA
220	4	CCVE_SUBPOOL	Subpool for storage
224	4	CCVEAMKV	Pointer to the AES MKVP block
228	4	CCVEMKVB	Pointer to the current Master Key Verification Pattern (MKVP) block.
232	32	CCVEMKB1	First MKVP block.
264	32	CCVEMKB2	Second MKVP block.
296	32	CCVEMKB3	Third MKVP block.
328	4	CCVEINPP	Pointer to installation optional parameter.
332	4	CCVEINPL	Length of the installation optional parameter.
336	4	CCVETRCN	Number of trace entries.
340	4	CCVEIOPB_PKDS	Address of PKDS IO subtask data.
344	4	CCVEIOST_TKDS	Address of TKDS IO subtask TCB.
348	4	CCVEIOPB_TKDS	Address of TKDS IO subtask data.
352	4	CCVEIOPB	Address of IO subtask data.
356	4	CCVECCPD	Pointer to CAJP Data.
360	4	CCVECCPV	Pointer to private CAJP Data .
364	4	CCVEWKAR	Work area for services.
368	4	CCVEMUST	Address of UDX service table.
372	8	CCVESECC	Reserved for security exit.
380	4	CCVEENTK	ENTE for security keys exit.
384	4	CCVEENTS	ENTE for security service exit.
388	8		Reserved
396	4	CCVEDSCB	Control block for the data manager.
400	16	CCVEAMB1	AES MKVP first block.
416	16	CCVEAMB2	AES MKVP second block .
432	16	CCVEAMB3	AES MKVP third block.
448	12	CCVE_CKDS_HASH_TABLES	CKDS hash tables.
460	12	CCVE_PKDS_HASH_TABLES	PKDS hash tables.

Table 14. Cryptographic Communication Vector Table Extension (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
472	4	CCVE_KEY_ STORE_POLICY	**Bit** — **Meaning When Set On** **0** — CKDS key store policy enabled **1** — CKDS control in fail mode **2** — CKDS control in warn mode **3** — CKDS default control enabled **4** — No duplicates in CKDS **8** — PKDS key store policy enabled **9** — PKDS control in fail mode **10** — PKDS control in warn mode **11** — PKDS default control enabled **12** — No duplicates in PKDS **16** — Granular keylabel access controls enabled in fail mode **17** — Granular keylabel access controls enabled in warn mode **18** — Enhanced export restrictions enabled for AES keys **19** — Enhanced export restrictions enabled for DES keys **24** — PKA key extensions enabled. **25** — PKCS #11 Token used for trusted certificate repository (SAF keyring when this bit is 0). **26** — PKA key extensions in WARNONLY mode.
476	4		Reserved
480	4	CCVEINQKP_ECB	INQKP ECB for waking up
484	4	CCVE_KSP_PKAKE_DATA_PTR	Pointer to PKA key management extensions data.
488	4	CCVE_FIPS	FIPS policy flags. **Bit** — **Meaning When Set On** **1** — FIPS startup known answer tests failed disabling PKCS#11. **2** — FIPS140(xxx,FAIL(YES)) specified **3** — Known answer test executed on accelerator for private key operation **4** — Known answer test executed on accelerator for public key operation
492	28		Reserved

Table 14. Cryptographic Communication Vector Table Extension (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
520	2	CCVELEN	Length. The length of the CCVE. The value of this field is 752 in decimal.