4767 PCIe Cryptographic Coprocessor CCA Support Program Installation Manual
Fourth edition (April 2019)

This edition applies to:

- IBM 4767 PCIe Cryptographic Coprocessor (hardware security module)
- Release 5.5 of the licensed CCA Cryptographic Coprocessor Support Program for the following 64-bit operating systems:
  - Red Hat Enterprise Linux Server 7.5 (RHEL 7.5)
  - SUSE (a Micro Focus company) Linux Enterprise Server 12 Service Pack 3 (SLES 12 SP3)
- Release 5.4 of the licensed CCA Cryptographic Coprocessor Support Program for the following 64-bit operating systems:
  - Microsoft Windows Server 2016 Version 1607
  - Red Hat Enterprise Linux Server 7.5 (RHEL 7.5)
  - SUSE Linux Enterprise Server 12 Service Pack 3 (SLES 12 SP3)
- Release 5.3 of the licensed CCA Cryptographic Coprocessor Support Program for the following 64-bit operating systems:
  - Microsoft Windows Server 2012 R2
  - Red Hat Enterprise Linux Server 7.2 (RHEL 7.2)
  - SUSE (a Micro Focus Company) Linux Enterprise Server Service Pack 1 (SLES 12 SP1)

Each release is designed to provide software and firmware support for an IBM 4767 PCIe Cryptographic Coprocessor installed into a computer running the accompanying operating system.

Changes are made periodically to the information herein. Before using this publication in connection with the operation of IBM systems, please check the product website at https://www.ibm.com/security/cryptocards for an updated version of this publication.

This and other publications that are related to the IBM 4767 coprocessor can be obtained in PDF format from the Library page at the product website.

Readers comments can be communicated to IBM by contacting the Crypto team at crypto@us.ibm.com.

IBM can use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright IBM Corporation 2016, 2019.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
Contents

Figures .................................................. v

Tables ................................................... vii

About this publication ............................... ix

Chapter 1. Installation process overview. ..................... 1
  Summary .................................................. 1

Chapter 2. Obtaining coprocessor hardware and software package .................................................. 3
  Ordering and installing coprocessor hardware .................................................. 3
    Ordering coprocessors .................................................. 3
    Ordering optional smart card readers .................................................. 3
    Maintaining coprocessor batteries .................................................. 3
    Installing IBM 4767 hardware .................................................. 4
    Downloading the software package .................................................. 4

Chapter 3. Installing and configuring the Support Program. ..................... 7
  Support Program components .................................................. 7
  Hardware and software requirements .................................................. 7
  Hardware and software installation .................................................. 8
    Installing and uninstalling coprocessor host software .................................................. 8
    Installing the Support Program .................................................. 9
  Configuring the Support Program .................................................. 10
    Setting CCA Support Program and Linux file permissions .................................................. 11
    Special security considerations .................................................. 11

Chapter 4. Loading and unloading coprocessor firmware ..................... 13
  Loading coprocessor firmware .................................................. 13
  Unloading coprocessor firmware and zeroizing the node .................................................. 18

Chapter 5. Using the CNM and CNI utilities to manage the cryptographic node ............. 19
  Overview .................................................. 19
  Starting the CCA Node Management utility .................................................. 20
  Sample scenarios for using the CNM and CNI utilities .................................................. 20
  Using the administrative functions of the CNM utility .................................................. 24
  Creating and managing access-control data .................................................. 26
    Access-control overview .................................................. 26
    Initializing the access-control system .................................................. 27
    Changing the initial state of the access-control system .................................................. 27
    Defining a role .................................................. 28
    Modifying existing roles .................................................. 28
    Defining a user profile .................................................. 29
    Modifying existing user profiles .................................................. 30
  Managing cryptographic keys .................................................. 31
    Managing the master keys .................................................. 31
    Managing key storage .................................................. 35
    Creating and storing primary DES KEKs .................................................. 37
    Establishing other nodes using CNI .................................................. 38

Chapter 6. Building applications to use with the CCA API ..................... 41
  Overview .................................................. 41
  Calling CCA verbs in C program syntax .................................................. 41

© Copyright IBM Corp. 2016, 2019
<table>
<thead>
<tr>
<th>Chapter 7. Building Java applications to use with the CCA JNI</th>
<th>45</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>45</td>
</tr>
<tr>
<td>Compiling and running a CCA JNI program</td>
<td>45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix A. Initial commands for DEFAULT role.</th>
<th>47</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Appendix B. Device driver error codes</th>
<th>49</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Appendix C. DES and PKA master-key cloning</th>
<th>51</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master-key cloning procedure</td>
<td>51</td>
</tr>
<tr>
<td>Access-control considerations when cloning</td>
<td>56</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix D. Threat considerations for a digital-signing server</th>
<th>59</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Appendix E. CCA sample code</th>
<th>69</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Appendix F. Coprocessor Load Utility reference</th>
<th>71</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Notices</th>
<th>77</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Trademarks</th>
<th>79</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>List of abbreviations and acronyms</th>
<th>81</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Glossary</th>
<th>83</th>
</tr>
</thead>
</table>

| Index | 89 |
Figures

1. Cloning-information worksheet ........................................ 53
## Tables

1. Activity checklist, CCA Cryptographic Coprocessor Support Program installation ........................................... 2
2. Part number for battery-replacement kit ........................................................................................................ 4
3. Default directory list files .................................................................................................................................. 10
4. Program segment contents .................................................................................................................................. 14
5. Class-key file for use with the CLU VA command in releases before Release 5.4 .................................................. 17
6. Initial commands for DEFAULT role .................................................................................................................. 47
7. Device driver error codes in the class X'8xxxxxxx' .......................................................................................... 49
8. Master-key cloning procedure phase overview .................................................................................................. 51
9. Cloning responsibilities, profiles and roles ........................................................................................................ 52
10. Master-key cloning procedure: establish SA node ............................................................................................ 53
11. Master-key cloning procedure: establish source (CSS) node ............................................................................ 54
12. Master-key cloning procedure: establish CSR node, clone master key ............................................................... 55
13. CCA commands related to master-key cloning .................................................................................................. 57
14. Threat considerations for a digital-signing server ............................................................................................. 59
15. Organization of memory segments .................................................................................................................... 71
16. Summary of CLU commands .......................................................................................................................... 73
17. CLU loader commands ....................................................................................................................................... 75
18. Typical CLU system status response ................................................................................................................ 75
19. Summary of shipped CLU files ........................................................................................................................ 76
About this publication

This installation manual describes Release 5.3 or later of the IBM® Common Cryptographic Architecture (CCA) Support Program (Support Program) for the IBM 4767 PCIe Cryptographic Coprocessor (hardware security module or HSM) installed on a Linux operating system. The Support Program includes device drivers, utilities, and the CCA coprocessor code.

You can obtain Support Program releases to use with the following 64-bit operating systems:

- Microsoft Windows Server
  - 2016 Version 1607
  - 2012 R2
- Red Hat Enterprise Linux Server (RHEL)
  - RHEL 7.5
  - RHEL 7.2
- SUSE (a Micro Focus Company) Linux Enterprise Server (SLES)
  - SLES 12 Service Pack 3 (SP3)
  - SLES 12 Service Pack 1 (SP1)

Use this manual to help with the following tasks:

- Obtain the Support Program and coprocessor firmware through the Internet.
- Load the software onto a host computer and the firmware into the coprocessors.
- Use the utilities that are supplied with the Support Program to:
  - Load the coprocessor function-control vector (FCV).
  - Initialize one or more coprocessors.
  - Create and manage access-control data.
  - Create master keys and primary key-encrypting keys (KEKs).
  - Manage key storage at the cryptographic node.
  - Create node-initialization file lists to set up and configure other cryptographic nodes.
- Link your application software to the CCA library.
- Obtain guidance for security consideration in application development and operational practices.

Audience

The audience for this publication includes:

- System administrators who install the host software and coprocessor firmware.
- Security officers responsible for the coprocessor access-control system.
- System programmers and application programmers who determine how the software is to be used.

Organization of this publication

- **Chapter 1, “Installation process overview,” on page 1** summarizes the installation and the operation of the CCA Cryptographic Coprocessor Support Program.
- **Chapter 2, “Obtaining coprocessor hardware and software package,” on page 3** describes how to obtain the cryptographic coprocessor hardware and the CCA Cryptographic Coprocessor Support Program software package.
- **Chapter 3, “Installing and configuring the Support Program,” on page 7** describes how to install the software onto the host computer.
- **Chapter 4, “Loading and unloading coprocessor firmware,” on page 13** describes how to load the operating system and the CCA firmware into the cryptographic coprocessor.
Chapter 5, “Using the CNM and CNI utilities to manage the cryptographic node,” on page 19 describes how to use the CCA Node Management and the CCA Node Initialization utilities to set up and manage cryptographic nodes.

Chapter 6, “Building applications to use with the CCA API,” on page 41 explains how to build applications for CCA, and how to link them to CCA library.

Chapter 7, “Building Java applications to use with the CCA JNI,” on page 45 explains how to build, compile, and run Java™ applications to use with the CCA Support Program.

Appendix A, “Initial commands for DEFAULT role,” on page 47 details the permissions granted to the DEFAULT role when the access-control system is initialized.

Appendix B, “Device driver error codes,” on page 49 provides error-code information that can be observed when operating the CLU utility.

Appendix C, “DES and PKA master-key cloning,” on page 51 provides a procedure for DES and PKA master-key cloning.

Appendix D, “Threat considerations for a digital-signing server,” on page 59 addresses threats to consider when employing the coprocessor and the CCA Support Program in a digital-signing application.

Appendix E, “CCA sample code,” on page 69 describes how to download CCA sample code from the product website.

“Notices” on page 77 provides legal notices.

A list of abbreviations and acronyms, a glossary, and an index are included.

IBM 4767 PCIe Cryptographic Coprocessor publications

For availability of these publications, check the PCIe Cryptographic Coprocessor Library page of the product website at [https://www.ibm.com/security/cryptocards](https://www.ibm.com/security/cryptocards). From the website, you can download, view, and print publications available in the Adobe Acrobat portable document format (PDF):

- IBM CCA Basic Services Reference and Guide for the IBM 4767 and IBM 4765 PCIe Cryptographic Coprocessors
- IBM 4767 PCIe Cryptographic Coprocessor Installation Manual
- IBM 4767 PCIe Cryptographic Coprocessor Smart Card User’s Guide

Software package for Linux or Windows

1. The IBM 4767 PCIe Cryptographic Coprocessor offering

   For instructions on how to download the Linux or Windows software package for the IBM 4767 PCIe Cryptographic Coprocessor offering, click the [HSM PCIeCC2](https://www.ibm.com/security/cryptocards) link in the site navigation section at [https://www.ibm.com/security/cryptocards](https://www.ibm.com/security/cryptocards) then click the Download software link in the HSM 4767 navigation section.

2. A supported operating system distribution

   The product requires a supported operating system distribution. For the latest information on supported distributions, click the [HSM PCIeCC2](https://www.ibm.com/security/cryptocards) link in the site navigation section, at [https://www.ibm.com/security/cryptocards](https://www.ibm.com/security/cryptocards) then click the Releases link in the secondary navigation section.

3. Linux device driver prerequisites

   An appropriate compiler and its associated development tools need to be installed from the Linux distribution prior to running the installer. The installer will attempt to build the kernel module for the device driver.

   Linux kernel module binaries are compiled for a specific kernel version and variant (PAE and so forth). Dependent upon the Linux distribution, kernel-source and kernel-syms RPMS or similar access to kernel source may also be required. The available kernel source must match the current kernel in both version and variant.
Note: See the installed readme.txt file if the compilation of the device driver fails.

4. CNM, CNI, and JNI prerequisites

Version 1.7.0 of a Java Runtime Environment (JRE) or Java Development Kit (JDK) is required to run:

- The CCA Node Management (CNM) utility.
- The CCA Node Initialization (CNI) utility.
- A CCA Java application using the Java Native Interface (JNI), created by an end-user (this requires a JDK).

A JRE might not be installed by default during a typical Linux or Windows installation.

Use this link to download the IBM Java Runtime Environment:


For Linux users, click on the Java 7 link to download JRE version 1.7.0. For Windows users, Java is packaged with Eclipse. After installing Eclipse, then Java 7 should be installed.
Chapter 1. Installation process overview

This chapter summarizes the installation and operation procedures discussed in this manual. A checklist is provided for you to use while installing the IBM 4767 PCIe Cryptographic Coprocessor and the IBM Common Cryptographic Architecture (CCA) Cryptographic Coprocessor Support Program (Support Program). See Table 1 on page 2. The IBM 4767 is a hardware security module (HSM). It is identified on the product website as HSM 4767.

Summary

The Support Program consists of several components, and includes:

- Device drivers and an operating system for the coprocessor hardware
- Support for the CCA security application program interface (API)
- A function-control vector (FCV)
  
  An FCV is a digitally signed value provided by IBM. Its use originated to enable the CCA application within the coprocessor to yield a level of cryptographic service consistent with applicable cryptographic implementation import and export regulations.
- Utility applications that run on the host machine which has at least one coprocessor (HSM) installed

See Chapter 3, “Installing and configuring the Support Program,” on page 7 for the supported operating system versions and prerequisite software.

To install the Support Program components and to establish a CCA cryptographic node, perform the following steps described in this manual:

1. **Obtain the hardware, firmware, and software.** Chapter 2, “Obtaining coprocessor hardware and software package,” on page 3 describes how to order the hardware from IBM, how to download the software package through the Internet, and how to install the downloaded files. Note that it is necessary to install the software package prior to installing the coprocessor hardware.

2. **Install the software on the host.** Chapter 3, “Installing and configuring the Support Program,” on page 7 describes how to install the software downloaded from the Internet onto the coprocessor host computer.

3. **Load the coprocessor firmware.** Chapter 4, “Loading and unloading coprocessor firmware,” on page 13 describes how to load both the embedded operating system and CCA application program.

4. **Set up the cryptographic node.** The utilities described in Chapter 5, “Using the CNM and CNI utilities to manage the cryptographic node,” on page 19 includes setup and management functions needed to:
  - Choose a specific coprocessor
  - Initialize a CCA node
  - Log on and off of a CCA node
  - Synchronize the clock-calendars
  - Obtain status information
  - Load the FCV
  - Create and manage access-control data
  - Manage the coprocessor master keys
  - Manage primary KEKs
  - Manage key storage
  - Create lists (“scripts”) for the CCA Node Initialization (CNI) utility
A CCA cryptographic node can be established by using the utilities provided with the Support Program, linking your C application programs to the CCA API, or compiling your Java Native Interface (JNI) CCA application programs. The access control and other setup requirements imposed by application software that you plan to use with the IBM 4767 should also be verified.

5. **Link application programs to the CCA library.** Chapter 6, “Building applications to use with the CCA API,” on page 41 describes how to build applications for CCA and how to link them to the CCA library.

Table 1 provides a checklist of activities to perform and the order to perform them when installing the Support Program.

**Table 1. Activity checklist, CCA Cryptographic Coprocessor Support Program installation**

<table>
<thead>
<tr>
<th>Step</th>
<th>Task</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Place an order with an IBM Z Systems sales representative.</td>
<td>“Ordering coprocessors” on page 3</td>
</tr>
<tr>
<td>2</td>
<td>Receive coprocessor hardware.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Download the coprocessor firmware and the host Support Program for your Linux operating system.</td>
<td>“Downloading the software package” on page 4</td>
</tr>
<tr>
<td>5</td>
<td>Install coprocessor hardware.</td>
<td>“Installing IBM 4767 hardware” on page 4</td>
</tr>
<tr>
<td>6</td>
<td>Load coprocessor firmware.</td>
<td>Chapter 4, “Loading and unloading coprocessor firmware,” on page 13</td>
</tr>
<tr>
<td>7</td>
<td>Set up a CCA test node. Review the first pages in Chapter 5, “Using the CNM and CNI utilities to manage the cryptographic node,” on page 19, and then set up a test node.</td>
<td>“Establishing a node in a test environment” on page 21</td>
</tr>
<tr>
<td>8</td>
<td>Run test programs that utilize the CCA library.</td>
<td>Chapter 6, “Building applications to use with the CCA API,” on page 41 or Chapter 7, “Building Java applications to use with the CCA JNI,” on page 45</td>
</tr>
</tbody>
</table>
Chapter 2. Obtaining coprocessor hardware and software package

The IBM CCA Cryptographic Coprocessor Support Program (Support Program) is available for download through the Internet by going to the product website at [https://www.ibm.com/security/cryptocards](https://www.ibm.com/security/cryptocards), then clicking on the **How to order** link in the site navigation section.

This chapter describes:

- How to order and install coprocessor hardware
- How to download the software package that contains the host software and coprocessor firmware

### Ordering and installing coprocessor hardware

The following sections describe how to:

- Order coprocessors
- Order optional smart card readers
- Maintain coprocessor batteries
- Install your coprocessor

### Ordering coprocessors

Currently there is only one model of the IBM 4767 PCIe Cryptographic Coprocessor, namely the Model 2. The IBM 4767 Model 2 is ordered from IBM as a Z Systems machine type-model (4767-002). The coprocessor is an adapter that can be installed in an x86 server. The slots must accept a standard height, half-length PCIe adapter card. If you encounter difficulty, contact IBM through the Support page at the [https://www.ibm.com/security/cryptocards](https://www.ibm.com/security/cryptocards) product website. IBM will endeavor to assist in problem determination and resolution, but makes no commitment that problems with other system types can be resolved.

The Support Program logically supports from one to eight coprocessors per system. The physical limit is determined based on the number of card slots available on the server.

To order the coprocessor hardware, contact your local IBM System z® sales representative, and order the model and part numbers that you have chosen. For available parts, see [Table 2 on page 4](#).

### Ordering optional smart card readers

IBM offers optional smart card support in the form of a Smart Card Utility Program (SCUP) and enhanced smart card feature for CNM that can be optionally installed when CNM is installed. For detailed information on smart card support, including how to order the optional smart card hardware, see *IBM 4767 PCIe Cryptographic Coprocessor Smart Card User’s Guide*. This document is available on the Library page of the HSM 4767.

### Maintaining coprocessor batteries

The coprocessor has two on-board batteries. These batteries provide critical backup power to a small quantity of internal memory, the clock-calendar, and the tamper-detection circuitry.

**Important:** It is imperative that the coprocessor always has batteries installed with sufficient stored energy to power the coprocessor during its entire useful life. When the coprocessor is not in a powered-on system and the batteries either fail or are removed from the coprocessor, the unit will zeroize and be rendered permanently inoperable. **There is no recovery from this situation.**
The useful life of the batteries mounted in an IBM 4767 that is continuously powered on is nearly the same as the useful shelf life of the batteries. The actual life of the batteries is anticipated to be in excess of seven years. IBM recommends changing the coprocessor batteries as a planned maintenance activity every five years. Before changing batteries, ensure that the replacement batteries measure 3.0V or higher and have not been in inventory for a long period of time.

Note: Each battery has a date-of-manufacture code stamped (not printed) on it in the format wwyy, where ww is the 2-digit week, and yy is the 2-digit year.

Replacement batteries are available through IBM so that you can maintain the functionality of the coprocessor. Table 2 shows the available battery-replacement part numbers.

Table 2. Part number for battery-replacement kit

<table>
<thead>
<tr>
<th>FRU PN</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>45D5803</td>
<td>Battery-replacement kit. Includes two replacement batteries, one battery-attention label, and one battery tray with connecting wires.</td>
</tr>
</tbody>
</table>

Special procedures are required to safely replace coprocessor batteries. See "Replacing coprocessor batteries" in the IBM 4767 PCIe Cryptographic Coprocessor Installation Manual.

Installing IBM 4767 hardware

The IBM 4767 is installed in a manner similar to other PCIe boards. Follow the process described in the IBM 4767 PCIe Cryptographic Coprocessor Installation Manual. Note that it is necessary to install the software package prior to installing the coprocessor hardware.

Ensure that the coprocessor batteries are never removed except as outlined in the battery-replacement procedure in the IBM 4767 PCIe Cryptographic Coprocessor Installation Manual. The coprocessor is certified at the factory. If it ever detects tampering, or if both battery power and system power are simultaneously removed, the factory certification is zeroized and the coprocessor is rendered non-functional. There is no recovery from this situation.

It is possible to inadvertently cause a tamper event if someone causes some of the coprocessor circuitry to short-circuit. Remember that the batteries on the coprocessor supply power to tamper sensors. If in handling the coprocessor someone causes a short circuit in this circuitry, this could result in a tamper event. This is very unlikely to occur, but be careful when installing the coprocessor to keep the circuitry on the board from contacting conductive portions of the host machine or other installed boards.

Downloading the software package

The Support Program software and coprocessor firmware are available for download through the Internet. Go to the product website at https://www.ibm.com/security/cryptocards, click HSM PCIeCC2 in the site navigation section, and then click the Download software link in the HSM 4767 navigation section. You will need your customer number and the IBM Tracking Serial Number of the coprocessor to complete the download. The serial number can be found on the black label affixed to the edge of the coprocessor.

Tip: To receive the latest version of the Support Program, wait to download the software package until you have received your coprocessor. See Step 3 of Table 1 on page 2. Check the website for any HSM 4767 available software and firmware updates.
From the software-package selection page, select the CCA software and firmware required by operating-system platform, release level, and Support Program. You are prompted to sign in or complete a registration procedure. After signing in with a universal IBM user ID, a page is presented from which to choose an offering to download.

If the Support Program is to be used on multiple host computers, copy the downloaded files to the other hosts.

At this point the Support Program can be installed. See Chapter 3, “Installing and configuring the Support Program,” on page 7.
Chapter 3. Installing and configuring the Support Program

After completing the steps as described in Chapter 2, “Obtaining coprocessor hardware and software package,” on page 3, follow the procedures in this chapter to install and configure the CCA Cryptographic Coprocessor Support Program onto the coprocessor host computer. (Loading firmware into the coprocessor is described in Chapter 4, “Loading and unloading coprocessor firmware,” on page 13 and initializing the CCA application within the coprocessor is described in Chapter 5, “Using the CNM and CNI utilities to manage the cryptographic node,” on page 19.)

This chapter:
- Lists the Support Program components that you are installing
- Lists hardware and software requirements
- Describes how to install the coprocessor host software, the coprocessor, and the coprocessor firmware host software
- Describes how to uninstall the coprocessor host software
- Describes how to configure coprocessor host software
- Discusses special security considerations

Support Program components

The procedures in this chapter describe how to install the following Support Program components onto the host computer:
- Device drivers for the IBM 4767 PCIe Cryptographic Coprocessor
- The Linux shared library or Windows DLL necessary to link the CCA application program interface (API) to the coprocessor driver
- The Coprocessor Load Utility (CLU) and firmware files necessary to load the operating system and the CCA application program into the coprocessor. CLU is described in Chapter 4, “Loading and unloading coprocessor firmware,” on page 13.
- The CCA Node Management (CNM) utility necessary to load the function-control vector (FCV) into the coprocessor and to set up a cryptographic node. CNM is described in Chapter 5, “Using the CNM and CNI utilities to manage the cryptographic node,” on page 19.
- The Smart Card Utility Program (SCUP) feature.

This chapter also describes how to uninstall the Support Program.

Hardware and software requirements

The hardware and software requirements for running the CCA Support Program are described below.

Hardware
- The IBM 4767 PCIe Cryptographic Coprocessor must be installed in an x86 architecture server.
- The coprocessor can only be installed in an x86 server that has at least one PCIe slot capable of accepting a standard height, half-length PCIe card and that meets the power and environmental requirements specified below.

Optional hardware
- There is an option to install smart card readers that work with the CNM utility of the coprocessor. For detailed information, see IBM 4767 PCIe Cryptographic Coprocessor Smart Card User’s Guide. This document is available on the Library page of the HSM 4767.
Hardware and software installation

Installing the coprocessor involves the following steps:
1. Install the host software as described in this chapter.
2. Install the hardware as described in the IBM 4767 PCIe Cryptographic Coprocessor Installation Manual.
3. Load coprocessor firmware as described in Chapter 4, “Loading and unloading coprocessor firmware,” on page 13.
4. Setting CCA Support Program and file permissions as described in this chapter.

Note that all components of the installed CCA package might not indicate the same version number. The version number for a component is updated only when the component is changed for a release. Unchanged components retain the version number from the last time that they were modified. For example, the Cryptographic Coprocessor Support Software might be at a different level from the coprocessor device driver.

Installing and uninstalling coprocessor host software

This section describes considerations for installing and uninstalling the hardware and software for the IBM Common Cryptographic Architecture (CCA) Support Program.

Backing up key storage files

The CCA software does not provide any automatic backup of key storage files. The default directory for key storage is:

Linux  /opt/ibm/4767/keys
Windows  \Program Files\IBM\4767\keys

The keys in these key storage files are encrypted by the appropriate AES, APKA, DES, or PKA master key securely contained within the coprocessor. To use these encrypted keys with another coprocessor, the same master key must be loaded into the other coprocessor. For DES and PKA keys, master-key cloning procedures can be used to accomplish the replication. The cloning of AES and APKA master keys is not supported.

Although the uninstall and install processes never intentionally delete existing key storage files, it is always good practice to create a backup copy of these files just prior to performing an uninstall. Use the backup copy to restore the key storage files following the installation of the Support Program. For this to work, the master keys used to encrypt the keys in key storage must not have been changed in the coprocessor during the installation process.

Note: Use of a CLU file of the form surrender_ownership_seg2_xipz_.#.#.#.clu zeroizes any preexisting master keys, retained keys, roles, and profiles. In this case, do not attempt to restore any key storage files.

Uninstalling a preexisting installation of predecessor IBM product

Having both the IBM 4765 and IBM 4767 installed at the same time is not supported. If applicable, uninstall the Support Program for the predecessor product “IBM 4765” prior to installing the IBM 4767 PCIe Cryptographic Coprocessor CCA Support Program.

Uninstalling a preexisting Support Program installation

Use the procedures in this section to uninstall a preexisting Support Program installation when any of the following are true:

• The installation is no longer needed.
A newer or older version needs to be installed.

Perform the following steps to uninstall the Support Program in Linux or Windows. The directory path shown is the default directory path.

1. Save a backup copy of your key storage files just before uninstalling the Support Program. The uninstall process should not delete or change key storage files, but creating a backup copy is good practice.
   - **Linux** /opt/ibm/4767/keys
   - **Windows** \Program Files\IBM\4767\keys

2. Linux users, log on as root (do not use the su command to switch user to root; su to root is not sufficient). Windows users log on as administrator.

3. Enter the command:
   - **Linux** /opt/ibm/4676/modify_installation
   - **Windows** \Program Files\IBM\4767\modify_installation

   The coprocessor device driver and other related information can be removed. The key storage directories and files remain.

**Installing the same or newer version of the Support Program**

When performing an install while an installation of the Support Program already exists, any attempt to install the same or newer version initiates execution of the preexisting code within the IBM4767_INSTALL_DIR directory. The existing installation can then be uninstalled. The newer version will not install until the existing version is uninstalled.

**Installing an older version of the Support Program**

If an installation of the Support Program already exists, then any attempt to install an older version will be blocked by the installer. An older version will not install until the existing version is uninstalled.

**Cleaning a corrupted installation**

If the installation directory is deleted without use of the uninstaller, then a new installation cannot proceed until the Support Program entries in the InstallAnywhere registry are removed.

The Linux InstallAnywhere registry entries are within /var/com.zerog.registry.xml.

**Installing the Support Program**

The CCA software does not provide any automatic backup of key storage files. Before installing the Support Program, back up these potentially critical files. See “Configuring the Support Program” on page 10 for information on the location of these files.

To install the Support Program:

1. Download the software package from the product website. Ensure that the permissions on the downloaded executable allow execution by root (Linux) or administrator (Windows).

2. Linux users, log on as root (do not use the su command to switch user to root; su to root is not sufficient). Windows users, log on as administrator.

3. Run the downloaded executable from a graphical desktop environment (GUI), and complete the prompts as needed. If a graphical desktop environment is not available, from the site navigation, click the HSM PCIeCC2 link, then click the FAQ link. Before running the executable, note the following:
• *Do not* attempt to install using the `-i` console option.

On Windows, if the message: Graphical installers are not supported by the VM appears, the installer will resort to console mode instead of GUI mode, but without a response file. Although no errors will be reported, the install will not be functional.

4. Log off and log on again as an appropriate user to activate changes in `/etc/profile`.

The installed `readme.txt` file contains the latest information about the product. If errors are encountered during the installation, check the `readme.txt` file for possible solutions.

---

**Configuring the Support Program**

When initializing a CCA key-storage dataset with either the CNM utility or with the Key_Storage_Initialization (CSNBKSI) verb, the DES, PKA, and AES key-storage dataset information entered during installation is used. These values are defined by the `CSUxxxDS` Linux or Windows environment variables, where `xxx` is DES, PKA, or AES.

<table>
<thead>
<tr>
<th>Verb</th>
<th>Linux default directory list file</th>
<th>Windows default directory list file</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES_KEY_RECORD_LIST</td>
<td><code>/opt/ibm/4767/keys/aeslist</code></td>
<td><code>\Program Files\IBM\4767\keys\aeslist</code></td>
</tr>
<tr>
<td>DES_KEY_RECORD_LIST</td>
<td><code>/opt/ibm/4767/keys/deslist</code></td>
<td><code>\Program Files\IBM\4767\keys\deslist</code></td>
</tr>
<tr>
<td>PKA_KEY_RECORD_LIST</td>
<td><code>/opt/ibm/4767/keys/pkalist</code></td>
<td><code>\Program Files\IBM\4767\keys\pkalist</code></td>
</tr>
</tbody>
</table>

Depending on your installation, the directory list file names shown might have been changed from their default values using the `CSUxxxLD` Linux environment variables, where `xxx` is DES, PKA, or AES.

Directory list files are created under the ownership of the environment of the user requesting the list service. **Linux:** Make sure that the files created keep the same group ID as your installation requires. This can also be accomplished by setting the “set-group-id-on-execution” bit on for each of the directory list file directories. See the g+s flags in the `chmod` command for full details. Not doing this could result in errors being returned by Key Record List verbs.

To assign a default CCA coprocessor, use the Linux `export` command or Windows `set` command to set the environment variable `CSU_DEFAULT_ADAPTER` to CRP0\(n\), where \(n = 1, 2, 3, \ldots, 8\), depending on which installed CCA coprocessor you want as the default. If this environment variable is not set when the first CCA verb of a process is called, the CCA software sets coprocessor CRP01 as the default. If this environment variable is set to an invalid value, you will get an error until the environment variable is set to a valid value or is unset.

To improve performance, the CCA implementation provides caching of key records obtained from key storage within the CCA host code. However, the host cache is unique for each host process. Caching can be a problem if different host processes access the same key record. An update to a key record caused by one process does not affect the contents of the key cache held for other processes. To avoid this problem, caching of key records within the key storage system can be suppressed so that all processes will access the most current key records. To suppress caching of key records, use the `export` command to set the environment variable `CSUCACHE` to ‘N’ or ‘n’ or to any string that begins with the character ‘N’ or ‘n’.

---

10 4767 PCIe Cryptographic Coprocessor CCA Support Program, Linux and Windows (April 2019)
Setting CCA Support Program and Linux file permissions

The CCA Support Program relies on file permissions at the group level to function correctly. This means that the users and administrators of the Support Program must have the correct group file permissions on the CCA shared objects, utilities, and key storage files and directories in order to be fully functional and run without errors.

**Note:** Key storage files and directories are defined as those files and directories that transverse the top level key-storage directory and below, including the top level directory. In the default configuration, the top level key-storage directory is:

**Linux**  
/opt/ibm/4767/keys

**Windows**  
\Program Files\IBM\4767\keys

**Linux:** For proper operation, the key storage files and directories must have a group ID of the application user group. Also, as a general rule, all key storage directories should have file permissions of 770 (drwxrws--) and be 'owned' by user root and group users. Once created, all key storage files should have file permissions of 660 (-rw-rw----). Note that the file mode creation mask of the current process affects the key-storage file creation permission modes by restricting the permissions from 660 to (660 & ~umask).

**Special security considerations**

The CNM utility provides a method to manage access control points. CNM obtains a list of access control points and their descriptions from the csuap.def file. If it becomes necessary to update this security-sensitive file, be sure to change the file permissions back to read-only.
Chapter 4. Loading and unloading coprocessor firmware

After completing the steps as described in Chapter 3, “Installing and configuring the Support Program,” on page 7, use the Coprocessor Load Utility (CLU) to load the coprocessor operating system and CCA application into the coprocessor.

This chapter includes instructions for using CLU to understand what coprocessors are installed and their status, and to install and uninstall the firmware that runs within the coprocessor.

If you obtain updates to the Support Program, use CLU to reload the necessary program segments. You can also load firmware from other vendors using CLU.

Note:
1. The file locations referenced in this chapter are the default directory paths.
2. Appendix B, “Device driver error codes,” on page 49 describes error codes returned by the coprocessor device driver. These are often presented in the form of a hexadecimal number such as X’8040xxxx’. You might encounter some of these error situations, especially when you first use the CLU utility and are less familiar with the product and its procedures.
3. The coprocessor function-control vector (FCV) is loaded by the CCA Node Management utility described in Chapter 5, “Using the CNM and CNI utilities to manage the cryptographic node,” on page 19.
4. For more details about CLU, such as coprocessor memory segments, CLU syntax, CLU commands, CLU files, and CLU return codes, see Appendix F, “Coprocessor Load Utility reference,” on page 71.

Loading coprocessor firmware

This section provides the procedures used in loading software into the coprocessor. Refer to the readme.txt file that accompanies the software package that you are installing for specific CLU file names (files with file extension “clu”). The readme.txt file might also provide additional information that augments or modifies these general procedures. For a summary of CLU files, see “Summary of shipped CLU files” on page 76.

You will be instructed in this section on how to perform this sequence of steps:
1. At a command prompt, run CLU. Refer to “Running CLU.”
2. Determine the firmware currently resident within the coprocessor by obtaining a CLU status response.
3. Change the contents of firmware Segments 1, 2, and 3, as appropriate.
4. Validate the final contents of the firmware segments.

Running CLU

The installation default directory path and name for the Coprocessor Load Utility (CLU) is:

Linux /opt/ibm/4767/csulclu

Windows \Program Files\IBM\4767\csunclu

From the CLU directory, enter this command at a command prompt to display the help menu for CLU:

Linux csulclu -h

Windows csunclu -h
Each time CLU is called, the name of the log file can optionally be specified. If no log file is specified, CLU reads the serial number of the specified adapter number (which defaults to adapter 0) and creates a log file named <serial number>.log. In general, when working with a specific coprocessor (called adapter by CLU), it is strongly recommended that you use its serial number as the log file name. The manufacturer’s serial number can be obtained from the label on its mounting bracket. By always using the serial number to name the log file, a complete history of status and code changes for the contents of each coprocessor is maintained.

When CLU writes to a log file, it creates the file if it does not exist, otherwise it appends to the existing file. CLU writes the same information that is normally displayed on the console.

CLU obtains the number of installed coprocessors from the device driver. These numbers (“adapter_number”) range from 0 - 7. To correlate these numbers to a particular coprocessor, use the System Status (SS) command to learn the number for each of the installed coprocessors. (See Table 18 on page 75)

Note: Except for the SS, ST and VA commands, the CLU utility can operate with a coprocessor only when it can obtain exclusive control of the coprocessor. If any other application (thread) is running that has performed CCA verb calls, all of the coprocessors that are loaded with CCA will be “busy” and unusable by CLU.

Important: When CLU needs exclusive control of the coprocessor, no applications that use CCA should be running.

Determining coprocessor program segment contents

The coprocessor has three program segments that CLU can load or reload, and one that CLU cannot change. The segment that CLU cannot change is called Segment 0, and is loaded at the factory and manages coprocessor initialization and hardware component interfaces. The remaining three segments are called Segment 1, Segment 2, and Segment 3, and all three have dependencies on Segment 0. Segment 1 provides program administration and cryptographic routines, and has no dependency on Segments 2 and 3. Segment 2 provides the embedded operating system, and requires Segment 1 to be loaded before it can be loaded, but has no dependency on Segment 3. Segment 3 provides the coprocessor support program, and requires both Segments 1 and 2 to be loaded before it can be loaded. Refer to Table 4. Each segment:

- Has a status
- Holds firmware
- Holds a validation public key
- Has an owner identifier (except Segment 1)

Table 4. Program segment contents

<table>
<thead>
<tr>
<th>Segment</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“Miniboot,” contains diagnostics and code loading controls</td>
</tr>
<tr>
<td>2</td>
<td>Embedded control program</td>
</tr>
<tr>
<td>3</td>
<td>CCA, or another application</td>
</tr>
</tbody>
</table>

Determine the current content and status of the coprocessor segments by using the ST command of CLU. Definitions of the fields, returned by the ST command grouped by Vital Product Data, ROM status, and Segment-n Information, are shown below:

- Vital Product Data (uniquely records each hardware element)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part Number</td>
<td>The part number (P/N) of the non-encapsulated portion of the coprocessor.</td>
</tr>
</tbody>
</table>
Secure Part Number
The part number of the encapsulated portion of the cryptographic coprocessor. This is the number used to derive the name of the class-key certificate file. Refer to "Validating the coprocessor segment contents" on page 17.

EC Number
The engineering change number of the coprocessor.

Serial Number
The manufacturer’s serial number of the coprocessor. This is different from the IBM tracking serial number used for warranty verification and download authorization.

Description
A statement that describes the type of coprocessor in general terms. Auditors should review this and other status information to confirm that an appropriate coprocessor is in use.

Manufacturing Site
A 2-digit number that identifies which site manufactured the coprocessor.

POST-0 Version
Power-on self-test version of Segment 0.

POST-0 Release
Power-on self-test release of Segment 0.

MiniBoot-0 Version
MiniBoot version of Segment 0.

MiniBoot-0 Release
MiniBoot release of Segment 0.

ROM Status
Field Description
Page 1 Certified

Segment-1 State
The Segment-1 State must be INITIALIZED to be functional. When the coprocessor detects a possible tamper event, it changes the state to ZEROIZED. Once in a ZEROIZED state, the coprocessor is in an unrecoverable, non-functional state.

To help avoid unintended “tamper” events, such as improper handling of the coprocessor, refer to the IBM 4767 PCIe Cryptographic Coprocessor Installation Manual. This manual includes instructions on how to properly replace the onboard batteries. It also include safe temperature ranges and so forth.

Segment-n State (for n = 2 or 3)
UNOWNED - The segment is currently not in use, and has no content.
RUNNABLE - The segment contains code and is in a generally usable state.

Segment-n Owner ID (for n = 2 or 3)
This value shows the current segment owner identifier. A value of 2 means that the segment is owned by the standard CCA Support Program. Any other segment owner identifier besides 2 indicates that the firmware is not the standard IBM CCA product code. Unauthorized or unknown firmware poses a security risk to your installation. Therefore, always ensure that the proper firmware is loaded in your coprocessor.

Segment-n Information (for n = 1, 2, or 3)

Field Description
Segment-n Image (for n = 1, 2, or 3)
The name and description of the firmware content of the segment.
For Segment-1, a factory-fresh image and its associated validation key will need to be changed by reloading Segment 1. The Segment-1 Information returned for a factory-fresh coprocessor includes these messages:

--INFO-- This appears to be a factory fresh card.
--INFO-- The segment-1 key swap CLU file (reload_seg1_xipz_factory_to_prod_keyswap <version>.clu).
--INFO-- must be loaded before loading segments 2 & 3.

Segment-n Revision (for n = 1, 2, or 3)
Be sure to observe the revision level. Ensure that the CLU Segment Revision values match those of the CLU files that were loaded.

Segment-n Hash (for n = 1, 2, or 3)
This is the hash digest calculated against the loaded code segment. Use this value to verify the integrity of the loaded code segment. Refer to the FAQ question "How do I verify the integrity of the IBM 4767?" for valid hash values, available by clicking the HSM PCIeCC2 link from the site navigation, then clicking on the FAQ link.

Changing segment contents

Use the CLU program load (PL) command to change the contents of Segments 1, 2, and 3. The program load command requires the specification of a data file which has a file extension of clu. The readme.txt file that accompanies the software distribution specifies the names of the CLU data files to use. Refer to Table 19 on page 76 for a list of all the CLU files that are shipped with the Support Program, along with a description of when and how to use each file with CLU.

Note: Generally the firmware within the coprocessor must be at the same release level as the CCA software in the hosting system. Do not attempt to mix and match different release levels except with specific instructions from IBM.

When using CLU, be advised that the first adapter is assigned the number 0, the second the number 1, and so on up to the number 7. CLU uses adapter number 0 by default. If multiple adapters are installed, the CLU system status (SS) command can be used to determine what adapter number goes with each installed adapter based on serial number.

In the following example, there is a server with two adapters installed that have serial numbers DV5CW901 and DV5CW302. The card with serial number DV5CW302 is factory fresh. When a card is factory fresh, the Segment-1 Information returned by a CLU ST command indicates "This appears to be a factory fresh card." When a card is factory fresh, it needs to be reloaded with a new Segment 1 image and its associated validation key before loading Segments 2 and 3 with the latest CCA release.

Since there are two adapters installed, the first step is to determine which card number is assigned to each adapter. Use the CLU system status command for this. For example:

Linux  csulclu -c SS -l DV5CW302.log
Windows csunclu -c SS -l DV5CW302.log

In this example, the output returned by the CLU system status command looks like this:

<table>
<thead>
<tr>
<th>Card #</th>
<th>P/N</th>
<th>S/N</th>
<th>Segment 3 Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00LV498</td>
<td>DV5CW901</td>
<td>5.2.23 CCA</td>
</tr>
<tr>
<td>1</td>
<td>00LV498</td>
<td>DV5CW302</td>
<td>(Segment 3 is not loaded.)</td>
</tr>
</tbody>
</table>

Note: The default log file name would have been DV5CW901.log since card number 0 is assigned to serial number DV5CW901.
The output shows that the card number to specify to CLU for serial number DV5CW302 is 1. Before continuing, check the status of adapter 1 and its segments by running the CLU status (ST) command. For example:

**Linux**  
csulclu -c ST -a 1

**Windows**  
csunclu -c ST -a 1

**Note:**

1. Once a factory fresh adapter is reloaded, it cannot normally be returned to its original factory-fresh condition.
2. Segment 1 comes from the factory with an image loaded, while Segments 2 and 3 do not come from the factory with an image loaded.

Refer to [Table 19 on page 76](#) for a list of all the CLU files that are shipped with the Support Program, along with a description of when and how to use each file with CLU. Since the image name and description in this example indicates a factory-fresh adapter, the name of the clu file to use is in the form `reloaded_seg1_xipz_factory_to_prod_keyswap.#.#.#.clu`. Refer to the readme.txt file that accompanies the software package being used. The readme.txt file includes the actual names of the CLU files, with each `.#.#` replaced with the version.release.mod level of the CCA release.

Here is an example of the CLU program load (PL) command to reload a factory fresh Segment 1 (replace the `.#.#` according to the readme.txt file):

**Linux**  
csulclu -c PL -a 1 -d reload_seg1_xipz_factory_to_prod_keyswap.#.#.#.clu

**Windows**  
csunclu -c PL -a 1 -d reload_seg1_xipz_factory_to_prod_keyswap.#.#.#.clu

After a period of time, CLU returns with a message that the program load ended successfully.

Using the readme.txt file, call the CLU program load command to load Segments 2 and 3.

**Validating the coprocessor segment contents**

Whenever the segment contents are loaded or replaced, it is a good practice to validate that those contents are from IBM. The CLU validate adapter (VA) command can be used to confirm that the segment contents are from IBM. The command does this by validating the IBM digital signature on the response created by the adapter.

The CLU VA command optionally takes as input the name of a class-key certificate file that has been digitally signed by IBM. In releases before Release 5.4, the name must be provided in order to validate that the contents of the segments are from IBM. Beginning with Release 5.4, the name is not needed and, if provided, the utility simply ignores the contents of the file.

IBM provides a different certificate file for each IBM 4767 secure part number, and the name of each file is the secure part number with "v.clu" appended to it. Table 5 shows the name of the class-key certificate file according to secure part number.

<table>
<thead>
<tr>
<th>Secure part number</th>
<th>Class-key certificate file</th>
</tr>
</thead>
<tbody>
<tr>
<td>00LU348</td>
<td>00LU348v.clu</td>
</tr>
<tr>
<td>00LV498</td>
<td>00LV498v.clu</td>
</tr>
</tbody>
</table>

If a class-key certificate file is needed but missing from the CLU directory, it is possible to get it from the product website. To find a class-key certificate file, first go to the product website at [https://](https://)
From the site navigation, click the HSM PCIeCC2 link, then click the FAQ link. The FAQ has a topic called "How do I verify the integrity of the IBM 4767?" that provides links to all of the certificate files for each 4767 coprocessor secure part number.

In releases before Release 5.4, find the proper class-key certificate file, then use the CLU validate adapter command to verify the integrity of your coprocessor. The following example assumes that the secure part number for adapter number 0 is 00LV498:

<table>
<thead>
<tr>
<th>O/S</th>
<th>CLU validate adapter command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>csulclu -c VA -a 0 -d 00LV498v.clu</td>
</tr>
<tr>
<td>Windows</td>
<td>csunclu -c VA -a 0 -d 00LV498v.clu</td>
</tr>
</tbody>
</table>

Here is the CLU validate adapter command to use for Release 5.4 or later, since no class-key certificate file is needed:

<table>
<thead>
<tr>
<th>O/S</th>
<th>CLU validate adapter command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>csulclu -c VA -a 0 -d</td>
</tr>
<tr>
<td>Windows</td>
<td>csunclu -c VA -a 0 -d</td>
</tr>
</tbody>
</table>

After a period of time, the validate adapter command returns with an indication of validation success or failure. An adapter that does not successfully validate cannot be trusted. Only use an adapter that has been successfully validated.

Unloading coprocessor firmware and zeroizing the node

When CLU processes a file that surrenders ownership of Segment 2, both Segment 2 and the subordinate Segment 3 are cleared:

- The code is removed.
- The validating ECC public key for the segment is cleared.
- The security-relevant data items held within the coprocessor for the segment are zeroized.
- The ROM Status of SEG2 and SEG3 is set to "UNOWNED".
- The ROM Status owner identifiers for SEG2 and SEG3 are cleared.

To unload coprocessor firmware and zeroize the node, provide a data file to the CLU program load (PL) command to surrender ownership. The file to use is in the form surrender_ownership_seg2_xipz_OID2.#.#.#.clu. (refer to Table 19 on page 76). Refer to the readme.txt file that accompanies the software package being used. The readme.txt file includes the actual names of the CLU files, with each #.#.# replaced with the version.release.mod level of the CCA release.

Here is an example of the CLU program load command to unload coprocessor firmware from adapter 1. Before running this command, be sure you want to destroy all of the security-relevant data items that are on the adapter:

Linux  csulclu -c PL -a 1 -d surrender_ownership_seg2_xipz_OID2.#.#.#.clu

Windows csunclu -c PL -a 1 -d surrender_ownership_seg2_xipz_OID2.#.#.#.clu

After the surrender ownership completes, confirm removal of Segments 2 and 3 and security-relevant data items by using the CLU status (ST) command. This example returns status on adapter 1:

Linux  csulclu -c ST -a 1

Windows csunclu -c ST -a 1
Chapter 5. Using the CNM and CNI utilities to manage the cryptographic node

A computer that provides cryptographic services, such as key generation and digital signature support, is defined here as a cryptographic node. The CCA Node Management (CNM) utility and the CCA Node Initialization (CNI) utility provided with the Support Program are tools to set up and manage the CCA cryptographic services provided by a node. This chapter describes:

- An overview of the CNM and CNI utilities
- How to start these utilities
- Three sample scenarios on how to use the utilities that you should consider

In addition, several sections are included with details on specific utility topics:

- Using the CNM utility administrative functions: Things that you should be aware of in the CNM utility. You should review this material after working through the topic “Establishing a node in a test environment” on page 21.
- Creating and managing access-control data: Some details about the access-control portion of the CNM utility.
- Managing cryptographic keys: Some of the key management things you can accomplish with the CNM utility.
- Using the CNI utility to establish other nodes: How you can automate use of the CNM utility using encapsulated procedures.

**Note:** This chapter describes the major functions of the CNM utility. For additional information about specific panels and fields, refer to the online help panels included with the utility.

These utilities are written in Java** and require use of a Java runtime environment (JRE). You can also use the Java Development Kit (JDK). For a description of the system setup required to run these utilities, see “Hardware and software requirements” on page 7.

**Overview**

Typical users of the CNM utility and the CNI utility are security administration personnel, application developers, system administrators and, in some cases, production-mode operators.

**Note:**

1. The CNM utility furnishes a limited set of the CCA API services. After becoming familiar with the utility, determine whether it meets your needs or whether a custom application to achieve more comprehensive administrative control and key management is required.
2. Files created through use of the CNM utility might be dependent on the release of the Java runtime environment. If the release of the Java runtime environment is changed, files created with the CNM utility might not function correctly with the new release.
3. The CNM utility has been designed for use with a mouse. Use the mouse click instead of the Enter key for consistent results.
4. No help panels are provided for the Master-Key Cloning portion of the utility. See “Cloning a DES or PKA master key” on page 33.
5. The CNM and CNI utilities use the IBM Common Cryptographic Architecture (CCA) API to request services from the coprocessor. The *IBM CCA Basic Services Reference and Guide* contains a comprehensive list of CCA verbs (also known as “callable services” or “procedure calls”). Refer to this
book and the individual services described herein to understand which access control commands might require enablement in the various roles that you will define using the procedures described in this chapter.

CCA Node Initialization utility overview

The CCA Node Initialization (CNI) utility runs scripts that can be created using the CNI Editor within the CNM utility. These scripts are known as CNI lists. The CNI utility can run the CNM utility functions necessary to set up a node. For example, it can be used to load access-control roles and profiles.

As you create a CNI list, specify the disk location of the data objects that the CNI utility will load into the target nodes. After creating a CNI list, the CNI list and any accompanying data files (for roles, profiles, and so forth) can be distributed to nodes where the CNI utility will be used for an “automated” setup. The source node and all nodes running the distributed CNI list must employ the same operating system and a compatible level of Java.

The CNI utility is further explained in “Establishing other nodes using CNI” on page 38.

CCA Node Management utility overview

The CCA Node Management (CNM) utility is a Java application that provides a graphical user interface to use in the setup and configuration of IBM 4767 CCA cryptographic nodes. The utility functions primarily to set up a node, create and manage access-control data, and manage the CCA master-keys necessary to administer a cryptographic node.

Data objects can be loaded directly into the coprocessor or saved to disk. It is important to note that any data objects saved to disk this way are in the clear and are not encrypted. Whoever saves data objects in the clear assumes responsibility for protecting those objects. The data objects are usable at other IBM 4767 CCA nodes that use the same operating system and a compatible level of Java.

Starting the CCA Node Management utility

To start the CCA Node Management utility:

1. Change directory to:

   Linux  /opt/ibm/4767/cnm
   Windows  \Program Files\IBM\4767\cnm

2. For Linux, enter csulcnm on the command line unless the optional Smart Card Utility Program (SCUP; Linux only) is to be included, in which case the command is csulcnm /sc. Note that SCUP is not available on Windows. For Windows, enter csuncnm without the /sc switch.

   The CNM main panel is displayed.

   Note: You can use only data objects that were saved while using CNM with the IBM 4767 PCIe Cryptographic Coprocessor. Non-PCIe data objects cannot be used.

Sample scenarios for using the CNM and CNI utilities

The following scenarios illustrate how to use the CNM and CNI utilities:

1. Establish a test node to be used to develop applications or establish procedures for using the CNM utility. First-time users should follow this procedure to begin experimentation with the utility and the coprocessor.
2. Establish nodes for a production environment using key parts. This scenario employs CNI lists to automate establishment of “target” production nodes.

3. Perform access-control administrator tasks.

4. Perform key-management officer tasks.

The purpose of the scenarios is to illustrate how the procedures described in this chapter can be used. Where appropriate, a scenario cross-references to sections with more detailed information.

To become familiar with the coprocessor’s CCA access-control system, refer to “Access-control overview” on page 26 and “Changing the initial state of the access-control system” on page 27. Here you will find an explanation of terms like role, initial-DEFAULT role, and user profile. The scenarios assume that the access-control system is in its initial state.

Note: These scenarios are instructional only. You are encouraged to determine the procedures best suited for your specific environment. Review the contents of Appendix H in the IBM CCA Basic Services Reference and Guide. This appendix describes observations on secure operations.

Establishing a node in a test environment

In this scenario, a single developer sets up a node to allow unlimited access to cryptographic services.

Important: The resulting cryptographic node should not be considered secure. Under this scenario, sensitive commands are permitted unrestricted use.

1. Install the coprocessor and the IBM CCA Cryptographic Coprocessor Support Program as described in the previous chapters. Start the CNM utility as described in “Overview” on page 19.

   Remember that an appropriate level of the Java Runtime Environment (JRE) or the Java Development Kit (JDK) must be installed.

2. If more than one coprocessor with CCA is installed, specify to the CNM utility which coprocessor to use. From the Crypto Node pull-down menu, select Select Adapter. You will see a drop-down list of available adapter numbers (ranging from one up to a maximum of eight). Choose an adapter (coprocessor) from the list. If you do not use the Select Adapter pull-down to choose an adapter, the default adapter (coprocessor) is used.

3. Synchronize the clocks within the coprocessor and host computer. From the Crypto Node pull-down menu, select Time; a submenu is displayed. From the submenu, select Set; the clock within the processor is synchronized with the host computer clock.

4. Use the CNM utility to enable all commands in the DEFAULT role. From the Access Control pull-down menu, select Roles. Highlight the DEFAULT entry and select Edit. You will see a screen that shows which commands are already enabled and which commands are not enabled by the DEFAULT role. Select Enable All. Then load the modified role back into the coprocessor by selecting Load and then OK.

   Before finishing this test by selecting Cancel, you can save a copy of this “all-commands-enabled” role to your file system using the Save button and assigning a file name. You must also select the folder (directory) where you will save the role.

   For more detail, see “Defining a role” on page 28.

   Finish this task by selecting Cancel.

5. Load the function-control vector (FCV) into the coprocessor. From the Crypto Node pull-down menu, select Authorization. A submenu is displayed. From the submenu, select Load to specify and load the FCV.

   The FCV file that you need to specify is the one that was placed on your server during the installation process. FCVs usually have file names such as fcv_td4kECC521_ECDSA.crt and can be found by using the file search utility available with your operating system.
6. Install a master key. From the Master Key pull-down menu, select either DES/PKA Master Keys, AES Master Key, or APKA Master Key, and then select Auto Set, then Yes. The coprocessor generates and sets a random master key.

A master key installed with Auto Set has actually passed through the main memory of your system processor as key parts. For production purposes, a more secure method of establishing a master key should be used, such as random generation or installation of known key-parts entered by two or more individuals (split knowledge). These options are also accessed from the pull-down menus mentioned above.

For more detail, see “Loading a master key automatically” on page 32.

7. Key storage is a CCA term that describes a place where the Support Program can store DES, PKA, HMAC, ECC, and AES cryptographic keys under names that you (or your applications) define. To use key storage, you must initialize the key storage file or files that correspond to the type of keys that you are using: DES, PKA, HMAC, ECC, or AES. For example, if you intend to use only DES keys, you must initialize the DES key storage file but not the others. See “Creating or initializing key storage” on page 36.

Note: ECC keys are in PKA key tokens that are stored in AES key-storage. PKA key tokens with an RSA private-key section identifier of X’30’ or X’31’ are also stored in AES key-storage.

Establishing a node in a production environment

In this scenario, the responsibility for establishing a cryptographic node is divided among three individuals, namely, an access-control administrator and two key-management officers. The administrator sets up the node and its access-control system, then the key-management officers load a master key and any required key-encrypting key(s). The key-encrypting keys can be used as transport keys to convey other keys between nodes.

Note that this scenario is focused on installing master keys and high-level, inter-node DES key-encrypting keys from key parts. The CCA implementation supports alternatives to the key-part technique such as random master-key generation and distribution of DES keys using techniques based on RSA public-key technology. The key-part technique assumes that there are two key-management officers who can be trusted to perform their tasks and to not share their key-part information. This implements a split knowledge, dual-control policy. The access-control system is set up to enforce dual control by separating the tasks of the first and second officers.

In this scenario, the access-control administrator uses the CNM utility to prepare CNI lists for the target node(s). The CNI lists automate the process of using the CNM utility at the target node. The administrator prepares a CNI list for the tasks performed by the target node access-control administrator and the two key-management officers. The administrator must know what commands require enablement in the target node under different conditions, including:

• Normal, limited operation (when the default role is used)
• When performing the access-control-administrator tasks
• When performing each of the key-management-officer tasks
• Under any other special circumstances using additional roles and profiles

The administrator is responsible for enabling commands in the various roles to ensure that only those commands actually required are authorized. Sensitive commands, such as loading a first key-part or loading subsequent key-part(s), are only enabled in roles for users with the responsibility and authority to utilize those commands. It is important to separate the responsibilities so that policies such as “split knowledge” and “dual control” are enforceable by the coprocessor's access-control system.

For more detail, see “Creating and managing access-control data” on page 26.
Performing access-control administrator tasks

In this task, the access-control administrator uses the CNM utility to prepare CNI lists for the target nodes. To set up the node and create its access-control data, the access-control administrator can:

1. On an established node, start the CNM utility.
2. Create and save to disk the access-control data for the target node, including:
   - Supervisory roles and user profiles for the access-control administrator and the key-management officers
   - A modified DEFAULT role to replace the initial DEFAULT role

For more detail, see "Creating and managing access-control data" on page 26. For information about creating a CNI list, see "Establishing other nodes using CNI" on page 38.

a. Create a CNI list to perform these operations:
   1) Synchronize the time within the coprocessor with the host computer’s time.
   2) Load the access-control data (load the user role and load the user profile).
   3) Log on as an access-control administrator
   4) Load the replacement DEFAULT role
   5) Load the FCV
   6) Log off

b. Create a CNI list for the first key-management officer:
   1) Log on as the first key-management officer
   2) As required, clear the new-master-key register
   3) Load a first master-key key-part
   4) As required, load first-part key-encrypting-key information
   5) Log off

c. Create a CNI list for the second key-management officer:
   1) Log on as the second key-management officer
   2) Load a second master-key key-part
   3) Optionally, load second-part key-encrypting-key information
   4) Log off

d. Create a CNI list for the third key-management officer:
   1) Log on as the third key-management officer
   2) Set the new master key.
   3) As required, reencipher any keys enciphered under the old master key to the new master key

3. Install the coprocessor and the Support Program onto the target node(s).
4. Transport to the target nodes the access-control data and the FCV specified in the CNI list.
5. With the involvement of the key-management officers, on each target node run the CNI lists developed in Steps 2a, 2b, and 2c. Refer to “Establishing other nodes using CNI” on page 38.

The target nodes are now ready to provide cryptographic service.

Performing key-management officer tasks

The key-management officers have two tasks:

- Prepare the key parts for eventual use at the target node(s)
- Load the key parts at the target nodes

A decision has to be made on how to transport the key parts from the point of generation to the point of installation. There are several reasonable scenarios:
1. Generate the key parts at a central place and transport these on portable media.
2. Generate the key parts at a central place and transport these on paper forms.
3. Generate the key parts at the point and time of (first) installation. If the key parts will be needed at another time, either to reload or to share with another node, then how the key parts will be transported must be decided.

You should review the specific capabilities of the CNM utility by working with the utility. Then you can review the specific approach that you select and test the CCA Node Initialization that has been prepared in conjunction with the access-control administrator.

For more detail, see “Managing cryptographic keys” on page 31.

---

**Using the administrative functions of the CNM utility**

This section describes how to use the CNM utility administrative functions to:

- Optionally choose among multiple coprocessors
- Initialize (or “zeroize”) the coprocessor
- Log on to and off of the coprocessor node
- Load the coprocessor FCV
- Configure the CNM utility defaults
- Synchronize the time within the coprocessor with the host computer
- Poll status information about the coprocessor and the CCA application

**Choosing a specific coprocessor**

CCA always has a coprocessor that is considered the default for processing. If your system has multiple coprocessors loaded with the CCA code, generally you will need to select the specific coprocessor upon which to operate. If no selection is made, CCA operates with the default coprocessor. Once a coprocessor selection is made, that selection remains in effect for the current utility session or until a different selection is made within the utility session.

To select an adapter (coprocessor) to use, from the Crypto Node pull-down menu, select Select Adapter. A drop-down list of available adapter numbers (ranging from one up to a maximum of eight) will appear. Choose a number from the list. If the Select Adapter pull-down is not used to choose an adapter, the default adapter is used.

**Note:**
1. When using the CLU utility, coprocessors are referenced as 0, 1, 2, ..., 7. Any particular coprocessor could possibly have the CCA application installed. With the CNM utility (and other applications that use the CCA API), the coprocessors loaded with the CCA application are designated 1, 2, 3, ..., 8. These new identifiers are assigned by CCA as it scans all of the installed coprocessors for those loaded with the CCA application.
2. When coding a CCA application, keywords CRP01, CRP02, CRP03, ..., CRP08 are used to “allocate” a coprocessor. These correspond to the numbers 1, 2, 3, ..., 8 used in the CNM utility pull-down.

**Initializing a CCA node**

The CCA node can be restored to its initial state, provided that the active role (the logged-on role or, in the absence of a logged-on role, the default role) permits use of the Reinitialize Device access-control command (offset X’0111’). This command clears (zeroizes) all:

- Master-key registers
- Retained RSA private keys and registered RSA public keys
Roles and profiles, restoring the access control to its initial state (see “Changing the initial state of the access-control system” on page 27).

To initialize the CCA node, select **Initialize...** from the Crypto Node pull-down menu, then confirm your intent to perform this major action.

**Logging on and off the node**

To log on to a node, select **Passphrase Logon...** from the File pull-down menu, then select Passphrase type: **Passphrase** or **Passphrase2**. To log off, select **Logoff** from the File pull-down menu.

**Note:** With the exception of the DEFAULT role, access to the coprocessor is restricted by passphrase authentication.

**Loading the FCV**

A function-control vector (FCV) is a digitally signed value provided by IBM. An FCV is required, and is used to enable the CCA application in the coprocessor to provide a level of cryptographic service consistent with applicable import and export regulations. Under current regulations, all users are entitled to the same level of cryptographic functionality. Therefore, IBM now supplies a single FCV with the CCA Support Program.

You use the CNM utility to load the FCV into the coprocessor. The FCV file is named `fcv_td4kECC521_ECDSA.crt`. You can locate this file using the filename search tool provided with your operating system.

To load the FCV:
1. From the Crypto Node pull-down menu, select **Authorization**; a submenu is displayed.
2. From the submenu, select **Load** to specify the FCV file on disk. Specify the file name and select **Update**. The utility loads the FCV.
3. Select **OK** to finish the task.

**Configuring the CCA Node Management utility**

The CNM utility displays a utility configuration panel when the **Configure Utility** menu item is selected. The utility configuration panel provides an easy way to navigate directory paths by allowing the user to specify which default directory to use when issuing a command. If the user does not set a default directory, the current directory is used. A default directory can be set for each of these categories:

1. Key storage
2. Master keys
3. Key encrypting keys
4. User profiles
5. User roles

**Synchronizing the clock-calendars**

The coprocessor uses its clock-calendar to record time and date and to prevent replay attacks in passphrase-based profile authentication. After installing the coprocessor, synchronize its clock-calendar with that of the host system.

To synchronize the clock-calendars:
1. From the Crypto Node pull-down menu, select **Time**; a submenu is displayed.
2. Select **Set...** from the submenu.
3. Answer **Yes** to synchronize the clock-calendar on the coprocessor with the clock-calendar on the host.
4. Select **OK** to finish this task.

**Obtaining status information**

The CNM utility can obtain the status of the coprocessor and the CCA application. The following status panels are available:

**CCA Application**
- Displays the version and the build date of the application. Also displays the status of the master key registers. For information about these registers, see “Managing the master keys” on page 31.

**Adapter**
- Displays the coprocessor serial number, ID, and hardware level.

**Command History**
- Displays the five most recent commands and subcommands sent to the coprocessor.

**Diagnostics**
- Indicates whether any of the coprocessor tamper-sensors have been triggered, whether any errors have been logged, and reflects the status of the coprocessor batteries.

**Export Control**
- Displays the maximum strength of the cryptographic keys used by the node, as defined by the FCV resident within the coprocessor.

To view the status panels:
1. From the **Crypto Node** pull-down menu, select **Status**. The CCA application status is displayed.
2. To select other status information, use the buttons at the bottom. The new panel is displayed.
3. Select **Cancel** to finish this task.

**Creating and managing access-control data**

The access-control system of the IBM CCA Cryptographic Coprocessor Support Program defines the circumstances under which the coprocessor can be used. It does this by controlling the use of CCA commands. For a list of these commands, refer to Appendix G of the *IBM CCA Basic Services Reference and Guide*. Also, see the “Required commands” section at the end of each verb description.

An administrator can give users differing authority, so that some users can use CCA services not available to others. This section includes an overview of the access-control system and instructions for managing your access-control data. You need to know which commands are required and under what circumstances. You also need to consider that some commands should be authorized only for select, trusted individuals, or for certain programs that operate at specific times. Generally, authorize only those commands that are required, so as not to inadvertently enable a capability that could be used to weaken the security of your installation(s). Obtain the information about command use from the documentation for the applications that you intend to support. Refer to Appendix H in the *IBM CCA Basic Services Reference and Guide* for additional guidance on this topic.

**Access-control overview**

The access-control system restricts or permits the use of commands based on roles and user profiles. Use the CNM utility to create roles that correspond to the needs and privileges of assigned users.

To access the privileges assigned to a role (those that are not authorized in the default role), a user must log on to the coprocessor using a unique user profile. Each user profile is associated with a role. (Multiple profiles can use the same role.) The coprocessor authenticates logons using the passphrase associated with the profile that identifies the user.
Note: The term user applies to both humans and programs.

The coprocessor always has at least one role—named the DEFAULT role. Use of the DEFAULT role does not require a user profile. Any user can use the services permitted by the DEFAULT role without logging onto or being authenticated by the coprocessor. In fact, any user who is not logged on to a role automatically gets assigned to the DEFAULT role.

A basic system might include the following roles:

Access-Control Administrator
   Can create new user profiles and modify the access rights of current users.

Key-Management Officer
   Can change the cryptographic keys. (This responsibility is best shared by two or more individuals making use of rights to enter “first” or “subsequent” key parts.)

General User
   Can use cryptographic services to protect his or her work, but has no administrative privileges. If your security plan does not require logon authentication for general users, address their requirements in the DEFAULT role.

Note: Few individuals would be assigned the roles of key-management officer or access-control administrator. Generally, the larger population would not log on and thus would have rights granted in the DEFAULT role.

Initializing the access-control system
When you initialize the access-control system, the CNM utility:
• Clears the access-control data in the coprocessor
• Furnishes the DEFAULT role with the commands required to load access-control data

Important: The DEFAULT role initially permits the loading of access-control data. The cryptographic node and the data that it protects are not secure while the DEFAULT role is permitted to load access-control data.

Successfully initializing the access-control system removes installation-installed access controls and keys. It is therefore a very sensitive operation that could render your node inoperable for production. Some installations will choose to remove authorization for this function from their coprocessor’s roles. In this event, if you wish to initialize the CCA cryptographic node, you must remove the CCA firmware from the coprocessor and reinstall it.

To initialize the access-control system:
1. From the Access Control pull-down menu, select Initialize...; a confirmation dialog box is displayed.
2. Select Yes to confirm; the utility initializes the access-control system.

Changing the initial state of the access-control system
After you have loaded the CCA firmware support into Segment 3 of the coprocessor, or after the access-control system is initialized, no access-control data exists except for an initial DEFAULT role that allows unauthenticated users to create and load access-control data. For a full description of this role, see Appendix A, “Initial commands for DEFAULT role,” on page 47.

After creating the roles and profiles needed for your environment—including the supervisory roles necessary to load access-control data and to manage cryptographic keys—remove all permissions assigned to the DEFAULT role. Then, add only those permissions you want to grant to unauthenticated users.
Important: The cryptographic node and the data that it protects are not secure while the DEFAULT role is permitted to load access-control data.

Defining a role

A role defines permissions and other characteristics of the users who use it. The DEFAULT role is the only role that is unauthenticated. All other roles require a passphrase to log on. To define a role:

1. From the Access Control pull-down menu, select Roles; a list of currently defined roles is displayed.
2. Select New to display the Role Management panel. At any time in the process, select List to return to the list of currently defined roles.
3. Define the role:
   - **Role ID**
     A character string, up to eight characters, that defines the name of the role. This name is contained in each user profile associated with this role. A Role ID cannot start with a space character.
   - **Comment**
     An optional character string, up to 20 characters, to describe the role.
   - **Required authentication strength**
     When a user logs on, the strength of the authentication provided is compared to the strength level required for the role. If the authentication strength is less than that required, the user cannot log on. Currently only the passphrase authentication method is supported; use a strength of 50.
   - **Valid times and valid days**
     These values determine when the user can log on. Note that times are Coordinated Universal Time. For more information about the access-control system, refer to Chapter 2 of the *IBM CCA Basic Services Reference and Guide*.
   - **Enabled operations and disabled operations**
     A list defining the commands that the role has enabled or disabled.

Some, but not all, CCA API verbs require one or more commands to obtain service from the coprocessor. The user requesting service must be assigned to a role that permits those commands needed to run the verb. Most commands when enabled permit an action to complete, but there are some commands that disallow an action when enabled. For this reason it is important to pay attention to each command name.

For more information about CCA verb calls and commands, refer to the *IBM CCA Basic Services Reference and Guide*.

4. Select Save... to save the role to disk.
5. Select Load to load the role into the coprocessor.

Modifying existing roles

Use the CNM utility to:
- Edit a disk-stored role
- Edit a coprocessor-stored role
- Delete a coprocessor-stored role

Tip: Any existing role can be used as a template to create a new role. When CNM opens a saved role, the existing information is displayed in the Role Definition panel. You need only modify or enter information specific to the new role, then give it a new Role ID and load or save it.
Editing a disk-stored role

To edit a role stored on disk:
1. From the Access Control pull-down menu, select Roles; a list of currently defined roles is displayed.
2. Select Open...; you are prompted to choose a file.
3. Open a file; data is displayed in the Role Definition panel.
4. Edit the role.
5. Select Save... to save the role to disk; select Load to load the role into the coprocessor.

Editing a coprocessor-stored role

To edit a role stored in the coprocessor:
1. From the Access Control pull-down menu, select Roles; a list of currently defined roles is displayed.
2. Highlight the role you want to edit.
3. Select Edit; data is displayed in the Role Definition panel.
4. Edit the role.
5. Select Save... to save the role to disk; select Load to load the role into the coprocessor.

Deleting a coprocessor-stored role

To delete a role stored in the coprocessor:
1. From the Access Control pull-down menu, select Roles; a list of currently defined roles is displayed.
2. Highlight the role you want to delete.
3. Select Delete...; the role is deleted.

Important: When CNM deletes a role, it does not automatically delete or reassign the user profiles associated with that role. Delete or reassign the user profiles associated with a role before you delete the role.

Defining a user profile

A user profile identifies a specific user to the coprocessor. To define a user profile:
1. From the Access Control pull-down menu, select Profiles; a list of currently defined profiles is displayed.
2. Select New to display the Profile Management panel.
3. Define the user profile:
   - User ID
     The name, up to 8 characters, given to a user profile of the cryptographic coprocessor. A user ID cannot start with a space character.
   - Comment
     An optional character string, up to 20 characters, to describe the user profile.
   - Activation Date and Expiration Date
     These values determine the first and last dates when the user can log on to the user profile.
   - Role
     The name of the role that defines the permissions granted to the user profile.
   - Passphrase and Confirm Passphrase
     The character string that the user must enter to gain access to the cryptographic node.
   - Passphrase Expiration Date
     The expiration date for the passphrase. The utility sets this by default to 3 months from the
current date. Change the expiration date as needed. Every passphrase contains an expiration date, which defines the lifetime of that passphrase. This is different from the expiration date of the profile itself.

4. Select Save... to save the profile to disk; select Load to load the profile into the coprocessor.
5. Select List to return to the list of currently defined profiles.

Modifying existing user profiles
Use the CNM utility to:
- Edit a disk-stored user profile
- Edit a coprocessor-stored user profile
- Delete a coprocessor-stored user profile
- Reset the user-profile-failure count

Editing a disk-stored user profile
To edit a profile stored on disk:
1. From the Access Control pull-down menu, select Profiles; a list of currently defined profiles is displayed.
2. Select Open...; you are prompted to choose a file.
3. Open a file; data is displayed in the User Profile Definition panel.
4. Edit the profile.
5. Select Save... to save the profile to disk; select Load to load the profile into the coprocessor.

Editing a coprocessor-stored user profile
To edit a profile stored in the coprocessor:
1. From the Access Control pull-down menu, select Profiles; a list of currently defined profiles is displayed.
2. Highlight the profile you want to edit.
3. Select Edit; data is displayed in the User Profile Definition panel.
4. Edit the profile.
5. Select Save... to save the profile to disk; select Replace to load the profile into the coprocessor.

Deleting a coprocessor-stored user profile
To delete a profile stored in the coprocessor:
1. From the Access Control pull-down menu, select Profiles; a list of currently defined profiles is displayed.
2. Highlight the profile you want to delete.
3. Select Delete...; the profile is deleted.

Resetting the user-profile failure count
To prevent unauthorized logons, the access-control system maintains a logon-attempt failure count (FC) for each user profile. If the number of failed attempts for a profile exceeds the limit defined in the profile, the offending profile is disabled. To reset the user-profile failure count:
1. From the Access Control pull-down menu, select Profiles; a list of currently defined profiles is displayed.
2. Highlight the profile.
3. Select Reset FC; a confirmation dialog box is displayed.
4. Select Yes to confirm; the logon-attempt failure count is set to zero.

**Managing cryptographic keys**

This section describes how to use the CNM utility to:
- Manage the master keys
- Manage primary key-encrypting keys (KEKs)
- Reset and manage DES, PKA, and AES key-storage

Key types are defined as follows:

**Master keys**
Special KEKs stored in the clear (not enciphered) and kept within the coprocessor secure module.
Four kinds of master keys are supported: DES, PKA, AES, and APKA. They are used to wrap other keys so that those keys can be stored outside of the secure module. DES and PKA master keys are 168-bit keys formed from three 56-bit DES keys. AES and APKA master keys are 256-bit (32-byte) AES keys.

**Primary KEKs**
DES keys shared by cryptographic nodes, sometimes referred to as transport keys. They are used to encipher other keys shared by the nodes. Primary KEKs, like the master key, are installed from key parts which get exclusive-ORED together to produce the final key. Knowledge of the key parts can be split among two people to effect a split-knowledge, dual-control security policy. In this manner, no single person ever knows the final value of the key.

**Other DES keys, PKA keys, and AES keys**
Enciphered keys used to provide cryptographic services, such as MAC keys, DATA keys and private PKA keys.

**Note:** When exchanging clear key-parts, ensure that each party understands how the exchanged data is to be used, since the management of key parts varies among different manufacturers and different encryption products.

**Managing the master keys**

A master key is used to encrypt local-node working keys, also called operational keys, while they are stored external to the coprocessor. CCA defines three master key registers for each master key:

**current-master-key register**
Stores the master key currently used by the coprocessor to encrypt and decrypt local keys

**old-master-key register**
Stores the previous master key and is used to decrypt keys enciphered by that master key

**new-master-key register**
Is an interim location used to store master-key information as accumulated to form a new master-key

The Support Program uses four sets of master key registers, one set for ciphering DES (symmetric) keys, one set for ciphering PKA private (asymmetric) keys, one set for ciphering AES (symmetric) keys, and one set for ciphering ECC (asymmetric) keys.

For information about checking the contents of these registers, see “Obtaining status information” on page 26.

**Note:**
1. The Master_Key_Distribution master key administration verb does not support AES or APKA master keys. Programs that use the CCA Master_Key_Process and Master_Key_Distribution master key
administration verbs can use the ASYM-MK keyword to steer operations to the PKA asymmetric master-key registers, the SYM-MK keyword to steer to the DES symmetric master key registers, or both the DES symmetric and PKA asymmetric sets of master key registers. The CNM utility uses the BOTH option. If you use another program to load master keys, and if this program specifically operates on either the SYM-MK or ASYM-MK master key registers, in general you will no longer be able to use the CNM utility to administer these master keys. Note that AES and APKA master keys work independently from DES and PKA master keys.

2. If your installation has multiple cryptographic coprocessors loaded with CCA, you will need to independently administer the master keys in each coprocessor.

3. If your installation has a server with multiple shared cryptographic coprocessors that are loaded with CCA, those coprocessors will need to be installed with identical master keys.

Verifying an existing master key

The CNM utility retrieves a key verification pattern for each master key stored in the master key registers. This pattern identifies the key, but does not reveal information about the actual key value.

To view a master-key verification pattern:
1. From the Master Key pull-down menu, select DES/PKA Master Keys, AES Master Key, or APKA Master Key, and then select Verify. A submenu is displayed.
2. From the submenu, select a master-key register. The verification pattern for the key stored in that register is displayed.

Loading a master key automatically

The CNM utility can auto-set a master key into the coprocessor; its key value cannot be viewed from the utility.

Important: If a master key of unknown value is lost, the keys enciphered under it cannot be recovered.

To automatically load the master key:
1. From the Master Key pull-down menu, select DES/PKA Master Keys, AES Master Key, or APKA Master Key, and then select Auto Set... or Random. You are prompted to verify the command.
2. Select Yes. The coprocessor generates and sets the selected master key.

Note:
1. Use of Random is preferred since the Auto-Set option passes clear key-parts through host-system memory.
2. After you change a master key, reencipher all keys enciphered under the former key to the current master key. See “Reenciphering stored keys” on page 36.

Loading a new master key from key parts

To set a new master-key into the coprocessor, first load the first key part, then any middle key parts, then the last key part into the new-master-key register, and then set the new master-key. To effect this:
1. From the Master Key pull-down menu, select DES/PKA Master Keys, AES Master Key, or APKA Master Key, and then select Parts. The Load Master Key panel is displayed.
2. Select the radio button to edit (First Part, Middle Part, or Last Part).
3. Enter data by one of the following:
   - Select Open... to retrieve preexisting data.
   - Select Generate to fill the fields with coprocessor-generated random numbers.
   - Manually enter data into the “Master Key Part” fields; each field accepts four hexadecimal digits.
Select **New** to clear data entered in error.

4. **Select Load** to load the “Master Key Part” into the new-master-key register. **Select Save**... to save the key part to disk.

   **Important:** Key parts saved to disk are not enciphered. Consider keeping a disk with a key part on it stored in a safe or vault.

   **Note:** When creating a key from parts, a first part and a last part are required, while middle part(s) are optional.

5. Repeating the preceding steps, load into the new-master-key register the remaining key parts.

   **Note:** For split-knowledge, dual-control security policy, different people must enter the separate key parts. To enforce a dual-control security policy, the access-control system should assign the right to enter a first key part to one role and the right to enter subsequent key parts to another role. Then, each authorized user can log on and perform the loading of that user’s respective key parts. A third role should be assigned the right to set the master key.

6. From the **Master Key** pull-down menu, select **DES/PKA Master Keys**, AES Master Key, or APKA Master Key, and then select **Set**... The utility transfers the data:
   a. In the current-master-key register to the old-master-key register, and deletes the former old-master key
   b. In the new-master-key register to the current-master-key register

   After setting a master key, reencipher the keys that are currently in storage. See "Reenciphering stored keys” on page 36.

**Cloning a DES or PKA master key**

This scenario explains the steps involved in **cloning** a DES or PKA master key from one coprocessor to another. (Cloning of an AES or APKA master key is not supported.) The term cloning is used rather than copying since the master key will be split into **shares** for transport between the coprocessors. The technique is explained at some length in “Understanding and managing master keys” in Chapter 2 of the *IBM CCA Basic Services Reference and Guide*. Appendix C, “DES and PKA master-key cloning,” on page 51 provides a step-by-step procedure to follow. The material in this chapter provides background information that can permit you to vary the procedure.

Cloning of the DES or PKA master key involves two or three nodes:

- The master-key source node (coprocessor share signing node)
- The master-key target node (coprocessor share receiving node)
- The share administration (SA) node. The SA node can also be either the source or the target node.

The CNM utility can store various data items involved in this process in a database that you can carry on portable media or FTP between the different nodes. One database is, by default, known as *sa.db* and contains the information about the SA key and keys that have been certified. The target node where the master key will be cloned also has a database known by default as the *csr.db*.

You can accomplish these tasks using the CNM utility:

1. **Set up the nodes in a secure manner with access-control roles and profiles and master keys.**

   You will need a role and profiles at the source and target nodes for each user who will obtain or store share, where 1 ≥ i ≤ n. Processing of share, is a separate access-control-point command so that, if you wish, your roles can ensure that independent individuals are involved with obtaining and installing the different shares.

   Consider the use of random master-key generation. Also consider roles that enforce a dual-control security policy; for example, permit one individual/role to register a hash and another...
individual/role to register a public key, have different individuals/roles for obtaining and installing the individual shares of the master key, and so forth.

See the guidance portion of Chapter 2 in the IBM CCA Basic Services Reference and Guide for the description of the Master_Key_Process and the Master_Key_Distribution verbs.

2. Install a unique 1 - 16 byte Environment ID (EID) of your choice into each node.

From the Crypto Node pull-down menu, select Set Environment ID, enter the identifier, and select Load. Use only these characters in an environment identifier (EID): A...Z, a...z, 0...9, and “@” (X'40'), space character (X'20'), “&” (X'26'), and “=” (X'3D').

You should enter a full 16-character identifier. For ‘short’ identifiers, complete the entry with space characters.

3. Initialize the master-key-sharing “m” and “n” values in the source and target nodes. These values must be the same in the source and the target nodes. The maximum number of shares is “n”, while “m” is the minimum number of shares that must be installed to reconstitute the master key in the target node.

From the Crypto Node pull-down menu, select Share Administration, and then select Set number of shares, enter the values, and select Load.

4. At the different nodes, generate these keys and have each public key certified by the share-administration (SA) key. You can use the utility’s sa.db database to transport the keys and the certificates.

Share Administration (SA)
This key is used to certify itself and the following keys. You must register the hash of the SA public-key, and the public key itself, in the SA, source, and target nodes.

When the SA key is created, the utility will supply an 8-byte/16-hex-character value that is a portion of the hash of the SA key. Be sure to retain a copy of this value. You will need this value to confirm the hash value recorded in the database to register the SA public-key at the source and target nodes.

Coprocessor Share Signing (CSS)
This key is used to sign shares distributed from the source node. The RSA private key is retained within the source node.

Coprocessor Share Receiving (CSR)
This key is used to receive a share-encrypting key into the target node. The SA-certified public CSR key is used at the source node to wrap (encrypt) the share-encrypting key that is unique for each share. The RSA private key is retained within the target node.

Generate the Key Pairs: SA, CSS, and CSR
From the Crypto Node pull-down menu, select Share Administration, select Create Keys, and one of Share Administration Key, CSS key, or CSR key, then select Create.

You also will need to supply key labels for the CSS and CSR keys that are retained in the source and target nodes. For example, IBM4767.CLONING.CSS.KEY and IBM4767.CLONING.CSR.KEY. The labels must not conflict with other key labels used in your applications.

When generating the CSR key at the coprocessor share-receiving node, also obtain the serial number of the coprocessor. From the Crypto Node pull-down menu, select Status. You must enter the serial-number value when certifying the CSR key.

5. Register the SA public-key in the coprocessor at the SA, source, and target nodes. This is a two-step process that should be done under a dual-control security policy.

One individual should install the SA public-key hash. From the Crypto Node pull-down menu, select Share Administration, select Register Share Administration, and select SA key hash, then enter the hash value obtained during SA key creation.
This section describes how to:

The other individual should install the actual SA public-key. From the Crypto Node pull-down menu, select Share Administration, select Register Share Administration, and select SA Key. By default, the public-key information is in the sa.db file.

6. Take the CSS key and the CSR key to the SA node and have the keys certified.
   From the Crypto Node pull-down menu, select Share Administration, select Certify Keys, and one of CSS key or CSR key.
   For the CSR key, you will need to supply the serial number of the target coprocessor as a procedural check that an appropriate key is being certified. Your procedures should include communicating this information in a reliable manner.

7. At the source node, have authorized individuals sign on to the role that permits them to obtain their shares. At least “m” shares must be obtained. These will be shares of the current master-key.
   From the Crypto Node pull-down menu, select Share Administration, select Get Share, and select the share number to be obtained. Observe the serial numbers and database identifiers. When these are agreed to be correct, select Get Share. The share information will be placed by default into the csr.db file and will obtain the CSR key-certificate, by default, from the sa.db file.
   Obtain current master key validation information for use later at the target node. From the Master Key pull-down menu, select DES/PKA Master Keys, then select Verify, then select Current.

8. At the target node, have authorized individuals sign on to the role that permits them to install each of their shares. At least “m” number of shares must be installed to reconstitute the master key into the new master key register.
   From the Crypto Node pull-down menu, select Share Administration, select Load Share, and select the share number to be installed. Observe the serial numbers and database identifiers. When these are agreed to be correct, select Install Share. The share information will be obtained by default from the csr.db file and the CSS key certificate will be obtained by default from the sa.db file.
   When “m” shares have been loaded, verify that the key in the new-master-key register is the same as the current master-key in the source node (when the shares were obtained). On the target node, from the Master Key pull-down menu, select DES/PKA Master Keys, then Verify, then select New.

9. When it is confirmed through master-key verification that the master key has been cloned, an authorized individual can set the master key. This action deletes any old master-key and moves the current master-key to the old-master-key register. Application programs that use keys encrypted by the master key can be impacted by this change, so ensure that setting of the master key is coordinated with the needs of your application programs. From the Master Key pull-down menu, select DES/PKA Master Keys, then select Set...

Managing key storage

The CNM utility allows basic key-storage management for keys. These utility functions do not form a comprehensive key-management system. Application programs are better suited to perform repetitive key-management tasks.

Key storage is a repository of keys that are accessed by key label using labels that you or your applications define. DES keys, RSA keys, ECC keys, HMAC keys, and AES keys are held in separate storage systems. (ECC keys, HMAC keys, and AES keys are held in AES key-storage, as well as RSA keys contained in an RSA key-token with a private-key section identifier of X'30' or X'31'.) Also, the coprocessor has a very limited internal storage for RSA keys. The coprocessor-stored keys are not considered part of key storage in this discussion.

Note: If your server has multiple cryptographic coprocessors that are loaded with CCA, those coprocessors must have identical master keys installed for key storage to work properly.

This section describes how to:
• Create or initialize key storage
• Reencipher stored keys
• Delete a stored key
• Create a key label

Note: The utility displays a maximum of 1,000 key labels. If you have more than 1,000 key labels in key storage, use an application program to manage them.

Creating or initializing key storage

To create or initialize key storage for your DES, PKA, ECC, HMAC, or AES keys:
1. From the Key Storage pull-down menu, select DES Key Storage, PKA Key Storage, or AES Key Storage; a submenu is displayed.
2. From the submenu, select Initialize; the Initialize DES Key Storage, Initialize PKA Key Storage, or Initialize AES Key Storage panel is displayed.

Note: ECC keys, HMAC keys, and AES keys are held in AES key-storage, as well as RSA keys contained in an RSA key-token with a private-key section identifier of X’30’ or X’31’.
3. Enter a description for the key-storage file, if desired.
4. Select Initialize; you are prompted to enter a name for the key-storage dataset.
   The locations that you set for the key storage datasets must match the locations defined by the CSUDESDS, CSUPKADS, and CSUAESDS registry entries. See “Configuring the Support Program” on page 10.
5. Enter a name for the file and save it. The key-storage file is created on the host.

Note: If a file with the same name exists, you are prompted to verify your choice because initializing the key storage modifies the file, and if it has any keys, these would be erased.

Reenciphering stored keys

To reencipher the keys in storage under a newly set current master-key:
1. From the Key Storage pull-down menu, select DES Key Storage, PKA Key Storage, or AES Key Storage. A submenu is displayed.
2. From the submenu, select Manage. The DES Key Storage Management, PKA Key Storage Management, or AES Key Storage Management panel is displayed. The panel lists the labels of the keys in storage.
3. Select Reencipher...; keys are re-enciphered under the key in the current master-key register.

Deleting a stored key

To delete a stored key from host key storage:
1. From the Key Storage pull-down menu, select DES Key Storage, PKA Key Storage, or AES Key Storage. A submenu is displayed.
2. From the submenu, select Manage. The selected Key Storage Management panel is displayed. The panel lists the labels of the keys in storage.
   You can set the filter criteria to list a subset of keys within storage. For example, entering *.mac as the filter criterion and refreshing the list limits it to key labels with two tokens that end in .mac (the asterisk is a wildcard character). Multiple wildcards are allowed.
3. Highlight the key label for the key to be deleted.
4. Select Delete... A confirmation dialog box is displayed.
5. Select Yes to confirm. The stored key is deleted from host key storage.
Creating a key label

To create a key label in host key storage:

1. From the Key Storage pull-down menu, select DES Key Storage, PKA Key Storage, or AES Key Storage. A submenu is displayed.
2. From the submenu, select Manage. The selected Key Storage Management panel is displayed. The panel lists the labels of the keys in storage.
3. Select New. You are prompted to enter a key label.
4. Select Load. The key label is loaded into storage.

Creating and storing primary DES KEKs

DES key-encrypting keys (KEKs) are encrypted under the DES master-key and stored in DES key-storage for local use. Key parts used to create a KEK can be randomly generated or entered as clear information. The parts can also be saved to disk or portable media in the clear for transport to other nodes or for re-creating the local KEK.

Note: The CNM utility supports only DES KEKs for the transport of keys between nodes. Applications can use the CCA API to furnish the services needed for public-key-based or AES-based key distribution.

To work with a DES KEK (or other double-length operational key):

1. From the Keys pull-down menu, select Primary DES Key-Encrypting Keys. The Primary DES Key-Encrypting Keys panel is displayed.
2. At any time you can select New to clear all data fields and reset all the radio buttons to their default settings.
3. Select the radio button for the desired key-part to be entered: First Part, Middle Part, or Last Part.
4. Enter data in the Key Part fields by using one of the following processes:
   • Select Open... to retrieve preexisting Key Part, Control Vector, and Key Label data previously stored on disk using the Save... command.
   • Select Generate to fill the Key Part fields with coprocessor-generated random numbers.
   • Manually enter data into the Key Part fields. Each of the Key Part fields accepts four hexadecimal digits.
5. Select a control vector for the key:
   • To use a default KEK control-vector, select the appropriate Default Importer or Default Exporter radio button.
   • To use a custom control-vector, select the Custom radio button. In the Control Vector fields, enter the left or right half of a control vector for any double-length key. Note that the KEY-PART bit (CV bit 44) must be on and that each byte of the control vector must have even parity. For detailed information about control vectors, refer to Appendix C of the IBM CCA Basic Services Reference and Guide.
6. Enter a key label to identify the key token in key storage.
7. Select Load to send the key part to the coprocessor for encryption and store the resulting key token into key storage. Select Save... to save the unencrypted Key Part and its associated Control Vector and Key Label values to disk.
8. Save to disk or Load to key storage the remaining key-part information by following Step 3 to Step 7. Be sure to use the same key label for each part of a single key.
Establishing other nodes using CNI

By creating a CNI list for the CCA Node Initialization (CNI) utility, you can load keys and access-control data stored on disk into other cryptographic nodes without running the CNM utility on those target nodes.

To set up a node using the CNI utility:

1. Start the CNM utility on an established node.
2. Save to the host or portable media the access-control data and keys you want to install on other nodes. When you run the CNI utility on the target node (Step 10 on page 39), it searches the identical directory path for each file. For example:
   - If a user profile is saved to this established node directory:
     - **Linux** /opt/ibm/4767/profiles
     - **Windows**\Program Files\IBM\4767\profiles
   - then the CNI utility will search the following target node directory:
     - **Linux** /opt/ibm/4767/profiles
     - **Windows**\Program Files\IBM\4767\profiles
   - If a user profile is saved to this portable media directory:
     - **Linux** /profiles
     - **Windows**\profiles
   - then the CNI utility will search the following target node directory:
     - **Linux** /profiles
     - **Windows**\profiles
3. From the File pull-down menu, select CNI Editor. The CCA Node Initialization Editor panel is displayed.

   The list in the top half of the panel displays the functions that can be added to the CNI list. The bottom half lists the functions included in the current CNI list. References to 'master keys' in the list refer to the DES/PKA master keys. The AES and APKA master key references are explicitly indicated.

   The CNI list can perform the following functions:
   - Select the active card
   - Logon to and logoff of the cryptographic node
   - Initialize the cryptographic facility (coprocessor)
   - Initialize the access control facility
   - Auto set the master key
   - Clear the new master-key register
   - Load master key parts
   - Set the master key
   - Auto set AES and APKA master keys
   - Clear the new AES and APKA master key registers
   - Load AES and APKA master key parts
• Set AES and APKA master keys
• Load APKA key wrapping options
• Load or delete user roles and user profiles
• Synchronize the clock-calendars
• Initialize DES, PKA, and AES host key storage

4. Add the functions you want. To add a function to the CNI list:
   a. Highlight it.
   b. Select Add. The function is added to the CNI list.

   **Note:** If the function you choose loads a data object—like a key part, key-storage file, user profile, or role—you are prompted to enter the file name or the ID of the object to be loaded.

5. Using the Move Up and Move Down buttons, organize the functions to reflect the same order that you follow when using the CNM utility. For example, if you are loading access-control data, you must first log on as a user with the authority to load access-control data.

6. Select Verify to confirm that objects have been created correctly.

7. Select Save…. You are prompted to choose a name and directory location for the CNI-list file.

8. Save the CNI-list file. The list file does not contain the data objects specified in the CNI list.

9. Copy the files needed by the CNI utility onto target host directory locations that mirror their location on the source host. If files are saved on portable media, insert the media into the target node.

10. From the target node, run the list using the CNI utility:
    a. Change directory to:
       - **Linux** /opt/ibm/4767/cnm
       - **Windows** \Program Files\IBM\4767\cnm
    b. Enter this command on the command line:
       - **Linux** csuclni listfile_name
       - **Windows** csuncni listfile_name
    If the CNI list includes a logon, enter csuclni (Linux) or csuncni (Windows) on the command line (without specifying a filename). The utility Help text describes the syntax for entering an ID and passphrase.

    The CNI utility loads files to the coprocessor from the host or portable media, as specified by the CNI list.
Chapter 6. Building applications to use with the CCA API

This chapter includes the following:
- An overview of the way in which applications obtain service from the Common Cryptographic Architecture (CCA) application program interface (API)
- The procedure for calling a CCA verb in the C programming language
- The procedure for compiling applications and linking them to the CCA API
- The procedure for viewing coprocessor hardware errors
- Enhancing throughput with CCA and the IBM 4767.

Sample CCA routines written in the C programming language can be downloaded from the product website. For information on downloading sample code, refer to Appendix E, “CCA sample code,” on page 69. You can use the samples to test the coprocessor and the Support Program.

Note: The file locations referenced in this chapter are the default directory paths.

Overview

Application and utility programs issue service requests to the cryptographic coprocessor by calling the security API verbs\(^1\). The Linux environment links requests to its shared object code. The operating system code in turn calls the coprocessor physical device driver (PDD). The hardware and software accessed through the API are themselves an integrated subsystem.

Verb calls are written in the standard syntax of the C programming language, and include an entry_point_name, verb parameters, and the variables for those parameters.

For a detailed listing of the verbs, variables, and parameters that can be used when programming for the security API, refer to the IBM CCA Basic Services Reference and Guide.

Calling CCA verbs in C program syntax

In every operating system environment, you can code CCA API verb calls using standard C programming language syntax.

Function call prototypes for all security API verbs are contained in a header file. The files and their default distribution locations are:

**Linux**

```
/opt/ibm/4767/include/csinkcl.h
```

**Windows**

```
Program Files\IBM\4767\include\csinkcl.h
```

To include these verb declarations, use the following compiler directive in your program:

**Linux**

```
#include "csinkcl.h"
```

**Windows**

```
#include "csinkcl.h"
```

\(^1\) The term *verb* implies an action that an application program can initiate. Some systems and publications use the term *callable service*. 

© Copyright IBM Corp. 2016, 2019
When issuing a call to a security API verb, code the verb entry-point name in uppercase characters. Separate the parameter identifiers with commas and enclose them in parentheses. End the call with a semicolon character. For example:

```
CSNBCKI (&return_code,
    &reason_code,
    &exit_data_length, /* exit_data_length */
    exit_data, /* exit_data */
    clear_key,
    key_token);
```

**Note:** The third and fourth parameters of a CCA call, `exit_data_length` and `exit_data`, are not currently supported by the Support Program. Although it is permissible to code null address pointers for these parameters, it is recommended that you specify a long integer valued to 0 with the `exit_data_length` parameter.

---

### Compiling and linking CCA application programs

The Support Program includes the C Language source code and the make file for a sample program. The file and its default distribution location is:

**Linux** /opt/ibm/4767/samples

**Windows**

\Program Files\IBM\4767\samples

Compile application programs that use CCA and link the compiled programs to the CCA library. The library and its default distribution location is:

**Linux** /usr/lib64/libcsulcca.so

**Windows**

\Program Files\IBM\4767\lib\csuncca.lib

\Windows\System32\CCA_JNI.dll

---

### Viewing coprocessor hardware errors

Most errors that occur in the HSM are logged in a file that can be viewed to look for or to investigate HSM errors.

In Linux, HSM messages are logged in the `/var/log/messages` file. To aid in problem determination, it is always a good idea to include this file and also the `/var/log/messages` file along with the `/var/log/debug` file when contacting the Crypto team about an HSM problem.

In Windows, HSM messages are logged in files located in the `\Program Files\IBM\4767\logs` directory. There are three types of files in this directory, namely message files from the HSM's asymmetric logging facility (file name contains "msg"), trace message files from the driver (file name contains "trc"), and emergency dump files generated by the HSM (file name contains "dmp"). To aid in problem determination, it is always a good idea to include these files when contacting the Crypto team about an HSM problem.

---

### Enhancing throughput with CCA and the coprocessor

When using the CCA API, the characteristics of your host application program will affect performance and throughput of the cryptographic coprocessor. There are two areas you should understand in order to evaluate performance and design your application to obtain the best performance from the coprocessor. One is multithreading and multiprocessing, and the other is caching DES, RSA, ECC, HMAC, and AES keys.
**Multithreading and multiprocessing**

The CCA application running inside the cryptographic coprocessor can process several CCA requests simultaneously. The coprocessor contains several independent hardware elements, including the RSA engine, DES engine, CPU, random-number generator, and PCIe communications interface. These elements can all be working at the same time, processing parts of different CCA verbs. By working on several verbs at the same time, the coprocessor can keep all of its hardware elements busy, maximizing the overall system throughput.

In order to take advantage of this capability, your host system must send multiple CCA requests to the coprocessor without waiting for each one to finish before sending the next one. The best way to accomplish this is to design a multithreaded host application program, in which each thread can independently send CCA requests to the coprocessor. For example, a Web server can start a new thread for each request it receives over the network. Each of these threads will send the required cryptographic requests to the coprocessor, independent of what the other threads are doing. By doing this, you guarantee that the coprocessor is not under-utilized. Another option is to have several independent host application programs all using the coprocessor at the same time.

**Caching DES, RSA, ECC, HMAC, and AES keys**

The CCA firmware for the coprocessor keeps copies of recently used DES, RSA, ECC, HMAC, and encrypted (not clear) AES keys in caches inside the secure module. The keys are stored in a form that has been decrypted, validated, and ready for use. If the same key is reused in a later CCA request, the coprocessor can use the cached copy, thus avoiding the overhead associated with decrypting and validating the key token. In addition, for retained RSA keys, the cache eliminates the overhead of retrieving the key from the internal flash EPROM memory.

As a result, applications that reuse a common set of keys can run much faster than those which use different keys for each transaction. Most common applications use a common set of DES keys, RSA and ECC private keys, HMAC keys, and encrypted AES keys, and the caching is very effective in improving throughput. RSA keys, and ECC public keys and AES clear keys which have very little processing overhead, are not cached.
Chapter 7. Building Java applications to use with the CCA JNI

This chapter includes the following:

- An overview of the way in which users can build Java programs to use with the CCA Support Program.
- The procedure for compiling a program calling the CCA Java Native Interface (JNI).

Note: The file locations referenced in this chapter are the default directory paths.

Overview

The CCA Support Program includes a CCA Java Native Interface (JNI). Application programmers can build Java applications to use with the CCA Support Program.

The CCA JNI provides a hikmNativeInteger class and, beginning with Release 5.3, the hikmNativeLong and hikmNativeNumber classes. The hikmNativeInteger class encapsulates the primitive type 'int' and, beginning with Release 5.3, inherits the hikmNativeNumber class. In Release 5.3 or later, the hikmNativeLong class encapsulates the primitive type 'long' and inherits the hikmNativeNumber class, while the hikmNativeNumber class is an empty abstract class for use in the native interface calls.

The file named hikmNativeInteger.html and, beginning with Release with 5.3, the files named hikmNativeLong.html and hikmNativeNumber.html, provide information about these classes. These files are typically located in the default directory:

Linux /opt/ibm/4767/samples

Windows
   \Program Files\IBM\4767\samples

Note: Beginning with Release 5.3, the hikmNativeInteger class is deprecated and any future releases may not provide it. Use the hikmNativeLong class instead.

All CCA JNI-related classes are in com.ibm.crypto.cca.jni. Be sure to include this statement in your Java application programs that call the CCA JNI:

```java
import com.ibm.crypto.cca.jni.*;
```

Compiling and running a CCA JNI program

After installing CCA, all of the required CCA Java Native Interface (JNI) libraries and files are contained in the following directories by default:

Linux /opt/ibm/4767/cnm
   /usr/lib64

Windows
   \Program Files\IBM\4767\cnm
   \Windows\System32

For Linux, the CCA JNI file is named CCA_JNI.so and is located in the /usr/lib64 directory, while for Windows, the CCA JNI file is named CCA_JNI.dll and is located in the \Windows\System32 directory.

The abstract class hikmNativeNumber, located in the directory for CNM, is in the com.ibm.crypto.cca.jni package of CNM.jar.
To compile or run a CCA JNI program, the classpath must point to the CNM.jar file. Assuming a default distribution, a command to compile a CCA JNI program named mac.java would look like this:

**Linux**

```
javac -classpath /opt/ibm/4767/cnm/CNM.jar mac.java
```

**Windows**

```
javac -classpath \Program Files\IBM\4767\cnm\CNM.jar mac.java
```

From the directory that contains the compiled output, set the load library path to include the following library before running a compiled CCA JNI program so that the CCA JNI library file can be found:

**Linux**

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib64
```

**Windows**

```
set LD_LIBRARY_PATH=$LD_LIBRARY_PATH;\Windows\System32
```

The mac.java program can be run using this command from the directory that contains the compiled output:

**Linux**

```
java -classpath /opt/ibm/4767/cnm/CNM.jar:. mac
```

**Windows**

```
java -classpath \Program Files\IBM\4767\cnm\CNM.jar;. mac
```
Appendix A. Initial commands for DEFAULT role

This appendix describes the characteristics of the default role after the coprocessor is initialized. When the coprocessor is initialized, no other access-control data exists. The following summarizes the initial default role:

- The role ID is DEFAULT.
- The required authentication strength is zero.
- It is valid at all times of the day and on all days of the week.
- The only functions permitted are those necessary to load access-control data.

All unauthenticated users are assigned to the DEFAULT role.

Important

The cryptographic node is not secure when unauthenticated users can load access-control data using the DEFAULT role. Restrict these commands to select supervisory roles.

Table 6 lists the access-control commands enabled in the DEFAULT role when the CCA firmware is initially loaded and when the CCA node is initialized.

<table>
<thead>
<tr>
<th>Access-control point (offset)</th>
<th>Command name</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0107'</td>
<td>One-Way Hash, SHA-1</td>
</tr>
<tr>
<td>X'0110'</td>
<td>Set Clock</td>
</tr>
<tr>
<td>X'0111'</td>
<td>Reinitialize Device</td>
</tr>
<tr>
<td>X'0112'</td>
<td>Initialize Access-Control System</td>
</tr>
<tr>
<td>X'0113'</td>
<td>Change User Profile Expiration Date</td>
</tr>
<tr>
<td>X'0114'</td>
<td>Change User Profile Authentication Data</td>
</tr>
<tr>
<td>X'0115'</td>
<td>Reset User Profile Logon-Attempt-Failure Count</td>
</tr>
<tr>
<td>X'0116'</td>
<td>Read Public Access-Control Information</td>
</tr>
<tr>
<td>X'0117'</td>
<td>Delete User Profile</td>
</tr>
<tr>
<td>X'0118'</td>
<td>Delete Role</td>
</tr>
<tr>
<td>X'0119'</td>
<td>Load Function-Control Vector</td>
</tr>
<tr>
<td>X'011A'</td>
<td>Clear Function-Control Vector</td>
</tr>
</tbody>
</table>
Appendix B. Device driver error codes

Each time that the coprocessor is reset, and the reset is not caused by a fault or tamper event, the coprocessor runs through Miniboot, power-on self-test (POST), code-loading, and status routines. During this process, the coprocessor attempts to coordinate with a host-system device driver. Coprocessor resets can occur because of power-on, a reset command sent from the device driver, or because of coprocessor internal activity such as completion of code updates. The coprocessor's fault or tamper-detection circuitry can also reset the coprocessor.

The coprocessor device driver monitors the status of its communication with the coprocessor and the coprocessor hardware-status registers. Programs such as the Coprocessor Load Utility (CLU) and the CCA Support Program can receive unusual status in the form of a 4-byte return code from the device driver.

There are a very large number of possible 4-byte codes, all of which are of the form X'8xxxxxxx'. The most likely codes that might be encountered are described in Table 7. If you encounter codes of the form X'8340xxxx' or X'8440xxxx', and the code is not in the Table, contact the IBM Crypto Team for advice by sending an email to Crypto@us.ibm.com.

<table>
<thead>
<tr>
<th>4-byte return code (hex)</th>
<th>Reason</th>
<th>Considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>8040FFBF</td>
<td>External intrusion</td>
<td>Arises due to the activation of an optional electrical switch connection to the enclosure of the server containing the coprocessor. This condition can be reset.</td>
</tr>
<tr>
<td>8040FFDA</td>
<td>Dead battery</td>
<td>The batteries have been allowed to run out of sufficient power, or have been removed. The coprocessor is zeroized and is no longer functional.</td>
</tr>
<tr>
<td>8040FFDB</td>
<td>Dead battery</td>
<td>The coprocessor is zeroized and is no longer functional.</td>
</tr>
<tr>
<td>8040FFDF</td>
<td>Dead battery</td>
<td>The coprocessor is zeroized and is no longer functional.</td>
</tr>
<tr>
<td>8040FFE8</td>
<td>Temperature tamper</td>
<td>High or low temperature limit has been exceeded. The coprocessor is zeroized and is no longer functional.</td>
</tr>
<tr>
<td>8040FFF3</td>
<td>Voltage tamper</td>
<td>The coprocessor is zeroized and is no longer functional.</td>
</tr>
<tr>
<td>8040FFF9</td>
<td>Mesh tamper</td>
<td>The coprocessor is zeroized and is no longer functional.</td>
</tr>
<tr>
<td>8040FFFF</td>
<td>Reset bit is on</td>
<td>Either low voltage was detected, the internal operating temperature of the coprocessor went out of limits, or the host driver sent a reset command. Try removing and reinserting the coprocessor into the bus slot.</td>
</tr>
<tr>
<td>8040FFFFFF</td>
<td>Battery warning</td>
<td>Battery power is marginal. The battery changing procedure described in the IBM 4767 PCIe Cryptographic Coprocessor Installation Manual should be followed to replace the batteries.</td>
</tr>
<tr>
<td>80xxxxxx (for example, 80400005)</td>
<td>General communication problem</td>
<td>Except for the prior X'8040xxxx' codes, there are additional conditions that arise in host-coprocessor communication. Determine that the host system in fact has a coprocessor. Try removing and reinserting the coprocessor into the bus slot. Run the CLU status command (ST). If problem persists, contact IBM Crypto Support by using the product website.</td>
</tr>
<tr>
<td>8340xxxx</td>
<td>Miniboot-0 codes</td>
<td>This class of return code arises from the lowest-level of reset testing. If codes in this class occur, contact IBM Crypto Support by using the product website.</td>
</tr>
<tr>
<td>4-byte return code (hex)</td>
<td>Reason</td>
<td>Considerations</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>8340038F</td>
<td>Random-number generation fault</td>
<td>Continuous monitoring of the random-number generator has detected a possible problem. There is a small statistical probability of this event occurring without indicating an actual ongoing problem. The CLU status (ST) command should be run at least twice to determine if the condition can be cleared.</td>
</tr>
<tr>
<td>8440xxxx</td>
<td>Miniboot-1 codes</td>
<td>This class of return code arises from the replaceable POST and code-loading code.</td>
</tr>
<tr>
<td>844006B2</td>
<td>Invalid digital signature</td>
<td>The digital signature on the data sent from the CLU utility to Miniboot could not be validated by Miniboot. Be sure that you are using an appropriate file (for example, CR1xxxxx.clu versus CE1xxxxx.clu). If the problem persists, obtain the output of a CLU status report and forward this and a description of what you are trying to accomplish to IBM Crypto Support by using the product website.</td>
</tr>
</tbody>
</table>
Appendix C. DES and PKA master-key cloning

This appendix includes:

- A procedure that outlines how to clone a DES or PKA master key from one coprocessor to another coprocessor using the CNM Utility
- Access-control considerations when cloning

Master-key cloning procedure

The following procedure outlines how to clone a DES or PKA master key from one coprocessor to another coprocessor using the CNM utility. Before using this procedure, you should familiarize yourself with the material presented at “Cloning a DES or PKA master key” on page 33 and “Understanding and managing master keys” in Chapter 2 of the IBM CCA Basic Services Reference and Guide.

Note: Ensure that the CNM utility is at the same level on all machines involved in the cloning procedure.

The master-key cloning procedure that follows makes no assumption about which computer contains the coprocessors used for:

- Share Administration ("SA node")
- Master-key source ("CSS" coprocessor Share-Signing node)
- Master-key target ("CSR" coprocessor Share-Receiving node)

Note: Cloning of AES and APKA master keys is not supported.

The SA key can reside in the same coprocessor as either the CSS or the CSR key, or it can reside in a separate coprocessor node. Any of the coprocessors can reside together in the same computer if multiple coprocessors with CCA installed are available.

The procedure ignores operator actions to:

- Log on and log off, as these steps depend on the specific roles in use at your installation
- Switch between coprocessors when you are using more than one coprocessor within a computer

The procedure is broken down into several phases as outlined in Table 8.

Table 8. Master-key cloning procedure phase overview

<table>
<thead>
<tr>
<th>Phase</th>
<th>Node</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SA</td>
<td>Establish the Share Administration node; create the SA database, generate the SA key, and store its public key and hash into the SA database.</td>
</tr>
<tr>
<td>2a</td>
<td>Source</td>
<td>Establish the source node; generate the “CSS” key and add the public key to the SA database; install the SA public-key.</td>
</tr>
<tr>
<td>2b</td>
<td>SA</td>
<td>Certify the CSS key and store the certificate into the SA database.</td>
</tr>
<tr>
<td>3a</td>
<td>Target</td>
<td>Establish the target node; create a CSR database, generate a “CSR” key, and add the public key to the CSR database for this node; install the SA public-key.</td>
</tr>
<tr>
<td>3b</td>
<td>SA</td>
<td>Certify the CSR key and store the certificate into the CSR database for the target node.</td>
</tr>
<tr>
<td>3c</td>
<td>Source</td>
<td>Obtain shares and current master-key verification pattern information.</td>
</tr>
<tr>
<td>3d</td>
<td>Target</td>
<td>Install shares and confirm new master-key; set the master key.</td>
</tr>
</tbody>
</table>

For each target node, repeat Phase 3 procedures.
Before undertaking the master-key cloning procedure, it is recommended that you complete the form found in Table 9 and the worksheet in Figure 1 on page 53.

**Table 9. Cloning responsibilities, profiles and roles**

<table>
<thead>
<tr>
<th>Task</th>
<th>Node</th>
<th>Profile</th>
<th>Role</th>
<th>Responsible individual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audit access controls</td>
<td>SA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate SA key</td>
<td>SA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register SA-key hash</td>
<td>SA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register SA key</td>
<td>SA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audit access controls</td>
<td>CSS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate CSS key</td>
<td>CSS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obtain CSS master key</td>
<td>CSS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register SA-key hash</td>
<td>CSS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register SA key</td>
<td>CSS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Certify CSS key</td>
<td>SA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audit access controls</td>
<td>CSR1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate CSR key</td>
<td>CSR1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register SA-key hash</td>
<td>CSR1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register SA key</td>
<td>CSR1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Certify CSR1 key</td>
<td>SA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obtain shares</td>
<td>CSS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Install shares</td>
<td>CSR1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verify CSR new</td>
<td>CSR1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set CSR master-key</td>
<td>CSR1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audit access controls</td>
<td>CSR2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate CSR key</td>
<td>CSR2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register SA-key hash</td>
<td>CSR2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register SA key</td>
<td>CSR2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Certify CSR2 key</td>
<td>SA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obtain shares</td>
<td>CSS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Install shares</td>
<td>CSR2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verify CSR new</td>
<td>CSR2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set CSR master-key</td>
<td>CSR2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Phase 1: Establish the Share Administration node

Using the coprocessor designated as the Share Administration (SA) node, follow the steps in Table 10. Note that this coprocessor can also serve as the master-key source or a master-key target node.

Table 10. Master-key cloning procedure: establish SA node

<table>
<thead>
<tr>
<th>Phase</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Audit the appropriateness of the access controls.</td>
</tr>
<tr>
<td>1.2</td>
<td>Perform date and time synchronization and ensure that the FCV authorization (fcv_td4kECC521_ECDSA.crt) is installed.</td>
</tr>
<tr>
<td>1.3</td>
<td>Confirm (or install) the master key.</td>
</tr>
<tr>
<td>1.4</td>
<td>Using the facilities of your operating system, erase any prior SA database from the SA database media.</td>
</tr>
<tr>
<td>1.5</td>
<td>If not already established, enter the Environment ID (EID):</td>
</tr>
<tr>
<td></td>
<td>• Crypto Node, Set Environment ID.</td>
</tr>
<tr>
<td></td>
<td>• Enter the EID, Load.</td>
</tr>
<tr>
<td>1.6</td>
<td>Generate the SA key:</td>
</tr>
<tr>
<td></td>
<td>• Crypto Node, Share Administration, Create Keys, Share Administration Key.</td>
</tr>
<tr>
<td></td>
<td>• Accept the default SA public-key and private-key labels, and enter the location and name of the SA database (“sa.db”).</td>
</tr>
<tr>
<td></td>
<td>• Create</td>
</tr>
<tr>
<td></td>
<td>• Record the SA-key hash value for use later in the procedure.</td>
</tr>
</tbody>
</table>
Table 10. Master-key cloning procedure: establish SA node (continued)

<table>
<thead>
<tr>
<th>Phase</th>
<th>Task</th>
</tr>
</thead>
</table>
| 1.7   | Register the SA public-key hash:  
|       | • **Crypto Node, Share Administration, Register Share Administration Key, SA-Key Hash.**  
|       | • Enter the SA database file name and location, *Next.*  
|       | • Enter the SA public-key label (or accept the default).  
|       | • Enter the SA-key hash, *Register.* |
| 1.8   | Register the SA public-key:  
|       | • **Crypto Node, Share Administration, Register Share Administration, SA Key.**  
|       | • Enter the SA database file name and location, *Next.*  
|       | • Enter the SA public-key label (or accept the default), *Register.* |

Phase 2: Establish the source node

Using the coprocessor designated as the master-key source node, follow the steps in Table 11. Note that this coprocessor can also serve as the SA node.

Table 11. Master-key cloning procedure: establish source (CSS) node

<table>
<thead>
<tr>
<th>Phase</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>2a.1</td>
<td>Audit the appropriateness of the access controls.</td>
</tr>
<tr>
<td>2a.2</td>
<td>Perform time synchronization and ensure that the FCV authorization <em>(fcv_id4kECC521_ECDSA.crt)</em> is installed.</td>
</tr>
</tbody>
</table>
| 2a.3  | Confirm the coprocessor serial number:  
|       | • **Crypto Node, Status.**  
|       | • **Adapter.**  
|       | • Note the coprocessor serial number, *Cancel.* |
| 2a.4  | Confirm (or install) the master key. |
| 2a.5  | Obtain the current master-key-verification information:  
|       | • **Master Key, Verify, Current.**  
|       | • **Save** to transport media, *Cancel.* |
| 2a.6  | If not already established, enter the Environment ID (EID):  
|       | • **Crypto Node, Set Environment ID.**  
|       | • Enter the EID, *Load.* |
| 2a.7  | If not already established, set the number of shares values, “m” and “n”:  
|       | • **Crypto Node, Share Administration, Set Number of Shares.**  
|       | • Set the maximum and minimum number of required shares, *Load.* |
| 2a.8  | Generate the CSS key:  
|       | • **Crypto Node, Share Administration, Create Keys, CSS Key.**  
|       | • Enter the CSS key label (for example, “CSS.KEY”).  
|       | • Confirm the coprocessor serial number.  
|       | • Confirm or enter the SA database name and location.  
|       | • *Create.* |
| 2a.9  | If this node is being used as the SA Node, you are done as the last two steps have already been completed. Otherwise, register the SA public-key hash:  
|       | • **Crypto Node, Share Administration, Register Share Administration Key, SA-Key Hash.**  
|       | • Enter the SA database file name and location, *Next.*  
|       | • Enter the SA public-key label (or accept the default).  
|       | • Enter the SA-key hash, *Register.* |
| 2a.10 | Register the SA public-key:  
|       | • **Crypto Node, Share Administration, Register Share Administration, SA Key.**  
|       | • Enter the SA database file name and location, *Next.*  
|       | • Enter the SA public-key label (or accept the default), *Register.* |
Phase 3: Establish target node and clone master key

Using the designated nodes, establish the target node and clone the master key following the steps in Table 12.

Note that this coprocessor can also serve as the SA node.

Table 12. Master-key cloning procedure: establish CSR node, clone master key

<table>
<thead>
<tr>
<th>Phase</th>
<th>Node</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>At the target node...</strong>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3a.1</td>
<td>Target</td>
<td>Audit the appropriateness of the access controls.</td>
</tr>
<tr>
<td>3a.2</td>
<td>Target</td>
<td>Perform date and time synchronization and ensure that the FCV authorization (fcv_td4kECC521_ECDSA.crt) is installed.</td>
</tr>
</tbody>
</table>
| 3a.3  | Target | Confirm the coprocessor serial number:  
|       |       | • Crypto Node, Status.  
|       |       | • Adapter.  
|       |       | • Note the coprocessor serial number, Cancel. |
| 3a.4  | Target | Ensure the existence of a (temporary) master key. |
| 3a.5  | Target | If not already established, enter the Environment ID (EID):  
|       |       | • Crypto Node, Set Environment ID.  
|       |       | • Enter the EID (for example, “CSR1 NODE” and extend with spaces to 16 entered characters).  
|       |       | • Load. |
| 3a.6  | Target | If not already established, set the number of shares values, “m” and “n”:  
|       |       | • Crypto Node, Share Administration, Set Number of Shares.  
|       |       | • Set the maximum and minimum number of required shares.  
|       |       | • Load. |
| 3a.7  | Target | Using the facilities of your operating system, erase the csr.db data file. |
| 3a.8  | Target | Generate the CSR key:  
|       |       | • Crypto Node, Share Administration, Create Keys, CSR Key.  
|       |       | • Enter the CSR key label (for example, “CSR1.KEY”).  
|       |       | • Confirm the coprocessor serial number.  
|       |       | • Select the key size.  
|       |       | • Provide the CSR database name and location (for example, “CSR1.DB”).  
|       |       | • Create. |
| 3a.9  | Target | If this node is being used as the SA Node, you are done as the last two steps have already been completed. Otherwise, register the SA public-key hash:  
|       |       | • Crypto Node, Share Administration, Register Share Administration, SA-Key Hash.  
|       |       | • Enter the SA database file name and location, Next.  
|       |       | • Enter the SA public-key label (or accept the default).  
|       |       | • Enter the SA-key hash, Register. |
| 3a.10 | Target | Register the SA public-key:  
|       |       | • Crypto Node, Share Administration, Register Share Administration Key, SA Key  
|       |       | • Enter the SA database file name and location, Next.  
|       |       | • Enter the SA public-key label (or accept the default), Register. |
| **At the SA node...**... |
| 3b.1  | SA | Certify the CSS key (as required):  
|       |       | • Crypto Node, Share Administration, Certify Keys, CSS Key.  
|       |       | • Enter the name and path for the SA database, Next.  
|       |       | • Confirm the CSS key label, the coprocessor serial number, and the SA Environment ID.  
|       |       | • Certify. |
Table 12. Master-key cloning procedure: establish CSR node, clone master key  (continued)

<table>
<thead>
<tr>
<th>Phase</th>
<th>Node</th>
<th>Task</th>
</tr>
</thead>
</table>
| 3b.2  | SA   | Certify the CSR key:  
|       |      | • Crypto Node, Share Administration, Certify Keys, CSR Key.  
|       |      | • Enter the name and path for the SA and CSR databases, Next.  
|       |      | • Confirm the SA key label, CSR key label, and the SA Environment ID.  
|       |      | • Enter the CSR serial number.  
|       |      | • Certify. |

At the source node...

3c.1 Source  
Obtain at least “m” of “n” shares. Perform the following for each share. Note that logon and logoff might be required to obtain each share.  
• Crypto Node, Share Administration, Get Share.  
• Select the share. Note that if you are obtaining an additional set(s) of shares, the “Distributed” messages might not be meaningful.  
• Enter the name and path for the SA and CSR databases, Next.  
• Confirm the CSS key label, CSS coprocessor serial number, and the CSR coprocessor serial number.  
• Get Share.  
Repeat as required.

At the target node...

3d.1 Target  
Install “m” (of “n”) shares. Perform the following for each share and observe the response. The response indicates when enough shares have been installed to form the new master-key. Note that logon and logoff might be required to install each share.  
• Crypto Node, Share Administration, Load Share.  
• Select the share.  
• Enter the name and path for the CSR and SA databases, Next.  
• Confirm the CSS key label, the CSS coprocessor serial number, and the CSR coprocessor serial number.  
• Load Share.  
Observe the response. Loading sufficient shares completes the new master-key.  
Repeat as required.

3d.2 Target  
Confirm the new master-key:  
• Master Key, Verify, New.  
• Compare, select the file, OK, Cancel

3d.3 Target  
Using the facilities of your operating system, erase the csr.db data file. This is not a security issue but rather to avoid complications should you perform another master-key cloning operation.

3d.4 Target  
As appropriate, set the master key:  
• Master Key, Set.  
• OK.

Access-control considerations when cloning

There are three classes of roles to consider for cloning operations:  
• Roles at the share-administration node  
• Roles at the source node, CSS  
• Roles at the target node, CSR  
You must also consider your security policy.

Your security policy needs to define who will have the authority to:
- Generate a random master-key at the source node.
- Set the master key, the action which brings a new master key into operation. Note that keys enciphered by the master key will need to be updated to the changed master key; you need to plan for this action and choose which role will provide this authority.
- Generate the retained RSA keys to certify the public keys of the source and target nodes (the SA key), and to generate the retained keys at the source (CSS) and target (CSR) nodes.
- Register the SA key and its hash, and decide if this will be a split responsibility.

In addition, it must be decided how many individuals must cooperate to clone a master key; of course, they must be selected to avoid collusion.

In deciding the “m” and “n” values, you need to consider when the cloning will take place and if there is the need to reconstitute the master key from a fewer number of shares than the total number obtained from the source node (perhaps because of share-corruption or the unavailability of one or more individuals who can obtain or install a share).

**Note:** The CNM utility places all of the shares from a node into the CSR.DB file. Each share is encrypted under a unique, triple-length DES key which itself is encrypted by the CSR public key of the target node.

Table 13 provides guidance for selecting the permissions applicable to roles related to cloning.

<table>
<thead>
<tr>
<th>Access-control point (offset)</th>
<th>Command name</th>
<th>Verb name</th>
<th>Consideration / role</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'001A'</td>
<td>Set and Activate CMK from NMK (SET SYM-MK)</td>
<td>Master-Key_Process</td>
<td>Critical, must be knowledgeable of the contents of the new master-key register and the implications of a master-key change.</td>
</tr>
<tr>
<td>X'001D'</td>
<td>Compute Verification Pattern</td>
<td>Many</td>
<td>All roles used for cloning.</td>
</tr>
<tr>
<td>X'0020'</td>
<td>Generate Random NMK (RANDOM SYM-MK)</td>
<td>Master-Key_Process</td>
<td>Not critical except that it fills the new master-key register.</td>
</tr>
<tr>
<td>X'0032'</td>
<td>Clear NMK Register (CLEAR SYM-MK)</td>
<td>Master-Key_Process</td>
<td>Probably assigned to the role that can set the master key. This can override the collected shares. Probably should be mutually exclusive with the Generate Random Master Key command (offset X'0020').</td>
</tr>
<tr>
<td>X'0033'</td>
<td>Clear OMK Register (CLR-OLD SYM-MK)</td>
<td>Master-Key_Process</td>
<td>Generally not used.</td>
</tr>
<tr>
<td>X'008E'</td>
<td>Generate Key</td>
<td>Key_Generate Random_Number_Generate</td>
<td>All roles used for cloning.</td>
</tr>
<tr>
<td>X'0090'</td>
<td>Reencipher to Current Master Key</td>
<td>Key-Token_Change</td>
<td>Consider who will update working keys encrypted by the master key.</td>
</tr>
<tr>
<td>X'0100'</td>
<td>PKA96 Digital Signature Generate</td>
<td>Digital_Signature_Generate</td>
<td>Certifier of the SA, CSS, and CSR keys.</td>
</tr>
<tr>
<td>X'0101'</td>
<td>PKA96 Digital Signature Verify</td>
<td>Digital_Signature_Verify</td>
<td>All roles used for cloning.</td>
</tr>
<tr>
<td>X'0102'</td>
<td>PKA96 Reencipher to Current Master Key</td>
<td>PKA_Key-Token_Change</td>
<td>Consider who will update working keys encrypted by the master key.</td>
</tr>
<tr>
<td>Access-control point (offset)</td>
<td>Command name</td>
<td>Verb name</td>
<td>Consideration / role</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------------------------</td>
<td>----------------------------------</td>
<td>---------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>X'0103'</td>
<td>PKA96 PKA Key Generate</td>
<td>PKA_Key_Generate</td>
<td>Required to generate the SA, CSS, and CSR keys.</td>
</tr>
<tr>
<td>X'0107'</td>
<td>One-Way Hash, SHA-1</td>
<td>One_Way_Hash</td>
<td>All roles used for cloning.</td>
</tr>
<tr>
<td>X'0114'</td>
<td>Change User Profile Authentication Data</td>
<td>Access_Control_Initialization</td>
<td>Allows changing the passphrase in ANY profile; use with discretion.</td>
</tr>
<tr>
<td>X'0116'</td>
<td>Read Public Access-Control Information</td>
<td>Access_Control_Maintenance</td>
<td>All roles used for cloning.</td>
</tr>
<tr>
<td>X'011C'</td>
<td>Set EID</td>
<td>Cryptographic_Facility_Control</td>
<td>Needed to set up the CSS and CSR nodes.</td>
</tr>
<tr>
<td>X'011D'</td>
<td>Initialize Master Key Cloning</td>
<td>Cryptographic_Facility_Control</td>
<td>Needed to set up the “m-of-n” values at the CSS and CSR nodes.</td>
</tr>
<tr>
<td>X'0200'</td>
<td>PKA Register Public Key Hash</td>
<td>PKA_Public_Key_Hash_Register</td>
<td>Use at the CSS and CSR nodes to ensure the SA key can be recognized; split responsibility with X'0201'.</td>
</tr>
<tr>
<td>X'0201'</td>
<td>PKA Public Key Register</td>
<td>PKA_Public_Key_Register</td>
<td>Use at the CSS and CSR nodes to ensure the SA key can be recognized; split responsibility with X'0200'.</td>
</tr>
<tr>
<td>X'0203'</td>
<td>Delete Retained Key</td>
<td>Retained_Key_Delete</td>
<td>Use to remove obsolete SA, CSS, and CSR keys; be careful about denial of service.</td>
</tr>
<tr>
<td>X'0204'</td>
<td>PKA Clone Key Generate</td>
<td>PKA_Key_Generate</td>
<td>Needed to generate the CSS and CSR keys.</td>
</tr>
<tr>
<td>X'021' - X'021F'</td>
<td>Clone-info (Share) Obtain n (n=1,2,3,...,15)</td>
<td>Master_Key_Distribution</td>
<td>Consider a profile and role for each share to enforce split responsibility.</td>
</tr>
<tr>
<td>X'022' - X'022F'</td>
<td>Clone-info (Share) Install n (n=1,2,3,...,15)</td>
<td>Master_Key_Distribution</td>
<td>Consider a profile and role for each share to enforce split responsibility.</td>
</tr>
<tr>
<td>X'0230'</td>
<td>List Retained Key</td>
<td>Retained_Key_List</td>
<td>All roles used for cloning.</td>
</tr>
</tbody>
</table>
Appendix D. Threat considerations for a digital-signing server

This appendix addresses threats which should be considered when employing the coprocessor with the Support Program in a digital-signing application. Much of the discussion is applicable to other environments in which you might apply the coprocessor.

An organization placing a certification authority (CA), registration authority (RA), Online Certificate Status Protocol (OCSP) responder, or time-stamping service into operation should consider how its installation will address various threats. Table 14 lists potential threats and presents product design and implementation solutions to many of these threats. Notes are included describing steps that you should consider to further mitigate your exposure to problems.

For the case of a digital-signing server, Appendix H of the IBM CCA Basic Services Reference and Guide describes actions that should be used in deploying the coprocessor, policies that should be considered, application functionality which should be included, and so forth.

Plan to reread the contents of Table 14 after first-pass decisions about your installation have been made.

Table 14. Threat considerations for a digital-signing server

<table>
<thead>
<tr>
<th>Threat discussion</th>
<th>Threat mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threats associated with physical attack on the coprocessor</td>
<td></td>
</tr>
</tbody>
</table>
### Table 14. Threat considerations for a digital-signing server (continued)

<table>
<thead>
<tr>
<th>Threat discussion</th>
<th>Threat mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical probing of the coprocessor</td>
<td>The coprocessor electronics incorporate a sophisticated set of active tamper-detection sensors and response mechanism. High and low temperature, voltage levels and sequencing, and physical penetration sensors are designed to detect unusual environmental situations.</td>
</tr>
<tr>
<td></td>
<td>All of the sensitive electronics are enclosed in a physically shielded package, called the security module. Upon detecting a potential tamper event, the coprocessor immediately clears all internal RAM memory which also zeroizes keys used to recover sensitive, persistent data from flash memory. An independent state controller is also reset which indicates that the coprocessor is no longer in a factory-certified condition.</td>
</tr>
<tr>
<td></td>
<td>The various tamper sensors are powered from the time of coprocessor manufacture through the end of life of the coprocessor. The coprocessor digitally signs a query response which can be verified to confirm that the coprocessor is genuine and is not tampered with.</td>
</tr>
<tr>
<td></td>
<td>Note that almost all of the firmware that runs on the main processor within the coprocessor security module is available in the clear on the Internet and is therefore subject to reverse engineering. However, the coprocessor validates digital signatures on code that it is requested to accept so that code modified by an adversary cannot be loaded into the coprocessor.</td>
</tr>
<tr>
<td></td>
<td>The design and implementation is being independently evaluated and is expected to be certified by the USA NIST under the FIPS PUB 140-2 Overall Level 4 standard.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> You must validate the condition of the coprocessor and the code content.</td>
</tr>
<tr>
<td>Physical modification of the coprocessor</td>
<td>The tamper-sensitive electronics are all packaged within the sealed tamper-responding package mounted on the base card. In the process of altering the sensitive electronics, the coprocessor factory certification would be destroyed rendering the device useless.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Physically confirm that a specific, serial-numbered coprocessor is in use and audit its status-query response to confirm that it remains an unaltered IBM coprocessor loaded with appropriate firmware.</td>
</tr>
</tbody>
</table>
Table 14. Threat considerations for a digital-signing server (continued)

<table>
<thead>
<tr>
<th>Threat discussion</th>
<th>Threat mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environmental manipulation of the coprocessor</td>
<td>The coprocessor has sensors to detect environmental stresses which might induce erroneous operation. Abnormal conditions can cause the unit to zeroize.</td>
</tr>
<tr>
<td>An adversary might utilize environmental conditions beyond those of the coprocessor specification to obtain or modify data or program flow for fraudulent coprocessor use. This modification might include manipulation of power lines, clock rates, or exposure to high and low temperatures. As an effect, the coprocessor might get into a situation where instructions are not correctly executed. As a result, security-critical data might get modified or disclosed in contradiction to the security requirements for the coprocessor.</td>
<td></td>
</tr>
<tr>
<td>Substituted process</td>
<td>Note: 1. Auditors need to complete the processes described for them to ensure that the signing key is indeed retained within the appropriate coprocessor. 2. Access to the host system should be supervised so that host-system security measures and proper operation can be relied upon.</td>
</tr>
<tr>
<td>Requests to, and responses from, the coprocessor might be directed to an alternative implementation enabling an adversary to influence results. An alternative implementation might be substituted with differing security features. For example, private-key generation and the production of digital signatures might be performed in an alternative implementation that would enable exposure of the private key.</td>
<td></td>
</tr>
<tr>
<td>Threats associated with logical attack on the coprocessor</td>
<td></td>
</tr>
<tr>
<td>Insertion of faults</td>
<td>The electronic design of the coprocessor renders classical approaches to smart-card attacks infeasible.</td>
</tr>
<tr>
<td>An adversary might determine security critical information through observation of the results of repetitive insertion of selected data. Insertion of selected inputs followed by monitoring the output for changes is a relatively well-known attack method for cryptographic devices. The intent is to determine information based on how the coprocessor responds to the selected inputs. This threat is distinguished by the deliberate and repetitive choice and manipulation of input data as opposed to random selection or manipulation of the physical characteristics involved in input/output operations.</td>
<td>Note: Supervision of the host system and controlling access to the system, both logically and physically, are important security steps to be taken by the using organization.</td>
</tr>
<tr>
<td>Forced reset</td>
<td>The coprocessor is designed to always run through its initial power-on sequence in the event of trap and reset conditions. Each application-level request is treated as a separate unit of work and processed from a single defined set of initial conditions.</td>
</tr>
<tr>
<td>An adversary might force the coprocessor into a nonsecure state through inappropriate termination of selected operations. Attempts to generate a nonsecure state in the coprocessor might be made through premature termination of transactions or communications between the coprocessor and the host, by insertion of interrupts, or by inappropriate use of interface functions.</td>
<td></td>
</tr>
<tr>
<td>Threat discussion</td>
<td>Threat mitigation</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------</td>
</tr>
<tr>
<td><strong>Invalid input</strong></td>
<td>Transaction requests carry authentication information applied in the caller’s domain and validated by the coprocessor. Each request is processed from a single known state with predefined conditions. The coprocessor firmware validates the characteristics of each request to address misuse scenarios.</td>
</tr>
<tr>
<td>An adversary or authorized user of the coprocessor might compromise the security features of the coprocessor through introduction of invalid inputs. Invalid input might take the form of operations which are not formatted correctly, requests for information beyond register limits, or attempts to find and execute undocumented commands. The result of such an attack might be a compromise in the security functions, generation of exploitable errors in operation, or release of protected data.</td>
<td></td>
</tr>
<tr>
<td><strong>Data loading malfunction</strong></td>
<td>As outlined in auditor procedures, the access-control setup should be verified along with confirming the installed coprocessor firmware.</td>
</tr>
<tr>
<td>An adversary might maliciously generate errors in setup data to compromise the security functions of the coprocessor. During the stages of coprocessor preparation which involve loading the coprocessor with special keys, identification of roles, and so forth, the data itself might be changed from the intended information or might be corrupted. Either event could be an attempt to penetrate the coprocessor security functions or to expose the security data in an unauthorized manner.</td>
<td></td>
</tr>
<tr>
<td><strong>Unauthorized program loading</strong></td>
<td>The coprocessor will only accept digitally signed firmware after the signature has been validated. An independent evaluation of IBM’s firmware build and signing procedures and the coprocessor design affirms the trust which can be placed in the identity of loaded firmware. <strong>Note:</strong> An auditor should follow procedures to affirm that specified firmware is in use.</td>
</tr>
<tr>
<td>An adversary might utilize unauthorized programs to penetrate or modify the security functions of the coprocessor. Unauthorized programs might include the execution of legitimate programs not intended for use during normal operation or the unauthorized loading of programs specifically targeted at penetration or modification of the security functions.</td>
<td></td>
</tr>
<tr>
<td><strong>Threats associated with control of access</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Invalid access</strong></td>
<td>An auditor can confirm the permissions granted in each established role and the set of profiles (users) associated with each role. An independent evaluation of the coprocessor firmware implementation and testing has reviewed the integrity of the access-control implementation.</td>
</tr>
<tr>
<td>A user or an adversary of the coprocessor might access information or services without having permission as defined in the role profile. Each role has defined privileges which allow access only to selected services of the coprocessor. Access beyond those specified services could result in exposure of secure information.</td>
<td></td>
</tr>
<tr>
<td><strong>Fraud on first use</strong></td>
<td>IBM’s manufacturing and distribution practice ensures that prior to factory certification the end-user of a coprocessor is unknown and unassigned. Factory-installed software is validated through checking of digital signatures. <strong>Note:</strong></td>
</tr>
<tr>
<td>An adversary might gain access to coprocessor information by unauthorized use of a new, not yet installed coprocessor. An adversary might try to get access to a coprocessor during or directly after the manufacturing process and load fraudulent software into the coprocessor or modify critical data stored within the coprocessor during the manufacturing and factory initialization process before it is shipped to the end customer.</td>
<td>1. The standard installation bring-up process replaces all of the runtime coprocessor firmware.</td>
</tr>
<tr>
<td>2. You should verify that Segments 2 and 3 are “UNOWNED” prior to loading coprocessor firmware for production. This ensures that no residual data remains to influence subsequent operations.</td>
<td></td>
</tr>
</tbody>
</table>

**Table 14. Threat considerations for a digital-signing server (continued)**
Table 14. Threat considerations for a digital-signing server (continued)

<table>
<thead>
<tr>
<th>Threat discussion</th>
<th>Threat mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Impersonation</strong></td>
<td>There are two “user” classes:</td>
</tr>
<tr>
<td></td>
<td>1. (IBM) coprocessor code signer: An independent evaluation of IBM’s procedure for building and signing code assures that legitimate code can be identified by an end-user auditor.</td>
</tr>
<tr>
<td></td>
<td>2. The CCA access-control design protects the integrity and confidentiality of an end-user access-control passphrase from the domain of the end-user process into the coprocessor. The correct passphrase and profile identification grant use of a role.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Host-system security, host-system application design, and administrative policies are required to assure that a designated user’s passphrase is secure.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Threats associated with unanticipated interactions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use of disallowed application functions</strong></td>
<td>The coprocessor design requires configuration of the access-control setup. The CCA firmware has been examined to ensure that functions are disallowed when required commands are not permitted.</td>
</tr>
<tr>
<td>An adversary might exploit interactions between applications to expose sensitive coprocessor or user data. Interactions might include execution of commands that are not required or allowed in the specific application being performed. Examples include use of functions related to master-key management or functions related to symmetric encryption or financial services. Those functions should not have any negative impact on the coprocessor functions required for the digital-signing application.</td>
<td><strong>Note:</strong></td>
</tr>
<tr>
<td></td>
<td>1. Your access-control configuration should follow the principles discussed in Appendix H of the IBM CCA Basic Services Reference and Guide such that only the functions needed for the operational phase can be invoked in this phase.</td>
</tr>
<tr>
<td></td>
<td>2. For the digital-signing application, establish guidelines for a set of roles with very limited capabilities and a setup sequence that restricts the coprocessor functionality to that essential to digital signing.</td>
</tr>
<tr>
<td></td>
<td>In some installations it might be desirable to accommodate a different approach to roles, to address additional application(s) functionalty(s), or both. In these cases, ensure that you review the guidelines and observations in Appendix H of the IBM CCA Basic Services Reference and Guide for applicability to your circumstances.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Threats regarding cryptographic functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cryptographic attack</strong></td>
<td>The coprocessor implements well-established and standardized cryptographic functions.</td>
</tr>
<tr>
<td>An adversary might defeat security functions through a cryptographic attack against the algorithm or through a brute-force attack. This attack might include either signature generation and verification functions or random-number generators.</td>
<td>The random-number generation implementation has been subjected to extensive evaluation under criteria published by the USA NIST and the German Information Security Agency (German Bundesamt für Sicherheit in der Informations Technik or German BSI).</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> For a digital-signing server, observe the guidelines in Appendix H of the IBM CCA Basic Services Reference and Guide.</td>
</tr>
</tbody>
</table>

| Threats regarding digital signatures | |

---

Appendix D. Threat considerations for a digital-signing server 63
<table>
<thead>
<tr>
<th>Threat discussion</th>
<th>Threat mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forging signed data</td>
<td>An adversary might modify data digitally signed by the coprocessor such that this modification is not detectable by the signatory or a third party. This attack might use weaknesses in the secure hash function, weaknesses in the signature encoding, or weaknesses in the cryptographic algorithm used to generate a forged signature.</td>
</tr>
<tr>
<td>Forging data before it is signed</td>
<td>An adversary might modify data to be digitally signed by the coprocessor before the signature is generated within the coprocessor. This attack might use weaknesses in the implementation that allow an adversary to modify data transmitted for signature to the coprocessor before the coprocessor actually calculates the signature.</td>
</tr>
<tr>
<td>Misuse of digital signature function</td>
<td>An adversary might misuse the coprocessor digital signature creation function to sign data that the coprocessor is not supposed to sign.</td>
</tr>
<tr>
<td>Forging signature-verification function</td>
<td>An adversary might modify the function for signature verification such that a false signature is accepted as valid. This attack might try to modify the signature-verification function or signed data to be verified such that the coprocessor returns a success message when this false signature is presented for verification.</td>
</tr>
<tr>
<td>Threat discussion</td>
<td>Threat mitigation</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------</td>
</tr>
</tbody>
</table>
| Deleting a private RSA signature key | By design, a retained private key is deleted only:  
1. Under CCA control with the Retained_Key_Delete verb  
2. By loading the coprocessor CCA firmware*  
3. By removing the coprocessor CCA firmware  
4. By causing a tamper event  

**Note:** To address these exposures:  
1. Selectively enable the Delete Retained Key command, offset X'0203'.  
2. Use host-system access controls to manage usage of the CLU Utility.  
3. Manage physical access to the coprocessor.  

* Reloading the coprocessor firmware with a file such as CEXxxxxx.clu does not zeroize the contents of persistent storage. The file CNWXxxxxx.clu will zeroize persistent storage. See Chapter 4, “Loading and unloading coprocessor firmware,” on page 13. |

<table>
<thead>
<tr>
<th>Threats that monitor information</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Information leakage</td>
<td>Practical means to interpret information leakage are the subject of ongoing research in commercial and governmental laboratories. An in-depth defense should include limiting access to the cryptographic environment and restrictions on the use of specialized equipment in and near the cryptographic environment.</td>
</tr>
</tbody>
</table>
| Linkage of multiple observations | **Note:**  
1. Usage of the cryptographic equipment should be controlled, including following the guidelines in Appendix H of the IBM CCA Basic Services Reference and Guide.  
2. An adversary might well have access to the signed data and signatures, so controls should be put in place to limit a user's ability to submit arbitrary signing requests.  
3. The use of standardized cryptographic procedures and monitoring of the cryptographic community's understanding of the vulnerabilities of these processes (SHA-1, RSA, ISO 9796, X9.31, HMAC, and triple-DES) can provide assurance of secure operation. |

<table>
<thead>
<tr>
<th>Miscellaneous threats</th>
<th></th>
</tr>
</thead>
</table>
### Table 14. Threat considerations for a digital-signing server (continued)

<table>
<thead>
<tr>
<th>Threat discussion</th>
<th>Threat mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Linked attacks</strong></td>
<td><strong>Note:</strong> 1. Use of the cryptographic system should be limited to authorized situations enforced through the coprocessor access controls and through use of host-system controls. 2. Host-system controls and organizational policies should restrict the access to the system for monitoring and the submission of arbitrary requests.</td>
</tr>
<tr>
<td>An adversary might perform successive attacks with the result that the coprocessor becomes unstable or some aspect of the security functionality is degraded. A following attack might then be successfully executed. Monitoring outputs while manipulating inputs in the presence of environmental stress is an example of a linked attack.</td>
<td></td>
</tr>
<tr>
<td><strong>Repetitive attack</strong></td>
<td>Use of the cryptographic system should be limited to authorized situations enforced through the coprocessor access controls and through use of host-system controls. Host-system controls and organizational policies should restrict the access to the system for monitoring and the submission of arbitrary requests.</td>
</tr>
<tr>
<td>An adversary might utilize repetitive undetected attempts at penetration to expose memory contents or to change security-critical elements in the coprocessor. Repetitive attempts related to some or all of the other threats discussed herein might be used to iteratively develop an effective penetration of the coprocessor security. If these attacks can, in all cases, remain undetected, there will be no warning of increased vulnerability.</td>
<td>Auditors must confirm that the digital-signing key, appropriate code, and access-control regime is resident in the authorized coprocessor.</td>
</tr>
<tr>
<td><strong>Cloning</strong></td>
<td></td>
</tr>
<tr>
<td>An adversary might clone part or all of a functional coprocessor to develop further attacks. The information necessary to successfully clone part or all of a coprocessor might derive from detailed inspection of the coprocessor itself or from illicit appropriation of design information.</td>
<td></td>
</tr>
<tr>
<td><strong>Threats addressed by the operating environment</strong></td>
<td><strong>Note:</strong> 1. An auditor must confirm through examination of a coprocessor-signed query response that the device is genuine and that the appropriate code is loaded. 2. The auditor must also confirm that the digital-signing key is a “retained” key in the coprocessor.</td>
</tr>
<tr>
<td><strong>Coprocessor modification and reuse</strong></td>
<td>An organization must establish, enforce, and audit policies which limit the access that a single individual has to the cryptographic system. The setup procedure must ensure that a single user does not have the opportunity to bring an inappropriate system into production.</td>
</tr>
<tr>
<td>An adversary might use a modified coprocessor to masquerade as an original coprocessor so that information assets can be fraudulently accessed. Removal, modification, and reinsertion of that coprocessor into a host system could be used to pass such a combination as an original. This might then be used to access or change the private-signature keys or other security-critical information to be protected.</td>
<td></td>
</tr>
<tr>
<td><strong>Abuse by privileged users</strong></td>
<td></td>
</tr>
<tr>
<td>A careless, willfully negligent, or hostile administrator or other privileged user might create a compromise of the coprocessor assets through execution of actions which expose the security functions or the protected data. A privileged user or administrator could directly implement or facilitate attacks based on any of the threats described here.</td>
<td></td>
</tr>
</tbody>
</table>
Table 14. Threat considerations for a digital-signing server (continued)

<table>
<thead>
<tr>
<th>Threat discussion</th>
<th>Threat mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data modification</td>
<td>Host-system security precautions and organization policies must be defined, enforced, and audited to thwart such attacks.</td>
</tr>
<tr>
<td>Data to be signed by the coprocessor might be modified by an adversary or by faults in the operational environment after it has been approved by the legitimate user, but before the data is submitted to the coprocessor to be signed. Data that has been approved by the legitimate user to be signed might be modified by an adversary, by false or malicious programs, or by environmental errors (for example, transmission errors) after the data has been approved by the legitimate user and before the data is transferred to the coprocessor to be signed.</td>
<td></td>
</tr>
<tr>
<td>Data verification</td>
<td>The coprocessor verifies the signature on code and certain code-loading commands.</td>
</tr>
<tr>
<td>Signed data to be verified by the coprocessor might be modified by an adversary or by faults in the operational environment before it is submitted to the coprocessor for signature verification such that the response of the coprocessor does not reflect the validity of the signature. Signed data submitted by a user might be modified within the coprocessor environment before it is passed to the coprocessor for verification. This might result in a response from the coprocessor that does not reflect the actual validity of the digital signature that should be verified.</td>
<td>The CCA design supports validation of the integrity of requests and responses between the coprocessor and the top layer of CCA code in the host system.</td>
</tr>
<tr>
<td>There is also the possibility that the response of the coprocessor is modified in the coprocessor environment before it is passed to the user that requested the signature verification.</td>
<td><strong>Note:</strong> Host-system security measures must address blocking the modification of request inputs and outputs.</td>
</tr>
</tbody>
</table>
Appendix E. CCA sample code

The CCA software package includes CCA sample routines. These samples can be found in the following default directory:

Linux  /opt/ibm/4767/samples

Windows
   \Program Files\IBM\4767\samples

The latest version of CCA sample routines is available from the CCA sample code page on the product website at https://www.ibm.com/security/cryptocards From there, use the links in the site navigation section to click on CCA sample code.
Appendix F. Coprocessor Load Utility reference

This appendix describes:
- The coprocessor memory segments into which you load the firmware
- The way in which the coprocessor validates firmware loads
- The syntax used to invoke the Coprocessor Load Utility (CLU)
- The loader commands supported by CLU
- A summary of shipped CLU files
- Return codes issued by CLU

Coprocessor memory segments

Coprocessor memory segments are organized as shown in Table 15.

Table 15. Organization of memory segments

<table>
<thead>
<tr>
<th>Segment</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Basic code</td>
</tr>
<tr>
<td></td>
<td>The basic code manages coprocessor initialization and the hardware component interfaces. This code cannot be changed after the coprocessor leaves the factory.</td>
</tr>
<tr>
<td>1</td>
<td>Firmware administration and cryptographic routines</td>
</tr>
<tr>
<td></td>
<td>Firmware in this segment:</td>
</tr>
<tr>
<td></td>
<td>• Administrates the replacement of firmware already loaded to Segment 1.</td>
</tr>
<tr>
<td></td>
<td>• Administrates the loading of data and firmware to Segments 2 and 3.</td>
</tr>
<tr>
<td></td>
<td>• Is loaded at the factory, but can be replaced using the CLU utility.</td>
</tr>
<tr>
<td>2</td>
<td>Embedded operating system</td>
</tr>
<tr>
<td></td>
<td>The coprocessor Support Program includes the operating system. The operating system supports applications loaded into Segment 3. Segment 2 is empty when the coprocessor is shipped from the factory.</td>
</tr>
<tr>
<td>3</td>
<td>Application firmware</td>
</tr>
<tr>
<td></td>
<td>The coprocessor Support Program includes a CCA application program that can be installed into Segment 3. The application functions according to the IBM CCA and performs access control, key management, and cryptographic operations. Segment 3 is empty when the coprocessor is shipped from the factory.</td>
</tr>
</tbody>
</table>

Validation of coprocessor firmware loads

When the coprocessor is shipped from the factory, it has within it the public key needed to validate replacement firmware for Segment 1.

Loading code into coprocessor Segment 2 and Segment 3 is a two-step process for each segment.
1. First, an “owner identifier” for a segment is sent to the coprocessor using an Establish Owner command. The owner identifier is only accepted if the digital signature associated with this identifier can be validated by the public key residing with the immediately lower segment. Once established, ownership remains in effect until a Surrender Owner command is processed by the coprocessor.
2. Second, a “code load” for a segment is sent to the coprocessor. Two different commands are available.
a. Initially use the **Load** command. Load-command data includes a public-key certificate that must be validated by the public key already residing with the next-lower segment. If the certificate is validated, and if the owner identifier in the Load-command data matches the current ownership held by the coprocessor for the segment, and if the complete Load-command data can be validated by the public key in the just-validated certificate, the coprocessor will accept the code and retain the validated public-key for the segment.

b. If a segment already has a public key, a **Reload** command can be used to replace the code in a segment. The coprocessor actions are the same as for a Load command, except that the included certificate must be validated by the public key associated with the target segment rather than the key associated with the next-lower segment.

The embedded operating system, working with the coprocessor hardware, can store security-relevant data items (SRDIs) on behalf of itself and an application in Segment 3. The SRDIs are zeroized upon tamper detection, loading of segment firmware, or a **Surrender Owner** of a segment. Note that the SRDIs for a segment are not zeroized when using the **Reload** command. The CCA application stores the master keys, the FCV, the access-control tables, and retained RSA private keys as SRDI information associated with Segment 3.

IBM signs its own firmware. Should another vendor intend to supply firmware for the coprocessor, that vendor’s **Establish Owner** command and code-signing public-key certificate must have been signed by IBM under a suitable contract. These restrictions ensure that:

* Only authorized code can be loaded into the coprocessor.
* Government restrictions are met relating to the import and export of cryptographic implementations.

**Coprocessor Load Utility syntax**

This section details the syntax used to invoke the Coprocessor Load Utility (CLU), and describes each function available in it. The installation default directory path for the CLU is:

**Linux**

```
/opt/ibm/4767/clu
```

**Windows**

```
Program Files\IBM\4767\clu
```

From the CLU directory, enter this command at a command prompt to display the help menu for CLU:

**Linux**

```
csulclu -h
```

**Windows**

```
csunclu -h
```

The CLU help menu that is displayed is shown:

**Coprocessor Load Utility (CLU) Version 5.5.nn Usage:**

```
CLU -c command [-l log_file] [-a adapter_number] [-d data_file] [-v]
where command is one of the following:

PL : Program load : Load firmware onto the specified adapter.
RS : Reset adapter : Reset the specified adapter.
SS : System status : Check status of all adapters.
ST : Status : Check status of specified adapter.
VA : Validate adapter : Validate the specified adapter's certificate chain.
VF : Verify file : Verify the specified CLU file was signed by IBM.
ZL : zUDX load : Perform a full load of a zUDX.
```

2. In this publication, the terms “load” and “reload” are employed. Other documentation might refer to these operations as “emergency burn” (EmBurn), and “regular burn” or “remote burn” (RemBurn), respectively.
HC : CCA load from HUL : Load CCA onto card with UNOWNED Seg 2.
RC : CCA reload from HUL : Reload CCA onto card with CCA currently loaded.

If no log_file is specified, CLU will read the serial number of the adapter by the adapter_number parameter (0 by default) and create a log file named <serial number>.log

The data_file option is valid only for the following commands:
- PL : The data file provides the signed imaged to load.
- VA : The data file provides the adapter validation CLU file.
- VF : The data file provides the CLU file to verify.
- ZL : The data file provides the zUDx file to use.

HC/RC/HE/RE : The data file provides the HUL file to use.

-v Verbose output. Enables extended output on certain commands.
Note: Adapter numbers are zero-based, so the first adapter in the system is adapter 0.
CLU -h - Invokes this help menu.
Deprecated CLU Invocation:
CLU log_file command [adapter_number] [data_file] [-Q]

Refer to Table 16 for a description of CLU commands and their usage.

Table 16. Summary of CLU commands

<table>
<thead>
<tr>
<th>CLU command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP</td>
<td>This command is not used in this implementation.</td>
</tr>
<tr>
<td>PL</td>
<td>Use the program load command to establish segment ownership and to load or reload microcode into a coprocessor. The CLU PL command requires a data_file. The data_file contains a series of commands that directs what CLU does. Miniboot commands REMBURN1, ESTOWN2, EMBURN2, REMBURN2, SUROWN2, ESTOWN3, EMBURN3, REMBURN3, and SUROWN3 are inferred from information contained in the data file. A single PL data_file can incorporate information for multiple ownership and loading commands.</td>
</tr>
<tr>
<td>RS</td>
<td>The reset adapter command the causes the coprocessor to perform a power-on reset. Although this command is not generally used, it can be helpful should the coprocessor and the host-system software lose synchronization. Be sure to end all host-system software processes that are operating within the coprocessor prior to issuing this command. This is necessary to enable the complete cryptographic subsystem to get to a reset state.</td>
</tr>
<tr>
<td>SS</td>
<td>Use the system status command to obtain the card number, IBM part number, manufacturer's serial number, and Segment 3 description. Here is an example of the output returned by the SS command:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Card #</th>
<th>P/N</th>
<th>S/N</th>
<th>Segment 3 Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00LV498</td>
<td>DV5CW901</td>
<td>5.4.33 CCA</td>
</tr>
<tr>
<td>1</td>
<td>00LV498</td>
<td>DV5CW302</td>
<td>5.4.33 CCA</td>
</tr>
</tbody>
</table>

Note:
1. The card (adapter) number is used for the adapter_number parameter.
2. The part number is used to determine the name of the CLU data_file parameter for the VA command.
3. The serial number is used to identify the physical card assigned to the card number. It is used as the default file name for the log file.
4. The Segment 3 description identifies which CCA release is installed.
Table 16. Summary of CLU commands (continued)

<table>
<thead>
<tr>
<th>CLU command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VF</td>
<td>Use the verify file command to verify that the file identified by the data_file parameter (with that file type clu) is digitally signed by IBM.</td>
</tr>
<tr>
<td>ST</td>
<td>Use the status command to obtain the status of loaded firmware and the release level of other components.</td>
</tr>
<tr>
<td>VA</td>
<td>Use the validate adapter command to confirm the validity of digitally signed messages returned by the coprocessor. Obtains the status of loaded software and the release level of other components. The data is transmitted in a message signed by the coprocessor device key, and then stored in the utility log file. The VA command requires a data_file. The name of the required file is the secure part number followed by v.clu. The utility uses its built-in ECC public key to validate the one-or-more class key certificates contained in the file identified by the data_file parameter. One of these certificates should validate the public key, or chain of public keys, obtained from the coprocessor, and confirm that the coprocessor has not been tampered with.</td>
</tr>
<tr>
<td>ZL</td>
<td>This command is not used in this implementation.</td>
</tr>
<tr>
<td>HC</td>
<td>This command is not used in this implementation.</td>
</tr>
<tr>
<td>RC</td>
<td>This command is not used in this implementation.</td>
</tr>
<tr>
<td>HE</td>
<td>This command is not used in this implementation.</td>
</tr>
<tr>
<td>RE</td>
<td>This command is not used in this implementation.</td>
</tr>
</tbody>
</table>

Note:
1. The CLU system status (SS) and the CLU status (ST) commands can be used while the coprocessor is active running application requests. All other CLU commands must ensure that the coprocessors are not “busy” by ending any applications that might have used a coprocessor. For example, end all applications that use the CCA API.
2. In general, CLU can be invoked by a script file.

Note:
1. log_file: Each time CLU is called, the name of the log file can optionally be specified. If no log file is specified, CLU reads the serial number of the specified adapter number (which defaults to adapter 0) and creates a log file named <serial number>.log. In general, when working with a specific coprocessor (called adapter by CLU), it is strongly recommended that you use its serial number as the log file name. The manufacturer's serial number can be obtained from the label on its mounting bracket. By always using the serial number to name the log file, a complete history of status and code changes for the contents of each coprocessor is maintained. When CLU writes to a log file, it creates the file if it does not exist, otherwise it appends to the existing file. CLU writes to an ASCII text file which contains the same information that is normally displayed on the console.
2. adapter_number: This parameter provides the coprocessor number as established by the device driver. This parameter defaults to 0. The utility logically supports up to eight coprocessors per machine. Coprocessors are designated to the device driver as numbers 0 - 7. You can use the serial number information that you obtain from the CLU system status command or the serial number printed on the end-bracket of the coprocessor to correlate a particular coprocessor to the adapter_number.
3. data_file: This parameter identifies the data file (drive, directory, and file name) used for the PL, VA, or VF loader commands (refer to Table 17 on page 75).
   a. For the PL command (firmware loads and reloads), it is the file name of the firmware image that you are loading into the coprocessor.
   b. For the VA command, this parameter is the class-key certificate file name used to validate the coprocessor response. Refer to Table 5, "Class-key file for use with the CLU VA command" on page 16. Should the class-key certificate file for your part number be missing from the CLU directory, it is possible to get it from the product website. To find a class-key certificate file, first go to the
product website at https://www.ibm.com/security/cryptocards. From there, click the HSM PCIeCC2 link, then click on the FAQ link. The FAQ has a topic called “How do I verify the integrity of the IBM 4767?” that provides links to all of the certificate files for each 4767 coprocessor part number. This FAQ also contains a description of the procedure for validating the coprocessor and its code.

**Coprocessor Load Utility commands**

The Coprocessor Load Utility (CLU) supports the loader commands described in **Table 17**.

**Table 17. CLU loader commands**

<table>
<thead>
<tr>
<th>Loader command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL (program load): Load microcode into coprocessor</td>
<td>Processes a series of the commands as directed by the contents of the data file to establish segment ownership and to load or reload segment firmware. Commands R1, E2, R2, S2, E3, L3, R3, and S3 are inferred from information contained in the data files that you use with the PL command. A single “PL” file can incorporate information for multiple ownership and loading commands.</td>
</tr>
<tr>
<td>RS: Reset the coprocessor</td>
<td>Resets the coprocessor. Generally you will not use this command. You might find this command helpful should the coprocessor and the host-system software lose synchronization. You should end all host-system software processes that are operating within the coprocessor prior to issuing this command to enable the complete cryptographic subsystem to get to a reset state. The command causes the coprocessor to perform a power-on reset.</td>
</tr>
<tr>
<td>SS (system status): Obtain system/box status</td>
<td>Obtains the part number, serial number, and a portion of the Segment 3 firmware image name for each of the installed coprocessors, provided that these are not being used by some application such as CCA. Refer to <strong>Table 18</strong>.</td>
</tr>
<tr>
<td>ST: Obtain coprocessor status</td>
<td>Obtains the status of loaded firmware and the release level of other components. The status is appended to the log files.</td>
</tr>
<tr>
<td>VA: Validate coprocessor status</td>
<td>Obtains the status of loaded firmware and the release level of other components. The data is transmitted in a message signed by the coprocessor device key, and then stored in the utility log file. The utility uses its built-in public key to validate the one-or-more class-key certificates contained in the file identified by the data_file parameter. One of these certificates should validate the public key, or chain of public keys, obtained from the coprocessor, and confirm that the coprocessor has not been tampered with.</td>
</tr>
<tr>
<td>VF: Verify CLU file</td>
<td>Verifies that the .clu file identified by the data file parameter is digitally signed by IBM.</td>
</tr>
<tr>
<td>EP, HC, RC, HE, RE</td>
<td>These commands are not used in this implementation.</td>
</tr>
</tbody>
</table>

In general, the utility can be invoked by a script file.

**Table 18. Typical CLU system status response**

<table>
<thead>
<tr>
<th>Card #</th>
<th>P/N</th>
<th>S/N</th>
<th>Segment 3 Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>-----------------------</td>
</tr>
<tr>
<td>0</td>
<td>00LU365</td>
<td>DV4CB301</td>
<td>5.5.nn CCA</td>
</tr>
</tbody>
</table>
Summary of shipped CLU files

Table 19 shows a list of all the CLU files that are shipped with the Support Program, along with a description of when and how to use each file with CLU. A CLU file name can be used as the data_file parameter of the CLU command indicated. Use the shipped CLU file name as the CLU data_file parameter.

Table 19. Summary of shipped CLU files

<table>
<thead>
<tr>
<th>Shipped CLU file name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>reload_seg1_xipz_factory_to_prod_keyswap_.#.#.#.clu</code></td>
<td>Use as data_file parameter of CLU PL command to reload Segment 1 if the image indicates a factory-fresh coprocessor (image name includes &quot;Factory&quot;). This swaps out the factory key with the production key.</td>
</tr>
<tr>
<td><code>reload_seg1_xipz_.#.#.#.clu</code></td>
<td>Use as data_file parameter of CLU PL command to reload Segment 1 if the image name does not include &quot;Factory&quot; and a different level of Segment 1 is desired.</td>
</tr>
<tr>
<td><code>establish_ownership_then_emergency_reload_seg2_seg3_xip_.#.#.#.clu</code></td>
<td>Use as data_file parameter of CLU PL command to load Segments 2 and 3 when ROM Status of SEG2 indicates &quot;UNOWNED&quot;.</td>
</tr>
<tr>
<td><code>reload_seg2_seg3_xip_.#.#.#.clu</code></td>
<td>Use as data_file parameter of CLU PL command to reload Segments 2 and 3 when ROM Status of SEG2 is &quot;OWNER2: 2&quot; and ROM Status of SEG3 is &quot;OWNER3: 2&quot; and a different level of firmware is desired.</td>
</tr>
<tr>
<td><code>surrender_ownership_seg2_xipz_OID2_.#.#.#.clu</code></td>
<td>Use as data_file parameter of CLU PL command to zeroize any preexisting master keys, retained keys, roles, and profiles.</td>
</tr>
<tr>
<td><code>00LV498v.clu</code></td>
<td>Use as data_file parameter of CLU VA command to validate the segment contents of a coprocessor with part number 00LV498.</td>
</tr>
</tbody>
</table>

Note: Substitute #.#.# above with the version.release.mod of the firmware as specified in the installation readme.txt file.

Coprocessor Load Utility return codes

When CLU finishes processing, it returns a code that can be tested in a script file or in a command file. The returned codes and their meanings are:

CLU return code

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>
Notices

This information was developed for products and services offered in the U.S.A.

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

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

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

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

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd..
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

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

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this
one) and (ii) the mutual use of the information which has been exchanged, should contact Product support using the product website at https://www.ibm.com/security/cryptocards.

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

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

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE: This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.
Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at:


Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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

Other product and service names might be trademarks of IBM or other companies.
## List of abbreviations and acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>APKA</td>
<td>Advanced Encryption Standard Public Key Architecture</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange</td>
</tr>
<tr>
<td>C</td>
<td>Celsius</td>
</tr>
<tr>
<td>CA</td>
<td>Certification Authority</td>
</tr>
<tr>
<td>CBC</td>
<td>Cipher Block Chain</td>
</tr>
<tr>
<td>CCA</td>
<td>Common Cryptographic Architecture</td>
</tr>
<tr>
<td>CLU</td>
<td>Coprocessor Load Utility</td>
</tr>
<tr>
<td>CNI</td>
<td>CCA Node Initialization</td>
</tr>
<tr>
<td>CNM</td>
<td>CCA Node Management</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CSR</td>
<td>Coprocessor Share Receiving</td>
</tr>
<tr>
<td>CSS</td>
<td>Coprocessor Share Signing</td>
</tr>
<tr>
<td>DES</td>
<td>Data Encryption Standard</td>
</tr>
<tr>
<td>ECC</td>
<td>Elliptic Curve Cryptography</td>
</tr>
<tr>
<td>EPROM</td>
<td>Erasable Programmable Read-Only Memory</td>
</tr>
<tr>
<td>FC</td>
<td>Failure Count</td>
</tr>
<tr>
<td>FCV</td>
<td>Function-Control Vector</td>
</tr>
<tr>
<td>FIPS</td>
<td>Federal Information Processing Standard</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>HSM</td>
<td>Hardware Security Module</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machines</td>
</tr>
<tr>
<td>IC</td>
<td>Integrated Circuit</td>
</tr>
<tr>
<td>ID</td>
<td>Identifier</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>KEK</td>
<td>Key-Encrypting Key</td>
</tr>
<tr>
<td>MAC</td>
<td>Message Authentication Code</td>
</tr>
<tr>
<td>OCSP</td>
<td>Online Certificate Status Protocol</td>
</tr>
<tr>
<td>PCIe</td>
<td>Peripheral Component Interconnect Express</td>
</tr>
<tr>
<td>PDD</td>
<td>Physical Device Driver</td>
</tr>
<tr>
<td>PDF</td>
<td>Portable Document Format</td>
</tr>
<tr>
<td>PKA</td>
<td>Public Key Algorithm</td>
</tr>
</tbody>
</table>
POST  Power-On Self-Test
RA    Registration Authority
RAM   Random access memory
RHEL  Red Hat Enterprise Linux
RSA   Rivest-Shamir-Adleman encryption algorithm
SA    Share Administration
SCUP  Smart Card Utility Program
SHA   Secure Hashing Algorithm
SLES  SUSE Linux Enterprise Server
Glossary

This glossary includes some terms and definitions from the *IBM Dictionary of Computing*, New York: McGraw Hill, 1994. This glossary also includes some terms and definitions taken from:

- The *American National Standard Dictionary for Information Systems*, ANS X3.172-1996, copyright 1996 by the American National Standards Institute (ANSI). Copies might be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are identified by the symbol (A) following the definition.

- The *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) following the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) following the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

access  In computer security, a specific type of interaction between a subject and an object that results in the flow of information from one to the other.

access control  Ensuring that the resources of a computer system can be accessed only by authorized users and in authorized ways.

access method  A technique for moving data between main storage and input/output devices.

**Advanced Encryption Standard (AES)**  A symmetric block cipher algorithm defined in Federal Information Processing (FIPS) Standard Number 197 in 2001 as the federal government approved encryption algorithm.

**American National Standard Code for Information Interchange (ASCII)**  The standard code, using a coded character set consisting of seven-bit characters (eight bits including parity check), that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters. (A)

**American National Standards Institute (ANSI)**  An organization consisting of producers, consumers, and general interest groups that establishes the procedures by which accredited organizations create and maintain voluntary industry standards for the United States. (A)

**Application Program Interface (API)**  A functional interface supplied by the operating system or by a separate program that allows an application program written in a high-level language to use specific data or functions of the operating system or the separate program.

authentication  A process used to verify the integrity of transmitted data, especially a message. (T) In computer security, a process used to verify the user of an information system or protected resource.

authorization  In computer security, the right granted to a user to communicate with or make use of a computer system. (T) The process of granting a user either complete or restricted access to an object, resource, or function.
authorize
To permit or give authority to a user to communicate with or make use of an object, resource, or function.

authorized program facility (APF)
A facility that permits identification of programs authorized to use restricted functions.

bus
In a processor, a physical facility along which data is transferred.

card
An electronic circuit board that is plugged into an expansion slot of a system unit. A plugin circuit assembly.

ciphertext
Text that results from the encipherment of plaintext. See also plaintext.

Cipher Block Chaining (CBC)
A mode of operation that cryptographically connects one block of ciphertext to the next plaintext block.

cleartext
Text that has not been altered by a cryptographic process. Synonym for plaintext. See also ciphertext.

Common Cryptographic Architecture (CCA) API
The application program interface described in the IBM CCA Basic Services Reference and Guide.

coprocessor
A supplementary processor that performs operations in conjunction with another processor. A microprocessor on an expansion card that extends the address range of the processor in the host system, or adds specialized instructions to handle a particular category of operations; for example, an I/O coprocessor, math coprocessor, or a network coprocessor.

cryptographic coprocessor (IBM 4767)
An expansion board that provides to a server a comprehensive set of cryptographic functions.

cryptographic node
A node that provides cryptographic services, such as key generation and digital signature support.

cryptography
The transformation of data to conceal its meaning. In computer security, the principles, means, and methods used to so transform data.

data encrypting key
A key used to encipher, decipher, or authenticate data. Contrast with key encrypting key.

Data Encryption Algorithm (DEA)
A 64-bit block cipher that uses a 64-bit key, of which 56 bits are used to control the cryptographic process and eight bits are used to check parity.

Data Encryption Standard (DES)
The National Institute of Standards and Technology (NIST) Data Encryption Standard, adopted by the U.S. government as Federal Information Processing Standard (FIPS) Publication 46 which allows only hardware implementations of the data encryption algorithm.

decipher
To convert enciphered data into clear data. Contrast with encipher.

driver
A program that contains the code needed to attach and use a device.
encipher
To scramble data or to convert data to a secret code that masks the meaning of the data. Contrast with decipher.

enciphered data
Data whose meaning is concealed from unauthorized users or observers. See also ciphertext.

expansion board
Synonym for expansion card.

expansion card
A circuit board that a user can install in an expansion slot to add memory or special features to a computer. Synonym for card.

exporter key
In the CCA, a type of DES KEK that can encipher a key at a sending node. Contrast with importer key.

feature
A part of an IBM product that can be ordered separately.

Federal Information Processing Standard (FIPS)
A standard that is published by the US National Institute of Science and Technology.

Flash-Erasable Programmable Read-Only Memory (Flash EPROM)
A specialized version of erasable programmable read only memory (EPROM) commonly used to store code in small computers.

function-control vector
A signed value provided by IBM to enable the CCA application in the IBM 4767 PCIe Cryptographic Coprocessor to yield a level of cryptographic service consistent with applicable export-and-import regulations.

hardware security module
A type of Secure Cryptographic Device that provides a set of cryptographic services to programs running in a host computer system. It has physical and logical security features which make it infeasible to obtain or alter sensitive values such as cryptographic keys which are protected by the HSM, or to alter the functions performed by the HSM except by authorized updates.

host computer
In regard to the CCA Cryptographic Coprocessor Support Program, the server into which the IBM 4767 PCIe Cryptographic Coprocessor is installed.

importer key
In CCA products, a type of DES KEK that can decipher a key at a receiving node. Contrast with exporter key.

inline code
In a program, instructions that are executed sequentially, without branching to routines, subroutines, or other programs.

interface
A boundary shared by two functional units, as defined by functional characteristics, signal characteristics, or other characteristics as appropriate. The concept includes specification of the connection between two devices having different functions. (T) Hardware, software, or both, that links systems, programs, and devices.
International Organization for Standardization (ISO)

An organization of national standards bodies established to promote the development of standards to facilitate the international exchange of goods and services, and to foster cooperation in intellectual, scientific, technological, and economic activity.

key
In computer security, a sequence of symbols used with an algorithm to encipher or decipher data.

key encrypting key (KEK)
A key used to cipher and decipher other keys. Contrast with data encrypting key.

key storage
In CCA products, a data file that contains cryptographic keys.

master key
In the CCA implementation, the key used to encrypt keys to process other keys or data at the node.

Message Authentication Code (MAC)
In computer security, (1) a number or value derived by processing data with an authentication algorithm, (2) the cryptographic result of block-cipher operations on text or data using the cipher block chain (CBC) mode of operation.

multiuser environment
A computer system that supports terminals and keyboards for more than one user at the same time.

National Institute of Science and Technology (NIST)
The current name for the US National Bureau of Standards.

node
In a network, a point at which one or more functional units connects channels or data circuits.
(I) The endpoint of a link or a junction common to two or more links in a network. Nodes can be processors, communication controllers, cluster controllers, or terminals. Nodes can vary in routing and other functional capabilities.

passphrase
In computer security, a string of characters known to the computer system and to a user; the user must specify it to gain full or limited access to the system and the data stored therein.

plaintext
Data that has not been altered by a cryptographic process. Synonym for cleartext. See also ciphertext.

power-on self-test (POST)
A series of diagnostic tests that runs automatically when device power is turned on.

private key
In computer security, a key that is known only to the owner and used with a public key algorithm to decipher data. Data is enciphered using the related public key. Contrast with public key. See also public key algorithm.

procedure call
In programming languages, a language construct for invoking execution of a procedure. (I) A procedure call usually includes an entry name and the applicable parameters.

profile
Data that describes the significant characteristics of a user, a group of users, or one-or-more computer resources.
**public key**
In computer security, a key that is widely known and used with a public key algorithm to encipher data. The enciphered data can be deciphered only with the related private key. Contrast with *private key*. See also *public key algorithm*.

**Public Key Algorithm (PKA)**
In computer security, an asymmetric cryptographic process that uses a public key to encipher data and a related private key to decipher data. Contrast with *data encryption algorithm* and *data encryption standard algorithm*. See also *RSA algorithm*.

**Random Access Memory (RAM)**
A storage device into which data is entered and from which data is retrieved in a nonsequential manner.

**Read-Only Memory (ROM)**
Memory in which stored data cannot be modified routinely.

**RSA algorithm**
A public key encryption algorithm developed by R. Rivest, A. Shamir, and L. Adleman.

**security**
The protection of data, system operations, and devices from accidental or intentional ruin, damage, or exposure.

**system administrator**
The person at a computer installation who designs, controls, and manages the use of the computer system.

**throughput**
A measure of the amount of work performed by a computer system over a given period of time; for example, number of jobs-per-day. (A) (I) A measure of the amount of information transmitted over a network in a given period of time; for example, a network’s data-transfer-rate is usually measured in bits-per-second.

**utility program**
A computer program in general support of computer processes. (T)

**verb**
A function possessing an entry-point name and a fixed-length parameter list. The procedure call for a verb uses the syntax standard to programming languages.

**4765**
IBM 4765 PCIe Cryptographic Coprocessor (predecessor to IBM 4767 PCIe Cryptographic Coprocessor).

**4767**
IBM 4767 PCIe Cryptographic Coprocessor.
Index

A
access-control system
  considerations when cloning 56
  initial state 27
  initialization 27
  overview 26
  role 28
  user profile 29
application programs
  compile 42
  link to CCA 42
auto-set, master key 32

B
batteries, coprocessor
  removal 4
  status 26
battery 3

C
C programming language
  verb calls 41
choosing among coprocessors 24
clock-calendars, synchronization 25
CLU (Coprocessor Load Utility)
  commands 75
  firmware validation 71
  overview 13
  return codes 76
CNI list 20
CNI utility (CCA Node Initialization Utility)
  overview 20
  using, node setup 38
CNM (CCA Node Management Utility)
  configure 25
  defaults 25
  overview 20
compile, application programs 42
components, Support Program 7
configure
  CNM utility 25
coprocessor
  load, firmware 13
  memory segments 71
  polling information 26
  status, batteries 26
create
  KEK 37
  key label 37
  key storage 36
  master key 32
  role 28
  user profile 29
cryptographic key management 31

D
DEFAULT role
  description 27
  initial use 27, 47
defaults
  CNM utility 25
define
  role 28
  user profile 29
delete
  role 29
  user profile 30
description
  DEFAULT role 27
  KEKs 31
  master key 31
download, Support Program 4

E
edit
  role 28
  user profile 30
  establish owner command 71

F
file permissions, Linux 11
firmware validation, CLU 71
function-control vector
  definition 1
  load 25

I
initial state, access-control system 27
initial use, DEFAULT role 27, 47
initialization
  access-control system 27
  key storage 36
  initialization of the CCA node 24
install host firmware
  Linux 8
installation, Support Program
  checklist 2
  into coprocessor 13
  onto host computer 7
  overview 1

K
KEKs
  create 37
  description 31
  primary 31
  storage 37
  key label, create 37
  key management, cryptographic 31
key storage
- create 36
- delete keys 36
- initialization 36
- key label, create 37
- management 35
- reencipher 36

L
- link to CCA, application programs 42
- Linux
  - file permissions 11
  - load command 72
  - load coprocessor firmware
    - commands 75
    - establish owner command 71
    - load command 72
    - owner identifier 71
    - reload command 72
    - surrender owner command 71
  - logon-attempt-failure count, reset 30

M
- makefile 42
- management
  - cryptographic key 31
  - key storage 35
  - master key 31
- master key
  - auto-set 32
  - create 32
  - description 31
  - management 31
  - new, set 32
  - registers 31
  - verification 32
- master-key administration 31
- master-key cloning 51
- memory segments, coprocessor 71

N
- node
  - setup, production-environment 22
  - test 22
- notices
  - notices statement 77

O
- ordering
  - batteries 3
  - overview 3
  - placing orders 3
- overview
  - access-control system 26
  - CLU 13
  - CNI 20
  - CNM 20
  - installation, Support Program 1
- owner identifier 71

P
- performance, enhancing 42
- permit, access-control commands 28
- placing orders 3
- polling information, coprocessor 26
- production-environment, node setup 22

R
- reencipher stored keys 36
- registers, master key 31
- reload command 72
- reset logon-attempt-failure count 30
- restrict, access-control commands 28
- return codes, CLU 76
- role
  - create 28
  - define 28
  - delete 29
  - modify 28

S
- sample routine, C programming language
  - makefile 42
  - source code 42
- security-relevant data item (SRDI) 72
- set new master-key 32
- setup
  - production-environment node 22
  - test node 22
- SRDI, security-relevant data item 72
- status, coprocessor batteries 26
- storage, KEKs 37
- stored keys, reencipher 36
- Support Program
  - components 7
  - coprocessor load 13
  - download 4
  - host install 7
  - host remove 7
  - overview, installation 1
  - surrender owner command 71
  - synchronization, clock-calendars 25
  - syntax
    - verb calls, C programming language 41
- system requirements
  - Linux 7

T
- test setup, node 22
- threat considerations, digital-signing server 59
- throughput, enhancing 42

U
- user profile
  - create 29
  - define 29
  - delete 30
  - edit 30
  - reset logon-attempt-failure count 30
- utilities
  - CLU 13
V
validation, coprocessor firmware 71
verb calls, C programming language 41

Z
zeroization of the CCA node 24

utilities (continued)
CNI 20, 38
CNM 20

verification, master key 32