Linux on System z

Exploiting Enterprise PKCS #11 using openCryptoki 3.1

SC34-2713-00

Linux on System z

Exploiting Enterprise PKCS #11 using openCryptoki 3.1

SC34-2713-00

Note

Before using this document, be sure to read the information in "Notices" on page 71.

Edition notice

This edition applies to the EP11 token for openCryptoki version 3.1 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures

About this document

Linux on System z® applications that are using a PKCS #11 API can take advantage of the Enterprise PKCS #11 (EP11) coprocessor mode of the IBM® Crypto Express4S (CEX4S) adapter.

EP11 is a stack architecture that provides a complete environment to use a library of standard cryptographic functions that are used to write applications on IBM System z with cryptographic hardware.

You can find the latest version of this document on the developerWorks® website at:

www.ibm.com/developerworks/linux/linux390/documentation_dev.html

and on the IBM Knowledge Center at:

ibm.com/support/knowledgecenter/linuxonibm/liaaf/lnz_r_lib.html

How this document is organized

Chapter 1, "Introduction," on page 1 contains general information about the Linux on System z EP11 enablement.

Chapter 2, "The EP11 crypto stack," on page 5 describes how the components of the Linux on System z EP11 enablement are positioned within the different layers between applications and hardware.

Chapter 3, "Building the EP11 crypto stack," on page 11 describes how to prepare or install the EP11 components within the stack.

Chapter 4, "Configuring openCryptoki for EP11 support," on page 41 describes the configuration and customization tasks for enabling the exploitation of the EP11 library functions from applications.

Chapter 5, "Using the EP11 token," on page 49 describes the APIs for invoking the EP11 library functions.

Chapter 6, "Managing master keys on the Crypto Express4S EP11 coprocessor," on page 53 tells how to use a tool to handle changes with secure master keys.

Chapter 7, "Troubleshooting EP11," on page 57 provides information how to resolve problems when using the Linux on System z EP11 enablement.

Chapter 8, "Programming examples for openCryptoki," on page 59 is a set of programming samples that use the EP11 library.

Who should read this document

This document is intended for C programmers who want to access IBM System z hardware support for cryptographic methods. It is also intended for system administrators who need to enable and configure the required cryptographic hardware.

Furthermore, this publication addresses users who want to enhance their existing openCryptoki applications with the new features of Enterprise PKCS #11.

Distribution independence

This publication does not provide information that is specific to a particular Linux distribution.

The tools it describes are distribution independent.

Other Linux on System z publications

You can find Linux on System z publications on developerWorks and on the IBM Knowledge Center.

These publications are available on developerWorks at

www.ibm.com/developerworks/linux/linux390/documentation_dev.html

- *Device Drivers, Features, and Commands*, SC33-8411
- *Using the Dump Tools*, SC33-8412
- *How to Improve Performance with PAV*, SC33-8414
- *How to use FC-attached SCSI devices with Linux on System z*, SC33-8413
- *How to use Execute-in-Place Technology with Linux on z/VM®*, SC34-2594
- *How to Set up a Terminal Server Environment on z/VM*, SC34-2596
- *Kernel Messages*, SC34-2599
- *libica Programmer's Reference*, SC34-2602
- *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294
- *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713
- *Linux on System z Troubleshooting*, SC34-2612
- *Linux Health Checker User's Guide*, SC34-2609

These publications are available on the IBM Knowledge Center at

ibm.com/support/knowledgecenter/linuxonibm/liaaf/lnz_r_lib.html

- *libica Programmer's Reference*, SC34-2602
- *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294
- *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713
- *Linux Health Checker User's Guide*, SC34-2609
- *Linux on System z Troubleshooting*, SC34-2612
- *Kernel Messages*, SC34-2599

Chapter 1. Introduction

The Linux on System z Enterprise PKCS #11 (EP11) enablement allows applications to use a PKCS #11 API to run secure key cryptographic operations on an IBM Crypto Express adapter that is configured as an Crypto Express4S EP11 coprocessor. The CEX4S adapter card is the first Crypto Express adapter which can be configured as an EP11 coprocessor.

The Linux on System z EP11 enablement comprises several components that need to be installed and configured within certain locations of the EP11 stack as described in Chapter 2, “The EP11 crypto stack,” on page 5.

An application's request is first submitted to a PKCS #11 API, implemented by the openCryptoki library and the EP11 token. From this token, the request is finally propagated to the Crypto Express4S EP11 coprocessor. The request is processed on this coprocessor. The resulting output is returned to the application across the involved interfaces.

The EP11 cryptography architecture offers a secure key infrastructure.

This introduction provides information about the standard software that is used in this implementation and about the used Crypto Express4S EP11 coprocessor (shortly referred to as CEX4P).

What is PKCS #11?

The Public-Key Cryptographic Standards (PKCS) comprise a group of cryptographic standards that provide guidelines and application programming interfaces (APIs) for the usage of cryptographic methods. As the name PKCS suggests, these standards put an emphasis on the usage of public key (that is, asymmetric) cryptography.

PKCS #11 is a cryptographic token interface standard, which specifies an API, called *Cryptoki*. With this API, applications can address cryptographic devices as tokens and can perform cryptographic functions as implemented by these tokens. This standard, first developed by the RSA Laboratories in cooperation with representatives from industry, science, and governments, is now an open standard lead-managed by the *OASIS PKCS 11 Technical Committee*.

It follows an object-based approach, addressing the goals of technology independence (any kind of HW device) and resource sharing. It also presents to applications a common, logical view of the device that is called a cryptographic *token*. *PKCS #11* , or *Cryptoki*, assigns a slot ID to each token. An application identifies the token that it wants to access by specifying the appropriate slot ID.

For more information about *PKCS #11*, refer to this URL:

`PKCS #11 Cryptographic Token Interface Standard`

What is openCryptoki?

openCryptoki is an open source implementation of the *Cryptoki* API defined by the PKCS #11 Cryptographic Token Interface Standard.

Thus, openCryptoki provides support for several cryptographic algorithms according to the industry-wide PKCS #11 standards. The openCryptoki library loads the so called tokens that provide hardware or software specific support for cryptographic functions.

The EP11 token extends the openCryptoki token library. It uses special hardware cryptographic functions that are provided by the IBM Crypto Express4S (CEX4S) adapter, which is configured by a certain firmware (see "Enabling the CEX4S adapter for EP11 firmware exploitation" on page 13).

openCryptoki can be used directly through the openCryptoki shared library (C API).

For more information about the openCryptoki services, or about the interfaces between the openCryptoki main module and its tokens, see `sourceforge.net/projects/opencryptoki`.

You can also read topic "openCryptoki overview" on page 7 for an introduction of the openCryptoki main features.

What is a Crypto Express4S EP11 coprocessor?

An IBM Crypto Express adapter, which is configured with the Enterprise PKCS #11 (EP11) firmware, is called an Crypto Express4S EP11 coprocessor. The Crypto Express4 adapter is the first adapter that can be configured as an EP11 coprocessor. Such an enabled adapter is also called CEX4P coprocessor. In the remainder of this document, an Crypto Express4S EP11 coprocessor is sometimes shortly referred to as CEX4P adapter.

The CEX4P adapter provides hardware-accelerated support for crypto operations that are based on the PKCS #11 Cryptographic Token Interface Standard. Access from applications to the functions of an CEX4P adapter is enabled through the EP11 stack. This EP11 stack consists of certain EP11 user space libraries and an EP11 extension in the Linux AP device driver. Using several layers of interfaces, the PKCS #11 standard requests are propagated to and returned from the CEX4P adapter by the device driver.

A CEX4P adapter is a *hardware security module (HSM)* that maintains and protects secrets (for example, master keys) such that these secrets cannot be revealed from outside the adapter: No operating system service or application can retrieve these secrets and any trial to physically break into the card destroys its data due to its tamper proof design.

The CEX4P adapter supports cryptographic operations with secure keys. A *secure key* is a key that is encrypted (wrapped) by a master key that is stored in the adapter. Therefore, on the CEX4P adapter, applications can decrypt (unwrap) a secure key and use it for cryptographic operations inside the adapter. Outside the adapter (for example, inside an operating system), a secure key cannot be used for cryptographic operations. To use a secure key, an application must call functions on the CEX4P adapter. It is therefore safe to keep a secure key in memory or to store it in a file system.

Cryptographic keys that are not encrypted are called *clear keys*. If a clear key is stored in memory or in a file, unauthorized access to that memory or file must carefully be prevented. Otherwise, the key can be stolen and used to decrypt protected information. The CEX4P adapter does not support clear key cryptography.

A Crypto Express4S EP11 coprocessor supports up to 16 domains. Each domain acts like an EP11 coprocessor, but maintains its own master key. That means, that the master key of one domain cannot be accessed by another domain. The CEX4P adapter supports 16 domains. Different domains of a crypto adapter may be assigned to different LPARs or z/VM guests, such that multiple LPARs or guests can share one Crypto Express4S EP11 coprocessor without sharing their master keys.

Chapter 2. The EP11 crypto stack

The EP11 crypto stack for Linux on System z consists of various components within the different layers from the application request down to the hardware and back again, with the returned request result. The stack thus provides an end-to-end solution for cryptographic operations.

For example, an application sends an encryption request to the crypto adapter. Through various interfaces, such a request is propagated from the application layer down to the target crypto adapter hardware. On its way down, the request passes through the involved layers: the standard crypto interfaces, the System z crypto libraries, and the operating system kernel. The zcrypt device driver finally sends the request to the CEX4P coprocessor. The resulting request output is sent back to the application just the other way round through the layer interfaces.

Figure 1 on page 6 illustrates the EP11 crypto stack within the Linux on System z environment. The components that make up the Linux on System z EP11 enablement are highlighted:

- the EP11 token within openCryptoki
- the host part of the EP11 library (located in user space, which is named `libep11.so`)
- the EP11 extension of the zcrypt device driver. This extension was included with kernel level 3.14 on https://www.kernel.org/. Note that distributions sometimes back-port features from newer *kernel.org* kernels into their current kernel versions. Therefore check with your distribution partner, whether your distribution release supports the EP11 enablement, if its kernel version is older than 3.14.
- the module part of the EP11 library, that is, the EP11 firmware that is installed on the CEX4S adapter hardware.

You can use the openCryptoki shared library directly (C API).

openCryptoki supports several tokens, which can offer different functionality for different hardware devices or software solutions. The EP11 token interacts with the host part of the EP11 library. EP11 can operate with the Crypto Express4S (CEX4S) adapter with EP11 firmware load for symmetric and asymmetric cryptographic functions.

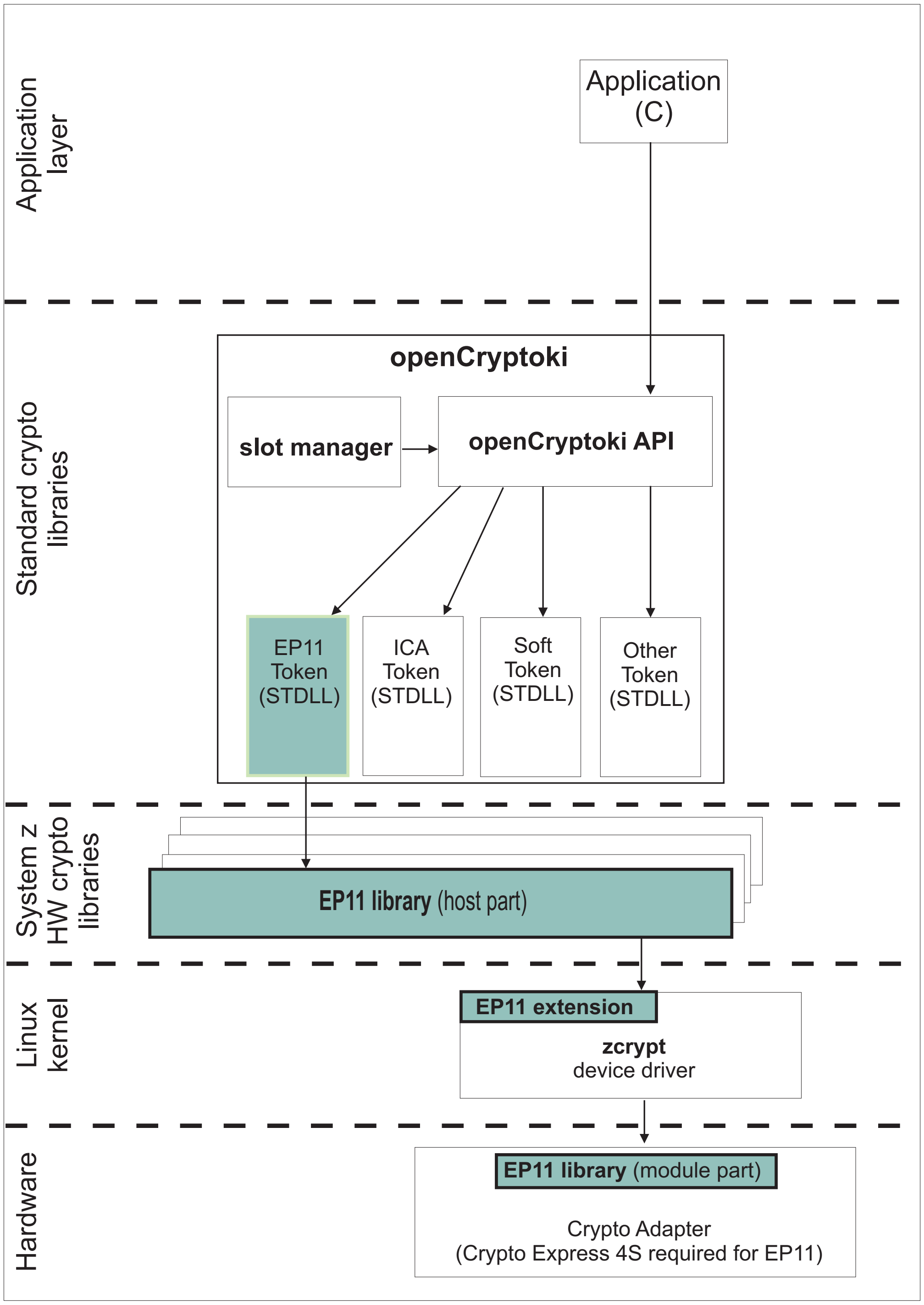


Figure 1. Stack and process flow with a configured EP11 token

The EP11 token itself does not implement PKCS #11 but provides services for accessing EP11 functions to openCryptoki. For a description of these services or the interface between the common part of openCryptoki and its tokens, see the openCryptoki documentation. Once the EP11 token is configured, cryptographic functions from the EP11 token are available to an application through the PKCS #11 API provided by the common openCryptoki code. The EP11 token itself accesses the EP11 library. The EP11 library is the host part of EP11, the module part of EP11 runs in the Crypto Express4S card. An installed EP11 library is a prerequisite for enabling openCryptoki to use the EP11 token. The EP11 library passes requests to the Crypto Express4S card through the zcrypt device driver of Linux on System z.

The host part of the EP11 library transforms cryptographic requests from the EP11 token into buffers. These transformed requests are sent to the CEX4P adapter. The host part also converts response buffers that are received from the adapter into data structures that are expected by the EP11 token. The EP11 token makes these APIs accessible to openCryptoki and thus the applications, but does not implement any cryptographic mechanism. The mechanisms available and their parameters depend on the EP11 implementation (EP11 library and Crypto Express4S card) and its configuration. The PKCS #11 Cryptographic Token Interface Standard defines methods for inquiring available mechanisms. All mechanisms and their parameters reported by PKCS #11 functions **`C_GetMechanismList`** and **`C_GetMechaninfo`** are available.

Besides the CEX4S adapter that is loaded with the EP11 firmware (also referred to as the EP11 module part), the EP11 token furthermore requires a zcrypt device driver that is loaded into the kernel, extended with the Linux on System z EP11 enablement support (see “Loading the Linux zcrypt device driver” on page 18). In addition, the EP11 token requires the availability of the host part of the EP11 library.

Therefore, check the following dependencies:

- **Dependencies on distributors:** Distributors build the openCryptoki RPM packages that comprise the EP11 support (EP11 token) for delivering them to customers. See also “Installing openCryptoki” on page 38.
- **Dependencies on hardware:** The EP11 library functions run on the IBM zEnterprise EC12 (zEC12) processor family (processor types 2827-H20, -H43, -H66, -H89, -HA1) or follow-on processors with an IBM Crypto Express4S (CEX4S) or follow-on adapter.

Note: In the remainder of this publication, the terms EP11 or Linux on System z EP11 enablement stand for the entirety of the implementation components that consists of the EP11 token, the EP11 extension of the *zcrypt* device driver, and the EP11 library (host part and module part) as shown in Figure 1 on page 6.

openCryptoki overview

openCryptoki consists of an implementation of the PKCS #11 API, a slot manager, an API for slot token dynamic link libraries (STDLLs), and a set of STDLLs (or tokens). The EP11 token is a new STDLL introduced with openCryptoki version 3.1.

The openCryptoki base library (`libopencryptoki.so`) provides the generic API as outlined in the PKCS #11 specification (version 2.20). This library also loads token-specific modules (STDLLs) that provide the token specific implementation of the PKCS #11 API and cryptographic functions (for example, session management, object management, and crypto algorithms). For a description of the PKCS #11 version 2.20 standard, refer to the following URL: PKCS #11 Cryptographic Token Interface Standard

A global configuration file (`/etc/opencryptoki/opencryptoki.conf`) is provided which describes the available tokens. This configuration file can be customized for the individual tokens. The openCryptoki package contains man pages that describe the format of the configuration files. For more information, see “Adjusting the openCryptoki configuration file” on page 41.

The EP11 token is a plug-in into the openCryptoki token library, providing support for several cryptographic algorithms.

Slot manager

The slot manager (**pkcsslotd**) runs as a daemon. Upon startup, it creates a shared memory segment and reads the openCryptoki configuration file to acquire the available token and slot information. The openCryptoki API attaches to this memory segment to retrieve token information. Thus, the slot manager provides the openCryptoki API with the token information when required. An application in turn links to or loads the openCryptoki API.

Slot token dynamic link libraries (STDLLs)

The EP11 token is an example of an STDLL within openCryptoki. STDLLs are plug-in modules to the openCryptoki (main) API. They provide token-specific functions that implement the interfaces. Specific devices can be supported by building an appropriate STDLL. Figure 1 on page 6 illustrates the stack and the process flow in a System z environment.

The STDLLs require local disk space to store persistent data, such as token information, personal identification numbers (PINs) and token objects. This information is stored in a separate directory for each token (for example in `/var/lib/opencryptoki/ep11tok` for the EP11 token). Within each of these directories there is a sub-directory `TOK_OBJ` that contains the token objects (token key store). Each private token object is represented by an encrypted file. Most of these directories are created during installation of openCryptoki.

The `pkcsconf` command line program

openCryptoki provides a command line program (`/usr/lib/pkcs11/methods/pkcsconf`) to configure and administer tokens that are supported within the system. The `pkcsconf` capabilities include token initialization, and security officer (SO) PIN and user PIN initialization and maintenance (see also “Initializing the token” on page 46).

`pkcsconf` operations that address a specific token must specify the slot that contains the token with the **-c** option. You can view the list of tokens present within the system by specifying the **-t** option (without **-c** option). For example, the following code shows the options for the `pkcsconf` command and displays slot information for the system:

```
# pkcsconf -?
pkcsconf: invalid option - ?
usage: pkcsconf [-itsmlIupPh] [-c slotnumber -U user-PIN -S SO-PIN -n new PIN]
```

The available options have the following meanings:

-i	display PKCS11 info
-t	display token info
-s	display slot info
-m	display mechanism list

-l	display slot description
-I	initialize token
-u	initialize user PIN
-p	set the user PIN
-P	set the SO PIN
-h	show this help
-c	specify the token slot for the operation
-U	the current user PIN (for use when changing the user pin with -u and -p options); if not specified, user will be prompted
-S	the current Security Officer (SO) pin (for use when changing the SO pin with -P option); if not specified, user will be prompted
-n	the new pin (for use when changing either the user pin or the SO pin with -u, -p or -P options); if not specified, user will be prompted

For more information about the `pkcsconf` command, see the `pkcsconf` man page.

Chapter 3. Building the EP11 crypto stack

The components of the Linux on System z EP11 enablement must be embedded into an infrastructure of hardware and software cryptographic components. In this environment, applications can start the provided functions by using the PKCS #11 openCryptokiAPI. This infrastructure is referred to as EP11 stack.

To enable the EP11 hardware cryptographic function support on System z, you must prepare some hardware components. You must also install and load specific driver modules and libraries, configure and start daemons, and set up your system environment.

Building this EP11 stack comprises several subtasks that are described in the following topics:

- "Preparing the Crypto Express4S EP11 coprocessor"
- "Loading the Linux zcrypt device driver" on page 18
- "Installing the host part of the EP11 library" on page 18
- "Setting a master key on the Crypto Express4S EP11 coprocessor" on page 19
- "Installing openCryptoki" on page 38
- "Configuring the EP11 token" on page 43

Preparing the Crypto Express4S EP11 coprocessor

To take advantage of the hardware-accelerated support for crypto operations from a CEX4P adapter, you must switch the CEX4S adapter into the CEX4P mode. This modification enables the installed and required EP11 firmware to run on this card.

The required information is presented in the following subtopics:

- "Purpose of domains"
- "Assigning adapters and domains to LPARs" on page 45
- "Enabling the CEX4S adapter for EP11 firmware exploitation" on page 13
- "Assigning EP11 adapters as dedicated adapters to z/VM guests" on page 17

Purpose of domains

When you configure your system on the Support Element (SE), you can specify how a logical partition (LPAR) uses coprocessors and accelerators. In this context, the Crypto Express cards support a concept of cryptographic domains. Each domain is protected by a master key, thus preventing access across domains and effectively separating the contained keys.

For information on how to configure domains, refer to *zEnterprise System Support Element Operations Guide*, which you can download from the IBM Resource Link.

There are two types of access to a cryptographic domain:

- for usage of cryptographic functions
- for management (control) of the domain, which includes the management of the master keys

A domain, which is assigned to an LPAR for usage access is called a usage domain of that LPAR. A domain, which is assigned to an LPAR for management (control) access is called a control domain of that LPAR. Every domain, which is a usage domain of an LPAR must also be a control domain of that LPAR, but not the other way round.

Usage domains

A logical partition's *usage domains* are domains in the coprocessors that can be used for cryptographic functions.

In Linux, you can use the **lszcrypt -b** command to find out which usage domain is configured for that Linux system:

```
$ lszcrypt -b

ap_domain=0
ap_interrupts are enabled
config_time=30 (seconds)
poll_thread is disabled
poll_timeout=250000 (nanoseconds)
```

Control domains

A logical partition's *control domains* are those cryptographic domains for which remote secure administration functions can be established and administered from this logical partition.

This logical partition's control domains must include its usage domains. So for each index that is selected in the *Usage domain index* list, you must select the same index in the *Control domain index* list.

But a logical partition's control domains can also include the control domains of other logical partitions. Assigning multiple logical partitions' control domains as control domains of a single logical partition allows using the partition to perform administrative functions from the TKE .

If you are using the Integrated Cryptographic Service Facility (ICSF) from z/OS, select at least one control domain with its matching usage domain. Refer to the ICSF documentation for information about ICSF basic operations.

If you are using a Trusted Key Entry (TKE) workstation to manage cryptographic keys, you can define your TKE host and the control domains for a logical partition. See “Setting a master key on the Crypto Express4S EP11 coprocessor” on page 19 for more information.

Control domain exposure

For configuration and management purposes the TKE needs to know which control domains are configured on the system.

In Linux, use a `sysfs` attribute called `ap_control_domain_mask` in `/sys/bus/ap/` to display the configured control domains. This information is set automatically from the device driver.

The attribute `ap_control_domain_mask` is read-only and contains a 32-byte field in hexadecimal notation, representing the installed control domain facilities. Each bit position represents a dedicated control domain. Thus, a maximum number of 256 domains could be addressed. For zEC12 processors, up to 16 domains are supported.

Example:

cat /sys/bus/ap/ap_control_domain_mask
0x0004000000000000000000000000000000000000000000000000000000000000

Byte	Meaning
1	domain 0-7
2	domain 8-15
3 - 32	reserved

In this example, the control domain 13 was configured.

Enabling the CEX4S adapter for EP11 firmware exploitation

You must have a CEX4S adapter, which is configured as an EP11 coprocessor card, and that is initialized and personalized in your z/VM guest or LPAR. Read this topic to learn how to check for the existence of a suitably configured CEX4P adapter and how to configure this adapter if it is missing yet.

About this task

A CEX4S Crypto Express card configured in the Enterprise PKCS #11 coprocessor mode (or shortly EP11 coprocessor mode) is also called a Crypto Express4S EP11 coprocessor (CEX4P). Such a coprocessor, which is installed in your z/VM guest or LPAR, is a prerequisite for using the functions of the EP11 library. This procedure shows you how to configure a CEX4S Crypto Express adapter into a CEX4P adapter by enabling the installed EP11 firmware from the Support Element.

Procedure

1. Check whether you have already plugged in and enabled your CEX4S Crypto Express card, and validate your model and type configuration (accelerator or coprocessor).

   To check, enter the **`lszcrypt`** command and check the output:

   ```
   $ lszcrypt
   card06: CEX4P
   ```

   If you see the output as shown, with an output line similar to

   ```
   cardxx: CEX4P
   ```

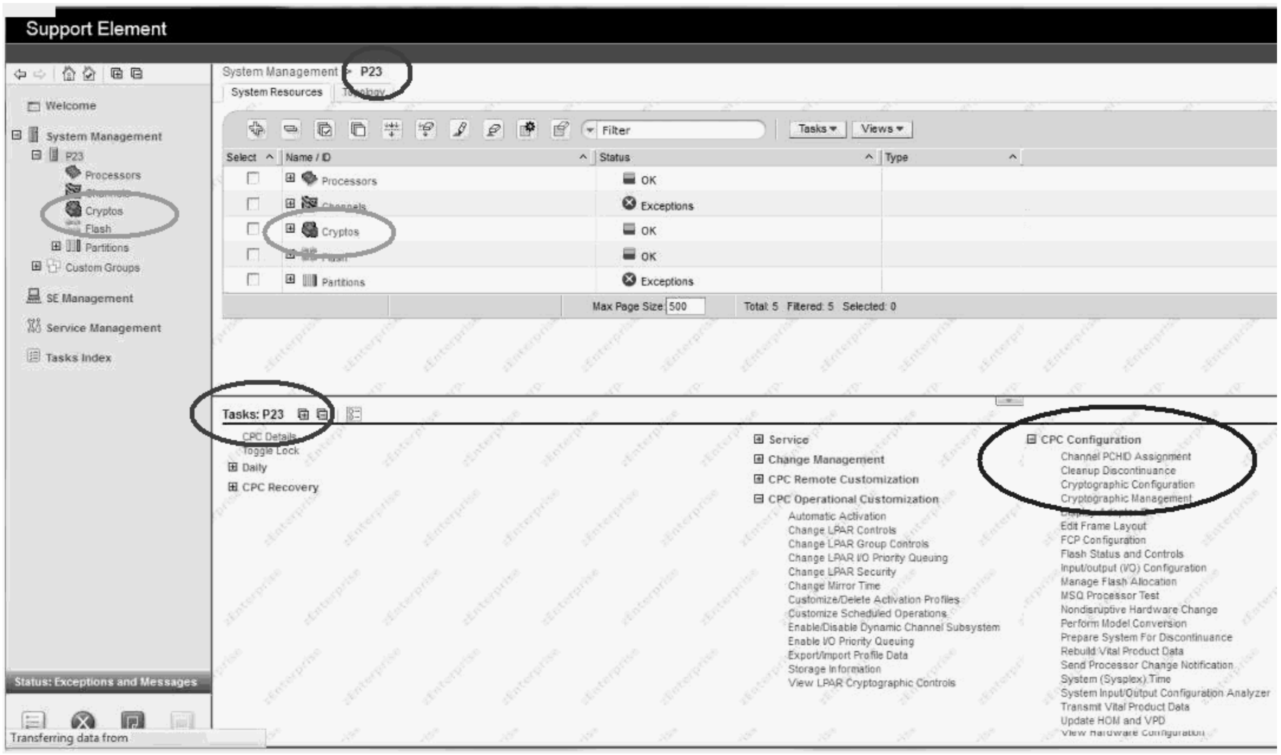
   then an CEX4P adapter is available and ready for use with EP11 and the task is completed.
2. If the following error message is displayed, the zcrypt device driver module must be installed.

   ```
   error - cryptographic device driver zcrypt is not loaded!
   ```

   For installation information, refer to “Loading the Linux zcrypt device driver” on page 18.
3. If the output from the **`lszcrypt`** command in step 1 does not show **`CEX4P`**, check the reason why this happend. If a CEX4S card is correctly assigned to the LPAR or z/VM guest, where the Linux is running in, but **`CEX4P`** is not shown, then you must activate the EP11 firmware on the CEX4S adapter. For

this purpose, log on to the Support Element with a user ID granted the appropriate access rights. You can either go directly to the Support Element, or you can use its web interface.

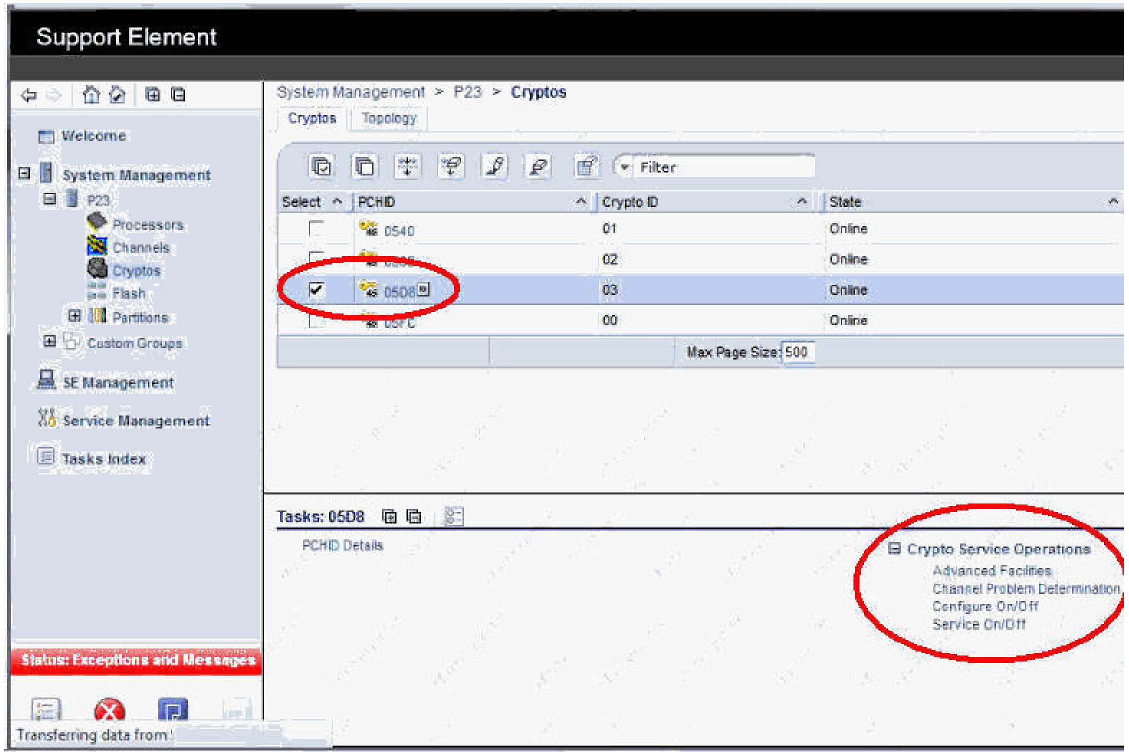
4. In the **System Management** window, select the CPC that holds the CEX4S adapter that you want to configure. In the sample screen from Figure 2, the selected CPC is *P23*.



Select	Name / ID	Status	Type
☐	Processors	OK	
☐	Channels	Exceptions	
☐	Cryptos	OK	
☐	Flash	OK	
☐	Partitions	Exceptions	

Figure 2. System Management in the Support Element

5. Select **Cryptos** from the dialog or from the navigation area to get a list of installed adapters as shown in Figure 3 on page 15.


Support Element

System Management > P23 > Cryptos

Select	PCHID	Crypto ID	State
☐	0540	01	Online
☐	[illegible]	02	Online
✓	05D8	03	Online
☐	05FC	00	Online

Max Page Size: 500

Tasks: 05D8

PCHID Details

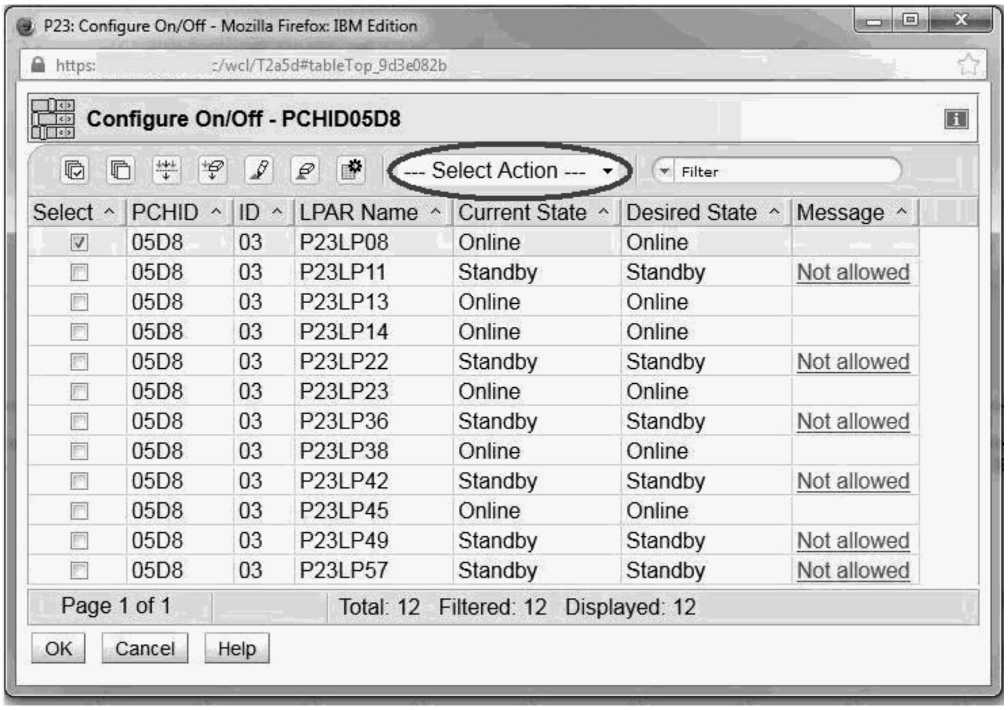
Crypto Service Operations
Advanced Facilities
Channel Problem Determination
Configure On/Off
Service On/Off

Status: Exceptions and Messages

Transferring data from

Figure 3. System Management - installed crypto adapters

6. Select the crypto card to be changed and also select **Configure On/Off** from the **Crypto Service Operations**.


P23: Configure On/Off - Mozilla Firefox: IBM Edition

https: :/wcl/T2a5d#tableTop_9d3e082b

Configure On/Off - PCHID05D8

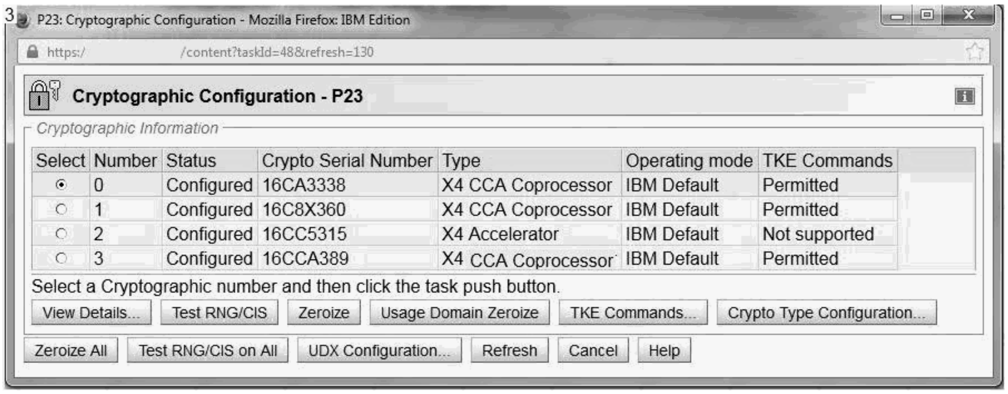
--- Select Action ---

Select	PCHID	ID	LPAR Name	Current State	Desired State	Message
✓	05D8	03	P23LP08	Online	Online	
☐	05D8	03	P23LP11	Standby	Standby	Not allowed
☐	05D8	03	P23LP13	Online	Online	
☐	05D8	03	P23LP14	Online	Online	
☐	05D8	03	P23LP22	Standby	Standby	Not allowed
☐	05D8	03	P23LP23	Online	Online	
☐	05D8	03	P23LP36	Standby	Standby	Not allowed
☐	05D8	03	P23LP38	Online	Online	
☐	05D8	03	P23LP42	Standby	Standby	Not allowed
☐	05D8	03	P23LP45	Online	Online	
☐	05D8	03	P23LP49	Standby	Standby	Not allowed
☐	05D8	03	P23LP57	Standby	Standby	Not allowed

Page 1 of 1 Total: 12 Filtered: 12 Displayed: 12

Figure 4. System Management - configure LPARs off

7. Select all LPARs, where this adapter is configured online, if any. The Crypto Express4S adapter must be configured offline in all LPARs, before you can change the configuration type. For this purpose, specify `Toggle` from the **Select Action** pull-down and press **OK**. This selection brings you back to the system level from Figure 2 on page 14.
8. From the **System Management** window, select **Cryptographic Configuration** from the **CPC Configuration** offerings (see Figure 2 on page 14).


P23: Cryptographic Configuration - Mozilla Firefox: IBM Edition

https:/ /content?taskId=48&refresh=130

Cryptographic Configuration - P23

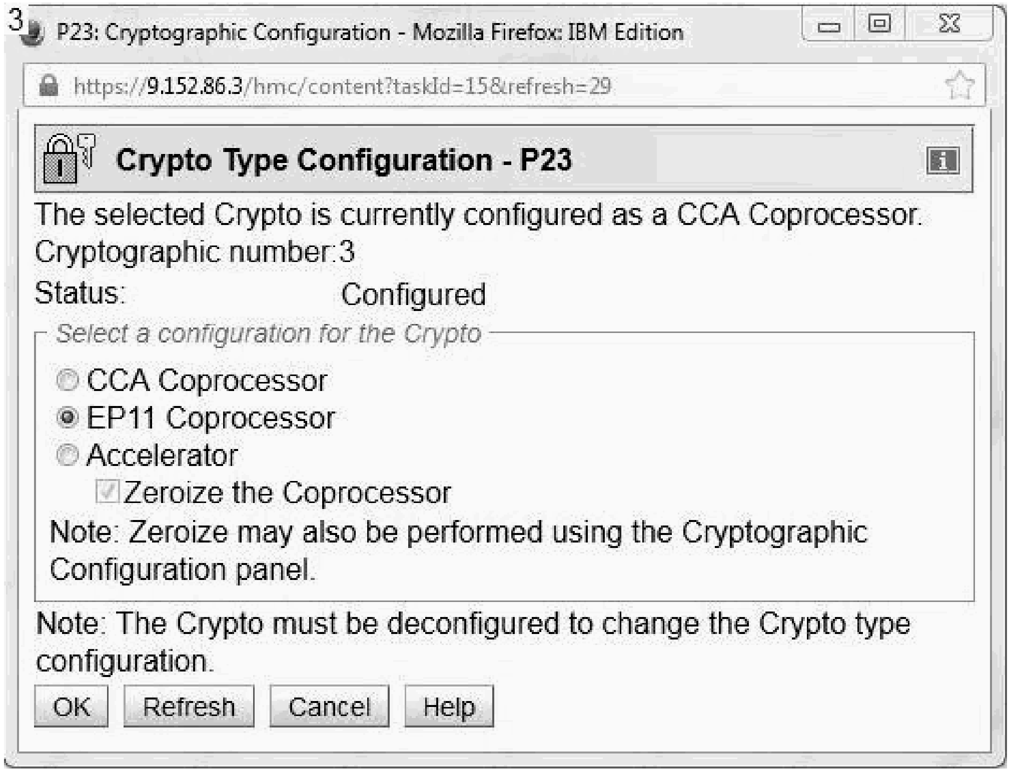
Cryptographic Information

Select	Number	Status	Crypto Serial Number	Type	Operating mode	TKE Commands
●	0	Configured	16CA3338	X4 CCA Coprocessor	IBM Default	Permitted
○	1	Configured	16C8X360	X4 CCA Coprocessor	IBM Default	Permitted
○	2	Configured	16CC5315	X4 Accelerator	IBM Default	Not supported
○	3	Configured	16CCA389	X4 CCA Coprocessor	IBM Default	Permitted

Select a Cryptographic number and then click the task push button.

Figure 5. System Management - Cryptographic Configuration

9. Select the desired adapter again (see step 4). Now press **Crypto Type Configuration** from the dialog shown in Figure 5. This selection brings you to the dialog shown in Figure 6.


P23: Cryptographic Configuration - Mozilla Firefox: IBM Edition

https://9.152.86.3/hmc/content?taskId=15&refresh=29

Crypto Type Configuration - P23

The selected Crypto is currently configured as a CCA Coprocessor.

Cryptographic number: 3

Status: Configured

Select a configuration for the Crypto

○ CCA Coprocessor

● EP11 Coprocessor

○ Accelerator

✓ Zeroize the Coprocessor

Note: Zeroize may also be performed using the Cryptographic Configuration panel.

Note: The Crypto must be deconfigured to change the Crypto type configuration.

Figure 6. System Management - Cryptographic Configuration

10. Select **EP11 Coprocessor** and press **OK**. This action makes the adapter to become a CEX4P adapter that is upgraded with the EP11 firmware. Also note, that TKE commands are always permitted for a CEX4P adapter, so that it can communicate with the TKE daemon *ep11TKEd*.
11. You must now select those LPARs that you want to allow to access and use the reconfigured adapter. For these LPARs, you need to configure back online the reconfigured adapter. Therefore, go to the dialog shown in Figure 4 on page 15, now toggling the status of the adapter for the LPAR back to online.
12. A restart of z/VM or the LPAR is required to activate the reconfiguration. For z/VM, check before, that the correct definitions have been applied to the EP11 coprocessor card. Also for the LPARs, on z/VM and on Linux, you must add the reconfigured adapter to the activation profile. Now deactivate and activate the LPAR. Then perform an IPL of Linux on that LPAR, respectively perform an IPL of z/VM and then start the guests using the reconfigured adapter.
13. Optionally, you can use the **`chzcrypt`** command to enable (online state) and disable (offline state) the IBM crypto adapter:

    ```
    $ chzcrypt -e 0x06    // set card06 online
    $ chzcrypt -d 0x06    // set card06 offline
    ```

    For more information about the IBM crypto adapter, see *Device Drivers, Features, and Commands*, SC33-8411 available at

    `www.ibm.com/developerworks/linux/linux390/documentation_dev.html`

Results

Now that the EP11 firmware has been enabled on your CEX4S Crypto Express card, this card turned into a so called CEX4P coprocessor which can take advantage of the Linux on System z EP11 enablement. To check the capability of a configured adapter, you can use the following **`lszcrypt -c <card-number>`** command:

```
$ lszcrypt -c 03
card03 provides capability for:
EP11 Secure Key
```

Assigning EP11 adapters as dedicated adapters to z/VM guests

On a z/VM guest, you can authorize the user to define virtual cryptographic facilities and provide the guest access to the AP queues on the PCI cryptographic cards. You achieve this with the help of the CRYPTO directory statement using the **`DOMAIN`** and **`APDEDicated`** operands.

The **`DOMAIN`** operand specifies up to 16 domains the virtual machine may use. The **`APDEDicated`** operand specifies up to 64 APs the virtual machine may use for dedicated access to the Adjunct Processor (AP) cryptographic facility. You can specify as many CRYPTO statements as you need to assign domains or APs to the virtual machine.

You can use the z/VM CP command **`QUERY CRYPto DOMains`** to request the display of the status of the cryptographic hardware and of installed AP domains.

Note: The **`CRYPTO APVIRTual`** directory statement cannot be used with the EP11 enablement.

For more information, see also *z/VM CP Planning and Administration* and *z/VM CP Commands and Utilities Reference* from the IBM Knowledge Center.

Restriction to extended evaluations

You can configure an CEX4P adapter to support extended evaluations to meet public sector requirements with regard to both FIPS and Common Criteria certifications. For more details, see the EP11 library structure documentation (by Tamas Visegrady).

The EP11 token of openCryptoki only supports the functions and mechanisms that are available on the adapter if the extended evaluations are configured. The available mechanisms and their attributes are reflected by the openCryptoki functions `C_GetMechanismList` and `C_GetMechaninfo`. However, there is one restriction on RSA mechanisms that cannot be reflected in the result of `C_GetMechanismInfo`: The `CKA_PUBLIC_EXPONENT` must have a value of at least 17.

Loading the Linux zcrypt device driver

You also need an installed Linux kernel that includes the zcrypt device driver with the EP11 extension.

About this task

With the EP11 enablement, the zcrypt device driver is no longer monolithic as in older distributions where the module was called **`z90crypt`**. The device driver is now loaded as separate modules, where the main module is called **`ap`**. There is, however, an alias name z90crypt that links to the ap main module.

Procedure

1. Check whether you loaded the current zcrypt device driver. To check, enter the **`lszcrypt`** command (see also "Enabling the CEX4S adapter for EP11 firmware exploitation" on page 13).
2. If the following error message is displayed, the device driver is not yet loaded:
   ```
   error - cryptographic device driver zcrypt is not loaded!
   ```
   To load the device driver ap main module, use the following command:
   `modprobe ap`

   See your Linux distribution documentation for how to load the module persistently.

Results

The zcrypt device driver that contains the EP11 extension is loaded.

Installing the host part of the EP11 library

Read the contained information about how to install the host part of the EP11 library as a component of the EP11 stack.

About this task

As a part of the EP11 stack, you need to install the host part of the EP11 library on your System z, as shown in Figure 1 on page 6.

Also, to use the EP11 functionality, the TKE daemon (*ep11TKEd*) must be available and running to perform certain communication tasks. This communication path is necessary, for example, for the initial key personalization or for key updates (see also "Setting a master key on the Crypto Express4S EP11 coprocessor").

Procedure

1. Obtain the RPM packages that contain the Linux on System z EP11 enablement from the IBM website:

   `http://www.ibm.com/security/cryptocards/pciecc/zlinuxsoftware.shtml`

   The names of the RPM packages are as follows:
   - `ep11-host-2.6-0release.s390x.rpm` contains the *ep11TKEd* and the `libep11.so`
   - `ep11-host-devel-2.6-0release.s390x.rpm` contains `ep11.h` (which are the API definitions for standard crypto functions) and `ep11adm.h` (which are the admin API definitions)

   Note: The host part of the EP11 library is developed and maintained by IBM and therefore not part of any commercial Linux distribution.

   .
2. Install the RPM. Use the command:
   ```
   rpm -ivh <rpm_packet>
   ```
3. The EP11 TKE daemon (`ep11TKEd`), which comes along with the RPM packages obtained in step 1 is also installed during RPM installation. It is required and must be running for handling administrative commands and for managing communication between the TKE workstation and the Crypto Express4S EP11 coprocessor.

Setting a master key on the Crypto Express4S EP11 coprocessor

To generate a secure master key, use the TKE workstation that is connected to the System z mainframe. Note that the master key is referred to as wrapping key in the respective documentation.

Note: This publication outlines a selection of the basic steps for creating and initializing EP11 smart cards and for generating a master key. It does not document the complete process of setting up a comprehensive security concept, nor does it demonstrate all security features available from the TKE workstation. For information about sophisticated features, for example, for a dual control security policy, for the zone concept, or for using TKE domain groups, refer to the *Trusted Key Entry Workstation User's Guide* from the IBM Resource Link.

Trusted Key Entry (TKE) is a priced optional feature that is used for managing System z secure coprocessors in a customer environment. Secure coprocessors operate with a master key that is located inside the coprocessor itself. These secure coprocessors use keys that are protected by being encrypted (wrapped) with the master key. These wrapped keys are called secure keys and are only decrypted inside the coprocessor's secure enclosure.

Information is provided in the following topics:
- "Setting up the TKE environment" on page 20
- "Create and initialize an EP11 smart card" on page 20
- "Creating a master key on the TKE workstation" on page 24

For more information about these tasks, refer to topics *Using the Crypto Module Notebook to administer EP11 crypto modules* and *Smart Card Utility Program (SCUP)* in the *Trusted Key Entry Workstation User's Guide* from the IBM Resource Link.

Setting up the TKE environment

For a Crypto Express4S EP11 coprocessor, a TKE workstation is required to perform certain key management functions.

A TKE version 7.3 is required to detect EP11 adapters and set and manage wrapping keys (master keys) correctly.

Note: For any master key transactions to the card (key generation or import) and for initialization/personalization purposes, you need at least two smart card readers. Furthermore, the described outline uses one CA (Certificate Authority) smart card and two smart cards that hold two separate key parts which make up the master key. The smart cards can be initialized from scratch by using the TKE interfaces.

To use the EP11 functions of the TKE, the EP11 library (*libep11.so*) and the TKE daemon (*ep11TKEd*) to handle administrative commands between the TKE and the crypto adapter must be installed.

To start the daemon, use the command

```
service ep11TKEd start
```

This command is redirected and performs the same processing as the command

```
systemctl start ep11TKEd.service
```

The *ep11TKEd* TKE daemon is listening on port 50004 for administrative TKE commands. These commands are translated into **ioctl** commands to talk to the zcrypt device driver.

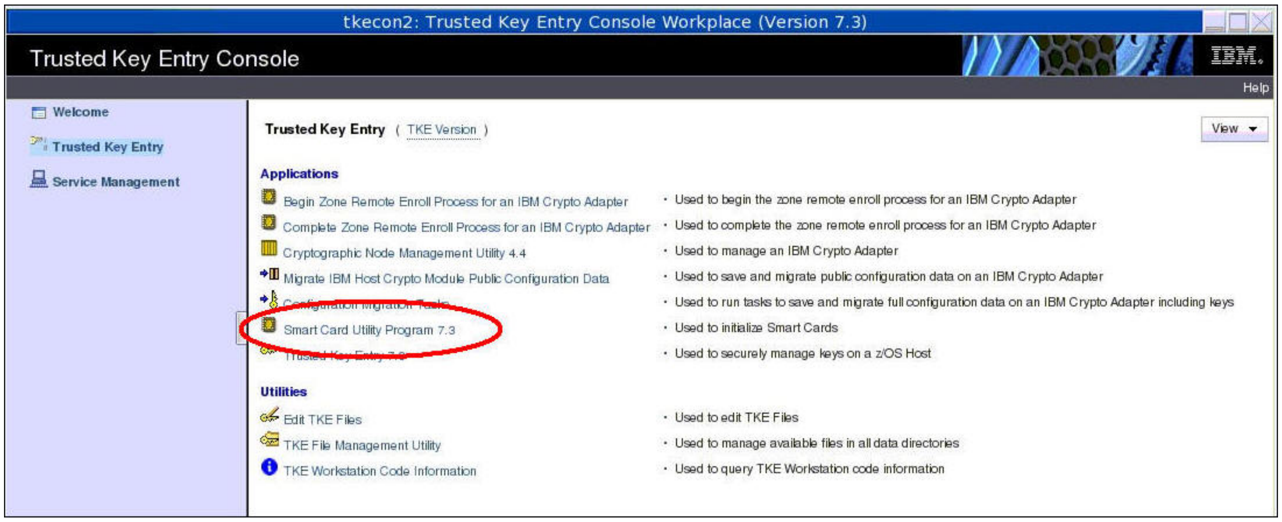
Create and initialize an EP11 smart card

Step 1

As a prerequisite, you need a valid CA (Certificate Authority) smart card to be authorized to create EP11 smart cards (see **Step 4**).

The **Trusted Key Entry** console automatically loads on start-up with a set of commonly used tasks. After the TKE console started, the initial **Trusted Key Entry Console** window appears.

This initial window provides access to applications and utilities available on the TKE workstation.



tkecon2: Trusted Key Entry Console Workplace (Version 7.3)

Trusted Key Entry Console

Help

- Welcome
- Trusted Key Entry
- Service Management

Trusted Key Entry (TKE Version)

View

Applications

Begin Zone Remote Enroll Process for an IBM Crypto Adapter	Used to begin the zone remote enroll process for an IBM Crypto Adapter
Complete Zone Remote Enroll Process for an IBM Crypto Adapter	Used to complete the zone remote enroll process for an IBM Crypto Adapter
Cryptographic Node Management Utility 4.4	Used to manage an IBM Crypto Adapter
Migrate IBM Host Crypto Module Public Configuration Data	Used to save and migrate public configuration data on an IBM Crypto Adapter
Configuration Migration Tasks	Used to run tasks to save and migrate full configuration data on an IBM Crypto Adapter including keys
Smart Card Utility Program 7.3	Used to initialize Smart Cards
Trusted Key Entry 7.3	Used to securely manage keys on a z/OS Host

Utilities

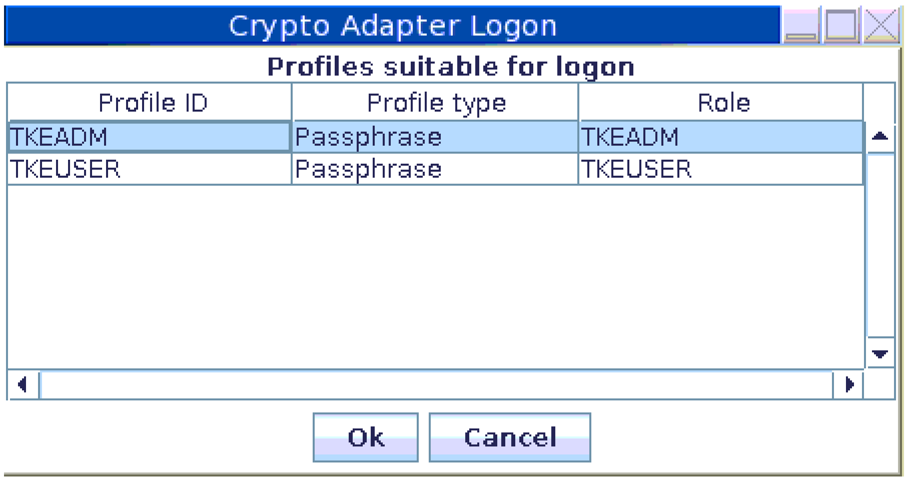
Edit TKE Files	Used to edit TKE Files
TKE File Management Utility	Used to manage available files in all data directories
TKE Workstation Code Information	Used to query TKE Workstation code information

Figure 7. TKE Console - initial window

Step 2

Click the **Smart Card Utility Program** application as shown in Figure 7.

When you open a TKE application or utility, you must sign on with a profile that is on the TKE workstation crypto adapter. Therefore, depending on how you have initialized your environment, the **Crypto Adapter Logon** window is displayed with profile IDs that represent a single or group passphrase logon. The individual or group profile you choose must have enough authority to do the functions that are performed by the application or utility. The steps described here use the default TKEADM user name.



Crypto Adapter Logon

Profiles suitable for logon

Profile ID	Profile type	Role
TKEADM	Passphrase	TKEADM
TKEUSER	Passphrase	TKEUSER

Figure 8. Crypto Adapter Logon

Step 3

After a successful log-on, the **Smart Card Utility Program** opens and shows a table for each smart card reader for all detected plugged-in smart card types. The tables are still empty at this point in time, because the EP11 smart card is not yet created. The missing information is provided during the process of initializing and personalizing the smart card as described in the remainder of this topic.

To continue, select **Initialize and enroll EP11 smart card** from the **EP11 Smart Card** pulldown choice.

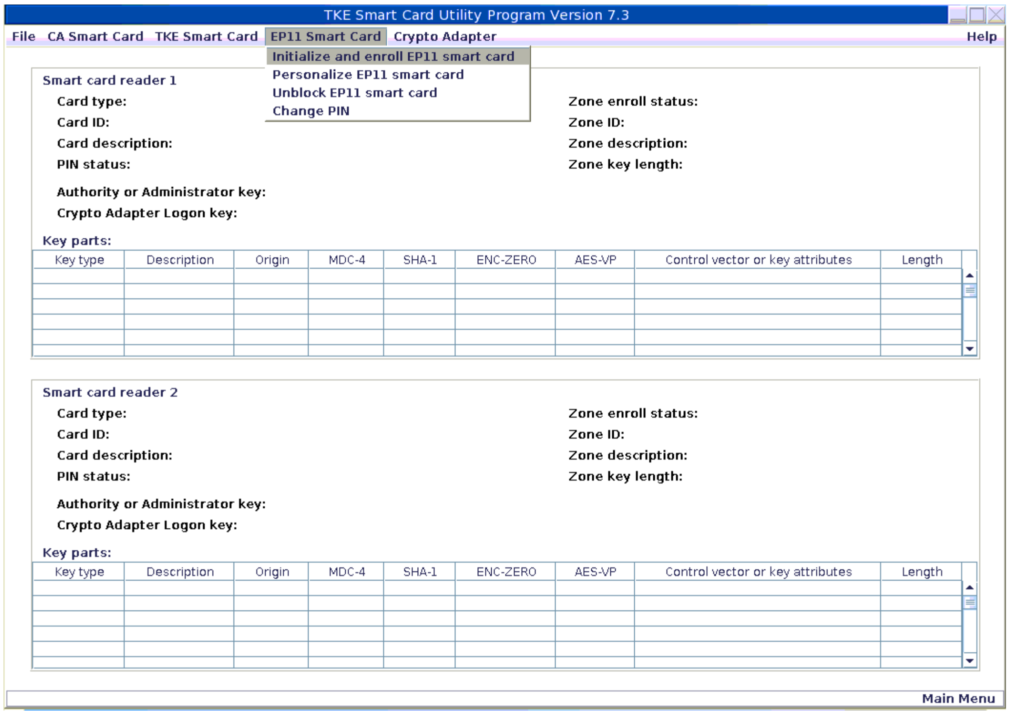
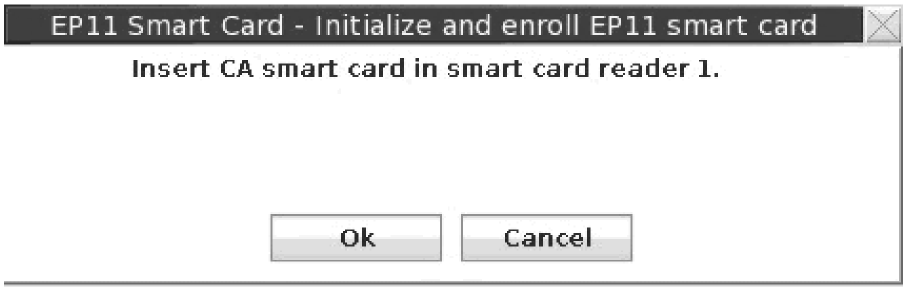

TKE Smart Card Utility Program Version 7.3
File CA Smart Card TKE Smart Card EP11 Smart Card Crypto Adapter Help
Initialize and enroll EP11 smart card
Personalize EP11 smart card
Unblock EP11 smart card
Change PIN
Smart card reader 1
Card type:
Card ID:
Card description:
PIN status:
Zone enroll status:
Zone ID:
Zone description:
Zone key length:
Authority or Administrator key:
Crypto Adapter Logon key:
Key parts:
Key type Description Origin MDC-4 SHA-1 ENC-ZERO AES-VP Control vector or key attributes Length
Smart card reader 2
Main Menu

Figure 9. Initialize and enroll EP11 smart card

Step 4

The **Smart Card Utility Program** prompts you to insert the CA smart card into the smart card reader 1 and then press the **OK** button. For detailed information, read the TKE documentation.

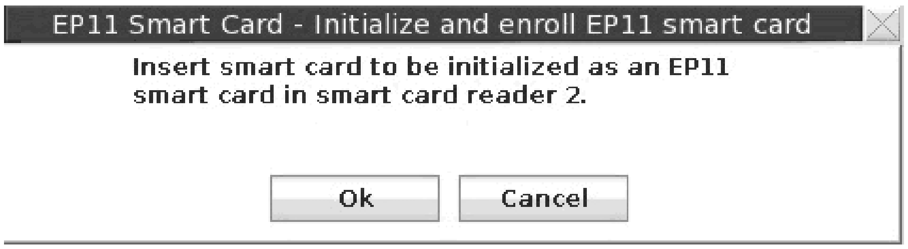

EP11 Smart Card - Initialize and enroll EP11 smart card

Insert CA smart card in smart card reader 1.

Figure 10. Insert CA smart card

Step 5

As next step, the **Smart Card Utility Program** prompts you to insert a smart card to be initialized as an EP11 smart card into smart card reader 2 and then press the **OK** button.

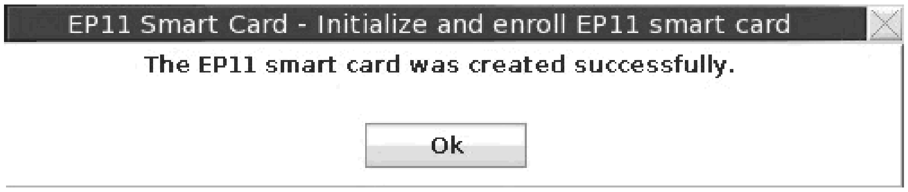

EP11 Smart Card - Initialize and enroll EP11 smart card

Insert smart card to be initialized as an EP11
smart card in smart card reader 2.

Figure 11. Insert smart card to be initialized as an EP11 smart card

Step 6

The **Smart Card Utility Program** creates the EP11 smart card and displays a message when the creation was successful. In this case, press the **OK** button.


EP11 Smart Card - Initialize and enroll EP11 smart card

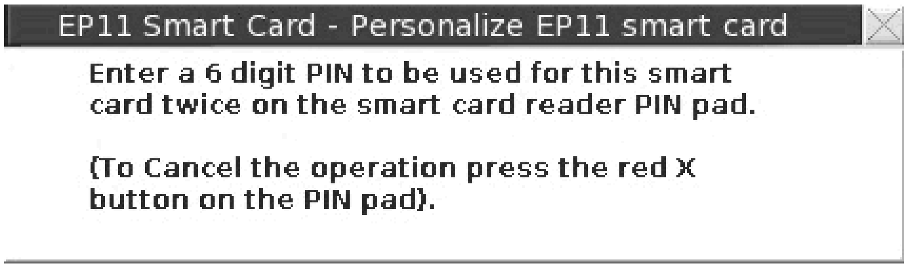
The EP11 smart card was created successfully.

Figure 12. EP11 smart card successfully created

Step 7

The **Smart Card Utility Program** now goes back to the window shown in Figure 9 on page 22, where you now select item **Personalize EP11 smart card** from the **EP11 Smart Card** pull-down choice.

To personalize the EP11 smart card, the **Smart Card Utility Program** prompts you to enter a PIN to be used for this smart card on the smart card reader PIN pad. The PIN must be entered twice for confirmation.

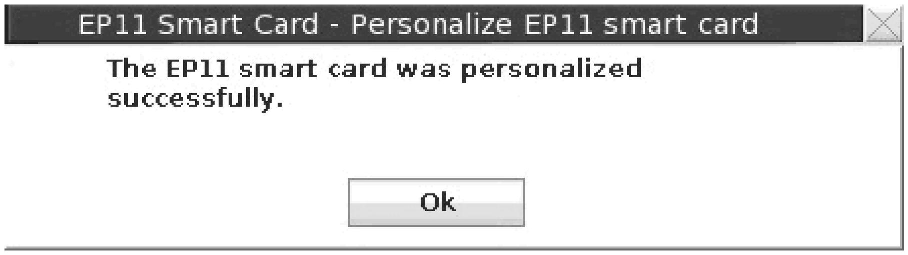

EP11 Smart Card - Personalize EP11 smart card

Enter a 6 digit PIN to be used for this smart
card twice on the smart card reader PIN pad.

(To Cancel the operation press the red X
button on the PIN pad).

Figure 13. Entering a PIN for the EP11 smart card

The **Smart Card Utility Program** informs you of a successful personalization of the EP11 smart card. This smart card now contains a certificate signed by the CA authority, and a PIN to access the smart card.



The EP11 smart card was personalized successfully.

Figure 14. EP11 smart card successfully personalized

Available smart card information is shown next time in the table shown in Figure 9 on page 22 next time you log on.

The EP11 smart card is needed whenever you want to set a new master key on the adapter.

Step 8

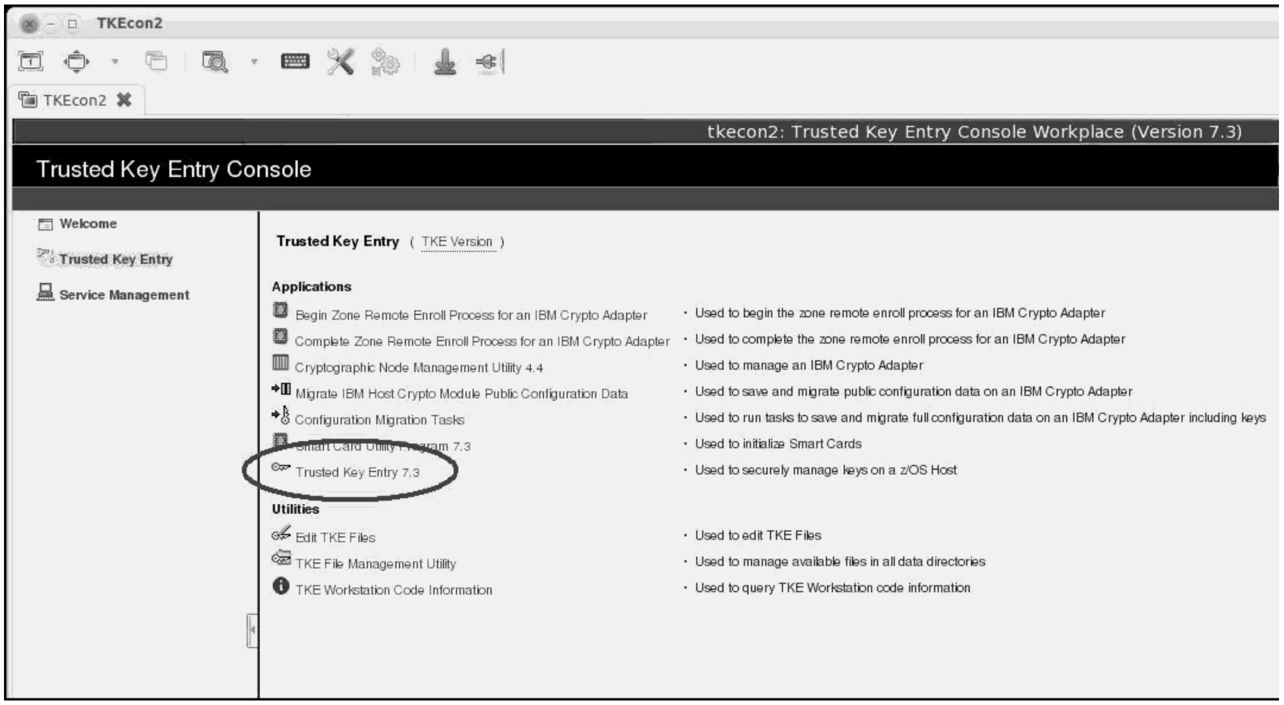
Repeat **Step 3** through **Step 7** to create the second EP11 smart card.

Creating a master key on the TKE workstation

Read an outline of the required steps for creating a master key and installing it on the CEX4P adapter. For detailed information about how to use the TKE workstation, refer to *z/OS Cryptographic Services ICSF TKE Workstation User's Guide, SA23-2211*.

Step 1

Go to the **Trusted Key Entry** console as described in Step 1 of "Create and initialize an EP11 smart card" on page 20.


TKEcon2

tkecon2: Trusted Key Entry Console Workplace (Version 7.3)

Trusted Key Entry Console

- Welcome
- Trusted Key Entry
- Service Management

Trusted Key Entry (TKE Version)

Applications

Begin Zone Remote Enroll Process for an IBM Crypto Adapter	• Used to begin the zone remote enroll process for an IBM Crypto Adapter
Complete Zone Remote Enroll Process for an IBM Crypto Adapter	• Used to complete the zone remote enroll process for an IBM Crypto Adapter
Cryptographic Node Management Utility 4.4	• Used to manage an IBM Crypto Adapter
Migrate IBM Host Crypto Module Public Configuration Data	• Used to save and migrate public configuration data on an IBM Crypto Adapter
Configuration Migration Tasks	• Used to run tasks to save and migrate full configuration data on an IBM Crypto Adapter including keys
Smart Card Utility Program 7.3	• Used to initialize Smart Cards
Trusted Key Entry 7.3	• Used to securely manage keys on a z/OS Host

Utilities

Edit TKE Files	• Used to edit TKE Files
TKE File Management Utility	• Used to manage available files in all data directories
TKE Workstation Code Information	• Used to query TKE Workstation code information

Figure 15. TKE Console - initial window

Step 2

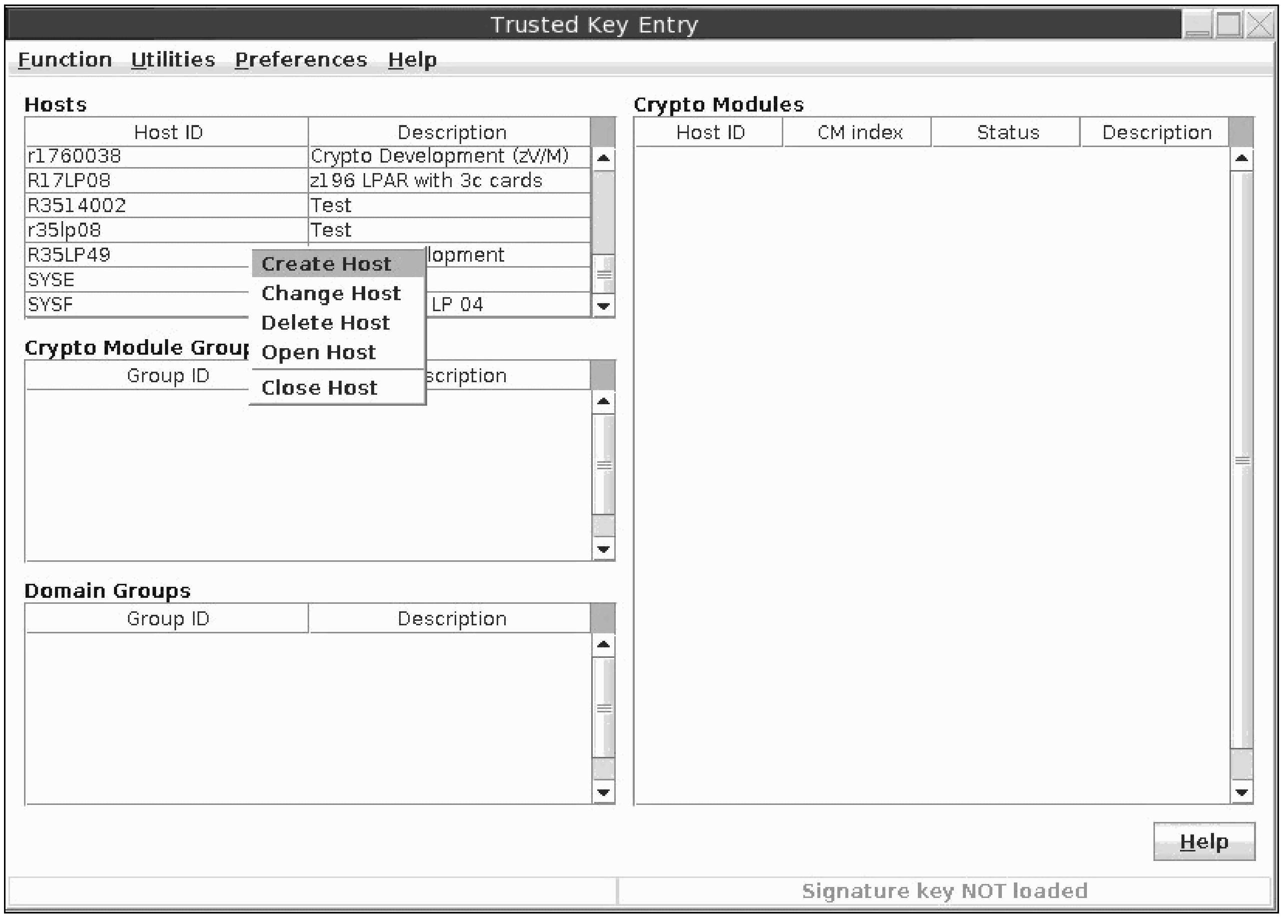
Click the **Trusted Key Entry** application as shown in Figure 15. Then proceed with the logon procedure as described in Step 2 of "Create and initialize an EP11 smart card" on page 20.

Step 3

Select the default profile ID TKEUSER, click **OK**, and in the upcoming **Passphrae Logon** dialog for this profile, logon with the passphrase associated to TKEUSER.

Step 4

The **Trusted Key Entry** main window is displayed (Figure 16 on page 26). Open the context menu for hosts and select action **Create Host**.



Host ID	Description
r1760038	Crypto Development (zV/M)
R17LP08	z196 LPAR with 3c cards
R3514002	Test
r35lp08	Test
R35LP49	lopment
SYSE	
SYSF	LP 04

Host ID	CM index	Status	Description

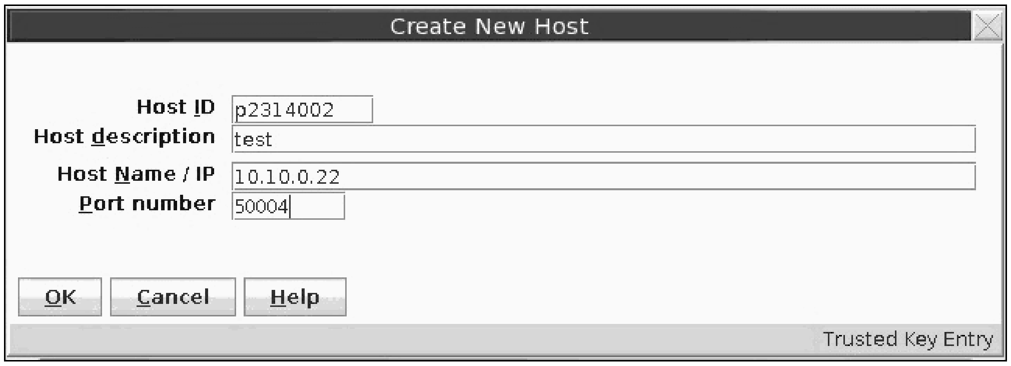
Group ID	scription

Group ID	Description

Figure 16. Trusted Key Entry - main window

Step 5

In the **Create New Host** dialog, enter the required values of the host for which you want to create the master key. It is assumed that this host is a Linux on System z system running the `ep11TKEd` TKE daemon. Press **OK** to return to the **Trusted Key Entry** main window.



Host ID	p2314002
Host description	test
Host Name / IP	10.10.0.22
Port number	50004

Figure 17. TKE - Create new Host

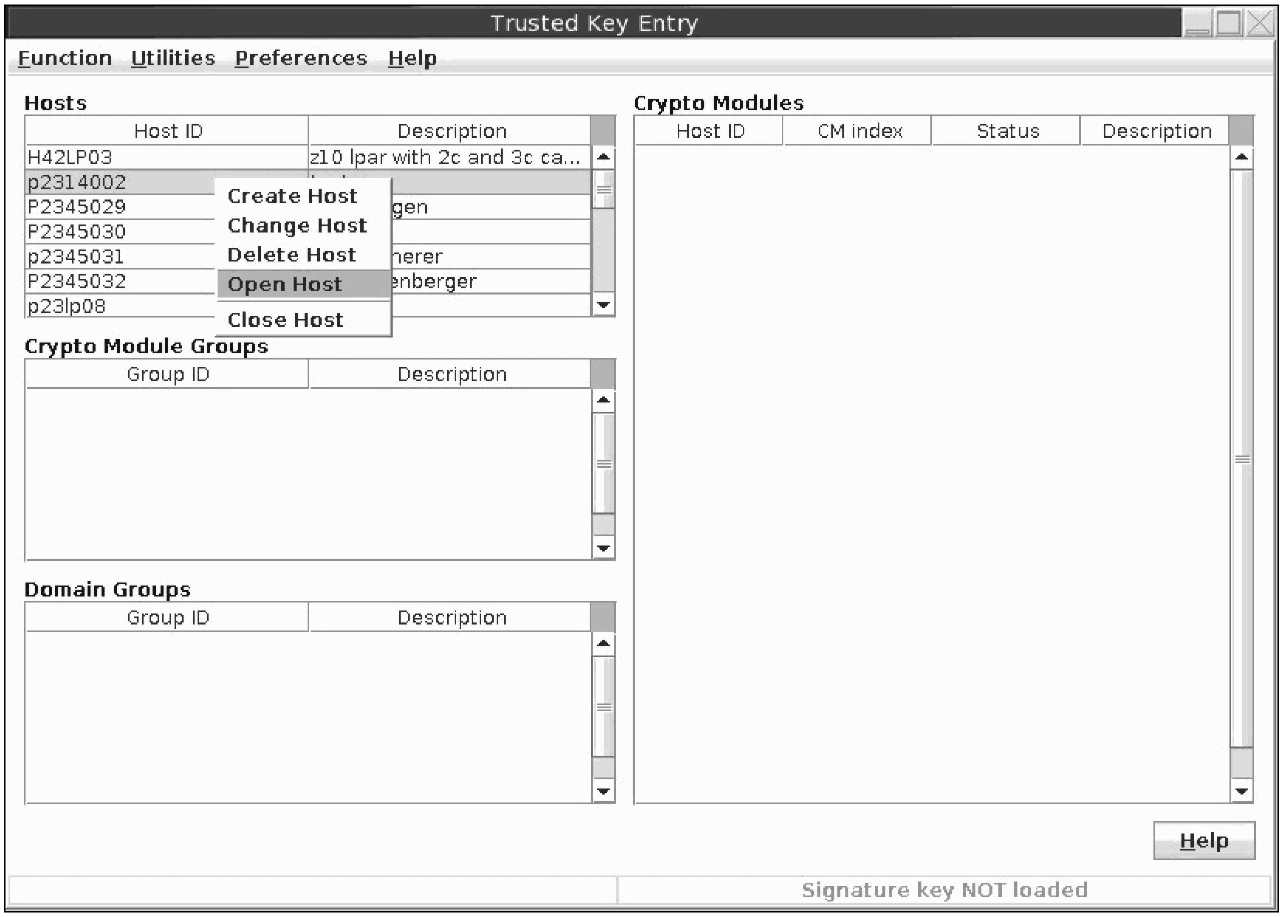
Step 6

The new host is visible now within the list of host IDs.

Before you continue to work on the new host, ensure the following:

- The `ep11TKEd` TKE daemon is started on the host.
- The TKE has connectivity to the host.

Then open the new host's context menu and select action **Open Host**.



Host ID	Description
H42LP03	z10 lpar with 2c and 3c ca...
p2314002	
P2345029	gen
P2345030	
p2345031	herer
P2345032	enberger
p23lp08	

Group ID	Description

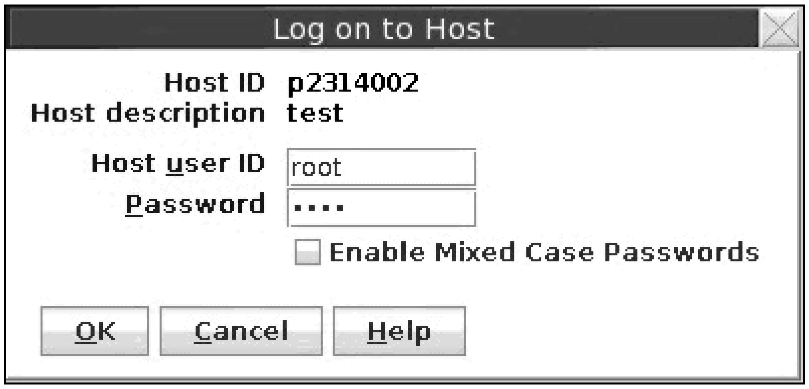
Group ID	Description

Host ID	CM index	Status	Description

Figure 18. Trusted Key Entry - main window with new created host

Step 7

When prompted by the TKE workstation, log on to the selected host with the appropriate credentials. The values that you enter as the user ID and password, are not relevant, because they are not validated. You just need to press the **OK** button.

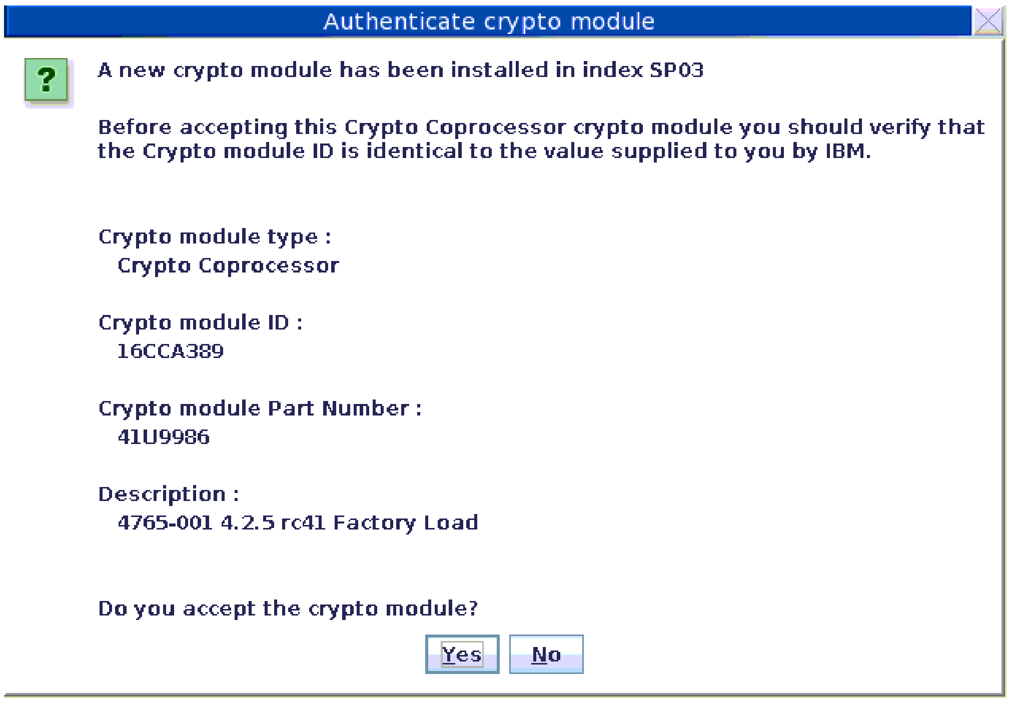

Log on to Host

Host ID p2314002
Host description test
Host user ID root
Password ••••
☐ Enable Mixed Case Passwords

Figure 19. Log on to new host

Step 8

The TKE now requests a verification of any new crypto adapter. Press the **Yes** button to continue.


Authenticate crypto module

A new crypto module has been installed in index SP03

Before accepting this Crypto Coprocessor crypto module you should verify that the Crypto module ID is identical to the value supplied to you by IBM.

Crypto module type :
Crypto Coprocessor

Crypto module ID :
16CCA389

Crypto module Part Number :
41U9986

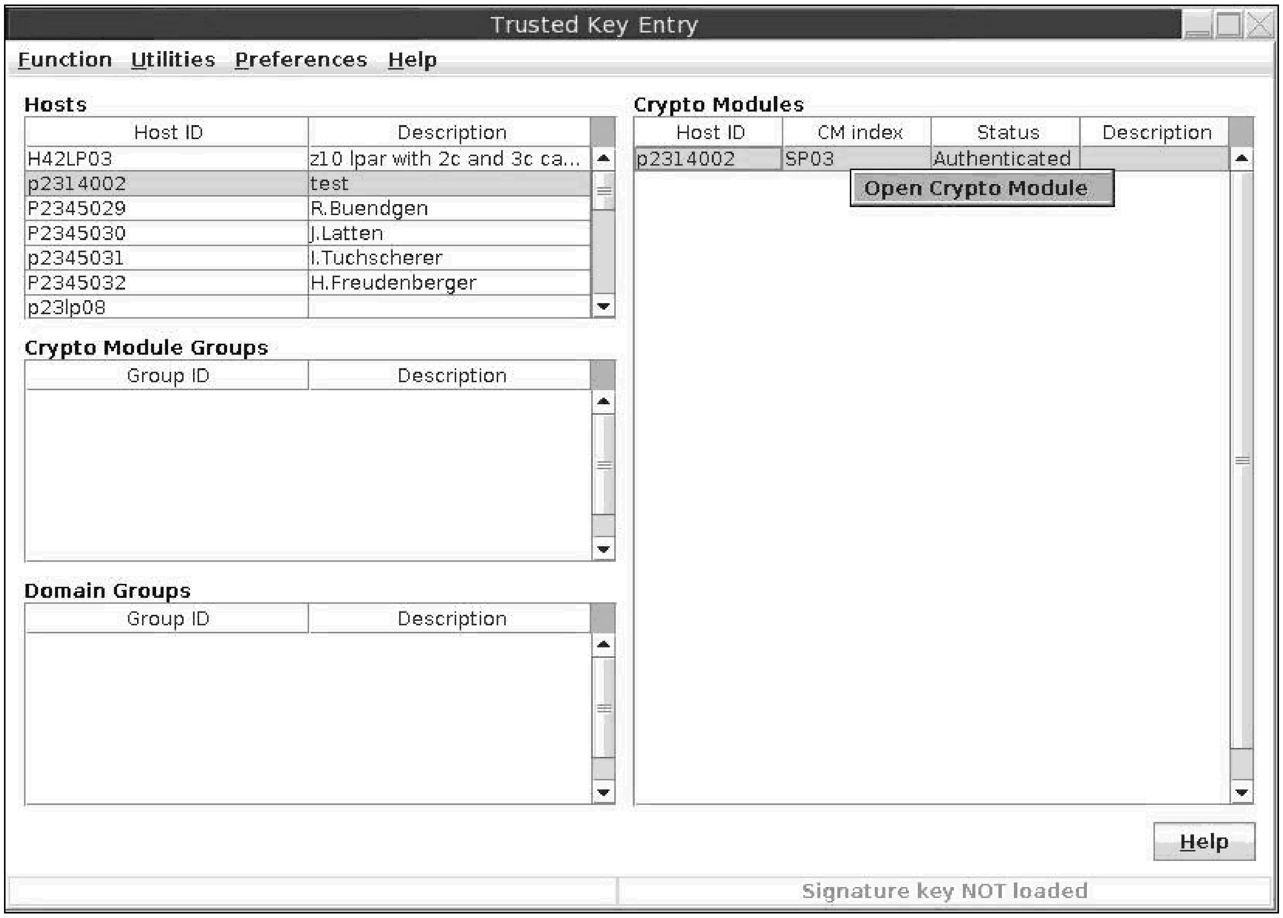
Description :
4765-001 4.2.5 rc41 Factory Load

Do you accept the crypto module?

Figure 20. Authenticate crypto module

Step 9

The **Crypto Modules** list now displays the available adapters. In the sample from Figure 21 on page 29, there is just one adapter available with host ID *p2314002*. Select a crypto adapter of your choice and trigger action **Open Crypto Module** from its context menu.



Hosts

Host ID	Description
H42LP03	z10 lpar with 2c and 3c ca...
p2314002	test
P2345029	R.Buendgen
P2345030	J.Latten
p2345031	I.Tuchscherer
P2345032	H.Freudenberger
p23lp08	

Crypto Module Groups

Group ID	Description

Domain Groups

Group ID	Description

Crypto Modules

Host ID	CM index	Status	Description
p2314002	SP03	Authenticated	

Signature key NOT loaded

Figure 21. Crypto Modules list

Step 10

The **Crypto Module Administration** window for the selected crypto adapter opens. Now you can start to configure the domains. Click on the **Domains** tab at the top. On the right side, the window now shows an **Index** tab for each available domain. Choose one of these indexes and select the **Domain Administrators** tab at the bottom of the window to add a new administrator role. In this documentation, the configuration is outlined for the domain with index *13*. For detailed information on domain configuration, refer to the TKE documentation.

Step 11

Now create a user ID with administrator role in the **Crypto Module Administration** window for the selected crypto adapter. Open the context menu by right-clicking into the white space of the window. Select action **Add Administrator**.

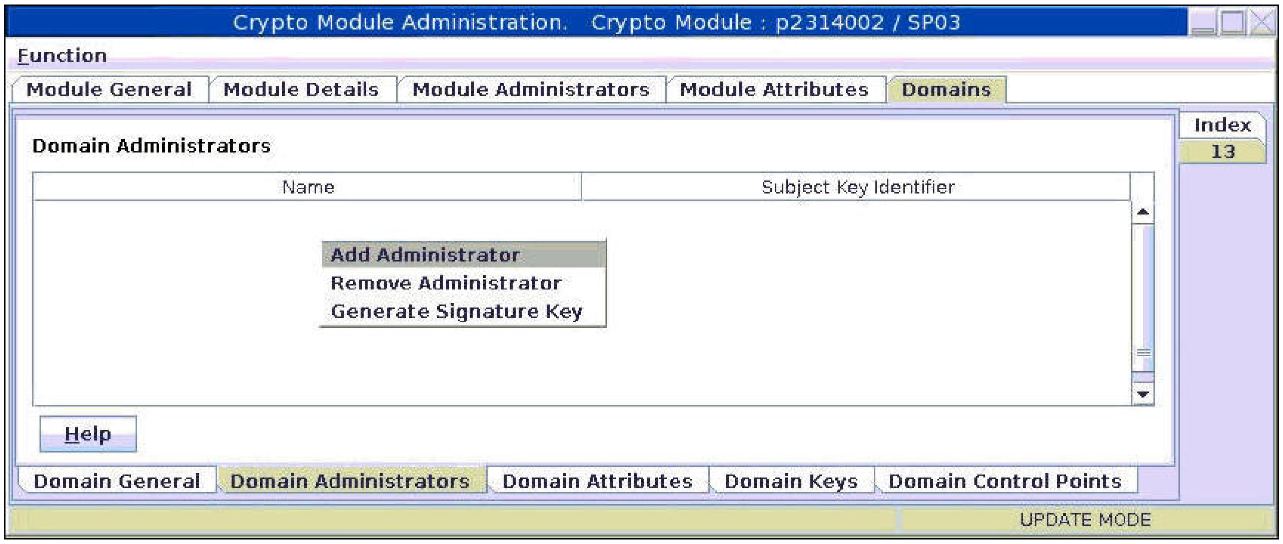


Figure 22. Crypto Module Administration - with context menu

From the opening **Select Source** window, TKE requests certain information from the previously CA prepared smart card that contains the administrator key and certificate.

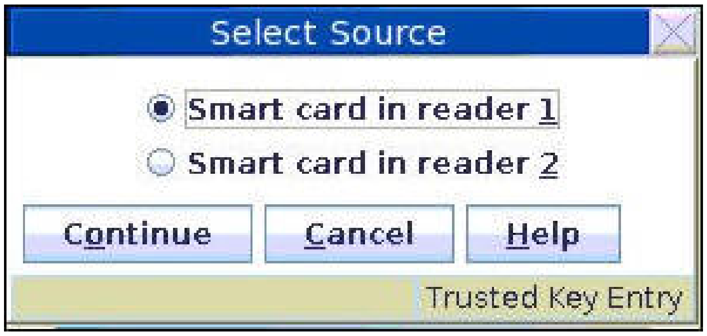
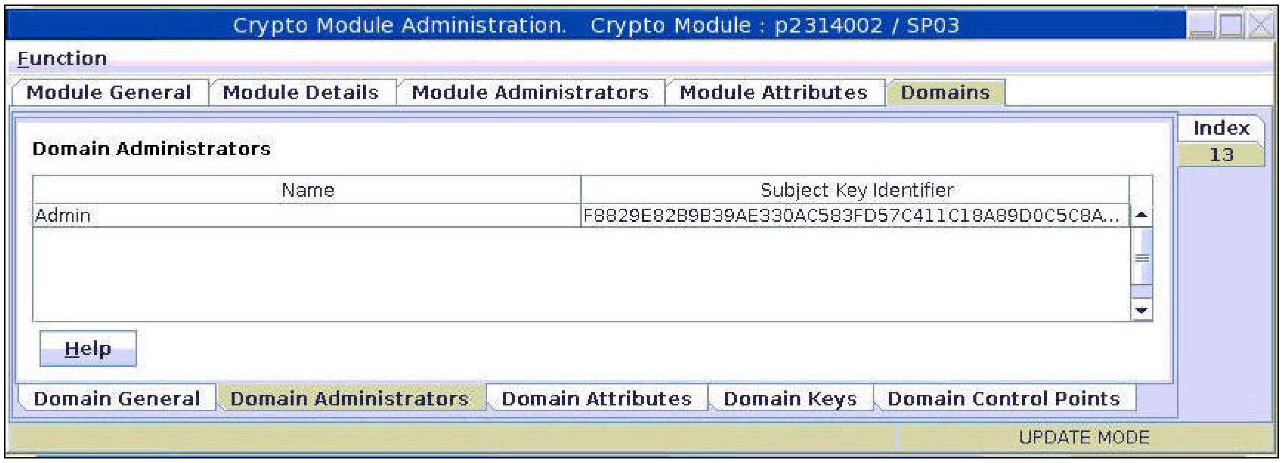


Figure 23. Select Source

After a successful authentication on the smart card reader, the TKE workstation imports the administrator key and certificate and creates an administrator profile.


Crypto Module Administration. Crypto Module : p2314002 / SP03

Function

Module General | Module Details | Module Administrators | Module Attributes | Domains

Index: 13

Domain Administrators

Name	Subject Key Identifier
Admin	F8829E82B9B39AE330AC583FD57C411C18A89D0C5C8A...

Help

Domain General | Domain Administrators | Domain Attributes | Domain Keys | Domain Control Points

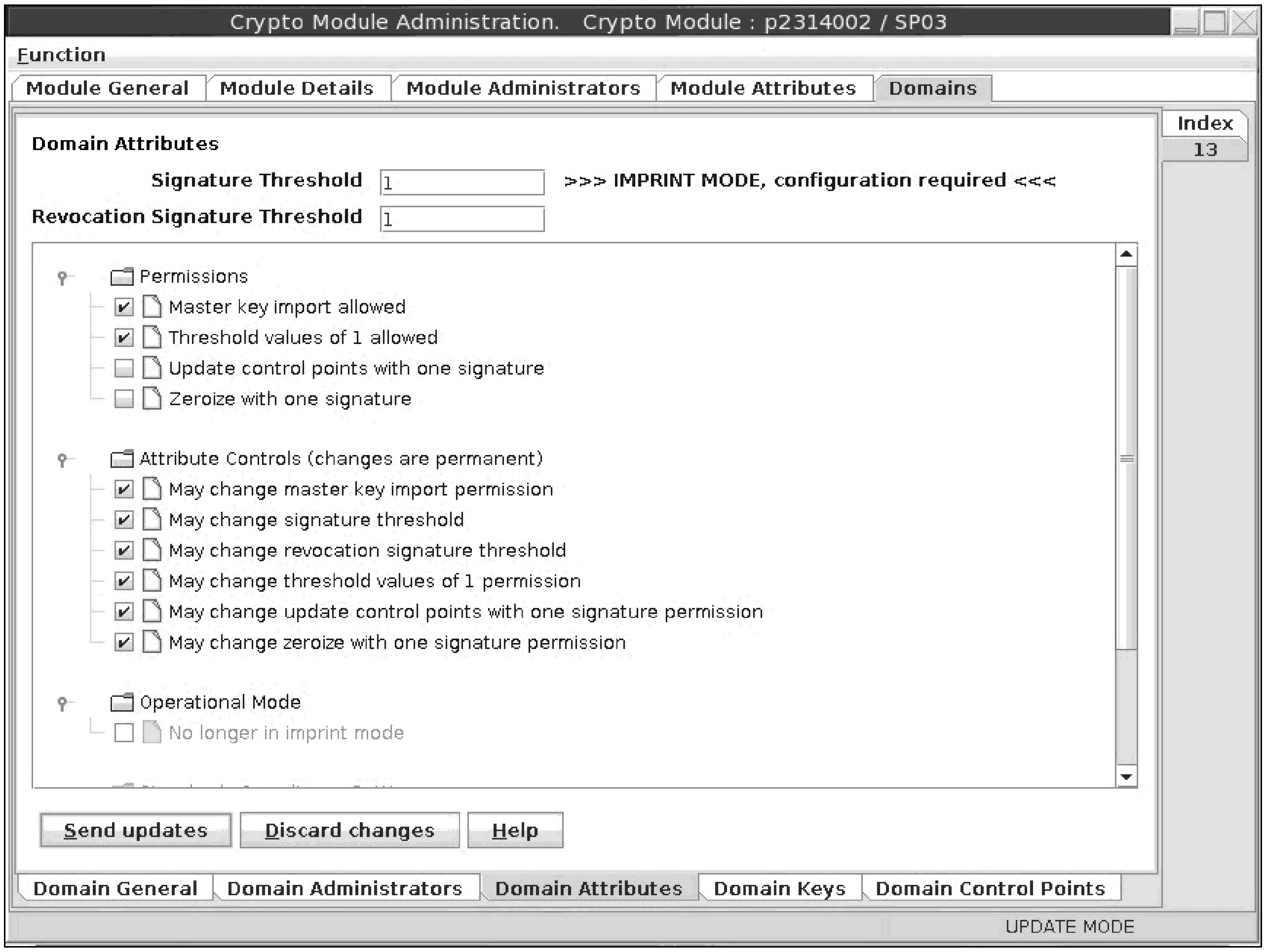
UPDATE MODE

Figure 24. Crypto Module Administration - Subject Key Identifier

Step 12

Now select the **Domain Attributes** tab at the bottom of the window. This selection opens the window that is shown in Figure 25 on page 32 where you can specify the required permissions and attribute controls for the current domain.

Per default, the *Signature Threshold* and the *Revocation Signature Threshold* are set to 0. Both values must be changed at least to 1 to release the card from the IMPRINT mode. For more information, see the TKE documentation. Press **Send updates** to apply your settings.


Crypto Module Administration. Crypto Module : p2314002 / SP03

Domain Attributes

Signature Threshold 1 >>> IMPRINT MODE, configuration required <<<

Revocation Signature Threshold 1

Permissions
- [✓] Master key import allowed
- [✓] Threshold values of 1 allowed
- [ ] Update control points with one signature
- [ ] Zeroize with one signature

Attribute Controls (changes are permanent)
- [✓] May change master key import permission
- [✓] May change signature threshold
- [✓] May change revocation signature threshold
- [✓] May change threshold values of 1 permission
- [✓] May change update control points with one signature permission
- [✓] May change zeroize with one signature permission

Operational Mode
- [ ] No longer in imprint mode

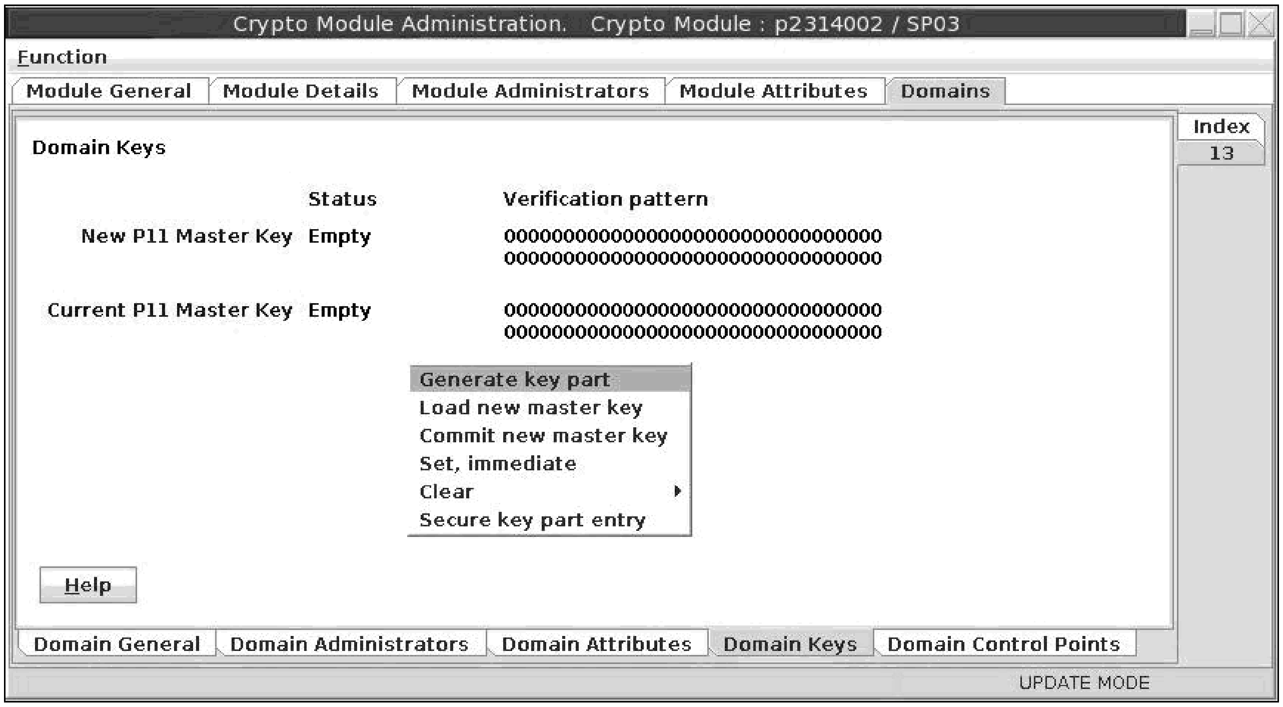
Figure 25. Crypto Module Administration - Setting permissions and attribute controls

Step 13

Now select the **Domain Keys** tab from the bottom of the **Crypto Module Administration** window.

The new **Crypto Module Administration** window with verification patterns for the new and current master key is displayed. The patterns are all set to 0, because the current and new master keys are empty yet.

Open the context menu by right-clicking in the white space, and select action **Generate key part**.



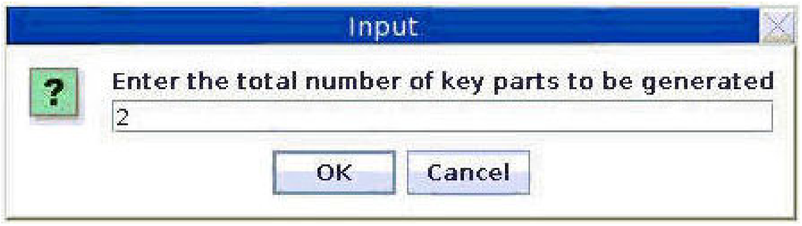
Domain Keys

	Status	Verification pattern
New P11 Master Key	Empty	00000000000000000000000000000000 00000000000000000000000000000000
Current P11 Master Key	Empty	00000000000000000000000000000000 00000000000000000000000000000000

Figure 26. Crypto Module Administration - Generate key part

Step 14

The TKE workstation now prompts you to enter the total number of key parts to be generated. You must at least generate two parts. Enter your input and press the **OK** button.



Enter the total number of key parts to be generated

2

Figure 27. Crypto Module Administration - Input for total number of key parts to be generated

Step 15

In a similar way as in the previous step, you are now prompted to insert an EP11 smart card and to enter a name and description for each generated key part. The TKE workstation informs you about a successful storage of all generated key parts and descriptions. The new master key can now be generated by the TKE component.

Step 16

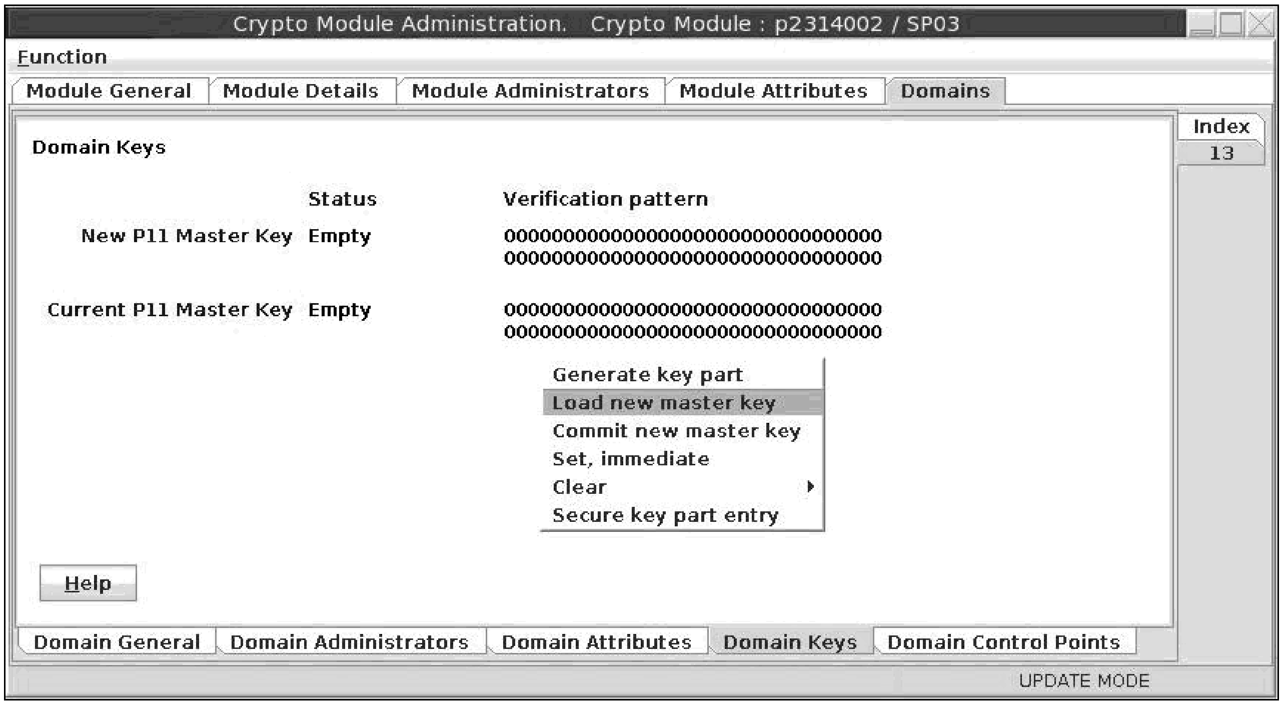
During the current process, the new master key now needs to go through three stages:

Load The key is just stored on the adapter, but not active.

Commit
The key is activated and is now present on the adapter as the new master key. In this state, the existing objects encrypted under the current master key can be re-encrypted by using this new master key.

Set The new master key is now switched to become the current key to be used.

Start with the **Load** step: Load the new generated master key parts from the cards to the target crypto adapter. For this purpose, open the context menu from the **Crypto Module Administration** window and select action **Load new master key**. TKE now prompts you for the total number of key parts to be loaded. Type the number of previously generated key parts. TKE then prompts you to load each key part separately.


Crypto Module Administration. Crypto Module : p2314002 / SP03

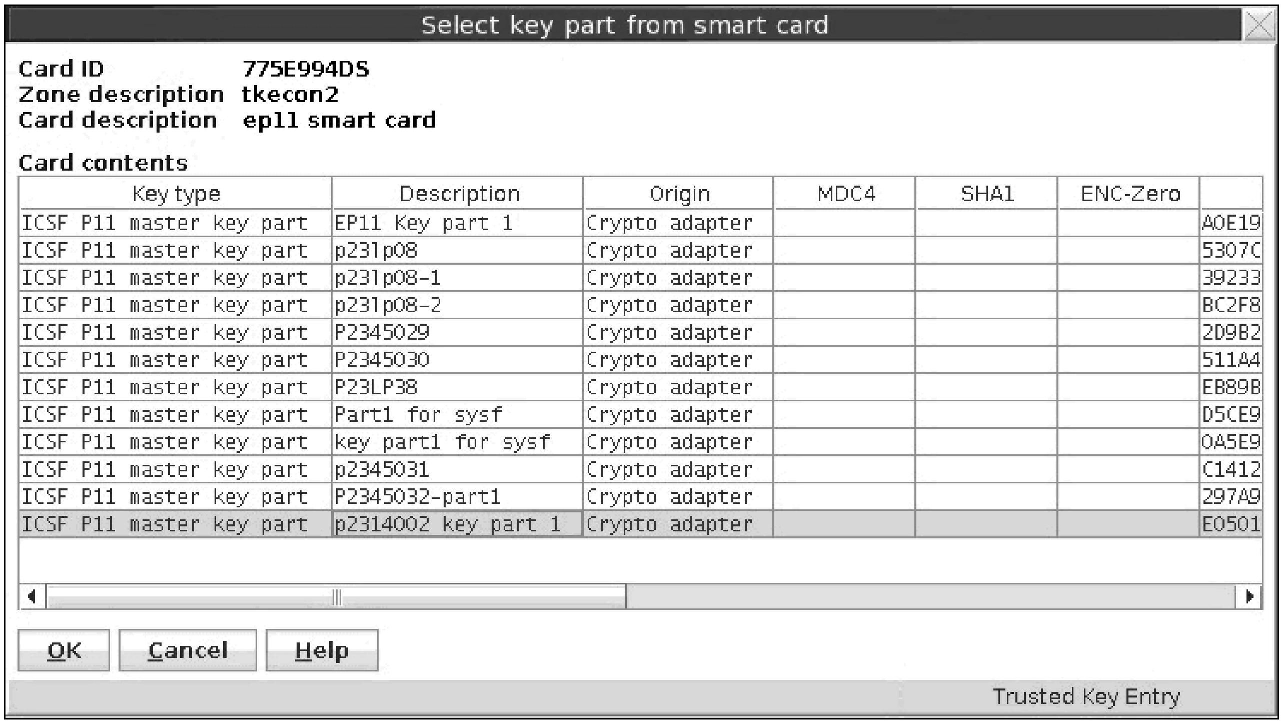
Domain Keys

```
                         Status     Verification pattern
     New P11 Master Key  Empty      00000000000000000000000000000000
                                    00000000000000000000000000000000

 Current P11 Master Key  Empty      00000000000000000000000000000000
                                    00000000000000000000000000000000
```

Figure 28. Crypto Module Administration - Load new master key

The TKE workstation opens the window **Select key part from smart card** as shown in Figure 29 on page 35. From this window, you can commit the single parts of your key. From the list of shown key parts, select that part that you now want to commit and press **OK**.



Select key part from smart card

Card ID 775E994DS
Zone description tkecon2
Card description ep11 smart card

Card contents

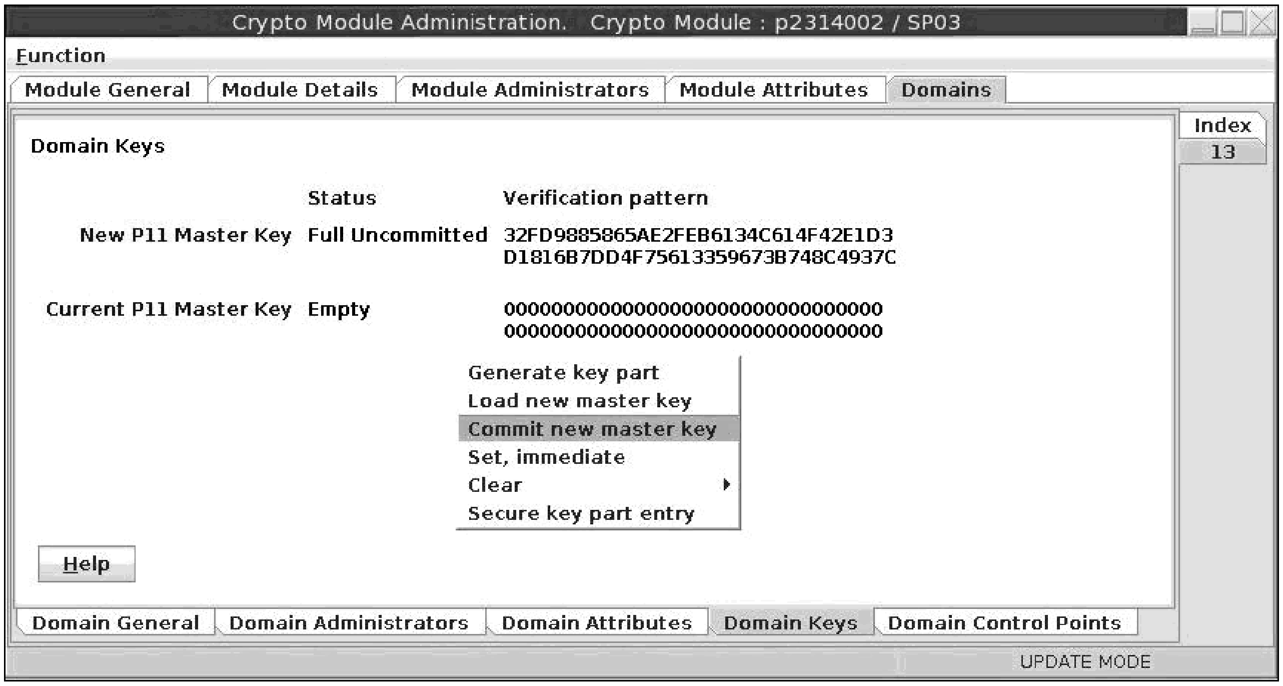
Key type	Description	Origin	MDC4	SHA1	ENC-Zero	
ICSF P11 master key part	EP11 Key part 1	Crypto adapter				A0E19
ICSF P11 master key part	p231p08	Crypto adapter				5307C
ICSF P11 master key part	p231p08-1	Crypto adapter				39233
ICSF P11 master key part	p231p08-2	Crypto adapter				BC2F8
ICSF P11 master key part	P2345029	Crypto adapter				2D9B2
ICSF P11 master key part	P2345030	Crypto adapter				511A4
ICSF P11 master key part	P23LP38	Crypto adapter				EB89B
ICSF P11 master key part	Part1 for sysf	Crypto adapter				D5CE9
ICSF P11 master key part	key part1 for sysf	Crypto adapter				0A5E9
ICSF P11 master key part	p2345031	Crypto adapter				C1412
ICSF P11 master key part	P2345032-part1	Crypto adapter				297A9
ICSF P11 master key part	p2314002 key part 1	Crypto adapter				E0501

Figure 29. Select key part from smart card

Step 17

After you loaded all single master key parts, the complete master key is successfully loaded onto the CEX4P adapter.

The TKE workstation switches back to the **Crypto Module Administration** window. You can see that the new master key is full/complete, but yet uncommitted. To commit the new master key, invoke the context menu and select action **Commit new master key**. The status switches to *Full Committed*, as shown in Figure 31 on page 36.


Crypto Module Administration. Crypto Module : p2314002 / SP03

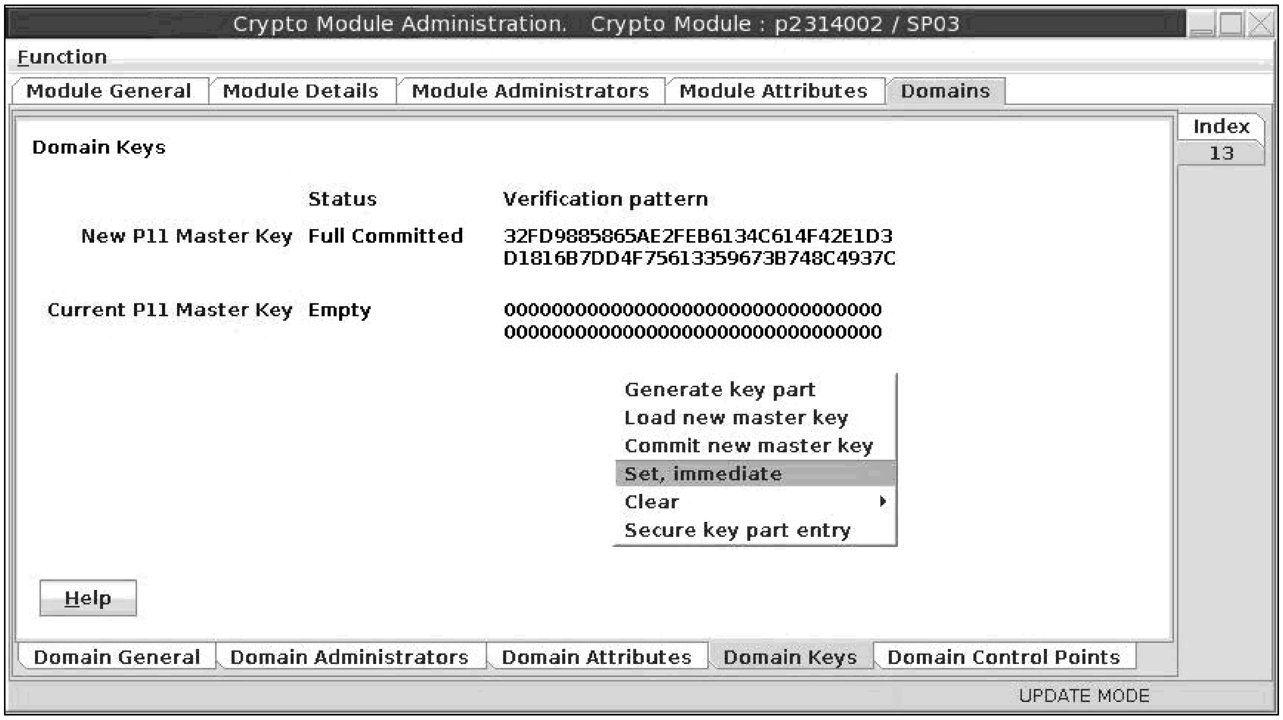
Domain Keys

	Status	Verification pattern
New P11 Master Key	Full Uncommitted	32FD9885865AE2FEB6134C614F42E1D3 D1816B7DD4F75613359673B748C4937C
Current P11 Master Key	Empty	00000000000000000000000000000000 00000000000000000000000000000000

Figure 30. Crypto Module Administration - Commit new master key

Step 18

You can now immediately set the new master key. From the context menu, start action **Set, immediate**.


Crypto Module Administration. Crypto Module : p2314002 / SP03

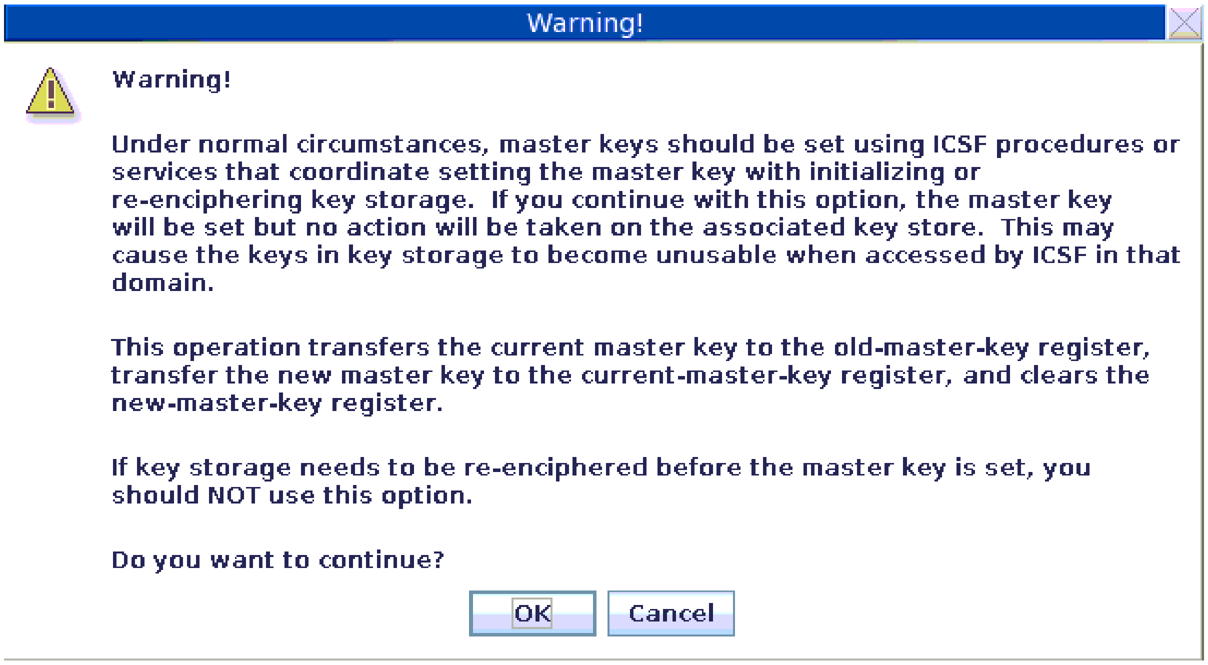
Domain Keys

	Status	Verification pattern
New P11 Master Key	Full Committed	32FD9885865AE2FEB6134C614F42E1D3 D1816B7DD4F75613359673B748C4937C
Current P11 Master Key	Empty	00000000000000000000000000000000 00000000000000000000000000000000

Figure 31. Crypto Module Administration - Set, immediate

Before you actually perform the action, the TKE comes up with a warning. If this is the first time you generated a master key, or if there are no keys stored on your host that are wrapped by the current master key, you can ignore the warning by pressing the **OK** button.

If there are keys wrapped by the current master key on your host, then you should not generate a new master key, but follow the procedure described in Chapter 6, "Managing master keys on the Crypto Express4S EP11 coprocessor," on page 53.


Warning!

Warning!

Under normal circumstances, master keys should be set using ICSF procedures or services that coordinate setting the master key with initializing or re-enciphering key storage. If you continue with this option, the master key will be set but no action will be taken on the associated key store. This may cause the keys in key storage to become unusable when accessed by ICSF in that domain.

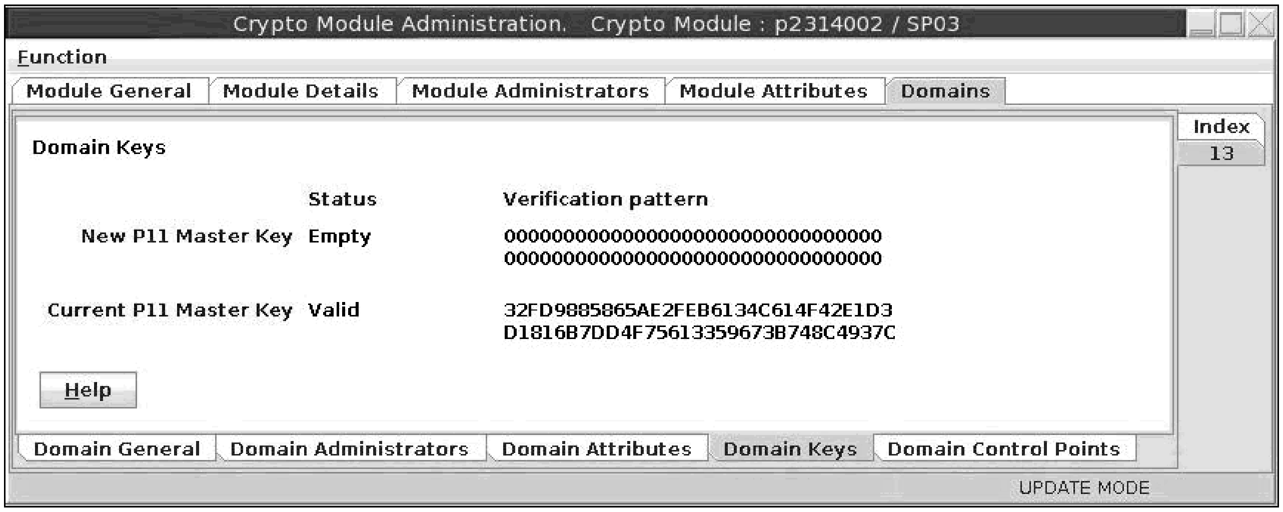
This operation transfers the current master key to the old-master-key register, transfer the new master key to the current-master-key register, and clears the new-master-key register.

If key storage needs to be re-enciphered before the master key is set, you should NOT use this option.

Do you want to continue?

Figure 32. Warning before setting the master key

See the result in Figure 33: The new master key is now switched to the *Current Master Key*, and its status is *Valid*.


Crypto Module Administration. Crypto Module : p2314002 / SP03

Domain Keys

	Status	Verification pattern
New P11 Master Key	Empty	00000000000000000000000000000000 00000000000000000000000000000000
Current P11 Master Key	Valid	32FD9885865AE2FEB6134C614F42E1D3 D1816B7DD4F75613359673B748C4937C

Figure 33. Crypto Module Administration - valid current master key

If you need to change the master key, see Chapter 6, "Managing master keys on the Crypto Express4S EP11 coprocessor," on page 53.

Installing openCryptoki

The EP11 token is part of openCryptoki package starting with version 3.1. openCryptoki in turn is shipped with the Linux on System z distributions.

Check whether you already installed openCryptoki in your current environment, for example:

```
$ rpm -qa | grep -i opencryptoki
```

Note: This command example is distribution dependent. `opencryptoki` must in certain distribution be specified as `openCryptoki` (case-sensitive).

You should see all installed openCryptoki packages. If required packages are missing, use the installation tool of your Linux distribution to install the appropriate openCryptoki RPM.

Note: You must remove any previous package of openCryptoki, before you can install the new package version 3.1.

Installing from the RPM

The openCryptoki version 3.1 or higher packages, that comprise the EP11 support (EP11 token) are delivered by the distributors. Distributors build these packages as RPM packages for delivering them to customers.

Customers can install these openCryptoki RPM packages by using the installation tool of their selected distribution.

If you received openCryptoki as an *RPM* package, follow the *RPM* installation process that is described in the *RPM* man page. This process is the preferred installation method.

Installing from the source package

As an alternative, for example for development purposes, you can get the latest openCryptoki version (inclusive latest patches) from the sourceforge repository (sourceforge.net/projects/opencryptoki) and build it yourself. But this version is not serviced. It is suitable for non-production systems and early feature testing, but you should not use it for production.

In this case, refer to the `INSTALL` file in the top level of the source tree. You can start from the instructions that are provided with the subtopics of this `INSTALL` file and select from the described alternatives. If you use this installation method parallel to the installation of an *RPM* package, then you should keep both installations isolated from each other.

1. Download the latest version of the openCryptoki sources from:
   `http://sourceforge.net/projects/opencryptoki/files/opencryptoki/v3.1/`
2. Decompress and extract the compressed tape archive (TGZ file). There is a new directory named `opencryptoki-3.1-x.x.x`.
3. Change to that directory and issue the following scripts and commands:

```
$ ./bootstrap
$ ./configure
$ make
$ make install
```

The scripts or commands perform the following functions:

bootstrap
Initial setup, basic configurations

configure
Check configurations and build the makefile

make Compile and link

make install
Install the libraries

Note: When installing openCryptoki from the source package, the location of some installed files will differ from the location of files installed from an RPM.

Chapter 4. Configuring openCryptoki for EP11 support

After a successful installation of openCryptoki, you need to perform certain configuration and customization tasks to enable the exploitation of the EP11 library functions from applications. Especially, you need to set up tokens and daemons and then initialize the tokens.

openCryptoki, and in particular the slot manager, can handle several tokens, which can have different support for different hardware devices or software solutions. As shown in Figure 1 on page 6, the EP11 token interacts with the host part of the EP11 library. EP11 can operate with the Crypto Express4S (CEX4S) adapter with EP11 firmware load for symmetric and asymmetric cryptographic functions.

For a complete configuration of the Linux on System z EP11 enablement, finish the tasks as described in the contained subtopics:
- "Adjusting the openCryptoki configuration file"
- "Configuring the EP11 token" on page 43
- "Assigning adapters and domains to LPARs" on page 45
- "Setting environment variables" on page 46
- "Initializing the token" on page 46

Finally, to control your configuration results, follow the instructions provided in "How to recognize the EP11 token" on page 47.

Adjusting the openCryptoki configuration file

A preconfigured list of all available tokens that are ready to register to the openCryptoki slot daemon is required before the slot daemon can start. This list is provided by the global configuration file. Read this topic for information on how to adapt this file according to your installation.

Table 1 lists the maximum number of available libraries that may be in place after you successfully installed openCryptoki. It may vary for different distributions and is dependent from the installed RPM packages.

Also, Linux on System z does not support the TPM token library.

A token is only available, if the token library is installed, and the appropriate software and hardware support pertaining to the stack of the token is also installed. For example, the EP11 token is only available if all parts of the EP11 library software are installed and a Crypto Express4S EP11 coprocessor is detected.

A token needs not be available, even if the corresponding token library is installed. Display the list of available tokens by using the command:

```
$ pkcsconf -t
```

Table 1. openCryptoki libraries.

Library	**Explanation**
`/usr/lib64/opencryptoki/libopencryptoki.so`	openCryptoki base library

Table 1. openCryptoki libraries (continued).

Library	Explanation
`/usr/lib64/opencryptoki/stdll/libpkcs11_ica.so`	libica token library
`/usr/lib64/opencryptoki/stdll/libpkcs11_sw.so`	software token library
`/usr/lib64/opencryptoki/stdll/libpkcs11_tpm.so`	TPM token library
`/usr/lib64/opencryptoki/stdll/libpkcs11_cca.so`	CCA token library
`/usr/lib64/opencryptoki/stdll/libpkcs11_ep11.so`	EP11 token library
`/usr/lib64/opencryptoki/stdll/libpkcs11_icsf.so`	ICSF token library

Note: An analogous set of libraries is available for 32-bit compatibility mode.

The `/etc/opencryptoki/opencryptoki.conf` file must exist and it must contain an entry for the EP11 token to make the token available. By default, this entry is available upon installation (see the `slot 4` entry in the provided sample configuration).

```
-------------- content of opencryptoki.conf ---------
version opencryptoki-3.1
# The following defaults are defined:
#       hwversion = 0.0
#       firmwareversion = 0.0
#       description = Linux
#       manufacturer = IBM
#
# The slot definitions below may be overriden and/or customized.
# For example:
#       slot 0
#       {
#          stdll = libpkcs11_cca.so
#          description = "OCK CCA Token"
#          manufacturer = "MyCompany Inc."
#          hwversion = 2.32
#          firmwareversion = 1.0
#       }
# See man(5) opencryptoki.conf for further information.
#
slot 0
{
stdll = libpkcs11_tpm.so
}

slot 1
{
stdll = libpkcs11_ica.so
}

slot 2
{
stdll = libpkcs11_cca.so
}

slot 3
{
stdll = libpkcs11_sw.so
}

slot 4
{
stdll = libpkcs11_ep11.so
confname = ep11tok.conf
}
---------------------------- end ----------------------------------
```

Note:

- The standard path for slot token dynamic link libraries (STDLLs) is: `/usr/lib64/opencryptoki/stdll/`.
- The standard path for the token-specific `ep11tok.conf` configuration file is `/etc/opencryptoki/`. You can change this path by using the `OCK_EP11_TOKEN_DIR` environment variable. For more information, read "Defining an EP11 token-specific configuration file" on page 44.

Use one of the following command to start the slot-daemon, which reads out the configuration information and sets up the tokens:

```
$ pkcsslotd
$ pkcsslotd start
$ systemctl start pkcsslotd.service
```

For a permanent solution, specify:

```
$ chkconfig pkcsslotd on
```

Configuring the EP11 token

You need to introduce the EP11 token into the openCryptoki library. For this purpose, you must define a slot entry in the global openCryptoki configuration file. You must also create a specific EP11 token configuration file.

Each token has its own token directory, which is used by openCryptoki to store token-specific information (like for example, key objects, user PIN, SO PIN, or hashes). The EP11 token directory is `/var/lib/opencryptoki/ep11tok/`.

Note: This configuration is token-based. It applies to all applications that use this EP11 token.

Defining the slot entry for the EP11 token in openCryptoki

Define a slot entry in the global openCryptoki configuration file that sets the `stdll` attribute to `libpkcs11_ep11.so`.

The default openCryptoki configuration file `opencryptoki.conf` provides a slot entry for the EP11 token. It is preconfigured to slot #4. Check this default entry to find out whether you can use it as is.

Among others, this slot entry must specify the EP11 token-specific configuration file. For this purpose, use the `confname` attribute with value `ep11tok.conf`. This EP11-specific configuration file defines the target adapters and target adapter domains to which the EP11 token sends its cryptographic requests. Insert the complete slot entry into the global openCryptoki configuration file called `opencryptoki.conf` to complete the EP11 token specification.

The following example defines the name of the configuration file of the EP11 token to be `ep11tok.conf`. Per default, this file is searched in the directory where openCryptoki searches its global configuration file.

```
slot 4
{
stdll = libpkcs11_ep11.so
confname = ep11tok.conf
description = "Ep11 Token"
manufacturer = "IBM"
hwversion = 4.11
firmwareversion = 2.0
}
```

Defining an EP11 token-specific configuration file

The configuration file for the EP11 token is delivered by openCryptoki, but must be adapted according to the installation's system environment. As described in the previous section, the slot entry defines this configuration file name as `ep11tok.conf`. If the environment variable `OCK_EP11_TOKEN_DIR` is set, then the EP11 token looks for file `ep11tok.conf` in this directory. If `OCK_EP11_TOKEN_DIR` is not set, then `ep11tok.conf` is searched in the global openCryptoki directory: `/etc/opencryptoki/ep11tok.conf`.

Because different EP11 hardware security modules (HSM) can use different wrapping keys (referred to as master keys in the TKE environment), users need to specify which HSM, in practice an adapter/domain pair, can be used by the EP11 token as a target for cryptographic requests. Therefore, an EP11 token configuration file contains a list of adapter/domain pairs to be used.

You can specify this list as a white list, starting with a line containing the key word `APQN_WHITELIST`. This keyword specifies up to 512 pairs of unsigned, decimal numbers in the range 0-65535, followed by the key word END. Each pair designates an adapter (first number) and a domain (second number). Alternatively, you can use the key word `APQN_ANY` to define that all adapter/domain pairs with EP11 firmware, that are available to the system, can be used as target adapters. This is the default.

An adapter number corresponds to the numerical part xx of an adapter ID of the form *cardxx*, as displayed by the **`lszcrypt`** tool or in the `sys` file system (for example, in `/sys/bus/ap/devices`).

Currently, Linux on System z supports only a single domain. That domain number can be displayed with **`lszcrypt -b`** (see the value of ap_domain) or alternatively as contents of `/sys/bus/ap/ap_domain`.

In addition to the target adapter, you can define a log level in the EP11 configuration file. For this purpose, use a line that consists of the key word LOGLEVEL followed by an integer between 0 and 9. For information about log levels, read topic "Enabling the logging support while running the EP11 token" on page 57.

Example of an EP11 token configuration file `ep11tok.conf`

```
#
# EP11 token configuration
#
APQN_ANY
LOGLEVEL 3
#
```

```
APQN_WHITELIST
5 2
6 2
END
```

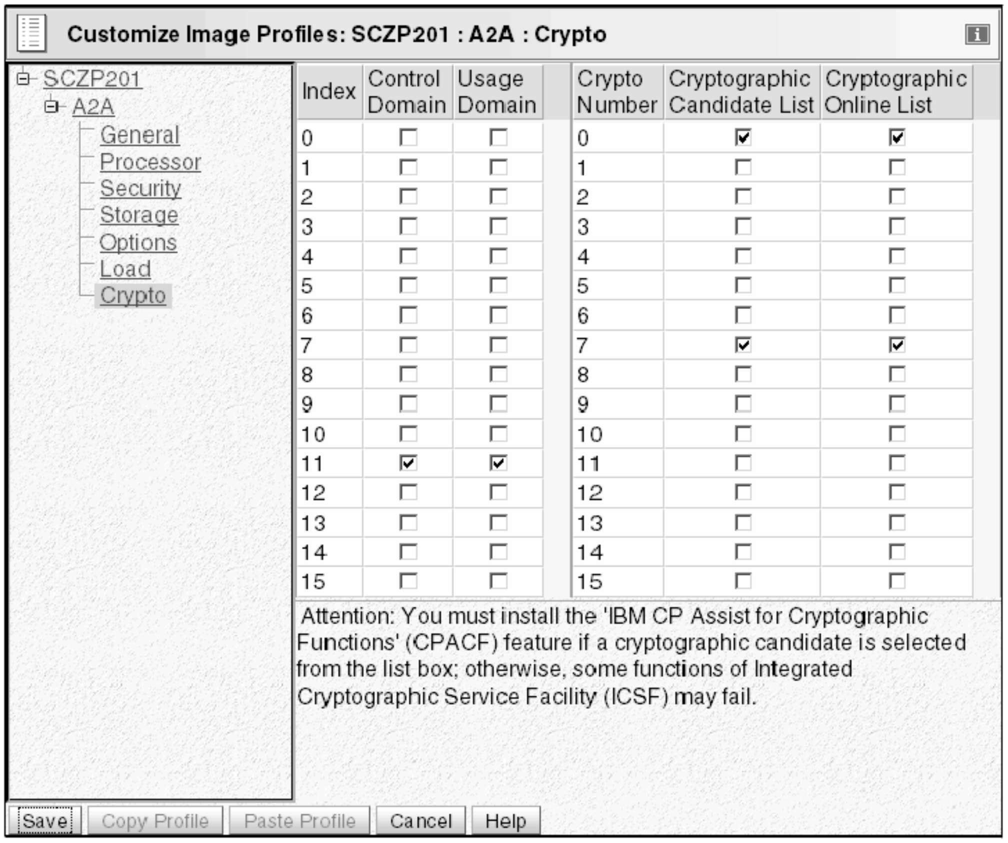
In this example, adapter 5 with domain 2 and adapter 6 with domain 2 is specified.

Note: At least one adapter/domain pair must be specified. If more than one APQN (adapter/domain pair) is used by a token, then all used adapter/domain pairs must be configured with the same master key.

Assigning adapters and domains to LPARs

After you set up the Crypto Express adapter in the Support Element, you must allow access to it from your LPAR. You achieve this by using the Hardware Management Console (HMC) or the Support Element (SE).

You can define a certain LPAR to use a domain (or multiple domains) as a usage domain and as a control domain, or as a control domain only. You can retrieve this information from the Support Element. Each adapter supports 16 domains (see Figure 34). The selected domains apply to all selected adapters. For a more detailed information about planning the cryptographic configuration, see *IBM System z10 Enterprise Class Configuration Setup, SG24-7571*.



Customize Image Profiles: SCZP201 : A2A : Crypto

Index	Control Domain	Usage Domain	Crypto Number	Cryptographic Candidate List	Cryptographic Online List
0	☐	☐	0	☑	☑
1	☐	☐	1	☐	☐
2	☐	☐	2	☐	☐
3	☐	☐	3	☐	☐
4	☐	☐	4	☐	☐
5	☐	☐	5	☐	☐
6	☐	☐	6	☐	☐
7	☐	☐	7	☑	☑
8	☐	☐	8	☐	☐
9	☐	☐	9	☐	☐
10	☐	☐	10	☐	☐
11	☑	☑	11	☐	☐
12	☐	☐	12	☐	☐
13	☐	☐	13	☐	☐
14	☐	☐	14	☐	☐
15	☐	☐	15	☐	☐

Attention: You must install the 'IBM CP Assist for Cryptographic Functions' (CPACF) feature if a cryptographic candidate is selected from the list box; otherwise, some functions of Integrated Cryptographic Service Facility (ICSF) may fail.

Figure 34. Cryptographic configuration for LPAR A2A

In Figure 34 on page 45, LPAR A2A is defined to use and control the cryptographic domain number 11. It is also allowed to access the crypto adapters, numbers 0 and 7. They are brought online if they are present in the system, if the LPAR is activated, and if the **zcrypt** device driver is loaded.

Linux can only use one crypto domain at a given time. If the LPAR contains multiple domains, the kernel selects one of them. If you want to use a different domain, you need to specify this domain as a parameter when loading the *ap* main module of the zcrypt device driver.

Setting environment variables

To customize your EP11 enablement, you can set environment variables. Setting environment variables overrides the settings in the `ep11tok.conf` configuration file.

The following variables are available:

OCK_EP11_TOKEN_DIR
specifies a new location for the EP11 token configuration file. The default is `/etc/opencryptoki/`. Example:

```
export OCK_EP11_TOKEN_DIR=/home/user/ep11token
```

OCK_EP11_TOKEN_LOGLEVEL
defines the granularity of logging support. Valid values are between 0 and 9. For information about log levels, read topic “Enabling the logging support while running the EP11 token” on page 57. Example:

```
export OCK_EP11_TOKEN_LOGLEVEL=2
```

Initializing the token

Once the configuration files of openCryptoki and the EP11 token are set up, and the **pkcsslotd** daemon is started, the EP11 token must be initialized.

Note: PKCS #11 defines two users for each token: a security officer (SO) whose responsibility is the administration of the token, and a standard user (User) who wants to use the token to perform cryptographic operations. openCryptoki requires that for both the SO and the User a log-in PIN is defined as part of the token initialization.

The following command provides some useful slot information:

```
# pkcsconf -s

Slot #0 Info
        Description: EP11 Token
        Manufacturer: IBM
        Flags: 0x1 (TOKEN_PRESENT)
        Hardware Version: 4.0
        Firmware Version: 2.11
Slot #1 Info
        Description: ICA Token
        Manufacturer: IBM
        Flags: 0x1 (TOKEN_PRESENT)
        Hardware Version: 4.0
        Firmware Version: 2.10
```

Find your preferred token in the details list and select the correct slot number. This number is used in the next initialization steps to identify your token:

```
$ pkcsconf -I -c <slot> // Initialize the Token and setup a Token Label

$ pkcsconf -P -c <slot> // change the SO PIN (recommended)

$ pkcsconf -u -c <slot> // Initialize the User PIN (SO PIN required)

$ pkcsconf -p -c <slot> // change the User PIN (optional)
```

`pkcsconf -I`
: During token initialization, you are asked for a token label. Provide a meaningful name, because you might need this reference for identification purposes.

`pkcsconf -P`
: For security reasons, openCryptoki requires that you change the default SO PIN (`87654321`) to a different value. Use the `pkcsconf -P` option to change the SO PIN.

`pkcsconf -u`
: When you enter the user PIN initialization you are asked for the newly set SO PIN. The length of the user PIN must be 4 - 8 characters.

`pkcsconf -p`
: You must at least once change the user PIN with `pkcsconf -p` option. After you completed the PIN setup, the token is prepared and ready for use.

Note: Define a user PIN that is different from `12345678`, because this pattern is checked internally and marked as default PIN. A log-in attempt with this user PIN is recognized as *not initialized*.

How to recognize the EP11 token

You can use the **`pkcsconf -t`** command to display a table that shows all available tokens. You can check the slot and token information, and the PIN status at any time.

The following information provided by the **`pkcsconf -t`** command about the EP11 token is returned in the *Token Info* section, where, for example, `Token #1 Info` displays information about the token plugged into slot number 1.

```
$ pkcsconf -t

Token #1 Info:
       Label: ep11
 Manufacturer: IBM Corp.
 Model: IBM EP11Tok
 Serial Number: 123
 Flags: 0x880445
          (RNG|LOGIN_REQUIRED|CLOCK_ON_TOKEN|TOKEN_INITIALIZED|USER_PIN_TO_BE_CHANGED
           |SO_PIN_TO_BE_CHANGED)
 Sessions: 0/-2
 R/W Sessions: -1/-2
 PIN Length: 4-8
 Public Memory: 0xFFFFFFFF/0xFFFFFFFF
 Private Memory: 0xFFFFFFFF/0xFFFFFFFF
 Hardware Version: 1.0
 Firmware Version: 1.0
 Time: 15:29:43
```

The most important information is as follows:

- The token **`Label`** you assigned at the initialization phase (`ep11`, in the example). You can initialize or change a token label by using the `pkcsconf -I` command.
- The **`Model`** name is unique and designates the token that is in use.
- The **`Flags`** provide information about the token initialization status, the PIN status, and features such as ***Random Number Generator*** (RNG). They also provide information about requirements, such as ***Login required***, which means that there is at least one mechanism that requires a session log-in to use that cryptographic function. For example, the mask for TOKEN_INITIALIZED is 0x00000400 and it is true, if the token has been initialized.

  The flag USER_PIN_TO_BE_CHANGED indicates that the user PIN must be changed before the token can be used. The flag SO_PIN_TO_BE_CHANGED indicates that the SO PIN must be changed before the token can be used.

  For more information about the flags provided in this output, see the description of the TOKEN_INFO structure and the Token Information Flags in the PKCS #11 Cryptographic Token Interface Standard.
- The **`PIN length`** range declared for this token.

Chapter 5. Using the EP11 token

You can take advantage of the EP11 library functions by using the openCryptoki standard interface (PKCS #11 standard C API).

The PKCS #11 Cryptographic Token Interface Standard describes the exact API.

Applications that are designed to work with openCryptoki are also able to use the Linux on System z EP11 enablement.

The EP11 token plugged into openCryptoki works only on IBM System z hardware, with further prerequisites as described in this publication.

Supported mechanisms for the EP11 token

View a list of the supported mechanisms for the EP11 token in the openCryptoki implementation.

Use the **pkcsconf** command with the shown parameters to retrieve a complete list of algorithms (or mechanisms) that are supported by the token:

```
$ pkcsconf -m -c <slot>
Mechanism #2
        Mechanism: 0x131 (CKM_DES3_KEY_GEN)
        Key Size: 24-24
        Flags: 0x8001 (CKF_HW|CKF_GENERATE)
...
Mechanism #10
        Mechanism: 0x132 (CKM_DES3_ECB)
        Key Size: 24-24
        Flags: 0x60301 (CKF_HW|CKF_ENCRYPT|CKF_DECRYPT|CKF_WRAP|CKF_UNWRAP)
Mechanism #11
        Mechanism: 0x133 (CKM_DES3_CBC)
        Key Size: 24-24
        Flags: 0x60301 (CKF_HW|CKF_ENCRYPT|CKF_DECRYPT|CKF_WRAP|CKF_UNWRAP)
...
```

The list displays all mechanisms that are supported by this token. The mechanism ID and name corresponds to the PKCS #11 specification. Each mechanism provides its supported key size and some further properties such as hardware support and mechanism information flags. These flags provide information about the PKCS #11 functions that may use the mechanism. In some cases, the flags also provide further attributes that describe the supported variants of the mechanism. Typical functions are for example, *encrypt*, *decrypt*, *wrap key*, *unwrap key*, *sign*, or *verify*.

The Crypto Express4S card in a zEC12 (at least GA1) together with openCryptoki version 3.1 and EP11 token support these PKCS #11 mechanisms:

Table 2. PKCS #11 mechanisms supported by the EP11 token.

Mechanism	Key sizes	Properties
CKM_RSA_PKCS	1024-4096	ENCRYPT, DECRYPT, SIGN,VERIFY, WRAP,UNWRAP
CKM_RSA_PKCS_KEY_PAIR_GEN	1024-4096	GENERATE_KEY_PAIR
CKM_RSA_X9_31_KEY_PAIR_GEN	1024-4096	GENERATE_KEY_PAIR

Table 2. PKCS #11 mechanisms supported by the EP11 token (continued).

Mechanism	**Key sizes**	**Properties**
CKM_RSA_PKCS_PSS	1024-4096	SIGN,VERIFY
CKM_SHA1_RSA_X9_31	1024-4096	SIGN,VERIFY
CKM_SHA1_RSA_PKCS	1024-4096	SIGN,VERIFY
CKM_SHA1_RSA_PKCS_PSS	1024-4096	SIGN,VERIFY
CKM_SHA256_RSA_PKCS	1024-4096	SIGN,VERIFY
CKM_SHA384_RSA_PKCS	1024-4096	SIGN,VERIFY
CKM_SHA512_RSA_PKCS	1024-4096	SIGN,VERIFY
CKM_AES_KEY_GEN	16,24,32	GENERATE
CKM_AES_ECB	16,24,32	ENCRYPT,DECRYPT
CKM_AES_CBC	16,24,32	ENCRYPT,DECRYPT, WRAP,UNWRAP
CKM_AES_CBC_PAD	16,24,32	ENCRYPT,DECRYPT, WRAP,UNWRAP
CKM_DES2_KEY_GEN	16,16	GENERATE
CKM_DES3_KEY_GEN	24,24	GENERATE
CKM_DES3_ECB	16,24	ENCRYPT,DECRYPT
CKM_DES3_CBC	16,24	ENCRYPT,DECRYPT, WRAP,UNWRAP
CKM_DES3_CBC_PAD	16,24	ENCRYPT,DECRYPT, WRAP,UNWRAP
CKM_SHA256	0,0	DIGEST
CKM_SHA256_KEY_DERIVATION	0,0	DERIVE
CKM_SHA256_HMAC	16,256	SIGN,VERIFY
CKM_SHA_1	0,0	DIGEST
CKM_SHA1_KEY_DERIVATION	0,0	DERIVE
CKM_SHA_1_HMAC	16,256	SIGN,VERIFY
CKM_SHA384	0,0	DIGEST
CKM_SHA384_HMAC	16,256	SIGN,VERIFY
CKM_SHA512	0,0	DIGEST
CKM_SHA512_HMAC	16,256	SIGN,VERIFY
CKM_EC_KEY_PAIR_GEN	192,521	GENERATE_KEY_PAIR, CKF_EC_F_P, EC_ECPARAMETERS, EC_NAMEDCURVE, EC_UNCOMPRESS
CKM_ECDSA	192,521	SIGN,VERIFY, CKF_EC_F_P, EC_ECPARAMETERS, EC_NAMEDCURVE, EC_UNCOMPRESS
CKM_ECDSA_SHA1	192,521	SIGN,VERIFY, CKF_EC_F_P, EC_ECPARAMETERS, EC_NAMEDCURVE, EC_UNCOMPRESS

Table 2. PKCS #11 mechanisms supported by the EP11 token (continued).

Mechanism	Key sizes	Properties
CKM_ECDH1_DERIVE	192,521	DERIVE, CKF_EC_F_P, EC_UNCOMPRESS
CKM_DSA_PARAMETER_GEN	1024-3072	GENERATE
CKM_DSA_KEY_PAIR_GEN	1024-3072	GENERATE_KEY_PAIR
CKM_DSA	1024-3072	SIGN,VERIFY
CKM_DSA_SHA1	1024-3072	SIGN,VERIFY
CKM_DH_PKCS_PARAMETER_GEN	1024-3072	GENERATE
CKM_DH_PKCS_KEY_PAIR_GEN	1024-3072	GENERATE_KEY_PAIR
CKM_DH_PKCS_DERIVE	1024-3072	DERIVE
CKM_RSA_X_509	1024-4096	ENCRYPT,DECRYPT, SIGN, VERIFY
CKM_RSA_X9_31	1024-4096	SIGN, VERIFY
CKM_PBE_SHA1_DES3_EDE_CBC	24,24	GENERATE

For explanation about the key object properties see the PKCS #11 Cryptographic Token Interface Standard.

Restrictions with using the EP11 library functions

In this topic, you find information about certain limitations of the EP11 library.

- The EP11 library implements the *secure key concept* (that is, a key is wrapped (encrypted) by a master key, which is kept within the EP11 adapter). That means, that EP11 key values are never accessible. The secure key concept ensures that clear keys never leave the hardware security module (HSM), which is the EP11 module part that is installed on the IBM Crypto Express4S card.

  Therefore, all keys must have the attribute CKA_SENSITIVE set to CK_TRUE. Since the PKCS #11 standard does not define a (token-specific) default for secret keys, the attribute must be explicitly provided whenever a secret key is generated, unwrapped, or build with **`C_CreateObject`**.

  In addition, all keys that are used with the EP11 token are extractable, that is, they must have the attribute `CKA_EXTRACTABLE` set to `CK_TRUE`.
- Keys leaving the hardware security module (HSM) are encrypted by the HSM master key (wrapping key) and come as binary large object (BLOB). In openCryptoki, objects can have special attributes that describe the key properties. Besides dedicated attributes defined by the application, there are some attributes defined as token-specific by openCryptoki.

  Table 3 and Table 4 on page 52 show the EP11 token-specific attributes and their default values for private and secure keys.

Table 3. Private key default attributes of the EP11 token

Private key attributes	value
CKA_SENSITIVE	CK_TRUE
CKA_EXTRACTABLE	CK_TRUE

Table 4. Secret key default attributes of the EP11 token

Secret key attributes	**value**
CKA_EXTRACTABLE	CK_TRUE

- When you create keys the default values of the attributes CKA_ENCRYPT, CKA DECRYPT, CKA_VERIFY, CKA_SIGN, CKA_WRAP and CKA_UNWRAP are CK_TRUE. Note, no EP11 mechanism supports the Sign/Recover or Verify/Recover functions.

  Note: During secret key generation, it is necessary to set the `CKA_SENSITIVE` attribute to `CKA_TRUE` explicitly, otherwise the key generation fails.

  Even if settings of `CKA_SENSITIVE`, `CKA_EXTRACTABLE`, or `CKA_NEVER_EXTRACTABLE` would allow accessing the key value, then openCryptoki returns `00..00` as key value (due to the secure key concept).

  For information about the key attributes, see the PKCS #11 Cryptographic Token Interface Standard.
- All RSA keys must have a public exponent (CKA_PUBLIC_EXPONENT) greater than or equal to 17.
- The Crypto Express EP11 coprocessor restricts RSA keys (primes and moduli) according to ANSI X9.31. Therefore, in the EP11 token, the lengths of the RSA primes (p or q) must be a multiple of 128 bits. Also, the length of the modulus (`CKA_MODULUS_BITS`) must be a multiple of 256.
- The mechanisms `CKM_DES3_CBC` and `CKM_AES_CBC` can only wrap keys, which have a length that is a multiple of the block size of DES3 or AES respectively. See the mechanism list and mechanism information (**`pkcsconf -m`**) for supported mechanisms together with supported functions and key sizes.
- The EP11 coprocessor adapter can be configured to restrict the cryptographic capabilities in order for the adapter to comply with specific security requirements and regulations. Such restrictions on the adapter impact the capability of the EP11 token.
- The EP11 library functions do not provide a key import for DSA, DH, and EC keys, but only support key generation on the adapter.

Chapter 6. Managing master keys on the Crypto Express4S EP11 coprocessor

There may be situations when the master key on (a domain of) a CEX4P or later adapter must be changed, for example, if company policies require periodic changes of all master keys. Simply changing the master keys using the TKE results in all secure keys stored in the EP11 token to become useless. Therefore all data encrypted by these keys are lost. To avoid this situation, you must accomplish a master key migration process, where activities on the TKE and on the Linux system must be interlocked.

The EP11 token stores all token key objects in the Linux file system in the `/var/lib/opencryptoki/ep11/TOK_OBJ` directory. All secret and private keys are secure keys, that means they are enciphered (wrapped) with the master key (*MK*) of the CEX4P adapter domain. Therefore, the master key is often also referred to as wrapping key. If master keys are changed in a domain of a CEX4P adapter, all key objects for secure keys in the EP11 token object repository become invalid. Therefore, all key objects for secure keys must be re-enciphered with the new *MK*. In order to re-encipher secure keys that are stored as EP11 key objects in the EP11 token object repository, openCryptoki provides the master key migration tool *pkcsep11_migrate*.

How to access the master key migration tool

The *pkcsep11_migrate* key migration utility is part of openCryptoki version 3.1, which includes the EP11 support.

Prerequisites for the master key migration process

The master key migration process for the EP11 token requires a TKE version 7.3 environment. How to set up this environment is described in “Setting up the TKE environment” on page 20.

To use the *pkcsep11_migrate* migration tool, the EP11 crypto stack including openCryptoki must be installed and configured. For information on how to set up this environment, refer to Chapter 3, “Building the EP11 crypto stack,” on page 11.

The master key migration process

Prerequisite for re-encipherment: The EP11 token may be configured to use more than one adapter/domain pair to perform its cryptographic operations. This is defined in the EP11 token configuration file. If the EP11 token is configured to use more than one adapter/domain pair, then all adapter/domain pairs must be configured to each have the same set of master keys. Therefore, if a master key on one of these adapter/domain pairs is changed, it must be changed on all those other adapter/domain pairs, too.

To migrate master keys on the set of adapter/domain pairs used by an EP11 token, you must perform the following steps:

1. On the Trusted Key Entry console (TKE), submit and commit the (same) new master key on all CEX4P adapter/domain combinations used by the EP11 token.

2. On Linux, stop all processes that are currently using openCryptoki with the EP11 token.
3. On Linux, back up the token object repository of the EP11 token. For example, you can use the following commands:

```
cd /var/lib/opencryptoki/ep11
tar -cvzf ~/ep11TOK_OBJ_backup.tgz TOK_OBJ
```

4. On Linux, migrate the keys of the EP11 token object repository with the *pkcsep11_migrate* migration tool (see the invocation information provided at the end of these process steps). The *pkcsep11_migrate* tool must only be called once for one of the adapter/domain pairs that the EP11 token uses. If a failure occurs, restore the backed-up token repository and try this step again.

   Attention: Do not continue with step 5 unless step 4 was successful. Otherwise you will lose your encrypted data.
5. On the TKE, activate the new master keys on all EP11 adapter/domain combinations that the EP11 token uses.
6. On Linux, restart the applications that used openCryptoki with the EP11 token.

In step 1 on page 53 of the master key migration process, the new master key must be submitted and committed via the TKE interface. That means the *new EP11 master key* must be in the state `Full Committed`. The current MK is in the state `Valid`. Now both (current and new) *EP11 master keys* are available and accessible. The utility can now decrypt all relevant key objects within the token and re-encrypt all these key objects with the new master key.

Note: All the decrypt and encrypt operations are done inside the EP11 coprocessor card, that means that at no time clear key values are visible within memory.

Invocation: `pkcsep11_migrate <-slot> <-adapter> <-domain>`

The following parameters are mandatory:

-slot - slot number for the EP11 token

-adapter
- the card ID, using the numerical suffix value form the card ID in the *sysfs* (to be retrieved from `/sys/devices/ap/cardxx` or with **`lszcrypt`**)

-domain
- the card domain number (to be retrieved from `/sys/bus/ap/ap_domain` or with **`lszcrypt -b`**)

All token objects representing secret or private keys that are found for the EP11 token, are re-encrypted.

Usage: The environment variable `PKCS11_USER_PIN_ENV_VAR` must be set with the USER pin of the EP11 token.

Example:

```
export PKCS11_USER_PIN_ENV_VAR=12345678
pkcsep11_migrate -slot 0 -adapter 8 -domain 13
```

Note: The program stops if the re-encryption of a token object fails. In this case, restore the back-up.

After this utility re-enciphered all key objects, the new master key must be activated. This activation must be done by using the TKE interface command **`Set, immediate`**. Finally, the new MK becomes the current MK and the previous MK must be deleted.

Note: This tool is embedded in the users `sbin` path and therefore callable from everywhere.

To prevent token object generation during re-encryption, openCryptoki with the EP11 token must not be running during re-encryption. It is recommended to make a back-up of the EP11 token object directory (/usr/local/var/lib/opencryptoki/ep11tok/TOK_OBJ).

Chapter 7. Troubleshooting EP11

Troubleshooting can provide helpful information, if problems occur while you work with the Linux on System z EP11 enablement.

The contained subtopics introduce different methods, which support troubleshooting:

- "Checking the device driver status"
- "Checking the EP11 token status"
- "Enabling the logging support while running the EP11 token"

Checking the device driver status

The first step of troubleshooting while working with the EP11 enablement may be to check the device driver status as described in this topic.

Use the **lszcrypt** command like shown to retrieve basic status information. Type **lszcrypt -VV** to achieve the following output:

```
$ lszcrypt -VV
card06: CEX4P online hwtype=10 depth=8 request_count=0
```

This call can be used to check whether the EP11 requests are sent to a specific crypto adapter.

For more information about using IBM cryptographic adapters with Linux on System z, see *Device Drivers, Features, and Commands*, SC33-8411 available at www.ibm.com/developerworks/linux/linux390/documentation_dev.html

Checking the EP11 token status

You can request information about the EP11 token status by using the **pkcsconf -t** command. The *Flags* entry shows the actual status flags for the token and whether the token is ready to be used. In the shown example, the SO PIN needs to be changed before the token can be used.

```
$ pkcsconf -t

Token #1 Info:
    ...
 Model: IBM EP11Tok
 ...
 Flags: 0x80044D
         (RNG|LOGIN_REQUIRED|USER_PIN_INITIALIZED|CLOCK_ON_TOKEN|TOKEN_INITIALIZED
             |SO_PIN_TO_BE_CHANGED)
 ...
 ...
 Time: 15:29:43
```

Enabling the logging support while running the EP11 token

Read about the tasks how to run the EP11 token with enabled logging support.

Logging support can be enabled either by setting the LOGLEVEL variable in the EP11 configuration file (ep11tok.conf) or by setting the environment variable OCK_EP11_TOKEN_LOGLEVEL. If neither the configuration file variable nor the environment variable was set, logging is disabled by default. If both variables are set, the environment variable overrules the settings in the configuration file.

Example of an EP11 token configuration file (ep11tok.conf) with log level 3:

```
#
# EP11 token configuration
#
APQN_ANY
LOGLEVEL 3
```

Table 5. EP11 log levels.

Log level	Description
0	No logging at all, not even for unrecoverable errors. This level is default in field mode.
1	Basic logging mode for unrecoverable errors and errors important for the user. No information logs are provided.
2	Like log level 1, including information logs, which can give additional information to the user. This level is favorite when users need debugging help.
3	Like log level 2, including warnings and additional information messages important to the developer. Select this level, if you need to forward logging information to the IBM support.
4	Like log level 3, including additional information, for example session and log-in tracing messages.
5	Like log level 4, including additional information, for example, API function entry and exit messages.
6 - 9	Tracing modes for increasingly elaborated debugging.

If a log level > 0 is defined in the environment variable OCK_EP11_TOKEN_LOGLEVEL or by the LOGLEVEL entry in the EP11 configuration file, then log entries are written to file /var/log/ock_ep11_token.<pid>.log. In this file name specification, <pid> denotes the ID of the running process that uses the EP11 token.

The log file is created with ownership *user*, and group *pkcs11*, and permission 640 (user: read, write; group: read only; others: nothing). For every application, which is using openCryptoki with the EP11 token, a new log file is created during token initialization. Prerequisite for a working EP11 stack is the existence of the EP11 coprocessor card and an appropriate device driver with EP11 support.

A log level > 3 is only recommended for developers.

Note: Future releases of openCryptoki may provide a different framework for logging and tracing which may lead to changes of the logging performed by the EP11 token.

Chapter 8. Programming examples for openCryptoki

The provided program segments in C illustrate some openCryptoki version 3.1 sample APIs to be used for EP11.

The contained openCryptoki code samples provide an insight into how to deal with the openCryptoki API's. After describing some basic functions such as initialization, session and log-in handling, the samples provide an introduction about how to create key objects and process symmetric encryption/decryption (AES). The last section shows RSA key generation with RSA encrypt and decrypt operations.

To develop an application that uses the openCryptoki library, you need to access the library. You achieve the loading of shared objects by using dynamic library calls (dlopen) as described in the sample provided in “Base procedures.”

At compile time, you need to specify the openCryptoki library:

```
gcc test_ock.c -g -O0 -o test_ock -lopencryptoki -ldl
-I /usr/include/opencryptoki/
```

The exact location of the include files depends on your Linux distribution.

The following sample categories are provided:

- Base procedures
- Session and log-in
- Object handling
- Cryptographic operations

Base procedures

View some openCryptoki code samples for base procedures, such as a main program, an initialization procedure, and finalize information.

Main program

```
/* Example program to test opencryptoki
 * build: gcc test_ock.c -g -O0 -o test_ock -lopencryptoki -ldl
                         -I /root/opencryptoki/usr/include/pkcs11/
 * execute: ./test_ock -c <slot> -p <PIN> */
#include <stdlib.h>
#include <errno.h>
#include <stdio.h>
#include <dlfcn.h>
#include <pkcs11types.h>
#include <string.h>
#include <unistd.h>
#define OCKSHAREDLIB "libopencryptoki.so"

void *lib_ock;
char *pin = NULL;
int count, arg;
CK_SLOT_ID  slotID = 0;
CK_ULONG rsaKeyLen = 2048, cipherTextLen = 0, clearTextLen = 0;
CK_BYTE *pCipherText = NULL, *pClearText = NULL;
CK_BYTE *pRSACipher = NULL, *pRSAClear = NULL;
CK_FLAGS rw_sessionFlags = CKF_RW_SESSION | CKF_SERIAL_SESSION;
CK_SESSION_HANDLE hSession;
CK_BYTE keyValue[] = {0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef,
                      0xCA,0xFE,0xBE,0xEF,0xCA,0xFE,0xBE,0xEF};
CK_BYTE msg[] = "The quick brown fox jumps over the lazy dog";
CK_ULONG msgLen = sizeof(msg);
CK_OBJECT_HANDLE hPublicKey, hPrivateKey;

/*** <insert helper functions (provided below) here> ***/
/*** usage / help ***/
void usage(void)
{
  printf("Usage:\n");
  printf(" -s <slot number> \n");
  printf(" -p <user PIN>\n");
  printf("\n");
  exit (8);  }

int main(int argc, char *argv[]) {
   while ((arg = getopt (argc, argv, "s:p:")) != -1) {
 switch (arg) {
 case 's':   slotID = atoi(optarg);
       break;
 case 'p':   pin = malloc(strlen(optarg));
             strcpy(pin,optarg);
       break;
 default:    printf("wrong option %c", arg);
       usage();
 }  }

  if ((!pin) || (!slotID)) {
 printf("Incorrect parameter given!\n");
 usage();
 exit (8);   }

  init();
  openSession(slotID, rw_sessionFlags, &hSession);
  loginSession(CKU_USER, pin, 8, hSession);
  createKeyObject(hSession, (CK_BYTE_PTR)&keyValue, sizeof(keyValue));
  AESencrypt(hSession, (CK_BYTE_PTR)&msg, msgLen, &pCipherText, &cipherTextLen);
  AESdecrypt(hSession, pCipherText, cipherTextLen, &pClearText, &clearTextLen);
  generateRSAKeyPair(hSession, rsaKeyLen, &hPublicKey, &hPrivateKey);
  RSAencrypt(hSession, hPublicKey, (CK_BYTE_PTR)&msg, msgLen, &pRSACipher, &rsaKeyLen);
  RSAdecrypt(hSession, hPrivateKey, pRSACipher, rsaKeyLen, &pRSAClear, &rsaKeyLen);
  logoutSession(hSession); closeSession(hSession);
  finalize();
  return 0;
}
```

C_Initialize

```
/*
 * initialize
 */
CK_RV init(void){
  CK_RV rc;
  lib_ock = dlopen(OCKSHAREDLIB, RTLD_GLOBAL | RTLD_NOW);
  if (!lib_ock) {
 printf("Error loading shared lib '%s' [%s]", OCKSHAREDLIB, dlerror());
 return 1;
  }
  rc = C_Initialize(NULL);
  if (rc != CKR_OK) {
     printf("Error initializing the opencryptoki library: 0x%X\n", rc);
  }
  return CKR_OK;
}
```

C_Finalize

```
/*
 * finalize
 */
CK_RV finalize(void) {
  CK_RV rc;
  rc = C_Finalize(NULL);
  if (rc != CKR_OK) {
 printf("Error during finalize: %x\n", rc);
 return rc;
  }
  if (pCipherText) free(pCipherText);
  if (pClearText)  free(pClearText);
  if (pRSACipher)  free(pRSACipher);
  if (pRSAClear)   free(pRSAClear);
  return CKR_OK;
}
```

Session and log-in procedures

When you use your sample code with a static linked library you can access the APIs directly. View some openCryptoki code samples for opening and closing sessions and for log-in.

C_OpenSession:

```
/*
 * opensession
 */

CK_RV openSession(CK_SLOT_ID slotID, CK_FLAGS sFlags,
                                     CK_SESSION_HANDLE_PTR phSession) {
 CK_RV rc;
  rc = C_OpenSession(slotID, sFlags, NULL, NULL, phSession);
  if (rc != CKR_OK) {
     printf("Error opening session: %x\n", rc);
     return rc;
  }
  printf("Open session successful.\n");
  return CKR_OK;
}
```

C_CloseSession:

```
/*
 * closesession
 */
CK_RV closeSession(CK_SESSION_HANDLE hSession) {
  CK_RV  rc;
  rc  = C_CloseSession(hSession);
  if (rc != CKR_OK) {
 printf("Error closing session: 0x%X\n", rc);
 return rc;
  }
  printf("Close session successful.\n");
  return CKR_OK;
}
```

C_Login:

```
/*
 * login
 */
CK_RV loginSession(CK_USER_TYPE userType, CK_CHAR_PTR pPin,
     CK_ULONG ulPinLen, CK_SESSION_HANDLE hSession) {
  CK_RV rc;
  rc = C_Login(hSession, userType, pPin, ulPinLen);
  if (rc != CKR_OK) {
 printf("Error login session: %x\n", rc);
 return rc;
  }
  printf("Login session successful.\n");
  return CKR_OK;
}
```

C_Logout:

```
/*
 * logout
 */
CK_RV logoutSession(CK_SESSION_HANDLE hSession) {
  CK_RV rc;
  rc = C_Logout(hSession);
  if (rc != CKR_OK) {
 printf("Error logout session: %x\n", rc);
 return rc;
  }
  printf("Logout session successful.\n");
  return CKR_OK;
}
```

Object handling procedures

When you use your sample code with a static linked library you can access the APIs directly. View some openCryptoki code samples for procedures dealing with object handling.

C_CreateKeyObject:

```
/*
 * createKeyObject
 */
CK_RV createKeyObject(CK_SESSION_HANDLE hSession, CK_BYTE_PTR key, CK_ULONG keyLength) {
  CK_RV rc;

  CK_OBJECT_HANDLE hKey;
  CK_BBOOL true = TRUE;
  CK_BBOOL false = FALSE;
  CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;
  CK_KEY_TYPE keyType = CKK_AES;
  CK_ATTRIBUTE keyTempl[] = {
    {CKA_CLASS, &keyClass, sizeof(keyClass)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_DECRYPT, &true, sizeof(true)},
    {CKA_SIGN, &true, sizeof(true)},
    {CKA_VERIFY, &true, sizeof(true)},
    {CKA_TOKEN, &true, sizeof(true)},   /* token object  */
    {CKA_PRIVATE, &false, sizeof(false)}, /* public object */
    {CKA_VALUE, keyValue, keyLength}, /* AES key       */
    {CKA_LABEL, "My_AES_Key", sizeof("My_AES_Key")}
  };
  rc = C_CreateObject(hSession, keyTempl, sizeof (keyTempl)/sizeof (CK_ATTRIBUTE), &hKey);
  if (rc != CKR_OK) {
 printf("Error creating key object: 0x%X\n", rc); return rc;
  }
  printf("AES Key object creation successful.\n");
}
```

C_FindObjects:

```
/*
 * findObjects
 */
CK_RV getKey(CK_CHAR_PTR label, int labelLen, CK_OBJECT_HANDLE_PTR hObject,
      CK_SESSION_HANDLE hSession) {
  CK_RV rc;
  CK_ULONG ulMaxObjectCount = 1;
  CK_ULONG ulObjectCount;
  CK_ATTRIBUTE objectMask[] = { {CKA_LABEL, label, labelLen} };
  rc = C_FindObjectsInit(hSession, objectMask, 1);
  if (rc != CKR_OK) {
 printf("Error FindObjectsInit: 0x%X\n", rc); return rc;
  }
  rc = C_FindObjects(hSession, hObject, ulMaxObjectCount, &ulObjectCount);
  if (rc != CKR_OK) {
 printf("Error FindObjects: 0x%X\n", rc); return rc;
  }
  rc = C_FindObjectsFinal(hSession);
  if (rc != CKR_OK) {
 printf("Error FindObjectsFinal: 0x%X\n", rc); return rc;
  }
}
```

Cryptographic operations

When you use your sample code with a static linked library you can access the APIs directly. View some openCryptoki code samples for procedures that perform cryptographic operations.

C_Encrypt (AES):

```
/*
 * AES encrypt
 */
CK_RV AESencrypt(CK_SESSION_HANDLE hSession,
   CK_BYTE_PTR pClearData,  CK_ULONG ulClearDataLen,
   CK_BYTE **pEncryptedData, CK_ULONG_PTR pulEncryptedDataLen) {
  CK_RV rc;
  CK_MECHANISM myMechanism = {CKM_AES_CBC_PAD, "01020304050607081122334455667788", 16};
  CK_MECHANISM_PTR pMechanism = &myMechanism;
  CK_OBJECT_HANDLE hKey;
  getKey("My_AES_Key", sizeof("My_AES_Key"), &hKey, hSession);
  rc = C_EncryptInit(hSession, pMechanism, hKey);
  if (rc != CKR_OK) {
 printf("Error initializing encryption: 0x%X\n", rc);
 return rc;
  }
  rc = C_Encrypt(hSession, pClearData, ulClearDataLen,
   NULL, pulEncryptedDataLen);
  if (rc != CKR_OK) {
 printf("Error during encryption (get length): %x\n", rc);
 return rc;
  }
  *pEncryptedData = (CK_BYTE *)malloc(*pulEncryptedDataLen * sizeof(CK_BYTE));

  rc = C_Encrypt(hSession, pClearData, ulClearDataLen,
   *pEncryptedData, pulEncryptedDataLen);
  if (rc != CKR_OK) {
 printf("Error during encryption: %x\n", rc);
 return rc;
  }
  printf("Encrypted data: ");
  CK_BYTE_PTR tmp = *pEncryptedData;
  for (count = 0; count < *pulEncryptedDataLen; count++, tmp++) {
 printf("%X", *tmp);
  }
  printf("\n");

  return CKR_OK;
}
```

C_Decrypt (AES):

```
/*
 * AES decrypt
 */
CK_RV AESdecrypt(CK_SESSION_HANDLE hSession,
   CK_BYTE_PTR pEncryptedData, CK_ULONG ulEncryptedDataLen,
   CK_BYTE **pClearData, CK_ULONG_PTR pulClearDataLen) {
  CK_RV rc;
  CK_MECHANISM myMechanism = {CKM_AES_CBC_PAD, "01020304050607081122334455667788", 16};
  CK_MECHANISM_PTR pMechanism = &myMechanism;
  CK_OBJECT_HANDLE hKey;
  getKey("My_AES_Key", sizeof("My_AES_Key"), &hKey, hSession);
  rc = C_DecryptInit(hSession, pMechanism, hKey);
  if (rc != CKR_OK) {
 printf("Error initializing decryption: 0x%X\n", rc);
 return rc;
  }
  rc = C_Decrypt(hSession, pEncryptedData, ulEncryptedDataLen, NULL, pulClearDataLen);
  if (rc != CKR_OK) {
 printf("Error during decryption (get length): %x\n", rc);
 return rc;
  }
  *pClearData = malloc(*pulClearDataLen * sizeof(CK_BYTE));
  rc = C_Decrypt(hSession, pEncryptedData, ulEncryptedDataLen, *pClearData,
       pulClearDataLen);
  if (rc != CKR_OK) {
 printf("Error during decryption: %x\n", rc);
 return rc;
  }
  printf("Decrypted data: ");
  CK_BYTE_PTR tmp = *pClearData;
  for (count = 0; count < *pulClearDataLen; count++, tmp++) {
 printf("%c", *tmp);
  }
  printf("\n");
  return CKR_OK;
}
```

C_GenerateKeyPair (RSA):

```
/*
 * RSA key generate
 */
CK_RV generateRSAKeyPair(CK_SESSION_HANDLE hSession, CK_ULONG keySize,
    CK_OBJECT_HANDLE_PTR phPublicKey, CK_OBJECT_HANDLE_PTR phPrivateKey ) {
  CK_RV rc;
  CK_BBOOL true = TRUE;
  CK_BBOOL false = FALSE;
  CK_OBJECT_CLASS keyClassPub = CKO_PUBLIC_KEY;
  CK_OBJECT_CLASS keyClassPriv = CKO_PRIVATE_KEY;
  CK_KEY_TYPE keyTypeRSA = CKK_RSA;
  CK_ULONG modulusBits = keySize;
  CK_BYTE_PTR pModulus = malloc(sizeof(CK_BYTE)*modulusBits/8);
  CK_BYTE publicExponent[] = {1, 0, 1};
  CK_MECHANISM rsaKeyGenMech = {CKM_RSA_PKCS_KEY_PAIR_GEN, NULL_PTR, 0};
  CK_ATTRIBUTE pubKeyTempl[] = {
 {CKA_CLASS, &keyClassPub, sizeof(keyClassPub)},
 {CKA_KEY_TYPE, &keyTypeRSA, sizeof(keyTypeRSA)},
 {CKA_TOKEN, &true, sizeof(true)},
 {CKA_PRIVATE, &true, sizeof(true)},
 {CKA_ENCRYPT, &true, sizeof(true)},
 {CKA_VERIFY, &true, sizeof(true)},
 {CKA_WRAP, &true, sizeof(true)},
 {CKA_MODULUS_BITS, &modulusBits, sizeof(modulusBits)},
 {CKA_PUBLIC_EXPONENT, publicExponent, sizeof(publicExponent)},
 {CKA_LABEL, "My_Private_Token_RSA1024_PubKey",
 sizeof("My_Private_Token_RSA1024_PubKey")},
 {CKA_MODIFIABLE, &true, sizeof(true)},
  };
  CK_ATTRIBUTE privKeyTempl[] = {
 {CKA_CLASS, &keyClassPriv, sizeof(keyClassPriv)},
 {CKA_KEY_TYPE, &keyTypeRSA, sizeof(keyTypeRSA)},
 {CKA_EXTRACTABLE, &true, sizeof(true)},
 {CKA_TOKEN, &true, sizeof(true)},
 {CKA_PRIVATE, &true, sizeof(true)},
 {CKA_SENSITIVE, &true, sizeof(true)},
 {CKA_DECRYPT, &true, sizeof(true)},
 {CKA_SIGN, &true, sizeof(true)},
 {CKA_UNWRAP, &true, sizeof(true)},
 {CKA_LABEL, "My_Private_Token_RSA1024_PrivKey",
 sizeof("My_Private_Token_RSA1024_PrivKey")},
 {CKA_MODIFIABLE, &true, sizeof(true)},
  };
  rc = C_GenerateKeyPair(hSession, &rsaKeyGenMech ,
    &pubKeyTempl, sizeof(pubKeyTempl)/sizeof (CK_ATTRIBUTE),
    &privKeyTempl, sizeof(privKeyTempl)/sizeof (CK_ATTRIBUTE),
    phPublicKey, phPrivateKey);
  if (rc != CKR_OK) {
 printf("Error generating RSA keys: %x\n", rc);
 return rc;
  }
  printf("RSA Key generation successful.\n");
}
```

C_Encrypt (RSA):

```
/*
 * RSA encrypt
 */
CK_RV RSAencrypt(CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hKey,
   CK_BYTE_PTR pClearData, CK_ULONG ulClearDataLen,
   CK_BYTE **pEncryptedData, CK_ULONG_PTR pulEncryptedDataLen) {
  CK_RV rc;
  CK_MECHANISM rsaMechanism = {CKM_RSA_PKCS, NULL_PTR, 0};
  rc = C_EncryptInit(hSession, rsaMechanism, hKey);
  if (rc != CKR_OK) {
 printf("Error initializing RSA encryption: %x\n", rc);
 return rc;
  }
  rc = C_Encrypt(hSession, pClearData, ulClearDataLen,
   NULL, pulEncryptedDataLen);
  if (rc != CKR_OK) {
 printf("Error during RSA encryption: %x\n", rc);
 return rc;
  }

  *pEncryptedData = (CK_BYTE *)malloc(rsaKeyLen * sizeof(CK_BYTE));
  rc = C_Encrypt(hSession, pClearData, ulClearDataLen,
   *pEncryptedData, pulEncryptedDataLen);
  if (rc != CKR_OK) {
 printf("Error during RSA encryption: %x\n", rc);
 return rc;
  }

  printf("Encrypted data: ");
  CK_BYTE_PTR tmp = *pEncryptedData;
  for (count = 0; count < *pulEncryptedDataLen; count++, tmp++) {
 printf("%X", *tmp);
  }
  printf("\n");
  return CKR_OK;
}
```

C_Decrypt (RSA):

```
/*
 * RSA decrypt
 */
CK_RV RSAdecrypt(CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hKey,
   CK_BYTE_PTR pEncryptedData, CK_ULONG ulEncryptedDataLen,
   CK_BYTE **pClearData, CK_ULONG_PTR pulClearDataLen) {
  CK_RV rc;
  CK_MECHANISM rsaMechanism = {CKM_RSA_PKCS, NULL_PTR, 0};
  rc = C_DecryptInit(hSession, rsaMechanism, hKey);
  if (rc != CKR_OK) {
 printf("Error initializing RSA decryption: %x\n", rc);
 return rc;
  }
  rc = C_Decrypt(hSession, pEncryptedData, ulEncryptedDataLen,
   NULL, pulClearDataLen);
  if (rc != CKR_OK) {
 printf("Error during RSA decryption: %x\n", rc);
 return rc;
  }

  *pClearData = malloc(rsaKeyLen*sizeof(CK_BYTE));
  rc = C_Decrypt(hSession, pEncryptedData, ulEncryptedDataLen,
   *pClearData, pulClearDataLen);
  if (rc != CKR_OK) {
 printf("Error during RSA decryption: %x\n", rc);
 return rc;
  }
  printf("Decrypted data: ");
  CK_BYTE_PTR tmp = *pClearData;
  for (count = 0; count < *pulClearDataLen; count++, tmp++) {
 printf("%c", *tmp);
  }
  printf("\n");
  return CKR_OK;
}
```

Accessibility

Accessibility features help users who have a disability, such as restricted mobility or limited vision, to use information technology products successfully.

Documentation accessibility

The Linux on System z publications are in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when you use the PDF file and want to request a Web-based format for this publication, use the Readers' Comments form in the back of this publication, send an email to eservdoc@de.ibm.com, or write to:

IBM Deutschland Research & Development GmbH
Information Development
Department 3282
Schoenaicher Strasse 220
71032 Boeblingen
Germany

In the request, be sure to include the publication number and title.

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

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility at

`www.ibm.com/able`

Notices

This information is for planning purposes only. The information herein is subject to change before the products described become available.

Trademarks

Glossary

Advanced Encryption Standard (AES)
A data encryption technique that improved upon and officially replaced the Data Encryption Standard (DES). AES is sometimes referred to as Rijndael, which is the algorithm on which the standard is based.

asymmetric cryptography
Synonym for public key cryptography..

Central Processor Assist for Cryptographic Function (CPACF)
Hardware that provides support for symmetric ciphers and secure hash algorithms (SHA) on every central processor. Hence the potential encryption/decryption throughput scales with the number of central processors in the system.

Chinese-Remainder Theorem (CRT)
A mathematical problem described by Sun Tsu Suan-Ching using the remainder from a division operation.

Cipher Block Chaining (CBC)
A method of reducing repetitive patterns in cipher-text by performing an exclusive-OR operation on each 8-byte block of data with the previously encrypted 8-byte block before it is encrypted.

Cipher block length
The length of a block that can be encrypted or decrypted by a symmetric cipher. Each symmetric cipher has a specific cipher block length.

clear key
Any type of encryption key not protected by encryption under another key.

CPACF instructions
Instruction set for the CPACF hardware.

Crypto Express4S (CEX4S)
Successor to the Crypto Express3 feature. The PCIe adapter on a CEX4S feature can be configured in three ways: Either as cryptographic accelerator (CEX4A), or as CCA coprocessor (CEX4C) for secure key encrypted transactions, or in EP11 coprocessor mode (CEX4P) for exploiting Enterprise PKCS #11 functionality.

A CEX4P only supports secure key mode.

electronic code book mode (ECB mode)
A method of enciphering and deciphering data in address spaces or data spaces. Each 64-bit block of plain-text is separately enciphered and each block of the cipher-text is separately deciphered.

libica Library for IBM Cryptographic Architecture.

master key (MK)
In computer security, the top-level key in a hierarchy of key-encrypting keys.

Mode of operation
A schema describing how to apply a symmetric cipher to encrypt or decrypt a message that is longer than the cipher block length. The goal of most modes of operation is to keep the security level of the cipher by avoiding the situation where blocks that occur more than once will always be translated to the same value. Some modes of operations allow handling messages of arbitrary lengths.

modulus-exponent (Mod-Expo)
A type of exponentiation performed using a modulus.

public key cryptography
In computer security, cryptography in which a public key is used for encryption and a private key is used for decryption. Synonymous with asymmetric cryptography.

Rivest-Shamir-Adleman (RSA)
An algorithm used in public key cryptography. These are the surnames of the three researchers responsible for creating this asymmetric or public/private key algorithm.

Secure Hash Algorithm (SHA)
An encryption method in which data is encrypted in a way that is mathematically impossible to reverse. Different data can possibly produce the same hash value, but there is no way to use the hash value to determine the original data.

secure key
: A key that is encrypted under a master key. When using a secure key, it is passed to a cryptographic coprocessor where the coprocessor decrypts the key and performs the function. The secure key never appears in the clear outside of the cryptographic coprocessor.

symmetric cryptogrphy
: An encryption method that uses the same key for encryption and decryption. Keys of symmetric ciphers are private keys.

zcrypt device driver
: Kernel device driver to access Crypto Express adapters. Formerly, a monolithic module called **`z90crypt`**. Today, it consists of multiple modules that are implicitly loaded when loading the **`ap`** main module of the device driver.

Index

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

Linux on System z
Exploiting Enterprise PKCS #11 using openCryptoki 3.1

Publication No. SC34-2713-00

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

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

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

Comments:

Thank you for your support.

Submit your comments using one of these channels:
- Send your comments to the address on the reverse side of this form.
- Send your comments via email to: eservdoc@de.ibm.com

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

Name	Address
Company or Organization	
Phone No.	Email address

Readers' Comments — We'd Like to Hear from You
SC34-2713-00

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Research & Development GmbH
Information Development
Department 3282
Schoenaicher Strasse 220
71032 Boeblingen
Germany

Fold and Tape | **Please do not staple** | Fold and Tape

SC34-2713-00

Cut or Fold
Along Line

SC34-2713-00