

z/OS



Cryptographic Services Integrated Cryptographic Service Facility Writing PKCS #11 Applications

z/OS



Cryptographic Services Integrated Cryptographic Service Facility Writing PKCS #11 Applications

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 53.

Second Edition, February 2008

This is a major revision of SA23-2231-00.

This edition applies to Version 1 Release 9 of z/OS (5694–A01) and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this document, or you may address your comments to the following address:

International Business Machines Corporation
MHVRCFS, Mail Station P181
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+845+432-9405

FAX (Other Countries):

Your International Access Code +1+845+432-9405

IBMLink™ (United States customers only): IBMUSM10(MHVRCFS)

Internet e-mail: mhvrcfs@us.ibm.com

World Wide Web: <http://www.ibm.com/systems/z/os/zos/webqs.html>

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this document
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2007, 2008. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	v
Tables	vii
About this document	ix
Who should read this document	ix
How this document is organized	ix
How to use this document	ix
Where to find more information	x
Chapter 1. Overview of z/OS support for PKCS #11	1
Tokens	2
The token data set (TKDS)	3
Controlling access to tokens	4
Managing tokens	7
Sample scenario for setting up z/OS PKCS #11 tokens	8
Auditing PKCS #11 functions	10
Component trace for PKCS #11 functions	10
Object types	10
Session objects	10
Token objects	11
Preparing to use PKCS #11 applications	11
Tasks for the system programmer	11
Tasks for the security administrator	11
Tasks for the auditor	12
Tasks for application programmers	12
Chapter 2. The C API	13
Using the C API	13
Deleting z/OS PKCS #11 tokens	13
Environment	13
Key types and mechanisms supported	13
Objects and attributes supported	15
Library, slot, and token information	25
Functions supported	27
Standard functions supported	27
Non-standard functions supported	31
Function return codes	32
Troubleshooting PKCS #11 applications	32
Chapter 3. The testpkcs11 program	35
Running the pre-compiled version of testpkcs11	35
Steps for running the pre-compiled version of testpkcs11	35
Building testpkcs11 from source code	36
Steps for building testpkcs11 from source code	36
Chapter 4. Token management callable services	39
SMP/E installation data sets, directories, and files	41
Source code for the testpkcs11 sample program	43
Accessibility	51

Using assistive technologies	51
Keyboard navigation of the user interface	51
z/OS information	51
Notices	53
Programming interface information.	54
Trademarks	54
Index	57

Figures

1. The z/OS ICSF library.	i
2. Sample job to define the TKDS	4

Tables

1. Token access levels	5
2. Resources in the CSFSERV class for token services	6
3. Mechanism information as returned by C_GetMechanismInfo (CK_MECHANISM_INFO)	14
4. Mechanisms supported by specific cryptographic hardware	15
5. Common footnotes for object attribute tables	15
6. Data object attributes that ICSF supports	16
7. X.509 certificate object attributes that ICSF supports.	17
8. Secret key object attributes that ICSF supports.	18
9. Public key object attributes that ICSF supports	21
10. RSA public key object attributes that ICSF supports	22
11. Private key object attributes that ICSF supports	23
12. RSA private key object attributes that ICSF supports.	24
13. Standard PKCS #11 functions that ICSF supports.	27
14. Syntax of the CK_RV CSN_FindAllObjects() function	31
15. Environment variables for capturing trace data	33

About this document

This document supports z/OS® (5694-A01). This document describes the support for PKCS #11 provided by the z/OS Integrated Cryptographic Service Facility (ICSF). ICSF is a component of z/OS Cryptographic Services, which includes the following components:

- z/OS Integrated Cryptographic Service Facility (ICSF)
- z/OS Open Cryptographic Services Facility (OCSF)
- z/OS System Secure Socket Level Programming (SSL)
- z/OS Public Key Infrastructure Services (PKI Services)

ICSF is a software element of z/OS that works with the hardware cryptographic feature and the Security Server (RACF®) to provide secure, high-speed cryptographic services. ICSF provides the application programming interfaces by which applications request the cryptographic services.

PKCS #11 is an industry-accepted standard provided by RSA Laboratories of RSA Security Inc. It specifies an application programming interface (API) to devices, referred to as *tokens*, that hold cryptographic information and perform cryptographic functions. PKCS #11 provides an alternative to IBM®'s Common Cryptographic Architecture (CCA).

Who should read this document

This document is primarily intended for application programmers who want to write PKCS #11 applications for z/OS. It also contains information for security administrators, system programmers, and auditors in installations that use PKCS #11 applications.

How this document is organized

- Chapter 1, “Overview of z/OS support for PKCS #11,” on page 1 provides an overview of ICSF support for PKCS #11. It discusses tokens, the token data set (TKDS), auditing and tracing PKCS #11 functions, session objects, and tasks that must be performed before using PKCS #11 applications.
- Chapter 2, “The C API,” on page 13 discusses the PKCS #11 C API provided by ICSF, highlighting differences between the z/OS implementation and the PKCS #11 standard.
- Chapter 3, “The testpkcs11 program,” on page 35 discusses how to build and run the testpkcs11 sample.
- Chapter 4, “Token management callable services,” on page 39 provides a brief introduction to the token management callable services, which are documented in *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

How to use this document

Application programmers should read the entire book.

Security administrators should read the section “Tasks for the security administrator” on page 11 and the information that it references.

System programmers should read the section “Tasks for the system programmer” on page 11 and the information that it references.

Auditors should read the section “Tasks for the auditor” on page 12 and the information that is references.

Where to find more information

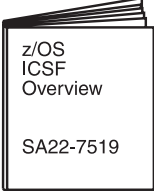

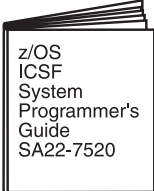
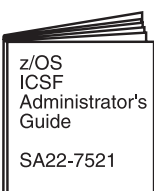

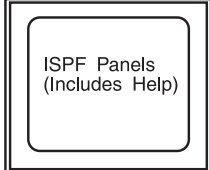
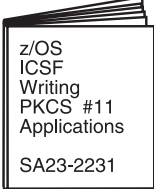
Before using this document, application programmers must be familiar with the PKCS #11 specification. The PKCS #11 standard is defined on the RSA Laboratories Web site at <http://www.rsa.com/rsalabs/>. Application programmers should also be familiar with the ICSF library and C programming.

Security administrators should be familiar with *z/OS Security Server RACF Security Administrator's Guide*.

Auditors should be familiar with *z/OS Security Server RACF Auditor's Guide*.

The callable services for PKCS #11 functions are documented in *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

The format of the token data set is documented in *z/OS Cryptographic Services ICSF System Programmer's Guide*.

Tasks		Tasks	
 <p>z/OS ICSF Overview SA22-7519</p>	Evaluating Planning	 <p>z/OS ICSF Messages SA22-7523</p>	Administrating Application Programming Diagnosis Operating
 <p>z/OS ICSF System Programmer's Guide SA22-7520</p>	Customizing Diagnosis Installing Operating	 <p>z/OS ICSF Administrator's Guide SA22-7521</p>	Administrating
 <p>z/OS ICSF Application Programmer's Guide SA22-7522</p>	Application Programming	 <p>ISPF Panels (Includes Help)</p>	Administrating
 <p>z/OS ICSF Writing PKCS #11 Applications SA23-2231</p>	Application Programming		

Optional Features

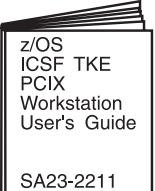

 <p>z/OS ICSF TKE PCIX Workstation User's Guide SA23-2211</p>	Available with the Trusted Key Entry Workstation (TKE Version 5)	 <p>IBM Online Library: z/OS Collection Kit SK3T-4269</p>	The ICSF Library, Trusted Key Entry Workstation User's Guide and Encryption Facility for z/OS Planning and Customizing are included on the IBM Online Library: z/OS Collection Kit SK3T-4269
----------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 1. The z/OS ICSF library

Chapter 1. Overview of z/OS support for PKCS #11

RSA Laboratories of RSA Security Inc. offers its Public Key Cryptography Standards (PKCS) to developers of computers that use public key and related technology. PKCS #11, also known as Cryptoki, is the cryptographic token interface standard. It specifies an application programming interface (API) to devices, referred to as *tokens*, that hold cryptographic information and perform cryptographic functions. The PKCS #11 API is an industry-accepted standard commonly used by cryptographic applications. ICSF supports PKCS #11, providing an alternative to IBM's Common Cryptographic Architecture (CCA) and broadening the scope of cryptographic applications that can make use of zSeries® cryptography. PKCS #11 applications developed for other platforms can be recompiled and run on z/OS.

The PKCS #11 standard is defined on the RSA Laboratories Web site at <http://www.rsa.com/rsalabs/>. This document describes how ICSF supports that standard. The support includes the following:

- A token data set (TKDS) that serves as a repository for cryptographic keys and certificates used by PKCS #11 applications
- A C application programming interface (API) that supports a subset of the V2.20 level of the PKCS #11 specification
- Token management callable services. The C API uses these callable services.

Tokens

On most single-user systems a token is a smart card or other plug-installed cryptographic device, accessed through a card reader or *slot*. The PKCS #11 specification assigns numbers to slots, known as *slot IDs*. An application identifies the token that it wants to access by specifying the appropriate slot ID. On systems that have multiple slots, it is the application's responsibility to determine which slot to access.

z/OS must support multiple users, each potentially needing a unique keystore. In this multiuser environment, the system does not give users direct access to the cryptographic cards installed as if they were personal smart cards. Instead, z/OS PKCS #11 tokens are virtual, conceptually similar to RACF (SAF) key rings. An application can have one or more z/OS PKCS #11 tokens, depending on its needs.

Typically, PKCS #11 tokens are created in a factory and initialized either before they are installed or upon their first use. In contrast, z/OS PKCS #11 tokens can be created using system software such as RACF, the *gskkyman* utility, or by applications using the C API. Each token has a unique token name, or label, that is specified by the end user or application at the time that the token is created.

Rules: A token name must follow these rules:

- Up to 32 characters in length
- Permitted characters are:
 - Alphanumeric
 - National: @ (X'5B'), # (X'7B'), or \$ (X'7C')
 - Period: . (X'4B')
- The first character must be alphabetic or national
- Lowercase letters can be used, but are folded to uppercase
- The IBM1047 code page is assumed

Because PKCS #11 tokens are typically physical hardware devices, the PKCS #11 specification provides no mechanism to delete tokens. However, because z/OS

PKCS #11 tokens are virtual, z/OS must provide a way to delete them. For information on how to delete tokens using the C API, see “Deleting z/OS PKCS #11 tokens” on page 13.

The token data set (TKDS)

The token data set (TKDS) is a VSAM data set that serves as the repository for cryptographic keys and certificates used by PKCS #11 applications. Before an installation can run PKCS #11 applications, the system programmer must create the TKDS and update the ICSF installation options data set to identify the data set name of the TKDS.

Rules: The token data set must follow these rules:

- It must be a key-sequenced VSAM data set with spanned variable length records.
- It must be allocated on a permanently resident volume.

Keys in the token data set are not encrypted. Therefore, it is important that the security administrator create a RACF profile to protect the token data set from unauthorized access.

For a sample job to define the TKDS data set, see “Sample job to define the TKDS” on page 4. For the format of the TKDS, see *z/OS Cryptographic Services ICSF System Programmer’s Guide*.

To optimize performance, ICSF creates a data space containing an in-storage copy of the TKDS.

Options for the TKDS in the ICSF installation options data set

The ICSF installation options data set contains two options related to the token data set:

- TKDSN
- SYSPLEXTKDS

The TKDSN option: The TKDSN option identifies the VSAM data set that contains the token data set. The format of this option is:

TKDSN(*data_set_name*)

data_set_name is the name of an existing token data set or an empty VSAM data set to be used as the token data set. If the TKDSN option is not specified in the ICSF installation options data set, ICSF provides no PKCS #11 services.

The SYSPLEXTKDS option: The SYSPLEXTKDS option specifies whether the token data set should have sysplex-wide data consistency. The SYSPLEXTKDS option is in effect only if the TKDSN option has also been specified. The format of this option is:

SYSPLEXTKDS(YES|NO, FAIL(YES|NO))

The SYSPLEXTKDS option can be YES or NO. The default value is NO.

- If SYSPLEXTKDS(NO, FAIL(*fail-option*)) is specified, no XCF signalling is performed when an update to a TKDS record occurs.
- If SYSPLEXTKDS(YES, FAIL(*fail-option*)) is specified, the system is notified of updates made to the TKDS by other members of the sysplex who have also specified SYSPLEXTKDS(YES, FAIL(*fail-option*)).

The value of FAIL can be YES or NO.

- If FAIL(YES) is specified, ICSF initialization terminates abnormally if there is a failure creating the TKDS latch set.
- If FAIL(NO) is specified, ICSF initialization processing continues even if the request to create a TKDS latch set fails with an environment failure. The system is not notified of updates to the TKDS by other members of the ICSF sysplex group.

The default is SYSPLEXTKDS(NO,FAIL(NO)).

Serialization of resources for the TKDS

To serialize updates to the TKDS and in-storage copies of the TKDS, ICSF uses sysplex-wide ENQs on the resource SYSZTKT and the resource SYSZTKKDS.TKDS dsn , where dsn is the data set name of the active TKDS. In addition, a latch set is defined for serializing reads to TKDS records.

Sample job to define the TKDS

A sample job illustrating the definition of the TKDS data set is shipped in SYS1.SAMPLIB, member CSFTKDS.

```
//CSFTKDS JOB <JOB CARD PARAMETERS>
//*****
//* Licensed Materials - Property of IBM *
//* 5694-A01 *
//* (C) Copyright IBM Corp. 2007 *
//* *
//* THIS JCL DEFINES A VSAM TKDS TO USE FOR PKCS #11 TOKENS *
//* AND OBJECTS *
//* *
//* CAUTION: This is neither a JCL procedure nor a complete JOB. *
//* Before using this JOB step, you will have to make the following *
//* modifications: *
//* *
//* 1) Add the job parameters to meet your system requirements. *
//* 2) Be sure to change CSF to the appropriate HLQ if you choose *
//* not to use the default. *
//* 3) The TKDS needs to be on a permanently resident volume. *
//* *
//*****
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
        DEFINE CLUSTER(NAME(CSF.CSFTKDS) -
            VOLUMES(XXXXXX) -
            RECORDS(100 50) -
            RECORDSIZE(2200 32756) -
            KEYS(72 0) -
            FREESPACE(0 0) -
            SPANNED -
            SHAREOPTIONS(2 3) ) -
        DATA (NAME(CSF.CSFTKDS.DATA) -
            BUFFERSPACE(100000) -
            ERASE -
            WRITECHECK) -
        INDEX (NAME(CSF.CSFTKDS.INDEX))
/*
```

Figure 2. Sample job to define the TKDS

Controlling access to tokens

The PKCS #11 standard was designed for systems that grant access to token information based on a PIN. The standard defines two types of users, the standard

user (*User*) and the security officer (*SO*), each having its own personal identification number (PIN). The SO can initialize a token (zero the contents) and set the User's PIN. The SO can also access the public objects on the token, but not the private ones. The User has access to the private objects on a token and has the power to change his or her own PIN. The User cannot reinitialize a token. The PIN that a user enters determines which role that user takes. A user can fill both roles by having knowledge of both PINs.

z/OS does not use PINs. Instead, profiles in the SAF CRYPTOZ class control access to tokens. For each token, there are two resources in the CRYPTOZ class:

- The resource *USER.token-name* controls the access of the User role to the token.
- The resource *SO.token-name* controls the access of the SO role to the token.

A user's access level to each of these resources (read, update, or control) determines the user's access level to the token.

There are six possible token access levels. Three are defined by the PKCS #11 standard, and three are unique to z/OS. The PKCS #11 token access levels are:

- User R/O: Allows the user to read the token including its private objects, but the user cannot create new token or session objects or alter existing ones.
- User R/W: Allows the user read/write access to the token object including its private objects.
- SO R/W: Allows the user to act as the security officer for the token and to read, create, and alter public objects on the token.

The token access levels unique to z/OS are:

- Weak SO: A security officer that can modify the CA certificates contained in a token but not initialize the token. (For example, a system administrator who determines the trust policy for all applications on the system.)
- Strong SO: A security officer that can add, generate or remove private objects in a token. (For example, a server administrator.)
- Weak User: A User that cannot change the trusted CAs contained in a token. (For example, to prevent an end-user from changing the trust policy of his or her token.)

Table 1 shows how a user's access level to a token is derived from the user's access level to a resource in the SAF CRYPTOZ class.

Table 1. Token access levels

CRYPTOZ resource	SAF access level		
	READ	UPDATE	CONTROL
<i>SO.token-label</i>	Weak SO Can read, create, delete, modify, and use public objects	SO R/W Same ability as Weak SO plus can create and delete tokens	Strong SO Same ability as SO R/W plus can read but not use (see Note1 on page 6) private objects; create, delete, and modify private objects

Table 1. Token access levels (continued)

CRYPTOZ resource	SAF access level		
	READ	UPDATE	CONTROL
USER.token-label	User R/O Can read and use (see Note 1) public and private objects	Weak User Same ability as User R/O plus can create, delete, and modify private and public objects. Cannot add, delete, or modify certificate authority objects	User R/W Same ability as Weak User plus can add, delete, and modify certificate authority objects

Notes:

1. "Use" is defined as any of the following:
 - Performing any cryptographic operation involving the key object; for example C_Encrypt
 - Searching for key objects using sensitive search attributes
 - Retrieving sensitive key object attributes.

The sensitive attribute for a secret key is CKA_VALUE. The sensitive attributes for a private key object are CKA_PRIVATE_EXPONENT, CKA_PRIME_1, CKA_PRIME_2, CKA_EXPONENT_1, CKA_EXPONENT_2, and CKA_COEFFICIENT.

2. The CRYPTOZ resources can be defined as "RACF-DELEGATED" if required. For information about delegated resources, see the section on delegated resources in *z/OS Security Server RACF Security Administrator's Guide*.
3. If the CSFSERV class is active, ICSF performs access control checks on the underlying callable services. The user must have READ access to the appropriate CSFSERV class resource. Table 2 lists the resources in the CSFSERV class for token services.

Table 2. Resources in the CSFSERV class for token services

Name of resource	Service	Called by
CSF1TRC	Token or object creation	C_InitToken, C_CreateObject, C_CopyObject
CSF1TRD	Token or object deletion	C_InitToken, C_DestroyObject
CSF1TRL	Token or object find	C_Initialize, C_FindObjects, C_FindALLObjects
CSF1SAV	Set object attributes	C_SetAttributeValue
CSF1GAV	Get object attributes	C_GetAttributeValue
CSF1GSK	Generate secret key	C_GenerateKey
CSF1GKP	Generate key pair	C_GenerateKeyPair
CSF1PKS	Private key sign	C_Decrypt, C_DecryptUpdate, C_DecryptFinal, C_Sign, C_SignFinal

Table 2. Resources in the CSFSERV class for token services (continued)

Name of resource	Service	Called by
CSF1PKV	Public key verify	C_Encrypt, C_EncryptUpdate, C_EncryptFinal, C_Verify, C_VerifyFinal
CSF1SKD	Secret key decrypt	C_Decrypt, C_DecryptUpdate, C_DecryptFinal
CSF1SKE	Secret key encrypt	C_Encrypt, C_EncryptUpdate, C_EncryptFinal
CSFOWH	One-way hash	C_Digest, C_DigestUpdate, C_DigestFinal
CSF1WPK	Wrap key	C_Wrapkey
CSF1UWK	Unwrap key	C_Unwrapkey
CSFIQF	PKCS #11 initialization	C_Initialize

Guidelines:

- If your organization controls access to ICSF callable services via the CSFSERV class, define the resources listed in Table 2 on page 6 and grant access accordingly.
Tip: Define generic profiles. For example, a profile named CSF* covers all the ICSF services. A profile named CSF1* covers the PKCS #11 subset of the ICSF services, with the exception of C_Digest, C_DigestUpdate, and C_DigestFinal, which are covered by the resource CSFOWH.
- The CRYPTOZ class supports generic profiles. Take advantage of this by creating a token naming convention for your organization and enforce it with generic profiles. For example, require users and applications to prefix their token names with their user IDs, as with data set names. (See “Sample scenario for setting up z/OS PKCS #11 tokens” on page 8.)
- For server applications, grant security officers (server administrators) Strong SO access and their end-users (server daemon user IDs) Weak User or User R/W access.
- For applications for which you do not wish to separate the security officer and end-user roles, grant the appropriate user IDs access to both the SO and USER profiles.

Managing tokens

z/OS provides several facilities to manage tokens:

- A C language application programming interface (API) that implements a subset of the PKCS #11 specification. For a description of this API, see Chapter 2, “The C API,” on page 13.
- Token management callable services. The C API uses these callable services. For information about these callable services, see Chapter 4, “Token management callable services,” on page 39.
- ISPF panels. The ICSF ISPF panels provide the capability to see a formatted view of TKDS objects, and make limited updates to them.
- The RACF RACDCERT command supports the certificate, public key, and private key objects, and provides the following subfunctions to manage these objects:

- ADDTOKEN - creates a new empty token
- DELTOKEN - deletes an existing token and everything in it
- LISTTOKEN - displays information on the certificate objects in a token and whether associated public and private key objects exist
- BIND - connects a RACF certificate, its public key, and potentially its private key to an existing token
- UNBIND - removes a certificate and its keys from a token
- IMPORT - defines a token certificate to RACF

For information about the RACDCERT command, see *z/OS Security Server RACF Command Language Reference* and *z/OS Security Server RACF Security Administrator's Guide*.

- The SAF CRYPTOZ class controls access to tokens. For information about this class, see “Controlling access to tokens” on page 4.
- The RACF R_Datalib callable service (IRRSDL00) allows applications to read tokens by providing a user ID of *TOKEN* to indicate that the key ring name is really a token name. For information about R_Datalib, see *z/OS Security Server RACF Callable Services*.

Note: IRRSDL00 was originally created to allow applications to read RACF (SAF) key rings, but has been enhanced to read PKCS #11 tokens as well. Thus applications written to read key rings can also read tokens without being modified.

Sample scenario for setting up z/OS PKCS #11 tokens

The following examples show how to control access to z/OS PKCS #11 tokens. In this scenario, a company wants to use z/OS PKCS #11 tokens as the key stores for its FTP and Web servers. The company has established a naming convention for their tokens requiring that all tokens have the owning user ID as the high-level qualifier. The owning user IDs for the FTP and Web server tokens are the daemons FTPSRV and WEBSRV, respectively. User ABIGAIL is the administrator for the servers.

The security administrator, who has the RACF SPECIAL attribute, creates the protection profiles for the tokens. His goal is to give user ABIGAIL the Security Officer role for these profiles, and to give the daemon user IDs the User role. To do this he issues RACF TSO commands. First, he activates the CRYPTOZ class with generics and RACLISTs it:

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

Next, the security administrator creates profiles for the security officer's access to the FTP and Web Server tokens:

```
RDEFINE CRYPTOZ SO.FTPSRV.* UACC(NONE)
RDEFINE CRYPTOZ SO.WEBSRV.* UACC(NONE)
```

Then he creates profiles for the standard user's access to the FTP and Web Server tokens:

```
RDEFINE CRYPTOZ USER.FTPSRV.* UACC(NONE)
RDEFINE CRYPTOZ USER.WEBSRV.* UACC(NONE)
```

The security administrator now gives user ABIGAIL Strong SO power for the tokens by giving her CONTROL access to the profiles that protect the tokens. The Strong SO power does not allow ABIGAIL to use the private objects in the tokens:

```
PERMIT SO.FTPSRV.* CLASS(CRYPTOZ) ID(ABIGAIL) ACC(CONTROL)
PERMIT SO.WEBSRV.* CLASS(CRYPTOZ) ID(ABIGAIL) ACC(CONTROL)
```

Next the security administrator gives the users FTPSRV and WEBSRV Weak User power for their respective tokens. This power allows them to use the private objects within the tokens, but not change the set of trusted CA certificates.

```
PERMIT USER.FTPSRV.* CLASS(CRYPTOZ) ID(FTPSRV) ACC(UPDATE)
PERMIT USER.WEBSRV.* CLASS(CRYPTOZ) ID(WEBSRV) ACC(UPDATE)
```

Finally, the security administrator refreshes the in-storage profiles for the CRYPTOZ class, so that the changes he just made take effect:

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

Now the set up is complete: ABIGAIL has Strong SO power over the tokens for the FTP server and the Web server, and can create the required tokens. FTPSERV and WEBSRV have User power over their respective tokens, and can use them as key stores after ABIGAIL has created them.

The task now is to create and populate the tokens for the servers with RACF certificates. The following certificates exist:

1. A root CA certificate installed under CERTAUTH with label 'Local Root CA for Servers'
2. An end-entity certificate and private key installed under user FTPSERV with label 'FTP Key'. This certificate was signed by the first certificate.
3. An end-entity certificate and private key installed under user WEBSRV with label 'Web Key'. This certificate was also signed by the first certificate.

User ABIGAIL issues the following TSO commands to create the tokens, using the company's naming conventions:

```
RACDCERT ADDTOKEN(ftpshr.ftp.server.pkcs11.token)
RACDCERT ADDTOKEN(websrv.web.server.pkcs11.token)
```

Next, issue the commands that bind the root CA certificate to the two tokens:

```
RACDCERT BIND(CERTAUTH LABEL('Local Root CA for Servers')
TOKEN(ftpshr.ftp.server.pkcs11.token)
RACDCERT BIND(CERTAUTH LABEL('Local Root CA for Servers')
TOKEN(websrv.web.server.pkcs11.token)
```

Now, bind the end-entity certificates to their respective tokens. Each should be the default in the token.

```
RACDCERT BIND(ID(FTPSRV) LABEL("FTP key")
TOKEN(ftpshr.ftp.server.pkcs11.token) DEFAULT)
RACDCERT BIND(ID(WEBSRV) LABEL("Web key")
TOKEN(websrv.web.server.pkcs11.token) DEFAULT)
```

The final step is for the user (ABIGAIL) to configure both servers to use their respective tokens: add directives to the servers' configuration files.

For the web server (IBM HTTP Server), the keyfile directive in the httpd.conf file is set as follows:

```
keyfile *TOKEN*/WEBSRV.WEB.SERVER.PKCS11.TOKEN SAF
```

The SAF keyword indicates to SSL that this is a key ring and is controlled by SAF; it is not a KDB file. The TOKEN keyword indicates that the key ring is a token. The FTP server configuration file also requires a token-qualified key ring name:

```
keyfile *TOKEN*/FTPSRV.FTP.SERVER.PKCS11.TOKEN
```

Auditing PKCS #11 functions

PKCS #11 functions are audited in the following ways:

- The SMF type 82 subtype 1 record that is written during ICSF initialization contains the data set name of the token data set (TKDS).
- The SMF type 82 subtype 21 record that is written when a member joins or leaves a sysplex group contains the cryptographic keys data set (CKDS) data set name if the member joined or left the ICSF CKDS sysplex group, or the TKDS data set name if the member joined or left the ICSF TKDS sysplex group.
- ICSF writes SMF type 82 subtype 23 records whenever a TKDS record for a token or token object is created, modified, or deleted. ICSF does not write SMF records for changes to session objects.

For descriptions of the SMF records that ICSF writes, see *z/OS MVS System Management Facilities (SMF)*.

Component trace for PKCS #11 functions

The following ICSF component trace entries trace events related to the token data set (TKDS):

- Type 16 (XCFTMSGGS) traces the broadcast of an XCF message related to TKDS I/O.
- Type 17 (XCFTMSGGR) traces the receipt of an XCF message related to TKDS I/O.
- Type 18 (XCFTENQ) traces the return of control to the TKDS I/O subtask following the request for an exclusive ENQ on the SYSZTKDS.TKDS`dsn` resource.

These trace entry types are always traced.

When viewed via IPCS, these entries show the ASCB address, the TCB address, the ASID, the general purpose registers, the GPR length, and the CSS address. For more information about IPCS, see *z/OS MVS IPCS User's Guide*.

Object types

ICSF supports PKCS #11 session objects and token objects. There are five classes of objects that can be associated with these object types:

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

Session objects

A session object exists for the life of a PKCS #11 session. ICSF creates session object data spaces to hold session objects; they are not maintained on DASD. ICSF associates a session object data space with the application that requested the creation of a session object. There is only one session object data space for an application, even if the application spawns multiple PKCS #11 sessions. The same session objects are available to all PKCS #11 sessions within an application.

ICSF creates a session object data space the first time a session object is created, if there is currently no session object data space associated with the application.

The session object data space exists as long as the PKCS #11 application's address space and job step TCB exist. ICSF deletes the data space if either the address space or job step TCB terminates. If ICSF terminates, all session object data spaces are destroyed.

Token objects

Token objects are stored in the token data set, with one record per object. They are visible to all applications that have sufficient permission to the token. They are persistent: they remain associated with the token even after a session is closed.

Preparing to use PKCS #11 applications

Before an installation can use PKCS #11 applications, some preparation is required on the part of the system programmer, security administrator, auditor, and application programmers. This section describes the preparation required.

Tasks for the system programmer

Before any PKCS #11 processing can be done, the system programmer must allocate a token data set (TKDS) for use by PKCS #11 functions, and specify the data set name of the TKDS in the TKDSN option of the ICSF installation options data set.

The system programmer must decide whether or not sysplex-wide consistency of the TKDS is required, and must specify the SYSPLEXTKDS option in the ICSF installation options data set to define the processing of TKDS updates in a sysplex environment.

For information on the TKDSN and SYSPLEXTKDS options of the ICSF installation options data set, see “Options for the TKDS in the ICSF installation options data set” on page 3. For a sample job to create the token data set, see “Sample job to define the TKDS” on page 4.

The system programmer should run the testpkcs11 utility program to test the PKCS #11 configuration. For information about running the testpkcs11 program, see “Running the pre-compiled version of testpkcs11” on page 35.

Tasks for the security administrator

The security administrator should create a RACF profile to protect the data set that contains the token data set. It is important to protect this data set, because keys in the token data set are not encrypted.

The security administrator needs to grant the appropriate access authority to users for accessing tokens and objects, by defining profiles in the CRYPTOZ class. For more information, see “Controlling access to tokens” on page 4.

The security administrator controls access to the token management callable services by defining profiles in the CSFSERV class. For information about defining profiles in the CSFSERV class, see *z/OS Cryptographic Services ICSF Administrator's Guide*. For a list of the resource names for token services, see Table 2 on page 6.

Tasks for the auditor

Auditors should become familiar with the data in SMF records that is related to PKCS #11 functions. For more information, see “Auditing PKCS #11 functions” on page 10.

Tasks for application programmers

Application programmers can write applications using the PKCS #11 API provided by ICSF. They should become familiar with the PKCS #11 specification, and with the information in this book. The PKCS #11 standard is defined on the RSA Laboratories Web site at <http://www.rsa.com/rsalabs/>.

Application programmers can use the sample program, `testpkcs11`, to learn about building and running PKCS #11 applications, and to troubleshoot problems. For information about the sample program, see Chapter 3, “The `testpkcs11` program,” on page 35.

Chapter 2. The C API

ICSF provides a PKCS #11 C language application program interface (API). This chapter highlights the differences between the z/OS API and the PKCS #11 V2.20 specification. To use this API, you must be familiar with both the PKCS #11 specification and the information in this chapter.

All manifest constants specified in this chapter can be found in the `csnpdefs.h` include file and (with the exceptions noted) in the PKCS #11 specification.

Using the C API

To create or use a z/OS PKCS #11 token, an application needs to do the following:

1. Implicitly or explicitly load the PKCS #11 API DLL (CSNPCAPI for applications running in 31-bit addressing mode not using XPLINK, CSNPCA3X for applications running in 31-bit addressing mode using XPLINK, CSNPCA64 for 64-bit addressing mode).
2. Locate the functions within that DLL, using the `C_GetFunctionList` function.
3. Call `C_Initialize`, which enables the application to call other functions in the API.
4. Determine the slots present, using the `C_GetSlotList` function. This function returns a slot number for each existing token to which the application has access.
5. To use an existing token, the application iterates through the slots using `C_GetTokenInfo` to find the token wanted.

To create a new token, the application uses the `C_WaitForSlotEvent` function to add a new slot containing an uninitialized token. The application then uses the `C_InitToken` function to initialize the new token and save it in the TKDS.

Deleting z/OS PKCS #11 tokens

Because PKCS #11 tokens are typically physical hardware devices, the PKCS #11 specification provides no mechanism to delete tokens. However, for z/OS PKCS #11 tokens, which are virtual, there must be a capability to delete tokens. An application does this by calling the `C_InitToken` function with a special label value `$$DELETE-TOKEN$$` (assuming code page IBM1047), left justified and padded on the right to 32 characters.

Tip: Use the constant `DEL_TOK` defined in `csnpdefs.h`.

Environment

Restriction: The calling program must be running as a Language-Environment-enabled (LE-enabled) application in TCB mode only. SRB mode is not supported.

Key types and mechanisms supported

ICSF supports the following PKCS #11 key types (`CK_KEY_TYPE`), depending on the limitations of the cryptographic hardware configuration. For the mechanisms supported by different hardware configurations, see Table 4 on page 15.

- `CKK_RSA` - key lengths 512 to 2048 bits
- `CKK_DES`
- `CKK_DES2`
- `CKK_DES3`

- CKK_AES - key lengths 128, 192, and 256 bits

Table 3. 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	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	Bits	CKF_HW CKF_ENCRYPT CKF_DECRYPT CKF_SIGN CKF_VERIFY CKF_SIGN_RECOVER CKF_VERIFY_RECOVER
CKM_MD5_RSA_PKCS	Bits	CKF_HW CKF_SIGN CKF_VERIFY
CKM_SHA1_RSA_PKCS	Bits	CKF_HW CKF_SIGN CKF_VERIFY
CKM_SHA256_RSA_PKCS	Bits	CKF_HW CKF_SIGN CKF_VERIFY
CKM_DES_ECB	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	not applicable	CKF_HW CKF_ENCRYPT CKF_DECRYPT
CKM_DES3_CBC	not applicable	CKF_HW CKF_ENCRYPT CKF_DECRYPT
CKM_DES3_CBC_PAD	not applicable	CKF_HW CKF_ENCRYPT CKF_DECRYPT CKF_WRAP CKF_UNWRAP
CKM_SHA_1	not applicable	CKF_HW CKF_DIGEST
CKM_SHA256	not applicable	CKF_HW CKF_DIGEST
CKM_RIPEMD160	not applicable	CKF_HW CKF_DIGEST
CKM_MD5	not applicable	CKF_HW CKF_DIGEST
CKM_AES_KEY_GEN	Bytes	CKF_HW CKF_GENERATE
CKM_AES_ECB	Bytes	CKF_HW CKF_ENCRYPT CKF_DECRYPT
CKM_AES_CBC	Bytes	CKF_HW CKF_ENCRYPT CKF_DECRYPT
CKM_AES_CBC_PAD	Bytes	CKF_HW CKF_ENCRYPT CKF_DECRYPT CKF_WRAP CKF_UNWRAP

Table 4. 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_RIPEMD160 CKM_MD5 CKM_AES_KEY_GEN CKM_AES_CBC CKM_AES_CBC_PAD CKM_AES_ECB	This is the base set. RSA private key operations limited to 1024 bits in length (maximum) and no key pair generation capability. The DES3 and AES mechanisms are not present on systems that are export-controlled.
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 or 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). The DES3 and AES mechanisms are not present on systems that are export-controlled.
z9™ 109 - CEX2C	Base set plus: CKM_RSA_PKCS_KEY_PAIR_GEN CKM_DES_ECB CKM_DES3_ECB CKM_SHA256_RSA_PKCS CKM_SHA256	RSA private key operations limited to 2048 bits in length (maximum). The DES3 and AES mechanisms are not present on systems that are export-controlled.

Objects and attributes supported

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

- CKO_DATA
- CKO_CERTIFICATE - CKC_X_509 only
- CKO_PUBLIC_KEY - CKK_RSA only
- CKO_PRIVATE_KEY - CKK_RSA only
- CKO_SECRET_KEY - CKK_DES, CKK_DES2, CKK_DES3, and CKK_AES only

The footnotes described in Table 5 are taken from the PKCS #11 specification and apply to the attribute tables that follow.

Table 5. Common footnotes for object attribute tables

Footnote number	Footnote meaning
1	Must be specified when object is created with C_CreateObject .
2	Must <i>not</i> be specified when object is created with C_CreateObject .

Table 5. Common footnotes for object attribute tables (continued)

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

Table 6. Data object attributes that ICSF supports. For the meanings of the footnotes, see Table 5 on page 15.

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

Table 6. Data object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 5 on page 15.

Attribute	Data type	Notes
CKA_OBJECT_ID	Byte array	DER-encoded OID. Default is empty. An application can set or change the value at any time.

Table 7. X.509 certificate object attributes that ICSF supports. For the meanings of the footnotes, see Table 5 on page 15.

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

Table 7. X.509 certificate object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 5 on page 15.

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

Table 8. Secret key object attributes that ICSF supports. For the meanings of the footnotes, see Table 5 on page 15.

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

Table 8. Secret key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 5 on page 15.

Attribute	Data type	Notes
CKA_LABEL	Printable EBCDIC string	Application-specific nickname. Limited to 32 characters. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
CKA_ID	Byte array	Default is empty. An application can set or change the value at any time.
CKA_KEY_TYPE ^{1, 3, 5}	CK_KEY_TYPE	Type of key: CKK_DES, CKK_DES2, CKK_DES3, or CKK_AES. An application can specify the value when the object is created (or generated) only.
CKA_START_DATE ⁸	CK_DATE	Start date for the key. Default is empty. An application can set or change the value at any time.
CKA_END_DATE ⁸	CK_DATE	End date for the key. Default is empty. An application can set or change the value at any time.
CKA_DERIVE ⁹	CK_BBOOL	TRUE if key supports key derivation (other keys can be derived from this one). Default is FALSE. An application can set or change the value at any time.
CKA_LOCAL ^{2, 4, 6}	CK_BBOOL	TRUE only if key was generated locally. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_GEN_MECHANISM ^{2, 4, 6}	CK_MECHANISM_TYPE	Identifier of the mechanism used to generate the key. Always CK_UNAVAILABLE_INFORMATION. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_ENCRYPT ⁸	CK_BBOOL	TRUE if key supports encryption ⁹ . Default is TRUE. An application can set or change the value at any time.
CKA_VERIFY ⁸	CK_BBOOL	TRUE if key supports verification where the signature is an appendix to the data. Default is TRUE. An application can set or change the value at any time.
CKA_WRAP ⁸	CK_BBOOL	TRUE if key supports wrapping (can be used to wrap other keys). ⁹ Default is TRUE. An application can set or change the value at any time.

Table 8. Secret key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 5 on page 15.

Attribute	Data type	Notes
CKA_DECRYPT ⁸	CK_BBOOL	TRUE if key supports decryption. ⁹ Default is TRUE. An application can set or change the value at any time.
CKA_SIGN ⁸	CK_BBOOL	TRUE if key supports signatures where the signature is an appendix to the data. ⁹ Default is TRUE. An application can set or change the value at any time.
CKA_UNWRAP ⁸	CK_BBOOL	TRUE if key supports unwrapping (can be used to unwrap other keys) ⁹ . Default is TRUE. An application can set or change the value at any time.
CKA_EXTRACTABLE ⁸	CK_BBOOL	TRUE if key is extractable. Caller can change from TRUE to FALSE only. Default is TRUE. An application can set or change the value, as per PKCS #11 restrictions.
CKA_SENSITIVE ⁸	CK_BBOOL	TRUE if key is sensitive. Caller can change from FALSE to TRUE only. Default is FALSE. An application can set or change the value, as per PKCS #11 restrictions.
CKA_ALWAYS_SENSITIVE ^{2, 4, 6}	CK_BBOOL	TRUE if key has always had the CKA_SENSITIVE attribute set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_NEVER_EXTRACTABLE ^{2, 4, 6}	CK_BBOOL	TRUE if key has never had the CKA_EXTRACTABLE attribute set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_VALUE ^{1, 4, 6, 7}	Byte array	The key. Sensitive key part. An application can specify the value when the object is created (or generated) only.
CKA_VALUE_LEN ^{2, 3, 6}	CK_ULONG	Length of the key in bytes (AES only). An application can specify the value when the object is created (or generated) only.
CKA_APPLICATION	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page. An application can specify the value when the object is created (or generated) only.

Table 9. Public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 5 on page 15.

Attribute	Data type	Notes
CKA_TOKEN ¹¹	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE. An application can specify the value when the object is created (or generated) only.
CKA_PRIVATE ¹¹	CK_BBOOL	Default value on create is FALSE. An application can specify the value when the object is created (or generated) only.
CKA_MODIFIABLE ¹¹	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_LABEL	Printable EBCDIC string	Application-specific nickname. Limited to 32 characters. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
CKA_TRUSTED	CK_BBOOL	Always set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_SUBJECT	Byte array	DER-encoding. Default empty. An application can set or change the value at any time.
CKA_ID	Byte array	Key identifier. Default empty. An application can set or change the value at any time.
CKA_KEY_TYPE ^{1, 3, 5}	CK_KEY_TYPE	Type of key. CKK_RSA only. An application can specify the value when the object is created (or generated) only.
CKA_START_DATE ⁸	CK_DATE	Start date for the key. Default empty. An application can set or change the value at any time.
CKA_END_DATE ⁸	CK_DATE	End date for the key. Default empty. An application can set or change the value at any time.
CKA_DERIVE ⁸	CK_BBOOL	TRUE if key supports key derivation (if other keys can be derived from this one). Default is FALSE. An application can set or change the value at any time.
CKA_LOCAL ^{2, 4, 6}	CK_BBOOL	TRUE only if key was generated locally. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.

Table 9. Public key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 5 on page 15.

Attribute	Data type	Notes
CKA_KEY_GEN_MECHANISM ^{2, 4, 6}	CK_MECHANISM_TYPE	Identifier of the mechanism used to generate the key. Always CK_UNAVAILABLE_INFORMATION. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_ENCRYPT ⁸	CK_BBOOL	TRUE if key supports encryption. ⁹ Default is TRUE. An application can set or change the value at any time.
CKA_VERIFY ⁸	CK_BBOOL	TRUE if key supports verification where the signature is an appendix to the data. Default is TRUE. An application can set or change the value at any time.
CKA_VERIFY_RECOVER ⁸	CK_BBOOL	TRUE if key supports verification where the data is recovered from the signature. ⁹ Default is TRUE. An application can set or change the value at any time.
CKA_WRAP ⁸	CK_BBOOL	TRUE if key supports wrapping (can be used to wrap other keys). ⁹ Default is TRUE. An application can set or change the value at any time.
CKA_APPLICATION	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page. An application can specify the value when the object is created (or generated) only.

Table 10. RSA public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 5 on page 15.

Attribute	Data type	Notes
CKA_MODULUS ^{1, 4}	Big integer	Modulus n An application can specify the value when the object is created (or generated) only.
CKA_MODULUS_BITS ^{2, 3}	CK_ULONG	Length in bits of modulus n An application can specify the value when the object is created (or generated) only.
CKA_PUBLIC_EXPONENT ¹	Big integer	Public exponent e An application can specify the value when the object is created (or generated) only.

Table 11. Private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 5 on page 15.

Attribute	Data type	Notes
CKA_TOKEN ¹¹	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE. An application can specify the value when the object is created (or generated) only.
CKA_PRIVATE ¹¹	CK_BBOOL	Default value on create is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_MODIFIABLE ¹¹	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_LABEL	Printable EBCDIC string	Application-specific nickname. Limited to 32 characters. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
CKA_SUBJECT	Byte array	DER-encoding. An application can set or change the value at any time.
CKA_ID	Byte array	Default is empty. An application can set or change the value at any time.
CKA_KEY_TYPE ^{1, 3, 5}	CK_KEY_TYPE	Type of key. CKK_RSA only. An application can specify the value when the object is created (or generated) only.
CKA_START_DATE ⁸	CK_DATE	Start date for the key. Default empty. An application can set or change the value at any time.
CKA_END_DATE ⁸	CK_DATE	End date for the key. Default empty. An application can set or change the value at any time.
CKA_DERIVE ⁸	CK_BBOOL	TRUE if key supports key derivation (if other keys can be derived from this one). Default is FALSE. An application can set or change the value at any time.
CKA_LOCAL ^{2, 4, 6}	CK_BBOOL	TRUE only if key was generated locally. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_KEY_GEN_MECHANISM ^{2, 4, 6}	CK_MECHANISM_TYPE	Identifier of the mechanism used to generate the key material. Always CK_UNAVAILABLE_INFORMATION. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.

Table 11. Private key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 5 on page 15.

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

Table 12. RSA private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 5 on page 15.

Attribute	Data type	Notes
CKA_MODULUS ^{1,4,6}	Big integer	Modulus <i>n</i> An application can specify the value when the object is created (or generated) only.

Table 12. RSA private key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 5 on page 15.

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

Library, slot, and token information

PKCS #11 maintains information about the library code, slots, and tokens, which can be set and queried by calling the library functions. For z/OS this information is as follows:

CK_INFO - Returned by C_GetInfo. not modifiable by applications.

- cryptokiVersion - 2.20
- manufacturerID - "IBM Corp. "
- libraryDescription - "z/OS PKCS11 library "
- libraryVersion - 2.9 (coincides with the current z/OS release)

CK_SLOT_INFO - Returned by C_GetSlotInfo. not modifiable by applications

- slotDescription - "z/OS PKCS11 - virtual smart card"
- manufacturerID - "IBM Corp."
- flags - for any slot returned by C_GetSlotList the following flags are set:
 - CKF_TOKEN_PRESENT=ON
 - CKF_REMOVABLE_DEVICE=ON
 - CKF_HW_SLOT=OFF
- hardwareVersion - 2.9 (coincides with current z/OS release)
- firmwareVersion - 2.9 (coincides with current z/OS release)

CK_TOKEN_INFO - Set by C_InitToken . Returned by C_GetTokenInfo

- label - As specified
- manufacturerID - "z/OS PKCS11 API" (Might be set to other values if the token was initialized outside of the C API.)
- model - "HCR7740" (coincides with the release that the token was created) (Might be set to other values if the token was initialized outside of the C API.)
- serialNumber - "0" (Might be set to other values if the token was initialized outside of the C API.)
- flags - the following flags are set ON for any initialized token. All others are OFF:
 - CKF_RNG
 - CKF_PROTECTED_AUTHENTICATION_PATH
 - CKF_TOKEN_INITIALIZED
 - CKF_USER_PIN_INITIALIZED
- ulMaxSessionCount - CK_UNAVAILABLE_INFORMATION
- ulSessionCount - CK_UNAVAILABLE_INFORMATION
- ulMaxRwSessionCount - CK_UNAVAILABLE_INFORMATION
- ulRwSessionCount - CK_UNAVAILABLE_INFORMATION
- ulMaxPinLen - CK_UNAVAILABLE_INFORMATION
- ulMinPinLen - 0
- ulTotalPublicMemory - CK_UNAVAILABLE_INFORMATION
- ulFreePublicMemory - CK_UNAVAILABLE_INFORMATION
- ulTotalPrivateMemory - CK_UNAVAILABLE_INFORMATION
- ulFreePrivateMemory - CK_UNAVAILABLE_INFORMATION
- hardwareVersion - 2.9 (coincides with current z/OS release)
- firmwareVersion - 2.9 (coincides with current z/OS release)
- utcTime - GMT date and time that the token was last updated

CK_SESSION_INFO - Returned by C_GetSessionInfo

- slotId - The slot in question
- state - CK_UNAVAILABLE_INFORMATION
- flags - As defined by the PKCS #11 specification
- ulDeviceError - A mapping of the last ICSF return and reason code values related to this session. For more information see "Function return codes" on page 32.

Functions supported

ICSF supports a subset of the standard PKCS #11 functions, and several non-standard functions.

Standard functions supported

Table 13 lists the standard PKCS #11 functions that ICSF supports. Any function not listed is not supported and returns the CKR_FUNCTION_NOT_SUPPORTED return code.

Table 13. Standard PKCS #11 functions that ICSF supports

Function	Usage notes
General purpose functions	
C_Initialize()	<ul style="list-style-type: none">• The library always uses OS locking for thread serialization. Therefore, if C_Initialize is called with the CreateMutex, DestroyMutex, LockMutex, and UnlockMutex function pointer arguments set and the CKF_OS_LOCKING_OK flag is not set, C_Initialize fails and returns the value CKR_CANT_LOCK.• When C_Initialize is called, the application-specific set of (virtual) slot IDs is allocated, one for each preexisting token that the application is authorized to use. (See the descriptions of C_GetSlotList and C_WaitForSlotEvent for information on how this set can increase in size.) The one exception to this occurs when C_Initialize is called by a child process after fork. If the PKCS #11 environment is inherited by the child process, the slot list and token state is not refreshed.
C_Finalize()	
C_GetInfo()	
C_GetFunctionList()	
Slot and token management functions	
C_GetSlotList()	<ul style="list-style-type: none">• If the pSlotList argument is NULL, this function returns only the number of allocated slots. In the process of returning this number C_GetSlotList searches for new tokens to which the application has access. If new tokens are found, slot IDs are allocated for them. This search is only performed if at least 5 seconds has passed since the last search was made.• If the pSlotList argument is non-NULL, this function returns the current list of virtual slot IDs. No attempt is made to discover new tokens created by other applications.• The tokenPresent argument flag is meaningless as all allocated slots have a token present.
C_GetSlotInfo()	
C_GetTokenInfo()	

Table 13. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
C_WaitForSlotEvent()	<ul style="list-style-type: none"> This function is used to dynamically allocate an additional slot in order to create a new token. There are no other slot events. The newly allocated slot ID is returned as the pSlot argument. The CKF_DONT_BLOCK argument flag is meaningless because this function never blocks. The dynamic slot allocation occurs synchronously.
C_GetMechanismList()	<p>The list of functions returned reflects the capabilities of the current cryptographic hardware configuration.</p> <p>Note: The loss or addition of hardware on the fly is not detected or reflected. (For example, on a z9-109, if the only CEX2C present is deactivated, this function still returns the mechanisms that require an active CEX2C to function.)</p>
C_GetMechanismInfo()	<p>The output of this function reflects the capabilities of the current cryptographic hardware configuration.</p>
C_InitToken()	<p>Tokens are protected by the security manager through profiles in the CRYPTOZ class. PINs are not used. The pPin and ulPinLen arguments are ignored.</p>
C_InitPIN()	<p>Tokens are protected by the security manager through profiles in the CRYPTOZ class. PINs are not used. This function performs no operation and always returns CKR_OK.</p>
C_SetPIN()	<p>Tokens are protected by the security manager through profiles in the CRYPTOZ class. PINs are not used. This function performs no operation and always returns CKR_OK.</p>
Session management functions	
C_OpenSession()	<p>The Notify and pApplication arguments are ignored.</p>
C_CloseSession()	
C_CloseAllSessions()	
C_GetSessionInfo()	<p>The state field returned is meaningless. It is always set to CK_UNAVAILABLE_INFORMATION .</p>
C_GetOperationState()	<p>Returns CKR_STATE_UNSAVEABLE if a find is active or more than one cryptographic operation is active.</p>
C_SetOperationState()	
C_Login()	<p>Tokens are protected by the security manager through profiles in the CRYPTOZ class. Applications are always logged in to the security manager. PINs are not used. This function has no effect on the session state and always returns CKR_OK.</p>
C_Logout()	<p>Tokens are protected by the security manager through profiles in the CRYPTOZ class. Applications are always logged in to the security manager. PINs are not used. This function has no effect on the session state and always returns CKR_OK.</p>
Object management functions	
C_CreateObject()	
C_CopyObject()	
C_DestroyObject()	
C_GetObjectSize()	

Table 13. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
C_GetAttributeValue()	
C_SetAttributeValue()	
C_FindObjectsInit()	
C_FindObjects()	Sensitive attributes cannot be used as search criteria when the object is marked sensitive or not exportable. Doing so results in no match found.
C_FindObjectsFinal()	
Encryption functions	
C_EncryptInit()	The following mechanisms are supported: <ul style="list-style-type: none"> • CKM_DES_ECB • CKM_DES_CBC • CKM_DES_CBC_PAD • CKM_DES3_ECB • CKM_DES3_CBC • CKM_DES3_CBC_PAD • CKM_RSA_PKCS • CKM_RSA_X_509 • CKM_AES_CBC • CKM_AES_ECB • CKM_AES_CBC_PAD
C_Encrypt()	
C_EncryptUpdate()	Multiple-part encryption is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms
C_EncryptFinal()	Multiple-part encryption is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
Decryption functions	
C_DecryptInit()	The following mechanisms are supported: <ul style="list-style-type: none"> • CKM_DES_ECB • CKM_DES_CBC • CKM_DES_CBC_PAD • CKM_DES3_ECB • CKM_DES3_CBC • CKM_DES3_CBC_PAD • CKM_RSA_PKCS • CKM_RSA_X_509 • CKM_AES_CBC • CKM_AES_ECB • CKM_AES_CBC_PAD
C_Decrypt()	
C_DecryptUpdate()	Multiple-part decryption is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
C_DecryptFinal()	Multiple-part decryption is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
Message digesting functions	
C_DigestInit()	The following mechanisms are supported: <ul style="list-style-type: none"> • CKM_MD5 • CKM_SHA_1 • CKM_SHA256 • CKM_RIPEMD160
C_Digest()	

Table 13. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
C_DigestUpdate()	
C_DigestFinal()	
Signing and message authentication coding (MACing) functions	
C_SignInit()	The following mechanisms are supported: <ul style="list-style-type: none"> • CKM_RSA_X_509 • CKM_RSA_PKCS • CKM_MD5_RSA_PKCS • CKM_SHA1_RSA_PKCS • CKM_SHA256_RSA_PKCS
C_Sign()	
C_SignUpdate()	Multiple-part signature is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
C_SignFinal()	Multiple-part signature is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
Functions for verifying signatures and message authentication codes (MACs)	
C_VerifyInit()	The following mechanisms are supported: <ul style="list-style-type: none"> • CKM_RSA_X_509 • CKM_RSA_PKCS • CKM_MD5_RSA_PKCS • CKM_SHA1_RSA_PKCS • CKM_SHA256_RSA_PKCS
C_Verify()	
C_VerifyUpdate()	Multiple-part verify is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
C_VerifyFinal()	Multiple-part verify is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
Key management functions	
C_GenerateKey()	The following mechanisms are supported: <ul style="list-style-type: none"> • CKM_DES_KEY_GEN • CKM_DES2_KEY_GEN • CKM_DES3_KEY_GEN • CKM_AES_KEY_GEN
C_GenerateKeyPair()	The following mechanisms are supported: <ul style="list-style-type: none"> • CKM_RSA_PKCS_KEY_PAIR_GEN
C_WrapKey()	The following mechanisms are supported for wrapping symmetric keys: <ul style="list-style-type: none"> • CKM_RSA_PKCS <p>The following mechanisms are supported for wrapping asymmetric keys:</p> <ul style="list-style-type: none"> • CKM_DES_CBC_PAD • CKM_DES3_CBC_PAD • CKM_AES_CBC_PAD

Table 13. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
C_UnwrapKey()	<p>The following mechanisms are supported unwrapping symmetric keys:</p> <ul style="list-style-type: none"> • CKM_RSA_PKCS <p>The following mechanisms are supported unwrapping asymmetric keys:</p> <ul style="list-style-type: none"> • CKM_DES_CBC_PAD • CKM_DES3_CBC_PAD • CKM_AES_CBC_PAD
Random number generation functions	
C_SeedRandom()	This function always returns the value CKR_RANDOM_SEED_NOT_SUPPORTED because the z/OS hardware random number generator is self-seeding.
C_GenerateRandom()	

Non-standard functions supported

The following non-standard function is also supported. Because it is non-standard, it does not appear in the PKCS #11 CK_FUNCTION_LIST structure returned by C_GetFunctionList(). To invoke this function, the caller must either locate the function in the main DLL using dlsym(), or link the application program with the main DLL's sidedeck.

Table 14. Syntax of the CK_RV CSN_FindALLObjects() function

CSN_FindALLObjects (
CK_SESSION_HANDLE	hSession,
CK_OBJECT_HANDLE_PTR	phObject,
CK_ULONG	ulMaxObjectCount
CK_ULONG_PTR	pu1ObjectCount
);	

CSN_FindALLObjects() is identical to C_FindObjects(), except that it uses the ALL rule array keyword when invoking the ICSF CSFPTRL callable service. This can result in CSN_FindALLObjects() returning handles to private objects even if the caller has insufficient SAF authority to view such objects. CSN_FindALLObjects() returns a private key handle (and C_FindObjects does not) when the following conditions are all met:

1. The private object matches the search criteria.
2. No sensitive attributes were specified in the search criteria. The sensitive values for this service are:
 - For a secret key object: CKA_VALUE
 - For a private key object: CKA_PRIVATE_EXPONENT, CKA_PRIME_1, CKA_PRIME_2, CKA_EXPONENT_1, CKA_EXPONENT_2, CKA_COEFFICIENT
3. The caller has only Weak SO or SO R/W permission to the token.

For more information about CSFPTRL processing with respect to the ALL rule array keyword, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

Function return codes

In general, the PKCS #11 function return codes are defined in the PKCS #11 specification. However, the following function return codes have a meaning specific to z/OS:

CKR_TOKEN_NOT_PRESENT

ICSF is not running or the TKDS is not properly configured. Note that this return code has no relationship to the slot flag CKF_TOKEN_PRESENT.

CKR_TOKEN_NOT_RECOGNIZED

The caller is not authorized to perform the action requested.

CKR_MECHANISM_INVALID

The specified mechanism is either unknown or not supported by the current cryptographic hardware configuration.

CKR_DEVICE_REMOVED

The token no longer exists. When this error is detected, the token flags are cleared indicating that the token is no longer initialized. It can be re-initialized as a new token, if desired.

Other ICSF-related errors are returned as vendor-defined error codes (CKR_VENDOR_DEFINED). The ICSF return and reason codes are combined into the single return code as follows:

```
#define CKR_IBM_ICSF_ERROR 0xC0000000 /* High order byte mask indicating ICSF error */
#define CKR_IBM_ICSF_ERROR_RET 0x00FF0000 /* Second byte is the return code */
#define CKR_IBM_ICSF_ERROR_RSN 0x0000FFFF /* low order half word is reason code */
```

This mapping is also used to store the ICSF return and reason code values in the CK_SESSION_INFO ulDeviceError field.

Troubleshooting PKCS #11 applications

Note: The information and techniques described in this section are for use primarily by IBM service personnel in determining the cause of a problem with the ICSF PKCS #11 C API.

You can capture trace data using environment variables. To do this, the trace environment variables CSN_PKCS11_TRACE and CSN_PKCS11_TRACE_FILE must be exported prior to the application's first call to any of the PKCS #11 functions.

Table 15. Environment variables for capturing trace data

Environment variable	Usage	Valid values
CSN_PKCS11_TRACE	Specifies the level of tracing to be performed.	<p>An integer value, 1-7. The higher the value, the greater the number of conditions traced. Each level includes the conditions of the levels below it:</p> <ul style="list-style-type: none"> 7 Debug information 6 Informational conditions 5 Normal but significant conditions 4 Warning conditions 3 Error conditions 2 Critical conditions 1 Immediate action required <p>Any other value causes tracing to be inactive (the default).</p>
CSN_PKCS11_TRACE_FILE	<p>Specifies the name of the trace file. Defaults to /tmp/csnpkcs11.%.trc.</p> <p>The current process identifier is included as part of the trace file name when the name contains a percent sign (%). For example, if CSN_PKCS11_TRACE_FILE is set to:</p> <pre>/tmp/csnpkcs11.%.trc</pre> <p>and the current process identifier is 247, the trace file name is</p> <pre>/tmp/csnpkcs11.247.trc</pre>	<p>Must be set to the full path name of an HFS file in a directory for which the executing application has write permission. The maximum length for the path name is 255 bytes. Values longer than 255 bytes are truncated.</p>

You can also use the utility program, testpkcs11, for troubleshooting. For information about running testpkcs11, see “Running the pre-compiled version of testpkcs11” on page 35.

Chapter 3. The testpkcs11 program

IBM provides a sample PKCS #11 program called testpkcs11 . The program is passed the name of a PKCS #11 token, and performs the following tasks:

1. Creates a token that has the name passed
2. Generates an RSA key-pair
3. Encrypts some test data using the public part of the key-pair
4. Decrypts the data using the private part of the key-pair
5. Deletes the key-pair and the token

You can use this program in several ways:

- As a utility program to test the system configuration for PKCS #11 and troubleshoot problems
- As a sample application to learn how to build and run a PKCS #11 application

IBM provides a pre-compiled version of this program installed in /usr/lpp/pkcs11/bin. IBM also provides the C source code for this program in the samples subdirectory, along with three Makefiles:

- Makefile - for 31-bit addressing mode
- Makefile3X - for 31-bit addressing mode with high performance (XPLINK) linkage
- Makefile64 - for 64-bit addressing mode with high performance (XPLINK) linkage

For the source code for this program, see “Source code for the testpkcs11 sample program” on page 43.

Running the pre-compiled version of testpkcs11

If you are testing the system configuration for PKCS #11, or troubleshooting problems with the configuration, you can run the pre-compiled version of testpkcs11.

Steps for running the pre-compiled version of testpkcs11

Before you begin: You need to know how to use z/OS UNIX® shells.

Perform the following steps to run the pre-compiled version of testpkcs11.

1. Change to the PKCS #11 bin directory by entering the following command:

```
cd /usr/lpp/pkcs11/bin
```

2. Choose a temporary token name to use. If you need to review the rules for token names, see “Tokens” on page 2.

3. Run testpkcs11, passing it your token name on the -t option. For example, to use a temporary token name of my.temp.token, enter the following command:

```
./testpkcs11 -t my.temp.token
```

If z/OS PKCS #11 has been set up properly and the you have sufficient authority to the token label specified, you should see the following output:

```
Getting the PKCS11 function list...
Initializing the PKCS11 environment...
Creating the temporary token...
```

```
Opening a session...
Generating keys. This may take a while...
Enciphering data...
Deciphering data...
Destroying keys...
Closing the session...
Deleting the temporary token...
Test completed successfully!
```

If you see different messages, there is an error in either your PKCS #11 set up or in the token label that you specified.

The most common user error is specifying token label that is unacceptable to ICSF or already in use. In that case the following is displayed:

```
Getting the PKCS11 function list...
Initializing the PKCS11 environment...
Creating the temporary token...
  C_InitToken #1 returned 7 (0x07) CKR_ARGUMENTS_BAD
  Make sure your the token name you specified meets ICSF rules:
  Contains only alphanumeric characters, nationals (@#$), and periods.
  The first character cannot be a numeric or a period.
```

If you see other error messages, there is probably an error in the setup for the PKCS #11 environment. Determine the error represented by the PKCS #11 error code returned. For information about error codes, see “Function return codes” on page 32.

To display the help text for the testpkcs11 program, run the program with the -h option:

```
cd /usr/lpp/pkcs11/bin
./testpkcs11 -h
```

Building testpkcs11 from source code

If you are learning how to build and run a PKCS #11 application, you can use the source code for testpkcs11 to build and run a sample application.

Steps for building testpkcs11 from source code

Before you begin: You need to know in which directory the PKCS #11 header file is located. By default it is located in the standard include subdirectory under /usr. If your standard include subdirectory it in a different location, you will need to modify the Makefile in step 2. You also need to know how to use z/OS UNIX shells.

Perform the following steps to build the sample application, testpkcs11. Issue the commands from the z/OS UNIX command shell.

1. Copy the testpkcs11.c program and appropriate Makefile to the current directory. For example, for 64-bit addressing enter the following commands:

```
cp /usr/lpp/pkcs11/samples/testpkcs11.c testpkcs11.c
cp /usr/lpp/pkcs11/samples/Makefile64 Makefile
```

2. If the standard include subdirectory is not located under /usr, edit the Makefile copied in step 1 and change the PKCS11_INSTALL_DIR variable as required.

3. Enter the following command to compile and link and produce the executable, testpkcs11:

make

4. Update your C/C++ environment variable `_CEE_RUNOPTS` to include `XPLINK(ON)` if it does not already include it. For example, execute the following command from a UNIX shell:

```
export _CEE_RUNOPTS=$_CEE_RUNOPTS' XPLINK(ON)'
```

When you are done, you have built the `testpkcs1164` application and can run it in your directory. For example, to run `testpkcs1164` in your directory and display its help text, enter the following command:

```
./testpkcs1164 -h
```

To run a test, choose a temporary token name and enter it with the `-t` option. If you need to review the rules for token names, see “Tokens” on page 2. For example, to use a temporary token name of `my.temp.token`, enter the following command:

```
./testpkcs1164 -t my.temp.token
```

For the output that should appear, see “Steps for running the pre-compiled version of `testpkcs11`” on page 35.

Chapter 4. Token management callable services

The token management callable services provide an application program interface for managing z/OS PKCS #11 tokens. The C API uses these services. These callable services include:

- CSFPTRC - token record create
- CSFPTRD - token record delete
- CSFPTRL - token record list
- CSFPSAV - set attribute value
- CSFPGAV - get attribute value

Calls to the system authorization facility (SAF) determine access authorization for the callable services. The CSFSERV class controls access to the token management callable services.

For details about the token management callable services, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

SMP/E installation data sets, directories, and files

The following dynamic link libraries (DLLs) are linked into SYS1.SIELNKE:

CSNPCAPI

The main DLL invoked by applications to use PKCS #11 functions. Also shipped as an HFS file at /usr/lpp/pkcs11/lib/csnp capi.so.

CSNPCA64

64-bit addressing mode version of CSNPCAPI. Also shipped as an HFS file at /usr/lpp/pkcs11/lib/csnpca64.so.

CSNPCA3X

31-bit addressing mode version of CSNPCAPI with XPLINK. Also shipped as an HFS file at /usr/lpp/pkcs11/lib/csnpca3x.so

CSNPCINT

An internal DLL loaded by CSNPCAPI..

CSNPCI64

64-bit addressing mode version of CSNPCINT.

CSNPCI3X

31-bit addressing mode version of CSNPCINT with XPLINK.

CSNPCUTL

An internal DLL implicitly loaded for utilities.

CSNPCU64

64-bit addressing mode version of CSNPCUTL.

CSNPCU3X

31-bit addressing mode version of CSNPCUTL with XPLINK.

CSFDLL31

CSFDLL in 31-bit addressing mode

CSFDLL64

CSFDLL in 64-bit addressing mode.

CSFDLL3X

31-bit addressing mode version of CSFDLL31 with XPLINK.

SMP/E installs the product files into the HFS directory /usr/lpp/pkcs11. This directory contains the following subdirectories and files:

- /usr/lpp/pkcs11/include subdirectory (members are symbolically linked to /usr/include)

csnpdefs.h

A header file that applications must include to use PKCS #11 functions. Also copied to SYS1.SIEAHDR.H(CSNPDEFS).

csfbext.h

A header file that applications must include to use the CSFDLLs. Also copied to SYS1.SIEAHDR.H(CSFBEXT).

- /usr/lpp/pkcs11/lib subdirectory (members are symbolically linked to /usr/lib)

CSNPCAPI.x

Side deck for CSNPCAPI. Also copied to SYS1.SIEASID(CSNPCAPI).

CSNPCA64.x

Side deck for CSNPCA64. Also copied to SYS1.SIEASID(CSNPCA64).

CSNPCA3X.x

Side deck for CSNPCA3X. Also copied to SYS1.SIEASID(CSNPCA3X).

csnpcapi.so**csnpca64.so****csnpca3x.so****CSFDLL31.x**

Side deck for CSFDLL31. Also copied to SYS1.SIEASID(CSFDLL31).

CSFDLL64.x

Side deck for CSFDLL64. Also copied to SYS1.SIEASID(CSFDLL64).

CSFDLL3X.x

Side deck for CSFDLL3X. Also copied to SYS1.SIEASID(CSFDLL3X).

- /usr/lpp/pkcs11/bin subdirectory

testpkcs11

Program to test system configuration for PKCS #11.

- /usr/lpp/pkcs11/samples subdirectory

testpkcs11.c

Source code for the testpkcs11 program.

Makefile

Makefile for the testpkcs11 program.

Makefile64

Makefile for 64-bit addressing mode version of the testpkcs11 program.

Makefile3X

Makefile for 31-bit addressing mode version of the testpkcs11 program with XPLINK.

Source code for the testpkcs11 sample program

For information about building and using the testpkcs11 sample program, see Chapter 3, "The testpkcs11 program," on page 35.

```
/*
/* *****
/* COMPONENT_NAME: testpkcs11.c
/*
/* Licensed Materials - Property of IBM
/* 5694-A01
/* Copyright IBM Corp. 2007
/* Status = HCR7740
/*
/* *****
/* *****
/*
/* This file contains sample code. IBM PROVIDES THIS CODE ON AN
/* 'AS IS' BASIS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR
/* IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
/* OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
/*
/* *****
/*
/* Change Activity:
/* $L0=P11C1 , HCR7740 , 060124 , JWS: PKCS11 initial release
/* $D1=MG08269, HCR7740, 061114 , JWS: Misc fixes
/* $D2=MG08740 , HCR7740 , 070302 , GL: XPLINK
/*
/* *****

#ifdef IBM
/* Customers may remove this copyright statement */
#pragma comment (copyright, "\
Licensed Materials - Property of IBM \
5694-A01 Copyright IBM Corp. 2007 \
All Rights Reserved \
US Government Users Restricted Rights - \
Use, duplication or disclosure restricted by \
GSA ADP Schedule Contract with IBM Corp.")
#endif

/* Install verification test for PKCS #11 */

#define _UNIX03_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>
#include <dlfcn.h>
#include <sys/timeb.h>
#include <csnpdefs.h>

int skip_token_obj;

CK_FUNCTION_LIST *funcs;
CK_SLOT_ID slotID = CK_UNAVAILABLE_INFORMATION;
CK_BYTE tokenName[32];

void ProcessRetCode( CK_RV rc )
{
    switch (rc) {
        case CKR_OK: printf(" CKR_OK"); break;
        case CKR_CANCEL: printf(" CKR_CANCEL"); break;
        case CKR_HOST_MEMORY: printf(" CKR_HOST_MEMORY"); break;
        case CKR_SLOT_ID_INVALID: printf(" CKR_SLOT_ID_INVALID"); break;
        case CKR_GENERAL_ERROR: printf(" CKR_GENERAL_ERROR"); break;
        case CKR_FUNCTION_FAILED: printf(" CKR_FUNCTION_FAILED"); break;
        case CKR_ARGUMENTS_BAD: printf(" CKR_ARGUMENTS_BAD"); break;
        case CKR_NO_EVENT: printf(" CKR_NO_EVENT"); break;
        case CKR_NEED_TO_CREATE_THREADS: printf(" CKR_NEED_TO_CREATE_THREADS"); break;
    }
}
```

```

case CKR_CANT_LOCK:                printf(" CKR_CANT_LOCK");                break;
case CKR_ATTRIBUTE_READ_ONLY:      printf(" CKR_ATTRIBUTE_READ_ONLY");      break;
case CKR_ATTRIBUTE_SENSITIVE:      printf(" CKR_ATTRIBUTE_SENSITIVE");      break;
case CKR_ATTRIBUTE_TYPE_INVALID:    printf(" CKR_ATTRIBUTE_TYPE_INVALID");    break;
case CKR_ATTRIBUTE_VALUE_INVALID:   printf(" CKR_ATTRIBUTE_VALUE_INVALID");   break;
case CKR_DATA_INVALID:             printf(" CKR_DATA_INVALID");             break;
case CKR_DATA_LEN_RANGE:           printf(" CKR_DATA_LEN_RANGE");           break;
case CKR_DEVICE_ERROR:             printf(" CKR_DEVICE_ERROR");             break;
case CKR_DEVICE_MEMORY:            printf(" CKR_DEVICE_MEMORY");            break;
case CKR_DEVICE_REMOVED:          printf(" CKR_DEVICE_REMOVED");          break;
case CKR_ENCRYPTED_DATA_INVALID:     printf(" CKR_ENCRYPTED_DATA_INVALID");     break;
case CKR_ENCRYPTED_DATA_LEN_RANGE:  printf(" CKR_ENCRYPTED_DATA_LEN_RANGE");  break;
case CKR_FUNCTION_CANCELED:        printf(" CKR_FUNCTION_CANCELED");        break;
case CKR_FUNCTION_NOT_PARALLEL:     printf(" CKR_FUNCTION_NOT_PARALLEL");     break;
case CKR_FUNCTION_NOT_SUPPORTED:    printf(" CKR_FUNCTION_NOT_SUPPORTED");    break;
case CKR_KEY_HANDLE_INVALID:        printf(" CKR_KEY_HANDLE_INVALID");        break;
case CKR_KEY_SIZE_RANGE:           printf(" CKR_KEY_SIZE_RANGE");           break;
case CKR_KEY_TYPE_INCONSISTENT:     printf(" CKR_KEY_TYPE_INCONSISTENT");     break;
case CKR_KEY_NOT_NEEDED:           printf(" CKR_KEY_NOT_NEEDED");           break;
case CKR_KEY_CHANGED:              printf(" CKR_KEY_CHANGED");              break;
case CKR_KEY_NEEDED:               printf(" CKR_KEY_NEEDED");               break;
case CKR_KEY_INDIGESTIBLE:         printf(" CKR_KEY_INDIGESTIBLE");         break;
case CKR_KEY_FUNCTION_NOT_PERMITTED: printf(" CKR_KEY_FUNCTION_NOT_PERMITTED"); break;
case CKR_KEY_NOT_WRAPPABLE:        printf(" CKR_KEY_NOT_WRAPPABLE");        break;
case CKR_KEY_UNEXTRACTABLE:        printf(" CKR_KEY_UNEXTRACTABLE");        break;
case CKR_MECHANISM_INVALID:        printf(" CKR_MECHANISM_INVALID");        break;
case CKR_MECHANISM_PARAM_INVALID:  printf(" CKR_MECHANISM_PARAM_INVALID");  break;
case CKR_OBJECT_HANDLE_INVALID:    printf(" CKR_OBJECT_HANDLE_INVALID");    break;
case CKR_OPERATION_ACTIVE:         printf(" CKR_OPERATION_ACTIVE");         break;
case CKR_OPERATION_NOT_INITIALIZED: printf(" CKR_OPERATION_NOT_INITIALIZED"); break;
case CKR_PIN_INCORRECT:           printf(" CKR_PIN_INCORRECT");           break;
case CKR_PIN_INVALID:             printf(" CKR_PIN_INVALID");             break;
case CKR_PIN_LEN_RANGE:           printf(" CKR_PIN_LEN_RANGE");           break;
case CKR_PIN_EXPIRED:             printf(" CKR_PIN_EXPIRED");             break;
case CKR_PIN_LOCKED:              printf(" CKR_PIN_LOCKED");              break;
case CKR_SESSION_CLOSED:          printf(" CKR_SESSION_CLOSED");          break;
case CKR_SESSION_COUNT:           printf(" CKR_SESSION_COUNT");           break;
case CKR_SESSION_HANDLE_INVALID:   printf(" CKR_SESSION_HANDLE_INVALID");   break;
case CKR_SESSION_PARALLEL_NOT_SUPPORTED: printf(" CKR_SESSION_PARALLEL_NOT_SUPPORTED"); break;
case CKR_SESSION_READ_ONLY:       printf(" CKR_SESSION_READ_ONLY");       break;
case CKR_SESSION_EXISTS:          printf(" CKR_SESSION_EXISTS");          break;
case CKR_SESSION_READ_ONLY_EXISTS: printf(" CKR_SESSION_READ_ONLY_EXISTS");  break;
case CKR_SESSION_READ_WRITE_SO_EXISTS: printf(" CKR_SESSION_READ_WRITE_SO_EXISTS"); break;
case CKR_SIGNATURE_INVALID:        printf(" CKR_SIGNATURE_INVALID");        break;
case CKR_SIGNATURE_LEN_RANGE:      printf(" CKR_SIGNATURE_LEN_RANGE");      break;
case CKR_TEMPLATE_INCOMPLETE:      printf(" CKR_TEMPLATE_INCOMPLETE");      break;
case CKR_TEMPLATE_INCONSISTENT:    printf(" CKR_TEMPLATE_INCONSISTENT");    break;
case CKR_TOKEN_NOT_PRESENT:        printf(" CKR_TOKEN_NOT_PRESENT - ICSF is not active or not configured for TKDS operations"); break;
case CKR_TOKEN_NOT_RECOGNIZED:     printf(" CKR_TOKEN_NOT_RECOGNIZED - You are not authorized to perform the token operation"); break;
case CKR_TOKEN_WRITE_PROTECTED:    printf(" CKR_TOKEN_WRITE_PROTECTED");    break;
case CKR_UNWRAPPING_KEY_HANDLE_INVALID: printf(" CKR_UNWRAPPING_KEY_HANDLE_INVALID"); break;
case CKR_UNWRAPPING_KEY_SIZE_RANGE: printf(" CKR_UNWRAPPING_KEY_SIZE_RANGE"); break;
case CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT: printf(" CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT"); break;
case CKR_USER_ALREADY_LOGGED_IN:   printf(" CKR_USER_ALREADY_LOGGED_IN");   break;
case CKR_USER_NOT_LOGGED_IN:       printf(" CKR_USER_NOT_LOGGED_IN");       break;
case CKR_USER_PIN_NOT_INITIALIZED:  printf(" CKR_USER_PIN_NOT_INITIALIZED");  break;
case CKR_USER_TYPE_INVALID:        printf(" CKR_USER_TYPE_INVALID");        break;
case CKR_USER_ANOTHER_ALREADY_LOGGED_IN: printf(" CKR_USER_ANOTHER_ALREADY_LOGGED_IN"); break;
case CKR_USER_TOO_MANY_TYPES:      printf(" CKR_USER_TOO_MANY_TYPES");      break;
case CKR_WRAPPED_KEY_INVALID:      printf(" CKR_WRAPPED_KEY_INVALID");      break;
case CKR_WRAPPED_KEY_LEN_RANGE:    printf(" CKR_WRAPPED_KEY_LEN_RANGE");    break;
case CKR_WRAPPING_KEY_HANDLE_INVALID: printf(" CKR_WRAPPING_KEY_HANDLE_INVALID"); break;
case CKR_WRAPPING_KEY_SIZE_RANGE:  printf(" CKR_WRAPPING_KEY_SIZE_RANGE");  break;
case CKR_WRAPPING_KEY_TYPE_INCONSISTENT: printf(" CKR_WRAPPING_KEY_TYPE_INCONSISTENT"); break;
case CKR_RANDOM_SEED_NOT_SUPPORTED: printf(" CKR_RANDOM_SEED_NOT_SUPPORTED"); break;
case CKR_RANDOM_NO_RNG:           printf(" CKR_RANDOM_NO_RNG");           break;
case CKR_BUFFER_TOO_SMALL:         printf(" CKR_BUFFER_TOO_SMALL");         break;
case CKR_SAVED_STATE_INVALID:      printf(" CKR_SAVED_STATE_INVALID");      break;
case CKR_INFORMATION_SENSITIVE:    printf(" CKR_INFORMATION_SENSITIVE");    break;
case CKR_STATE_UNSAVEABLE:        printf(" CKR_STATE_UNSAVEABLE");        break;
case CKR_CRYPTOKI_NOT_INITIALIZED: printf(" CKR_CRYPTOKI_NOT_INITIALIZED");  break;
case CKR_CRYPTOKI_ALREADY_INITIALIZED: printf(" CKR_CRYPTOKI_ALREADY_INITIALIZED"); break;
case CKR_MUTEX_BAD:               printf(" CKR_MUTEX_BAD");               break;
case CKR_MUTEX_NOT_LOCKED:        printf(" CKR_MUTEX_NOT_LOCKED");        break;

```



```

        /* Otherwise - Value does not match a known PKCS11 return value */
    }
}

void showError( char *str, CK_RV rc )
{
    printf("%s returned: %d (0x%0x)", str, rc, rc );
    ProcessRetCode( rc );
    printf("\n");
}

CK_RV createToken( void )
{
    CK_VOID_PTR      p = NULL; // @D1C
    CK_RV            rc;
    CK_FLAGS         flags = 0;

    printf("Creating the temporary token... \n");
    /* wait for slot event. On z/OS this creates a new slot synchronously */
    rc = funcs->C_WaitForSlotEvent(flags, &slotID, p);
    if (rc != CKR_OK) {
        showError(" C_WaitForSlotEvent #1", rc );
        return !CKR_OK;
    }
    /* The slot has been created. Now initialize the token in the slot */
    /* On z/OS no PIN is required, so we will pass NULL. */
    rc = funcs->C_InitToken(slotID, NULL, 0, tokenName);
    if (rc != CKR_OK) {
        showError(" C_InitToken #1", rc );
        if (rc == CKR_ARGUMENTS_BAD) {
            printf(" Make sure your the token name you specified meets ICSF rules:\n");
            printf(" Contains only alphanumeric characters, nationals (@#$), and periods.\n");
            printf(" The first character cannot be a numeric or a period.\n");
        }
        return !CKR_OK;
    }

    return CKR_OK;
}

CK_RV deleteToken( void )
{
    CK_VOID_PTR      p;
    CK_RV            rc;
    CK_FLAGS         flags = 0;

    if (slotID != CK_UNAVAILABLE_INFORMATION) {
        printf("Deleting the temporary token... \n");
        /* C_InitToken with the reserved label $$DELETE-TOKEN$$ is the way to delete a token */
        /* on z/OS */
        memset(tokenName, ' ', sizeof(tokenName));
        memcpy(tokenName, DEL_TOK, sizeof(DEL_TOK));
        rc = funcs->C_InitToken(slotID, NULL, 0, tokenName);
        if (rc != CKR_OK) {
            showError(" C_InitToken #2 (for delete)", rc );
            return !CKR_OK;
        }
    }

    return CKR_OK;
}

CK_RV encryptRSA( void )
{
    CK_BYTE          data1[100];
    CK_BYTE          data2[256];
    CK_BYTE          cipher[256];
    CK_SLOT_ID      slot_id;
    CK_SESSION_HANDLE session;
    CK_MECHANISM     mech;
    CK_OBJECT_HANDLE publ_key, priv_key;
    CK_FLAGS         flags;
    CK_ULONG         i;
    CK_ULONG         len1, len2, cipherlen;

```

```

CK_RV          rc;
static CK_OBJECT_CLASS  class = CKO_PUBLIC_KEY;
static CK_KEY_TYPE      type=  CKK_RSA;
static CK_OBJECT_CLASS  privclass = CKO_PRIVATE_KEY;
static CK_BBOOL         true  = TRUE;
static CK_BBOOL         false = FALSE;

static CK_ULONG  bits = 1024;
static CK_BYTE   pub_exp[] = { 0x01, 0x00, 0x01 };

/* Attributes for the public key to be generated */
CK_ATTRIBUTE pub_tmpl[] = {
    {CKA_MODULUS_BITS,  &bits,  sizeof(bits)  },
    {CKA_ENCRYPT,        &>true,  sizeof(true) },
    {CKA_VERIFY,        &>true,  sizeof(true) },
    {CKA_PUBLIC_EXPONENT, &pub_exp, sizeof(pub_exp) }
};

/* Attributes for the private key to be generated */
CK_ATTRIBUTE priv_tmpl[] =
{
    {CKA_DECRYPT, &true, sizeof(true) },
    {CKA_SIGN,   &true, sizeof(true) }
};

slot_id = slotID;
flags = CKF_SERIAL_SESSION | CKF_RW_SESSION;
printf("Opening a session... \n");
rc = funcs->C_OpenSession( slot_id, flags, (CK_VOID_PTR) NULL, NULL, &session );
if (rc != CKR_OK) {
    showError(" C_OpenSession #1", rc );
    return !CKR_OK;
}

printf("Generating keys. This may take a while... \n");
mech.mechanism = CKM_RSA_PKCS_KEY_PAIR_GEN;
mech.ulParameterLen = 0;
mech.pParameter = NULL;

rc = funcs->C_GenerateKeyPair( session, &mech,
                               pub_tmpl, 4,
                               priv_tmpl, 2,
                               &publ_key, &priv_key );

if (rc != CKR_OK) {
    showError(" C_GenerateKeyPair #1", rc );
    return !CKR_OK;
}

/* now, encrypt some data */
len1 = sizeof(data1);
len2 = sizeof(data2);
cipherlen = sizeof(cipher);

for (i=0; i < len1; i++)
    data1[i] = (i) % 255;

mech.mechanism = CKM_RSA_PKCS;
mech.ulParameterLen = 0;
mech.pParameter = NULL;

printf("Enciphering data... \n");
rc = funcs->C_EncryptInit( session, &mech, publ_key );
if (rc != CKR_OK) {
    showError(" C_EncryptInit #1", rc );
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

rc = funcs->C_Encrypt( session, data1, len1, cipher, &cipherlen );
if (rc != CKR_OK) {
    showError(" C_Encrypt #1", rc );
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

```

```

/* now, decrypt the data */
printf("Deciphering data... \n");
rc = funcs->C_DecryptInit( session, &mech, priv_key );
if (rc != CKR_OK) {
    showError("    C_DecryptInit #1", rc );
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

rc = funcs->C_Decrypt( session, cipher, cipherlen, data2, &len2 );
if (rc != CKR_OK) {
    showError("    C_Decrypt #1", rc );
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

/* PKCS - returns clear data as is */
if (len1 != len2) {
    printf("    ERROR: lengths don't match\n");
    printf("    Length of original data = %d, after decryption = %d\n", len1, len2);
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

for (i=0; i < len1; i++) {
    if (data1[i] != data2[i]) {
        printf("    ERROR: mismatch at byte %d\n", i );
        funcs->C_CloseSession( session );
        return !CKR_OK;
    }
}

printf("Destroying keys... \n");
rc = funcs->C_DestroyObject( session, priv_key );
if (rc != CKR_OK) {
    showError("    C_DestroyObject #1", rc );
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

rc = funcs->C_DestroyObject( session, publ_key );
if (rc != CKR_OK) {
    showError("    C_DestroyObject #2", rc );
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

printf("Closing the session... \n");
rc = funcs->C_CloseSession( session );
if (rc != CKR_OK) {
    showError("    C_CloseSession #1", rc );
    return !CKR_OK;
}

return CKR_OK;
}

```

```

CK_RV getFunctionList( void )
{
    CK_RV      rc;
    CK_RV      (*pFunc)();
    void      *d;
#ifdef _LP64
    char      e[]="CSNPCA64";
#elif _XPLINK
    char      e[]="CSNPCA3X";          /* @D2A */
#else
    char      e[]="CSNPCA3I";
#endif

```

```

printf("Getting the PKCS11 function list...\n");

```

```

d = dlopen(e,RTLD_NOW);
if ( d == NULL ) {
    printf("%s not found in linklist or LIBPATH\n",e); // @D1A
    return !CKR_OK;
}

pFunc = (CK_RV (*)( ))dlsym(d,"C_GetFunctionList");
if (pFunc == NULL ) {
    printf("C_GetFunctionList() not found in module %s\n",e); // @D1A
    return !CKR_OK;
}
rc = pFunc(&funcs);

if (rc != CKR_OK) {
    showError(" C_GetFunctionList", rc );
    return !CKR_OK;
}

return CKR_OK;
}

void displaySyntax(char *pgm) {
    printf("usage: %s { -t <token-name> | -h }\n\n", pgm );
    printf(" -t <token-name> = The name of a temporary token to create for the test. The\n");
    printf(" name must be less than 33 characters in length and contains only alphanumeric\n");
    printf(" characters, nationals (@#$), and periods. The first character cannot be a\n");
    printf(" numeric or a period. The token will be deleted when the test is complete.\n\n");
    printf(" -h = Displays this help.\n\n");
}

void main( int argc, char **argv )
{
    CK_C_INITIALIZE_ARGS cinit_args;
    CK_RV rc, i;

    memset(tokenName, ' ', sizeof(tokenName)); /* Token name is left justified, padded with blanks */

    if (argc == 3) {
        if (strcmp(argv[1], "-t") == 0)
            if (strlen(argv[2]) > 0 && strlen(argv[2]) < 33) {
                memcpy(tokenName, argv[2], strlen(argv[2]));
            }
            else {
                displaySyntax(argv[0]);
                return;
            }
        else {
            displaySyntax(argv[0]);
            return;
        }
    }
    else {
        displaySyntax(argv[0]);
        return;
    }

    rc = getFunctionList();
    if (rc != CKR_OK) {
        printf("getFunctionList failed!\n"); // @D1C
        return;
    }

    memset( &cinit_args, 0x0, sizeof(cinit_args) );
    cinit_args.flags = CKF_OS_LOCKING_OK;

    printf("Initializing the PKCS11 environment...\n");
    rc = funcs->C_Initialize( &cinit_args );
    if (rc != CKR_OK) {
        showError(" C_Initialize", rc );
        return;
    }
}

```

```

rc = createToken();
if (rc != CKR_OK) {
    funcs->C_Finalize( NULL );
    return;
}

rc = encryptRSA();
if (rc != CKR_OK) {
    deleteToken();
    funcs->C_Finalize( NULL );
    return;
}

rc = deleteToken();
if (rc != CKR_OK) {
    funcs->C_Finalize( NULL );
    return;
}

rc = funcs->C_Finalize( NULL );
if (rc != CKR_OK) {
    showError(" C_Initialize", rc );
    return;
}
printf("Test completed successfully!\n");
}

```

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Vol I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

z/OS information

z/OS information is accessible using screen readers with the BookServer/Library Server versions of z/OS books in the Internet library at:

<http://www.ibm.com/systems/z/os/zos/bkserv/>

Notices

This information was developed for products and services offered in the USA.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Programming interface information

This publication documents intended programming interfaces that allow the customer to write programs to obtain the services of z/OS Integrated Cryptographic Services Facility (ICSF).

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries, or both:

- AIX
- IBM
- IBMLink
- RACF
- z/OS
- z/VM
- z/VSE
- z9
- zSeries

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product or service names may be trademarks or service marks of others.

Index

A

accessibility 51
auditing PKCS #11 functions 10

C

C application program interface (API), using 13
CCA (Common Cryptographic Architecture) 2
CK_RV CSN_FindAllObjects() function 31
Common Cryptographic Architecture (CCA) 2
component trace entries for TKDS events 10
constants, manifest
 where defined 13
Cryptoki 2
CRYPTOZ class 5
CSFSERV class
 resources for token services 6

D

data object
 attributes that ICSF supports 16
decryption functions supported 29
deleting token 13
disability 51
DLLs provided by ICSF 41
dynamic link libraries (DLLs) provided by ICSF 41

E

encryption functions supported 29
environment variables for tracing 32

F

function
 non-standard PKCS #11 supported by ICSF 31
 standard PKCS #11 supported by ICSF 27
function return code
 unique to z/OS, list of 32

G

general purpose functions supported 27

H

header file for C API 41

I

installation options data set
 options for the TKDS 3
 SYSPLEXTKDS option 3
 TKDSN option 3

J

job to define TKDS 4

K

key management functions supported 30
key types supported 13
keyboard 51

L

Language Environment 13
library
 information that can be set and queried 25

M

MAC verification functions supported 30
MACing functions supported 30
manifest constants
 where defined 13
mechanism
 information returned by C_GetMechanismInfo 14
 which cryptographic hardware supports 15
message authentication code verification functions
 supported 30
message authentication coding functions supported 30
message digesting functions supported 29

N

Notices 53

O

object
 data
 attributes that ICSF supports 16
 private key
 attributes that ICSF supports 23
 RSA private key
 attributes that ICSF supports 24
 RSA public key
 attributes that ICSF supports 22
 secret key
 attributes that ICSF supports 18
 X.509 certificate
 attributes that ICSF supports 17
object management functions supported 28
object type
 supported by ICSF, list of 15

P

PIN 5
PKCS (Public Key Cryptography Standards) 2

- private key object
 - attributes that ICSF supports 23
- program, sample
 - building and using 35
 - source code for 43
- Public Key Cryptography Standards (PKCS) 2
- public key object
 - attributes that ICSF supports 21

R

- random number generation functions supported 31
- RSA private key object
 - attributes that ICSF supports 24
- RSA public key object
 - attributes that ICSF supports 22

S

- sample program, testpkcs11
 - building and using 35
 - source code for 43
- secret key object
 - attributes that ICSF supports 18
- serialization of updates to TKDS 4
- session management functions supported 28
- session object 10
- shortcut keys 51
- signature verification functions supported 30
- signing functions supported 30
- slot 2
 - information that can be queried 26
- slot functions supported 27
- slot ID 2
- SMF records written for PKCS #11 functions 10
- SO R/W
 - description 5
- SO role 5
- SRB mode 13
- Strong SO
 - description 5
- SYSPLEXTKDS option in the installation options data set 3
- SYSZTCKDS.TKDS*dsn* resource 4
- SYSZTKT resource 4

T

- tasks
 - building testpkcs11 application
 - steps 36
 - running the sample program
 - steps 35
- TCB mode 13
- testpkcs11 program
 - building and using 35
 - source code for 43
- TKDS (token data set)
 - description 3
 - sample job to define 4
 - serialization of resources 4

- TKDSN option in the installation options data set 3
- token
 - access levels 5
 - deleting 13
 - information that can be set and queried 26
 - managing, interfaces for 7
- token data set (TKDS)
 - description 3
 - sample job to define 4
 - serialization of resources 4
- token management functions supported 27
- token object 11
- token, on z/OS
 - description 2
 - rules for name 2
- trace data 32
- trace entries for TKDS events 10
- troubleshooting applications 32

U

- User R/O
 - description 6
- User R/W
 - description 6
- User role 5

W

- Weak SO
 - description 5
- Weak User
 - description 6

X

- X.509 certificate object
 - attributes that ICSF supports 17

Z

- z/OS PKCS #11 token
 - deleting 13
 - description 2
 - rules for name 2

Readers' Comments — We'd Like to Hear from You

z/OS

Cryptographic Services Integrated Cryptographic Service Facility Writing PKCS #11 Applications

Publication No. SA23-2231-01

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:

- Send your comments to the address on the reverse side of this form.
- Send your comments via e-mail to: mhvrfs@us.ibm.com

If you would like a response from IBM, please fill in the following information:

Name

Address

Company or Organization

Phone No.

E-mail address



Fold and Tape

Please do not staple

Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
MHVRCFS, Mail Station P181
2455 South Road
Poughkeepsie, NY
12601-5400



Fold and Tape

Please do not staple

Fold and Tape



Program Number: 5694-A01

Printed in USA

SA23-2231-01

