4765 PCIe Cryptographic Coprocessor CCA Support Program Installation Manual

Release 44
Note
Before using this information and the product it supports, be sure to read the general information printed under "Notices" on page 85.

Third Edition, October 2015
This edition applies to:
• IBM 4765 PCIe Cryptographic Coprocessor
• Release 4.4 of the licensed CCA Cryptographic Coprocessor Support Program for the following 32-bit operating system:
  – SUSE Linux Enterprise Server 11 Service Pack 2 from Novell (SLES 11 SP2)
  – SUSE Linux Enterprise Server 11 Service Pack 3 from Novell (SLES 11 SP3)

Each release is designed to provide software support for an IBM 4765 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 http://www.ibm.com/security/cryptocards for an updated version of this publication.

This and other publications related to the IBM 4765 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 2013, 2015.
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 ........................................ 3
Ordering and installing coprocessor hardware ........................................ 3
  Ordering coprocessors ......................................... 3
  Ordering optional smart card readers ................................. 3
  Maintaining coprocessor batteries ................................... 3
  Installing IBM 4765 hardware ..................................... 4
  Downloading the software ....................................... 4

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

Chapter 4. Loading and unloading coprocessor software ........................................ 13
Loading coprocessor software ....................................... 13
Unloading coprocessor software and zeroizing the node .................. 17

Chapter 5. Using the CNM and CNI utilities to manage the cryptographic node ............... 19
Overview .................................................. 19
Using the CNM and CNI utilities, sample scenarios ..................... 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 .................................................. 27
  Modifying existing roles ......................................... 29
  Defining a user profile .......................................... 29
  Modifying existing user profiles ................................... 30
Managing cryptographic keys .......................................... 31
  Managing the master keys ........................................ 32
  Managing key storage .......................................... 37
  Creating and storing primary DES KEKs ............................ 38
Establishing other nodes using CNI ................................... 39

Chapter 6. Building applications to use with the CCA API ....................................... 41
Overview .................................................. 41
Calling CCA verbs in C program syntax ................................ 41
Compiling and linking CCA application programs ...................... 42

© Copyright IBM Corp. 2013, 2015
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viewing coprocessor hardware errors</td>
<td>42</td>
</tr>
<tr>
<td>Enhancing throughput with CCA and the coprocessor</td>
<td>42</td>
</tr>
<tr>
<td>Chapter 7. Building Java applications to use with the CCA JNI</td>
<td>45</td>
</tr>
<tr>
<td>Overview</td>
<td>45</td>
</tr>
<tr>
<td>Compiling and running a CCA JNI program</td>
<td>45</td>
</tr>
<tr>
<td>Appendix A. Initial commands for DEFAULT role.</td>
<td>47</td>
</tr>
<tr>
<td>Appendix B. Machine-readable-log contents</td>
<td>49</td>
</tr>
<tr>
<td>Appendix C. Device driver error codes</td>
<td>51</td>
</tr>
<tr>
<td>Appendix D. DES and PKA master-key cloning</td>
<td>53</td>
</tr>
<tr>
<td>Master-key cloning procedure</td>
<td>53</td>
</tr>
<tr>
<td>Access-control considerations when cloning</td>
<td>58</td>
</tr>
<tr>
<td>Appendix E. Threat considerations for a digital-signing server</td>
<td>61</td>
</tr>
<tr>
<td>Appendix F. Sample programs</td>
<td>71</td>
</tr>
<tr>
<td>Appendix G. Coprocessor Load Utility reference</td>
<td>79</td>
</tr>
<tr>
<td>Notices</td>
<td>85</td>
</tr>
<tr>
<td>Trademarks</td>
<td>87</td>
</tr>
<tr>
<td>List of abbreviations and acronyms</td>
<td>89</td>
</tr>
<tr>
<td>Glossary</td>
<td>91</td>
</tr>
<tr>
<td>Index</td>
<td>97</td>
</tr>
</tbody>
</table>
## Figures

1. Typical CLU status response .................................................. 15
2. Role Management panel .......................................................... 28
3. Profile Management panel ........................................................ 30
4. Load Master Key panel ............................................................. 34
5. CCA Node Initialization Editor panel ......................................... 39
6. Cloning-information worksheet ............................................... 55
7. Typical CLU system status response ......................................... 82
### Tables

1. Activity checklist, CCA Cryptographic Coprocessor Support Program installation .......................................................... 2
2. Battery-replacement part numbers ........................................................................................................................................ 4
3. Default directory list files ............................................................................................................................................... 10
4. Software segment contents .............................................................................................................................................. 15
5. Class-key file for use with the CLU VA command .............................................................................................................. 17
6. Initial commands for DEFAULT role .................................................................................................................................... 47
7. Device driver error codes in the class X'8xxxxxxx' ............................................................................................................. 51
8. Master-key cloning procedure phase overview .............................................................................................................. 53
9. Cloning responsibilities, profiles and roles ......................................................................................................................... 54
10. Master-key cloning procedure: establish SA node ........................................................................................................ 55
11. Master-key cloning procedure: establish source (CSS) node ........................................................................................ 56
12. Master-key cloning procedure: establish CSR node, clone master key .......................................................................... 57
13. CCA commands related to master-key cloning ............................................................................................................ 59
14. Threat considerations for a digital-signing server ........................................................................................................ 61
15. Verbs called by the sample routine .................................................................................................................................. 71
16. Organization of memory segments ..................................................................................................................................... 79
17. CLU loader commands ....................................................................................................................................................... 81
18. Summary of shipped CLU files ............................................................................................................................................ 82
About this publication

This installation manual describes Release 4.4 of the IBM® Common Cryptographic Architecture (CCA) Support Program (Support Program) for the IBM 4765 PCIe Cryptographic Coprocessor. The Support Program includes device drivers, utilities, and the CCA coprocessor code.

You can obtain Support Program releases to use with the following 32-bit operating systems:
@ • SUSE Linux Enterprise Server 11 Service Pack 2 from Novell (SLES 11 SP2)
@ • SUSE Linux Enterprise Server 11 Service Pack 3 from Novell (SLES 11 SP3)

Use this manual to help with the following tasks:
• Obtain the Support Program through the Internet
• Load the software onto a host computer and into the coprocessors
• Use the utilities 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 libraries
• Obtain guidance for security consideration in application development and operational practices

Audience

The audience for this publication includes:
• System administrators who install the software
• 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,” on page 3 describes how to obtain the cryptographic coprocessor hardware and the CCA Cryptographic Coprocessor Support Program.
• 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 software,” on page 13 describes how to load the operating system and the CCA software 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 libraries.
• 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, “Machine-readable-log contents,” on page 49 details the content of the machine-readable log created by the Coprocessor Load Utility.

• Appendix C, “Device driver error codes,” on page 51 provides error-code information that can be observed when operating the CLU utility.

• Appendix D, “DES and PKA master-key cloning,” on page 53 provides a procedure for DES and PKA master-key cloning.

• Appendix E, “Threat considerations for a digital-signing server,” on page 61 addresses threats to consider when employing the coprocessor and the CCA Support Program in a digital-signing application.

• Appendix F, “Sample programs,” on page 71 provides sample programs in C, including one for the Java Native Interface (JNI).

• “Notices” on page 85 provides legal notices.

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

IBM 4765 PCIe Cryptographic Coprocessor publications

For availability of these publications, check the PCIe Cryptographic Coprocessor Library page of the product website at [http://www.ibm.com/security/cryptocards](http://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 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors
• IBM 4765 PCIe Cryptographic Coprocessor Installation Manual
• IBM 4765 PCIe Cryptographic Coprocessor Smart Card User’s Guide
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 PCIe Cryptographic Coprocessor and the IBM Common Cryptographic Architecture (CCA) CCA Cryptographic Coprocessor Support Program (Support Program). See Table 1 on page 2.

Summary

The Support Program consists of several components, and includes:

- Device drivers and an operating system for the coprocessor hardware
- Support for the CCA application program interface (API)
- A function-control vector (FCV)
- Utility applications that run on the host machine into which the coprocessor has been installed

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.

See the Linux portion of 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 and software.** Chapter 2, “Obtaining coprocessor hardware and software,” on page 3 describes how to order the hardware from IBM, how to download the software through the Internet, and how to install the downloaded files.

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

4. **Set up the cryptographic node.** The CCA Node Management (CNM) utility 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 using the utilities provided with the Support Program, or by either 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 4765 should also be verified.

5. **Link application programs to the CCA libraries.** 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 libraries.

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 System z® 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 Support Program for your operating system.</td>
<td>“Downloading the software” on page 4</td>
</tr>
<tr>
<td>5</td>
<td>Install coprocessor hardware.</td>
<td>“Installing IBM 4765 hardware” on page 4</td>
</tr>
<tr>
<td>6</td>
<td>Load coprocessor software.</td>
<td>Chapter 4, “Loading and unloading coprocessor software,” 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 libraries.</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 43</td>
</tr>
</tbody>
</table>
Chapter 2. Obtaining coprocessor hardware and software

The IBM CCA Cryptographic Coprocessor Support Program (Support Program) is available for download through the Internet on the PCIe Cryptographic Coprocessor Order page at [http://www.ibm.com/security/cryptocards].

This chapter describes:
- How to order and install coprocessor hardware
- How to download the software

Note: Check the IBM product website, [http://www.ibm.com/security/cryptocards] for the approved server list.

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 4765 PCIe Cryptographic Coprocessor, namely the Model 1. The IBM 4765 Model 1 is ordered from IBM as a System z machine type-model (4765-001). The coprocessor is an adapter that can be installed in an IBM-approved x86 server. For thermal reasons, the coprocessor requires at least two, possibly three, PCIe slots that are adjacent to one another. The slots must accept a standard short-type PCIe adapter card. If you encounter difficulty, contact IBM through the Support page at [http://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 up 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 4765 PCIe Cryptographic Coprocessor Smart Card User’s Guide. This document is available on the Library page of the PCIe Cryptographic Coprocessor.

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 4765 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 on it in the format mmyy, where mm is the 2-digit month, 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>
<thead>
<tr>
<th>Part number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>45D5803</td>
<td>Battery-replacement kit. Includes two replacement batteries and one temporary-battery tray.</td>
</tr>
<tr>
<td>74Y0465</td>
<td>Multi-battery replacement pack. Includes 20 replacement batteries. Requires at least one temporary-battery tray.</td>
</tr>
</tbody>
</table>

Special procedures are required to safely replace coprocessor batteries. See "Replacing coprocessor batteries" in the IBM 4765 PCIe Cryptographic Coprocessor Installation Manual. Note especially that the positive and negative ends are opposite what they are on normal batteries. Do not rely on the shape of the batteries to determine polarity. Instead, rely on the plus and minus signs printed on the side of each battery. When installing, you must match the plus sign on the battery to the plus sign marked on the coprocessor battery holder or the temporary-battery tray.

**Installing IBM 4765 hardware**

The IBM 4765 is installed in a manner similar to other PCIe boards. Follow the process described in the IBM 4765 PCIe Cryptographic Coprocessor Installation Manual.

Ensure that you never remove the coprocessor batteries except as outlined in the battery-replacement procedure in the IBM 4765 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**

You download the Support Program software through the Internet. Go to the product website at [http://www.ibm.com/security/cryptocards](http://www.ibm.com/security/cryptocards) select PCIe Cryptographic Coprocessor on the left sidebar, and click the **Software download** link. 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 until you have received your coprocessor. See Step 3 of Table 1 on page 2. At that time you should also check the website for any available fixes. See the Spotlight section of the product website at [http://www.ibm.com/security/cryptocards](http://www.ibm.com/security/cryptocards).

From the Software download page, select the CCA software and firmware that you require by operating-system platform, release level, and Support Program as indicated on the Software download page of the product website, [http://www.ibm.com/security/cryptocards](http://www.ibm.com/security/cryptocards). You are prompted to sign in or complete a registration procedure. After you sign in with a universal IBM user ID, then you are presented with a page from which you choose an offering to download.

If you plan to use the Support Program on multiple host computers, you can copy the downloaded files to the other hosts.

Now you are able to install the Support Program. See Chapter 3, “Installing and configuring the Support Program,” on page 7.
Chapter 3. Installing and configuring the Support Program

After downloading the software as described in Chapter 2, “Obtaining coprocessor hardware and software,” on page 3, follow the procedures in this chapter to install the CCA Cryptographic Coprocessor Support Program onto the coprocessor host computer. (Loading software into the coprocessor is described in Chapter 4, “Loading and unloading coprocessor software,” 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 and 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 4765 PCIe Cryptographic Coprocessor
• The shared libraries necessary to link the CCA application program interface (API) to the coprocessor driver
• The Coprocessor Load Utility and software files necessary to load the operating system and the CCA application program into the coprocessor. The utility is described in Chapter 4, “Loading and unloading coprocessor software,” 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. The utility 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), which is an optional feature. SCUP is not included in the typical install. It can only be installed by using the custom install selection.

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 4765 PCIe Cryptographic Coprocessor must be installed on an IBM-approved x86 server.

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 4765 PCIe Cryptographic Coprocessor Smart Card User’s Guide. This document is available on the Library page of the PCIe Cryptographic Coprocessor.

Software for Linux

1. The IBM 4765 PCIe Cryptographic Coprocessor offering
For instructions on how to download the software for the IBM 4765 PCIe Cryptographic Coprocessor offering, click on the PCIe Cryptographic Coprocessor link at http://www.ibm.com/security/cryptocards then click on the Software download link.

2. A supported operating system distribution
The product requires a supported operating system distribution. For the latest information on supported distributions, click on the PCIe Cryptographic Coprocessor link at http://www.ibm.com/security/cryptocards Click on the Product support link, then click on the Software updates link at the top of the page.

3. 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
A Java Runtime Environment (JRE) version 1.6.0 or later 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 installation.

To download the IBM Java Runtime Environment, go to: http://www.ibm.com/developerworks/java/jdk/linux/download.html
The JRE version 1.6.0 is located in the section titled "Java SE Version 6". A link to installation instructions is located on the download page.

---

**Installing coprocessor host software**

The following sections:
• Describe how to install and uninstall the Support Program.
• Describe how to configure the Support Program.

Installing the coprocessor involves the following steps:
1. Install the software as described in this chapter.
2. Install the hardware as described in the IBM 4765 PCIe Cryptographic Coprocessor Installation Manual.
3. Install software into the coprocessor as described in Chapter 4, “Loading and unloading coprocessor software,” on page 13.
4. Setting CCA Support Program and Linux file permissions

Note that all components of the installed CCA package might not indicate the same version number. The version number for a software 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.
System requirements for the Support Program

This section describes system requirements and 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 /opt/IBM/4765/keys. The keys in these key storage files are encrypted by a 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 an AES or APKA master key is not supported.

Although the uninstall and installation 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

If applicable, uninstall the Support Program for the predecessor product “IBM 4764” prior to installing the IBM 4765 PCIe Cryptographic Coprocessor CCA Support Program. Having both the IBM 4764 and IBM 4765 installed at the same time is not supported.

Uninstalling a preexisting Support Program installation

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

- The installation is no longer needed.
- A newer version needs to be installed.
- An older version needs to be installed.

Perform the following steps to uninstall the Support Program in Linux. 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.
2. Log on as root (do not use the su command to switch user to root; su to root is not sufficient).
3. Enter the command /opt/IBM/4765/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 _IBM4765_installation directory. The existing installation can then be modified or uninstalled. The newer version will not install until the existing version is uninstalled.
Installing an older version of the Support Program

When performing an install and 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.

Modifying an installed version of the Support Program

An existing installation can be modified using the command `/opt/IBM/4765/modify_installation`.

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.

For Linux

The 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” for information on the location of these files.

To install the Support Program in Linux:

1. Download the software package from the product website. Ensure that the permissions on the downloaded executable allow execution by root.
2. Log on as root (do not use the su command to switch user to root; su to root is not sufficient).
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, refer to the product FAQ at [http://www.ibm.com/security/cryptocards/pciecc/4765FAQ.shtml](http://www.ibm.com/security/cryptocards/pciecc/4765FAQ.shtml) Before running the executable, note the following:
   • Do not attempt to install using the -i console option.
   • Do not attempt to install from an ssh or telnet session without a response file, as described in the FAQ.
   • 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 environment variables, where xxx is DES, PKA, or AES.

Table 3 shows the default directory list file for each Key Record List verb:

<table>
<thead>
<tr>
<th>Verb</th>
<th>Default directory list file</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES_Key_Record_List</td>
<td><code>/opt/IBM/4765/keys/aeslist</code></td>
</tr>
</tbody>
</table>
Depending on your installation, the directory 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. 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 `export` command to set the environment variable `CSU_DEF_ADAPTER` to CRP0n, where `n = 1, 2, 3, ..., or 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 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 in 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'.

### 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 `/opt/IBM/4765/keys`.

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 (drw+xwrs--) 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 software

After installing the Support Program on the host computer, 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.

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

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

Note:
1. The file locations referenced in this chapter are the default directory paths.
2. Appendix C, “Device driver error codes,” on page 51 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 the CLU, such as coprocessor memory segments, CLU syntax, CLU commands, CLU files, and CLU return codes, see Appendix G, “Coprocessor Load Utility reference,” on page 79.

Loading coprocessor software

This section provides the procedures used in loading software into the coprocessor. Refer to the readme.txt file that accompanies the software distribution 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 82.

You will be instructed to follow this sequence of steps:
1. At a command prompt, run CLU.
2. Determine the software currently resident within the coprocessor by obtaining a CLU status response.
3. Change the contents of software Segments 1, 2, and 3, as appropriate.
4. This step performs an update that corrects a problem with a non-maskable interrupt (NMI) being detected following a shutdown of the operating system when an IBM 4765 PCIe Cryptographic Coprocessor is installed in an IBM System x M4+ (M4, M5, etc.) class server. CCA Release 4.4 or later is required.
   a. The Coprocessor is shipped with a 2-pin Berg jumper (connector) installed on the pins located at J1002. Refer to “Figure 1. Jumpers on the 4765 PCIe Cryptographic Coprocessor (front side)” in IBM 4765 PCIe Cryptographic Coprocessor Installation Manual. Verify that this jumper is installed (flash write enabled).
   b. All requests to the Coprocessor must be halted and remain halted while this update is in process.
   As a user with root authority, invoke the following code load utility command located in the path /opt/IBM/4765/clu:
   | csulclu RigolinoUpdate.txt mr 0 Rinonnnn.clu

© Copyright IBM Corp. 2013, 2015
# where:
# csulclu
# is the code load utility command
# RigolinoUpdate.txt
# is the name of the log file
# mr is the CLU command refresh communication microcode
# 0 is the number of the coprocessor to use
# Rinonnnn.clu
# is the data file

# Wait for the update to complete, then run this system command:
# cat /proc/driver/ycrypto/0 > RigolinoApplied.txt

# Confirm the microcode refresh by viewing the file produced by the cat command. Expect to see
# the following in the RigolinoApplied.txt file:
# Rigolino: Dev ID: 64 Sub ID: 90
# c. If more than one Coprocessor requiring this update is installed on the server, repeat (b) for each
# Coprocessor.
# d. To ensure that the update completes, perform a physical shutdown of the server. Halt the server
# with this command:
# shutdown -h now

# After the shutdown completes, reboot the server. Requests to the Coprocessor can be restarted.

5. Validate the final contents of the software segments.

Running CLU

Start CLU by entering this command at the command prompt:

/opt/IBM/4765/clu/csulclu

You can provide parameters interactively to the CLU utility, or you can include these on the command line input. See “Coprocessor Load Utility syntax” on page 80. Each time that you use CLU you will need to specify a log file name. This is the first parameter and can be included on the command line. In general, when working with a specific coprocessor, it is strongly recommended you use the coprocessor serial number as the log file name. You can obtain the manufacturer’s serial number from the label on the bracket (not the black label) at the end of the coprocessor. By always naming the log file with the serial number, you can keep a complete history of status and code changes for the contents of each coprocessor.

CLU will append information to two log files. If a log file does not exist, it will be created. One log file contains the same information that is normally displayed on your console. The other log file, to which CLU will assign MRL as the file name extension, contains a “machine-readable log.” The MRL file is intended for use with an analysis utility.

Subsequent instructions in this section assume that you use CLU interactively. Change to the directory that contains the coprocessor code files. Start CLU, then respond to the prompts as requested.

CLU obtains the number of installed coprocessors from the device driver. If more than one coprocessor is installed, CLU interactively prompts for the “number” of the coprocessor with which you intend to interact. These numbers (“coprocessor_#”) 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 Figure 7 on page 82).
Note: Except for the ST, PL of Segment 3, 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 software segment contents

The coprocessor has three “segments”: Segment 1, Segment 2, and Segment 3 (see Table 4). Each segment:

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

<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. Figure 1 shows a typical ST response.

---

CSULCLU V4.2.0 4.4.16.status.log begun Wed Nov 13 13:27:59 2013
*********** Command ST started. ---- Wed Nov 13 13:27:59 2013

*** VPD data; PartNum = 41U9986
*** VPD data; EC Num = N441780
*** VPD data; Ser Num = 16D2C3B2
*** VPD data; Description = IBM 4765-001 PCI-e Cryptographic Coprocessor
*** VPD data; Mfg. Loc. = 91
** ROM Status; POST0 Version 1, Release 43
*** ROM Status; MiniBoot0 Version 1, Release 20
*** ROM Status; INIT: INITIALIZED
*** ROM Status; SEG2: RUNNABLE , OWNER2: 2
*** ROM Status; SEG3: RUNNABLE , OWNER3: 2
*** Page 1 Certified: YES
*** Segment 1 Image: 4.3.5 E P1v060C M011D P2v0708 F5540 201208281457403A000022000000000000
*** Segment 1 Revision: 40305
*** Segment 1 Hash: 722DF07C6C6B49395FFC5B6F77C58B8A358F36B5B733F499164604989B5107
*** Segment 2 Image: 4.4.16 y4_13-lnx-2013-08-09-18 201310311121404A000000000416041600
*** Segment 2 Revision: 40416
*** Segment 2 Hash: 01BAB729C7ACDD0CF6FBE292FF8CFDD0B07F00CB257DD6588B7CA09C5E63E
*** Segment 3 Image: 4.4.16 CCA 201310311121404A000000000416041600
*** Segment 3 Revision: 40416
*** Segment 3 Hash: 2BBC0FEB2C812ABE56C9E41D F3052D68588F44936B916B911C28DD16A8032868
*** Query Adapter Status successful ***
Obtain Status ended successfully!

Figure 1. Typical CLU status response

Definitions of the fields on the ST response are:

<table>
<thead>
<tr>
<th>Item</th>
<th>Discussion</th>
</tr>
</thead>
</table>

Chapter 4. Loading and unloading coprocessor software 15
VPD  Vital product data that uniquely records each hardware element.

PartNum  The part number (P/N) of the coprocessor. See Table 5 on page 17.

EC Num  The engineering change number of the coprocessor.

Ser Num  The manufacturer's serial number of the coprocessor. This is not 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.

ROM Status  The coprocessor must always be in an INITIALIZED state to be functional. If the status is ZEROIZED, the coprocessor has detected a possible tamper event and is in an unrecoverable, non-functional state. (Unintended “tamper” events can be created by improper handling of the coprocessor. Remove the batteries only when following the recommended battery changing procedure, maintain the coprocessor in the safe temperature range, and so forth. See the IBM 4765 PCIe Cryptographic Coprocessor Installation Manual.)

ROM Status SEG2 / SEG3  Several status conditions for Segment 2 and Segment 3 exist, including:

  UNOWNED  Currently not in use, no content

  RUNNABLE  Contains code and is in a generally usable state

Owner identifiers are also shown. The standard CCA Support Program is assigned identifier 2 for both Segment 2 and Segment 3. Any other owner identifier indicates that the software is not the standard IBM CCA product code. In all cases, ensure that the proper software is loaded in your coprocessor. Unauthorized or unknown software poses a security risk to your installation.

Segment 1 Image  The name and description of the software content of Segment 1. For a factory-fresh coprocessor, the name will include "Factory". A factory-fresh image and its associated validation key will need to be changed by reloading Segment 1.

Segment 2 and Segment 3 Images  If Segments 2 and 3 have owned status (that is, their ROM status is not UNOWNED), observe the name for each segment image and their revision levels. IBM incorporates "CCA" in the Segment 3 image name to indicate that the image is provided as part of the CCA Support Program.

Segment Revision values  Be sure to observe the revision level.

Segment Hash values  These are the hash values calculated against each segment of the loaded code. Use these values to verify the integrity of the loaded code. See the Validating the IBM 4765 page at http://www.ibm.com/security/cryptocards/pciecc/validation.shtml for valid hash values.

Changing software segment contents  Generally the software 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.
Start the CLU utility and enter the parameters interactively (see “Running CLU” on page 14).

- Enter the log file name (########.log, where ######## is the serial number of the coprocessor).
- Enter the command, PL.
- If there are multiple coprocessors, enter the coprocessor number.
- Enter the CLU file name as indicated in the readme.txt file.

Repeat as required so that the proper software is loaded for Segments 1, 2, and 3.

**Validating the coprocessor segment contents**

After you have loaded or replaced the code in Segments 1, 2, and 3, validate the coprocessor segment contents. Use the CLU VA command to confirm the segment contents and validate the digital signature on the response created by the coprocessor. Depending on the IBM 4765 coprocessor (PartNum) in use, issue this command and substitute the class-key certificate file name from Table 5 for the data_file parameter. Note that the name of the data file is “v.clu” appended to the coprocessor part number, all in lower case.

csuclu #######.log VA [coprocessor_#] data_file

The part number can be obtained using the CLU ST command.

*Table 5. Class-key file for use with the CLU VA command*

<table>
<thead>
<tr>
<th>PartNum</th>
<th>Class-key certificate file</th>
</tr>
</thead>
<tbody>
<tr>
<td>45D7930</td>
<td>45d7930v.clu</td>
</tr>
<tr>
<td>41U8608</td>
<td>41u8608v.clu</td>
</tr>
</tbody>
</table>

The “[coprocessor_#]” parameter is the optional designator for a particular coprocessor. It defaults to 0.

If the required class-key certificate file is not found in the clu subdirectory, try finding it at this URL: [http://www.ibm.com/security/cryptocards/pciecc/validation.shtml](http://www.ibm.com/security/cryptocards/pciecc/validation.shtml). This Web page provides detailed information on how to verify the integrity of a coprocessor, along with a copy of each class-key certificate file.

**Unloading coprocessor software and zeroizing the node**

When you use CLU to process a file that surrenders ownership of Segment 2, both Segment 2 and the subordinate Segment 3 are cleared:

- The code is removed.
- The validating 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.

Refer to the readme.txt file that accompanies the software distribution that you are using. This file might provide additional information that augments or modifies this general procedure.

Perform these actions:

1. Change to the directory that contains the CLU files, typically /opt/IBM/4765/clu.
2. Start the CLU utility, csuclu.
3. Respond to the prompts and use the serial number of the coprocessor in the logfile name.
4. Use the CLU PL command to surrender ownership of Segment 2 and 3 using a data_file parameter that has the CLU file name shown in Table 18 on page 82.

---

1. You can refer to the IBM product website [http://www.ibm.com/security/cryptocards](http://www.ibm.com/security/cryptocards) FAQ section for the procedure to validate coprocessor integrity. That topic carries the current list of class-key certificate files.
5. Confirm removal of Segments 2 and 3 software and security-relevant data items by using the CLU ST command. The ROM Status of SEG2 and SEG3 should indicate "UNOWNED".

Note:
1. You can also zeroize CCA without removing the software by using the CCA reinitialize process. See "Initializing a CCA node" on page 24.
2. IBM does not normally make available a file to restore the factory Segment 1 validating key to put the coprocessor into a condition similar to a factory-fresh product. Segment 1 can only be changed a limited number of times before the available Device Key certificate space is exhausted and the coprocessor is potentially rendered unusable. If you require a capability to restore the Segment 1 factory validating key, and are willing to expose your coprocessor to a possible lock-up condition, you can obtain the required file from IBM by submitting a query using the Contact page on the product website, [http://www.ibm.com/security/cryptocards](http://www.ibm.com/security/cryptocards)
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.

- 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) or Java Software Development Kit (SDK). 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, you can determine whether it meets your needs or whether you require a custom application to achieve more comprehensive administrative control and key management.
2. Files created through use of the CNM utility might be dependent on the release of the Java runtime environment. If you change the release of the Java runtime environment that you use, files that you have 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 34.
5. These utilities use the IBM Common Cryptographic Architecture (CCA) API to request services from the coprocessor. The IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors contains a comprehensive list of the verbs (also known as “callable services” or “procedure calls”) provided by the CCA API. Refer to this book and the individual
services described herein to understand which commands might require authorization 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 you create 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, you 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 39.

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 4765 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.

You can load data objects directly into the coprocessor or save them 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 4765 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 /opt/IBM/4765/cnm
2. Enter csulcnm on the command line unless the optional Smart Card Utility Program is to be included, in which case the command is csulcnm /sc.

The CNM main panel is displayed.

Note: You can use only data objects that were saved while using CNM with the IBM 4765 PCIe Cryptographic Coprocessor. Non-PCIe data objects cannot be used.

Using the CNM and CNI utilities, sample scenarios

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. Clone a DES or PKA master key from one coprocessor to another coprocessor. This is a procedure of interest to very high-security installations that employ multiple coprocessors.

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, see “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 for the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors. 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 “Starting the CCA Node Management utility” on page 20.

   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 clock 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 clocks are synchronized.

4. Use the CNM utility to permit 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 Permit 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 27.

   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.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. These options are also accessed from the pull-down menus mentioned above.

   For more detail, see “Loading a master key automatically” on page 33.
7. Key storage is a CCA term that describes a place where the Support Program can store DES, PKA, ECC, and AES cryptographic keys under names that you (or your applications) define. If you intend 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, 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. If you intend to use DES and PKA keys, you must initialize the DES and PKA key storage files, but not the AES key storage file. If you intend to use all three, you must initialize all three. See "Creating or initializing key storage" on page 37.

@ Note: ECC keys are 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 authorization 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 authorizes commands in the various roles to ensure that only those commands actually required are enabled. 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 39.

a. Create a CNI list to perform these operations:
   1) Synchronize the clock-calendars within the coprocessor and host computer
   2) Load the access-control data
   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) Load a first master-key key-part
   3) As required, load first-part key-encrypting-key information
   4) 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) As required, load second-part key-encrypting-key information
   4) Log off

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. See “Establishing other nodes using CNI” on page 39.

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 has to 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 clock-calendars within the coprocessor with the host computer
- Poll status information about the coprocessor and the CCA application

Choosing a specific coprocessor

If your system has multiple coprocessors loaded with the CCA code, there is a default coprocessor. Generally you will need to select the specific coprocessor upon which to operate. If you do not make a selection, you will operate with the default coprocessor. Once you make a coprocessor selection, that selection remains in effect for the current utility session or until you make a different selection within the utility session.

To select an adapter (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 a number from the list. If you do not use the Select Adapter pull-down to choose an adapter, the default adapter is used.

Note:
1. When using the CLU utility, coprocessors are referenced as 0, 1, 2. 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. 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

You can restore the CCA node to its initial state, provided that the role you are operating under (the default role or a logged-on role) permits use of the Reinitialize Device command (offset X'0111'). This command clears (zeroizes) all:

- Master-key registers
- Retained PKA and registered PKA 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. You will be asked to 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. 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 signed value provided by IBM to enable the CCA application in the coprocessor to provide a level of cryptographic service consistent with applicable import and export regulations. Under the 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.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-calendars with 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 32.

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 for the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors. 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 selected, trusted individuals, or for certain programs that operate at specific times. Generally, you should 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). You will 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 the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors 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 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 software from the coprocessor and reinstall the CCA software.

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 software 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 can log on to that role. 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; see Figure 2 on page 28 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.

   **Comment**
   An optional character string 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 these times are Coordinated Universal Time. If you are not already familiar with the access-control system, refer to Chapter 2 of the *IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors*.

   **Restricted operations and permitted operations**
   A list defining the commands that the role is allowed to use.

   Each CCA API verb might 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.

   For more information about CCA verb calls and commands, refer to the *IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors*.

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 you open 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 you delete a role, the CNM utility does not automatically delete or reassign the user profiles associated with that role. You should 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; see Figure 3 on page 30.
3. Define the user profile:

   **User ID**
   The name given to a user profile of the cryptographic coprocessor.

   **Comment**
   An optional character string 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 will set this by default to 3 months from the current date. You can change the expiration date. 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) keys.

**Primary KEKs**

DES keys shared by cryptographic nodes and are 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. Knowledge of the key parts can be shared in part by two people to effect a split-knowledge, dual-control security policy.

**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 while they are stored external to the coprocessor. CCA defines three master-key registers:

**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 25.

*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 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 retrieve a 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, you cannot recover the keys enciphered under it.

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 37.

Loading a new master key from key parts

To set a new master-key into the coprocessor, load the first, any middle, and the last key parts 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. See Figure 4 on page 34.
2. Select the radio button for the key part you are editing (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 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 key parts on it stored in a safe or vault.

   Note: When you create a key from parts, you must have both a first part and a last part; 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.

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 new master-key, reencipher the keys that are currently in storage. See “Reenciphering stored keys” on page 37.

**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 for the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors. Appendix D, “DES and PKA master-key cloning,” on page 53 provides a step-by-step procedure that you can 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
- The master-key target 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_i, where 1 ≤ i ≤ n. Processing of share_i 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 IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors 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 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 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, IBM4765.CLONING.CSS.KEY and IBM4765.CLONING.CSR.KEY. The labels that you use must not conflict with other key labels used in your applications.

When generating the CSR key at the 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.

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 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
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 you access by key label using labels that you or your applications define. DES keys, PKA keys, ECC keys, and AES keys are held in separate storage systems. (ECC keys and AES keys are held in AES key-storage.) Also, the coprocessor has a very limited internal storage for PKA 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 keys, 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 and AES keys are held in AES key-storage.
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.

Note to Linux users: 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 had any keys, these would be erased.

Reenciphering stored keys

To reencipher the keys in storage under a current master-key that is new:
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**... The keys are re-enciphered under the key in the current master-key register.

**Deleting a stored key**

To delete a stored 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 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 keys with 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.

**Creating a key label**

To create a key label:

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 (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 for the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors.

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 on page 38 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 40), it searches the identical directory path for each file. For example:

   • If you save a user profile to the established node directory /IBM4765/profiles, the CNI utility will search the target node directory /IBM4765/profiles.

   • If you save a user profile to the diskette directory /profiles, the CNI utility will search the target node directory /profiles.

3. From the File pull-down menu, select CNI Editor. The CCA Node Initialization Editor panel is displayed. See Figure 5.

![Figure 5. CCA Node Initialization Editor panel](image)
The list in the top portion of the panel displays the functions that can be added to the CNI list. The bottom portion 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 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 storage for DES keys, PKA keys, and AES keys

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 you saved the files to 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 /opt/IBM/4765/cnm
    b. Enter csulcni listfile_name on the command line

    If the CNI list includes a logon, enter csulcni or csuncni 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 4765.

A sample routine written in the C programming language can be found in Appendix F, “Sample programs,” on page 71. Source code for the sample routine is shipped with the software. You can use the sample included 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. 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 for the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors.

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:

```
/opt/IBM/4765/include/csulincl.h
```

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

```
#include "csulincl.h"
```

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:

2. The term verb implies an action that an application program can initiate. Some systems and publications use the term callable service.
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:

```
/opt/IBM/4765/samples
```

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

```
/usr/lib/csulcca.so
```

### Viewing coprocessor hardware errors

Most errors that occur in the coprocessor hardware are logged in the `/var/log/messages` file. View this file to look for or investigate coprocessor errors.

When contacting the Crypto team about a problem with a coprocessor, it is always a good idea to include the `/var/log/messages` file and also the `/var/log/debug` file. These files can frequently provide useful information for problem determination.

### 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 4765. 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, PKA, ECC, and AES keys.

#### Multithreading and multiprocessing

The CCA application running inside the 4765 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, PKA, ECC, and AES keys

The CCA software for the coprocessor keeps copies of recently used DES, PKA, ECC, 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 and avoid the overhead associated with decrypting and validating the key token. In addition, for retained PKA 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, PKA and ECC private keys, ECC and encrypted AES keys, and the caching is very effective in improving throughput. PKA 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).

A sample Java routine for calling the CCA JNI can be found in "Syntax, sample JNI routine" on page 75.

Source code for the sample routine is shipped with the software as part of the installation package.

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. To illustrate how to use the CCA JNI to call CCA verbs, a sample module named mac.java is provided as part of the installation package. See "Syntax, sample JNI routine" on page 75. The default distribution location of the sample code is:

/opt/IBM/4765/samples

The CCA JNI provides a hikmNativeInteger class that encapsulates the primitive type 'int' for use in the native interface calls. A file named hikmNativeInteger.html provides information about this class and is located in the same directory as the mac.java file.

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.*;
```

See "Syntax, sample JNI routine" on page 75 for an example.

The mac.java sample calls the same CCA verbs as the sample C language program named mac.c. See Chapter 6, “Building applications to use with the CCA API,” on page 41.

The Java entry points of CCA verbs are very similar to the C entry points except that a letter "J" is appended to the entry point name. For example, CSNBKGN is the C entry point for Key_Generate, while CSNBKGNJ is the Java entry point.

Compiling and running a CCA JNI program

To compile the sample JNI program, invoke the following command from the directory that contains the sample source code:

```bash
javac -classpath /opt/IBM/4765/cnm/CNM.jar mac.java
```

Note:

1. The classpath option points to the CNM.jar file because the hikmNativeInteger class is in that file.
2. The path shown for the CNM.jar file is the default distribution location of that file.
3. Notice to JNI users of releases before Release 4.2.5. Beginning with CCA Release 4.2.5, the files HIKM.zip and libHIKM.so.#.#.# are deprecated by CNM.jar and libCCA_JNI.so. These files will continue to be provided, on an as-is basis, for customers who want to continue using them for JNI access to the CCA API only. No additional support for these legacy files is provided.
After the sample Java program is compiled, it can be run using these commands from the directory that contains the compiled output:

```bash
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib
java -classpath /opt/IBM/4765/cnm/CNM.jar:. mac
```

**Note:**

1. The path shown for the CNM.jar file is the default distribution location of that file.
2. The libCCA_JNI.so library is the CCA Java Native Interface. After installing the CCA code, the path will include the library along with all other required CCA libraries.
3. For Linux users who downloaded a different Java Runtime Environment, add the new path to the `java` command shown above. For example, if you downloaded Java SE Version 6, change the `java` command above to `/opt/ibm/java-i386-60/jre/bin/java`. For information on downloading Java SE Version 6, see "Hardware and software requirements" on page 7.
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 selected supervisory roles.

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

Table 6. Initial commands for DEFAULT role

<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. Machine-readable-log contents

The CLU utility creates two log files, one intended for reading and the other for possible input to a program. This latter log file, the machine-readable log or MRL file, contains the binary outputs from the coprocessor in response to various commands input to the coprocessor.

Appendix C. 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, its 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 via email from the Support page on the IBM product website (http://www.ibm.com/security/cryptocards).

Table 7. Device driver error codes in the class X'8xxxxxxx'

<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 optional electrical connection to 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>8040FFEB</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>8040FFFF3</td>
<td>Voltage tamper</td>
<td>The coprocessor is zeroized and is no longer functional.</td>
</tr>
<tr>
<td>8040FFFF9</td>
<td>Mesh tamper</td>
<td>The coprocessor is zeroized and is no longer functional.</td>
</tr>
<tr>
<td>8040FFFFB</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 4765 PCIe Cryptographic Coprocessor Installation Manual should be followed to replace the batteries.</td>
</tr>
<tr>
<td>804xxxxxx (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 via the 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 via the website.</td>
</tr>
<tr>
<td>4-bytereturn 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 signature</td>
<td>The 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 via the website.</td>
</tr>
</tbody>
</table>
Appendix D. 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 34 and “Understanding and managing master keys” in Chapter 2 of the IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessor.

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>

© Copyright IBM Corp. 2013, 2015
Before undertaking the master-key cloning procedure, it is recommended that you complete the forms found in Table 9 and Figure 6 on page 55.

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 time synchronization and insure that the FCV authorization (fcv_td4kECC521.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>
</tbody>
</table>
| 1.5   | If not already established, enter the Environment ID (EID):
|     | • Crypto Node, Set Environment ID. |
|     | • Enter the EID, Load. |
| 1.6   | Generate the SA key:
|     | • Crypto Node, Share Administration, Create Keys, Share Administration Key. |
|     | • Accept the default SA public-key and private-key labels, and enter the location and name of the SA database ("sa.db"). |
|     | • Create |
|     | • Record the SA-key hash value for use later in the procedure. |
### 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_td4kECC521.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  | 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 time synchronization and insure that the FCV authorization (fcv_td4kECC521.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 | 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>
<tbody>
<tr>
<td>3b.2</td>
<td>SA</td>
<td>Certify the CSR key:  &lt;ul&gt;&lt;li&gt;Crypto Node, Share Administration, Certify Keys, CSR Key.&lt;/li&gt;&lt;li&gt;Enter the name and path for the SA and CSR databases, Next.&lt;/li&gt;&lt;li&gt;Confirm the SA key label, CSR key label, and the SA Environment ID.&lt;/li&gt;&lt;li&gt;Enter the CSR serial number.&lt;/li&gt;&lt;li&gt;Certify.&lt;/li&gt;&lt;/ul&gt;</td>
</tr>
</tbody>
</table>

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.  <ul><li>Crypto Node, Share Administration, Get Share.</li><li>Select the share. Note that if you are obtaining an additional set(s) of shares, the “Distributed” messages might not be meaningful.</li><li>Enter the name and path for the SA and CSR databases, Next.</li><li>Confirm the CSS key label, CSS coprocessor serial number, and the CSR coprocessor serial number.</li><li>Get Share.</li></ul> 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.  <ul><li>Crypto Node, Share Administration, Load Share.</li><li>Select the share.</li><li>Enter the name and path for the CSR and SA databases, Next.</li><li>Confirm the CSS key label, the CSS coprocessor serial number, and the CSR coprocessor serial number.</li><li>Load Share.</li></ul> Observe the response. Loading sufficient shares completes the new master-key. Repeat as required. |

| 3d.2  | Target | Confirm the new master-key:  <ul><li>Master Key, Verify, New.</li><li>Compare, select the file, OK, Cancel</li></ul> |

| 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:  <ul><li>Master Key, Set.</li><li>OK.</li></ul> |

---

**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 Master Key</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 Master Key</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 New Master Key Register</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 Old Master Key Register</td>
<td>Master_Key_Process</td>
<td>Generally not used.</td>
</tr>
<tr>
<td>X’008E’</td>
<td>Generate Key</td>
<td>Key_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_SignatureVerify</td>
<td>All roles used for cloning.</td>
</tr>
<tr>
<td>X’0102’</td>
<td>PKA96 Key Token Change</td>
<td>PKA_Key_Token_Change</td>
<td>Consider who will update working keys encrypted by the master key.</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>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'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_Initiation</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'0211' - X'021F'</td>
<td>Clone-info (Share) Obtain</td>
<td>Master_Key_Distribution</td>
<td>Consider a profile and role for each share to enforce split responsibility.</td>
</tr>
<tr>
<td>X'0221' - X'022F'</td>
<td>Clone-info (Share) Install</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 E. 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 for the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors 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>
<tr>
<td>Threat discussion</td>
<td>Threat mitigation</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------</td>
</tr>
<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, radiation, and physical penetration sensors are designed to detect unusual environmental situations. 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. 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 you can verify to confirm that the coprocessor is genuine and is not tampered with. Note that almost all of the software that runs on the main processor within the coprocessor is available 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. The design and implementation has been independently evaluated and certified by the USA NIST under the FIPS PUB 140-2 Level 4 standard (FIPS certification number 1505). <strong>Note:</strong> You must validate the condition of the coprocessor and the code content.</td>
</tr>
<tr>
<td>An adversary might perform physical probing of the coprocessor to reveal design information and operational contents. Such probing might include electrical functions but is referred to here as physical since it requires direct contact with the coprocessor internals. Physical probing might entail reading data from the coprocessor through techniques commonly employed in IC failure analysis and IC reverse-engineering efforts. The goal of the adversary is to identify such design details as hardware security mechanisms, access-control mechanisms, authentication systems, data-protection systems, memory partitioning, or cryptographic programs. Determination of software design, including initialization data, passwords, PINs, or cryptographic keys might also be a goal.</td>
<td></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 4765 base card. In the process of altering the sensitive electronics, the coprocessor factory certification would be destroyed rendering the device useless. <strong>Note:</strong> You will need to 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 software.</td>
</tr>
<tr>
<td>An adversary might physically modify the coprocessor in order to reveal design or security-related information. This modification might be accomplished through techniques commonly employed in hardware failure analysis and reverse-engineering efforts. The goal is to identify such design details as hardware-security mechanisms, access-control mechanisms, authentication systems, data-protection systems, memory partitioning, or cryptographic programs. Determination of software design, including initialization data, passwords, PINs, or cryptographic keys, might also be a goal.</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>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 and radiation. 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><strong>Substituted process</strong></td>
<td></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><strong>Note:</strong></td>
<td></td>
</tr>
<tr>
<td>1. Auditors need to complete the processes described for them to ensure that the signing key is indeed retained within the appropriate coprocessor.</td>
<td></td>
</tr>
<tr>
<td>2. Access to the host system should be supervised so that host-system security measures and proper operation can be relied upon.</td>
<td></td>
</tr>
<tr>
<td><strong>Threats associated with logical attack on the coprocessor</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Insertion of faults</strong></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><strong>Note:</strong> 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><strong>Forced reset</strong></td>
<td></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>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>
</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>Invalid input</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 software validates the characteristics of each request to address misuse scenarios.</td>
</tr>
<tr>
<td>Data loading malfunction</td>
<td>As outlined in auditor procedures, the access-control setup should be verified along with confirming the installed coprocessor software.</td>
</tr>
<tr>
<td>Unauthorized program loading</td>
<td>The coprocessor will only accept digitally signed software after the signature has been validated. An independent evaluation of IBM’s software build and signing procedures and the coprocessor design affirms the trust which can be placed in the identity of loaded software. <strong>Note:</strong> An auditor should follow procedures to affirm that specified software is in use.</td>
</tr>
<tr>
<td>Threats associated with control of access</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 software implementation and testing has reviewed the integrity of the access-control implementation.</td>
</tr>
<tr>
<td>Invalid access</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 software implementation and testing has reviewed the integrity of the access-control implementation.</td>
</tr>
<tr>
<td>Fraud on first use</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> 1. The standard installation bring-up process replaces all of the runtime coprocessor software. 2. You should verify that Segments 2 and 3 are “unowned” prior to loading coprocessor software for production. This ensures that no residual data remains to influence subsequent operations.</td>
</tr>
<tr>
<td>Threat discussion</td>
<td>Threat mitigation</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Impersonation</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>
<tr>
<td>Threats associated with unanticipated interactions</td>
<td>The coprocessor design requires you to configure the access-control setup. The CCA software has been examined to ensure that functions are disallowed when required commands are not enabled.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> 1. Your access-control configuration should follow the principles discussed in Appendix H of the IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors 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 and/or to address additional application(s) functionality. In these cases, ensure that you review the guidelines and observations in Appendix H of the IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors for applicability to your circumstances.</td>
</tr>
<tr>
<td>Threats regarding cryptographic functions</td>
<td>The coprocessor implements well-established and standardized cryptographic functions.</td>
</tr>
<tr>
<td></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 fur 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 for the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors.</td>
</tr>
<tr>
<td>Threats regarding digital signatures</td>
<td></td>
</tr>
<tr>
<td>Threat discussion</td>
<td>Threat mitigation</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Forging signed data</td>
<td>The coprocessor implements well-established and standardized cryptographic functions.</td>
</tr>
<tr>
<td>An adversary might modify data digitally signed by the coprocessor such that this</td>
<td><strong>Note:</strong> 1. There are precautions in the use of CCA which should be observed as documented in Appendix H of the IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors.</td>
</tr>
<tr>
<td>modification is not detectable by the signatory or a third party. This attack</td>
<td>2. Users should maintain an awareness of vulnerabilities discussed in (open) forums regarding the strength of cryptographic algorithms and processes that they employ.</td>
</tr>
<tr>
<td>might use weaknesses in the secure hash function, weaknesses in the signature</td>
<td></td>
</tr>
<tr>
<td>encoding, or weaknesses in the cryptographic algorithm used to generate a forged</td>
<td></td>
</tr>
<tr>
<td>signature.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Forging data before it is signed</td>
<td>Requests from user host-application process memory carry an integrity check value which the coprocessor confirms prior to incorporating the hash in a digital signature.</td>
</tr>
<tr>
<td>An adversary might modify data to be digitally signed by the coprocessor before the</td>
<td><strong>Note:</strong> Users must review host-system and host-application program security to ensure that authenticated hash values received into the coprocessor have not been compromised and are representative of the data to be protected.</td>
</tr>
<tr>
<td>signature is generated within the coprocessor. This attack might use weaknesses in</td>
<td></td>
</tr>
<tr>
<td>the implementation that allow an adversary to modify data transmitted for signature</td>
<td></td>
</tr>
<tr>
<td>to the coprocessor before the coprocessor actually calculates the signature.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Misuse of signature function</td>
<td>An independent review of the coprocessor software is expected to affirm that:</td>
</tr>
<tr>
<td>An adversary might misuse the coprocessor signature creation function to sign data</td>
<td>• The digital-signature generation service requires an appropriate permission in a role</td>
</tr>
<tr>
<td>that the coprocessor is not supposed to sign.</td>
<td>• The processing of requests and the integrity of the design prevent data alteration</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> 1. The integrity of the coprocessor and its code must be affirmed by an auditor who reviews a coprocessor status query.</td>
</tr>
<tr>
<td>The adversary might try to submit data to the coprocessor and get it signed</td>
<td>2. An auditor must confirm that appropriate access-control roles and profiles have been established which exclude unauthorized users from use of the digital-signature functionality.</td>
</tr>
<tr>
<td>without passing the authorization checks of the coprocessor it has to perform</td>
<td></td>
</tr>
<tr>
<td>before generating a digital signature.</td>
<td></td>
</tr>
<tr>
<td>As an alternative, an adversary might try to modify data within the coprocessor</td>
<td></td>
</tr>
<tr>
<td>through the use of coprocessor functions or by trying to influence the coprocessor</td>
<td></td>
</tr>
<tr>
<td>such that the data in the coprocessor gets modified.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Forging signature-verification function</td>
<td>The signature-verification function of primary interest here occurs in the coprocessor's code-loading process (in Miniboot). With this product:</td>
</tr>
<tr>
<td>An adversary might modify the function for signature verification such that a false</td>
<td>• Miniboot code, like the control program and (CCA) application program code, is only accepted into the coprocessor when the coprocessor validates the signature on the signed code.</td>
</tr>
<tr>
<td>signature is accepted as valid. This attack might try to modify the signature-</td>
<td>• The initial Miniboot code loaded in the factory is also subject to digital-signature verification.</td>
</tr>
<tr>
<td>verification function or signed data to be verified such that the coprocessor</td>
<td>• Standardized cryptographic processes are used (SHA-1, RSA, ISO 9796) for the signature.</td>
</tr>
<tr>
<td>returns a success message when this false signature is presented for verification.</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>Deleting a private RSA signature key</td>
<td>By design, a retained private key is deleted only:</td>
</tr>
<tr>
<td>An adversary might use a function that deletes a private RSA signature key without</td>
<td>1. Under CCA control with the Retained_Key_Delete verb</td>
</tr>
<tr>
<td>being authorized to do so and without physically tampering with the coprocessor.</td>
<td>2. By loading the coprocessor CCA software*</td>
</tr>
<tr>
<td></td>
<td>3. By removing the coprocessor CCA software</td>
</tr>
<tr>
<td></td>
<td>4. By causing a tamper event</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> To address these exposures:</td>
</tr>
<tr>
<td></td>
<td>1. Selectively enable the Delete Retained Key command, X'0203'</td>
</tr>
<tr>
<td></td>
<td>2. Use host-system access controls to manage usage of the CLU Utility</td>
</tr>
<tr>
<td></td>
<td>3. Manage physical access to the coprocessor</td>
</tr>
<tr>
<td></td>
<td>* Reloading the coprocessor software with a file such as CEXxxxxx.clu does not zeroize the contents of persistent storage. The file CNWxxxxx.clu will zeroize persistent storage. See Chapter 4, “Loading and unloading coprocessor software,” on page 13.</td>
</tr>
</tbody>
</table>

### Threats that monitor information

<table>
<thead>
<tr>
<th>Information leakage</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>An adversary might exploit information that is leaked from the coprocessor during</td>
<td></td>
</tr>
<tr>
<td>normal usage. Leakage might occur through emanations, variations in power</td>
<td></td>
</tr>
<tr>
<td>consumption, I/O characteristics, clock frequency, or by changes in processing-</td>
<td></td>
</tr>
<tr>
<td>time requirements. This leakage might be interpreted as a covert channel</td>
<td></td>
</tr>
<tr>
<td>transmission but is more closely related to measurement of operating parameters,</td>
<td></td>
</tr>
<tr>
<td>which might be derived either from direct (contact) measurements or measurement of</td>
<td></td>
</tr>
<tr>
<td>emanations and can then be related to the specific operation being performed.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Linkage of multiple observations</th>
<th><strong>Note:</strong> 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 for the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors.</th>
</tr>
</thead>
<tbody>
<tr>
<td>An adversary might observe multiple uses of resources or services and, by linking these</td>
<td>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.</td>
</tr>
<tr>
<td>observations, deduce information that would reveal critical security information. The</td>
<td>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.</td>
</tr>
<tr>
<td>combination of observations over a period of many uses of the coprocessor, or the</td>
<td></td>
</tr>
<tr>
<td>integration of knowledge gained from observing different operations, might reveal</td>
<td></td>
</tr>
<tr>
<td>information that allows an adversary to either learn information directly or to</td>
<td></td>
</tr>
<tr>
<td>formulate an attack that could further reveal information that the coprocessor is required to keep secret.</td>
<td></td>
</tr>
</tbody>
</table>

### Miscellaneous threats

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Threat discussion</td>
<td>Threat mitigation</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------</td>
</tr>
</tbody>
</table>
| **Linked attacks** | **Note:**  
| 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. | 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. |
| **Repetitive attack** | 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. |
| **Cloning** | Auditors must confirm that the digital-signing key, appropriate code, and access-control regime is resident in the authorized coprocessor. |
| **Threats addressed by the operating environment** | **Note:**  
| **Coprocessor modification and reuse** | 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. |
| 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. | 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. |
| **Abuse by privileged users** |  |
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</td>
<td></td>
</tr>
<tr>
<td>by an adversary or by faults in the operational</td>
<td></td>
</tr>
<tr>
<td>environment after it has been approved by the legitimate</td>
<td></td>
</tr>
<tr>
<td>user, but before the data is submitted to the coprocessor</td>
<td></td>
</tr>
<tr>
<td>to be signed. Data that has been approved by the</td>
<td></td>
</tr>
<tr>
<td>legitimate user to be signed might be modified by an</td>
<td></td>
</tr>
<tr>
<td>adversary, by false or malicious programs, or by</td>
<td></td>
</tr>
<tr>
<td>environmental errors (for example, transmission errors)</td>
<td></td>
</tr>
<tr>
<td>after the data has been approved by the legitimate user</td>
<td></td>
</tr>
<tr>
<td>and before the data is transferred to the coprocessor</td>
<td></td>
</tr>
<tr>
<td>to be signed.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Data verification</td>
<td></td>
</tr>
<tr>
<td>Signed data to be verified by the coprocessor might be</td>
<td></td>
</tr>
<tr>
<td>modified by an adversary or by faults in the operational</td>
<td></td>
</tr>
<tr>
<td>environment before it is submitted to the coprocessor</td>
<td></td>
</tr>
<tr>
<td>for signature verification such that the response of the</td>
<td></td>
</tr>
<tr>
<td>coprocessor does not reflect the validity of the signature.</td>
<td></td>
</tr>
<tr>
<td>Signed data submitted by a user might be modified</td>
<td></td>
</tr>
<tr>
<td>within the coprocessor environment before it is passed</td>
<td></td>
</tr>
<tr>
<td>to the coprocessor for verification. This might result in</td>
<td></td>
</tr>
<tr>
<td>a response from the coprocessor that does not reflect the</td>
<td></td>
</tr>
<tr>
<td>actual validity of the digital signature that should be</td>
<td></td>
</tr>
<tr>
<td>verified.</td>
<td></td>
</tr>
<tr>
<td>There is also the possibility that the response of the</td>
<td></td>
</tr>
<tr>
<td>coprocessor is modified in the coprocessor environment</td>
<td></td>
</tr>
<tr>
<td>before it is passed to the user that requested the</td>
<td></td>
</tr>
<tr>
<td>signature verification.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: Host-system security measures must address blocking the modification of request inputs and outputs.
Appendix F. Sample programs

There are two sample routines here. One is a program written in C that illustrates the use of some CCA verbs. The other is a sample JNI routine written in C.

Sample C routine

To illustrate the practical application of CCA verb calls, this section describes the sample C programming language routine included with the Support Program. For reference, a copy of the sample routine is shown in "Syntax, sample routine."

The sample routine generates a message authentication code (MAC) on a text string, and then verifies the MAC. To effect this, the routine:

1. Calls the Key_Generate (CSNBKGN) verb to create a MAC/MACVER key pair.
2. Calls the MAC_Generate (CSNBMGN) verb to generate a MAC on a text string with the MAC key.
3. Calls the MAC_Verify (CSNBMVR) verb to verify the text string MAC with the MACVER key.

As you review the sample routine shown in "Syntax, sample routine," refer to the IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and IBM 4764 PCI-X Cryptographic Coprocessors for descriptions of the called verbs and their parameters. These verbs are listed in Table 15.

Table 15. Verbs called by the sample routine

<table>
<thead>
<tr>
<th>Verb</th>
<th>Entry-point name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key_Generate</td>
<td>CSNBKGN</td>
</tr>
<tr>
<td>MAC_Generate</td>
<td>CSNBMGN</td>
</tr>
<tr>
<td>MAC_Verify</td>
<td>CSNBMVR</td>
</tr>
</tbody>
</table>

Syntax, sample routine

/*******************************                        */
/*/ Module Name: mac.c */
/*/ */
/*/ DESCRIPTIVE NAME: Cryptographic Coprocessor Support Program */
/*/ C language source code example */
/*/ */
/*/------------------------------*/
/*/ */
/*/ Licensed Materials - Property of IBM */
/*/ */
/*/ (C) Copyright IBM Corp. 1997-2001. */
/*/ */
/*/ US Government Users Restricted Rights - Use duplication or */
/*/ disclosure restricted by GSA ADP Schedule Contract with IBM Corp. */
/*/ */
/*/-------------------------------*/
/*/ */
/*/ NOTICE TO USERS OF THE SOURCE CODE EXAMPLES */
/*/ */
/*/ */
/*/ The source code examples provided by IBM are only intended to */
/*/ assist in the development of a working software program. The */
/*/ source code examples do not function as written: additional */
/*/ code is required. In addition, the source code examples may */
/*/ not compile and/or bind successfully as written. */
/*/ */
/*/ International Business Machines Corporation provides the source */

© Copyright IBM Corp. 2013, 2015
/ * code examples, both individually and as one or more groups, */
/* "as is" without warranty of any kind, either expressed or */
/* implied, including, but not limited to the implied warranties of */
/* merchantability and fitness for a particular purpose. The entire */
/* risk as to the quality and performance of the source code */
/* examples, both individually and as one or more groups, is with */
/* you. Should any part of the source code examples prove defective, */
/* you (and not IBM or an authorized dealer) assume the entire cost */
/* of all necessary servicing, repair or correction. */
/* */
/* IBM does not warrant that the contents of the source code */
/* examples, whether individually or as one or more groups, will */
/* meet your requirements or that the source code examples are */
/* error-free. */
/* */
/* IBM may make improvements and/or changes in the source code */
/* examples at any time. */
/* */
/* Changes may be made periodically to the information in the */
/* source code examples; these changes may be reported, for the */
/* sample code included herein, in new editions of the examples. */
/* */
/* References in the source code examples to IBM products, programs, */
/* or services do not imply that IBM intends to make these */
/* available in all countries in which IBM operates. Any reference */
/* to the IBM licensed program in the source code examples is not */
/* intended to state or imply that IBM's licensed program must be */
/* used. Any functionally equivalent program may be used. */
/* */
/* ------------------------------------------------------------------------*/
/* */
/* This example program: */
/* */
/* 1) Calls the Key_Generate verb (CSNBKGN) to create a MAC (message */
/*    authentication code) key token and a MACVER key token. */
/* */
/* 2) Calls the MAC_Generate verb (CSNBMGN) using the MAC key token */
/*    from step 1 to generate a MAC on the supplied text string */
/*    (INPUT_TEXT). */
/* */
/* 3) Calls the MAC_Verify verb (CSNBMVVR) to verify the MAC for the */
/*    same text string, using the MACVER key token created in */
/*    step 1. */
/* */
/* ***************************************************************************/
#include <stdio.h>
#include <string.h>

#ifndef _AIX
#include <csufincl.h>
#else
#include "csunincl.h"
#endif
else
#include "csulincl.h"    /* else linux */
#endif

/* Defines */
#define KEY_FORM    "OPOP"
#define KEY_LENGTH  "SINGLE "
#define KEY_TYPE_1  "MAC   
#define KEY_TYPE_2  "MACVER  
#define INPUT_TEXT  "abcdefgijklmn0987654321"
#define MAC_PROCESSING_RULE  "X9.9-1 "
#define SEGMENT_FLAG  "ONLY  
#define MAC_LENGTH   "HEX-9  
#define MAC_BUFFER_LENGTH 10

72 4765 PCIe Cryptographic Coprocessor CCA Support Program, Release 4.4
void main()
{
    static long return_code;
    static long reason_code;
    static unsigned char key_form[4];
    static unsigned char key_length[8];
    static unsigned char mac_key_type[8];
    static unsigned char macver_key_type[8];
    static unsigned char kek_key_id_1[64];
    static unsigned char kek_key_id_2[64];
    static unsigned char mac_key_id[64];
    static unsigned char macver_key_id[64];
    static unsigned char rule_array[3][8];
    static unsigned char rule_array_count;
    static unsigned char chaining_vector[18];
    static unsigned char mac_key_type[8];
    static unsigned char macver_key_type[8];
    static unsigned char kek_key_id_1[64];
    static unsigned char kek_key_id_2[64];
    static unsigned char mac_key_id[64];
    static unsigned char macver_key_id[64];
    static unsigned char text_length;
    static unsigned char text[26];
    static unsigned char rule_array_count;
    static unsigned char rule_array[3][8];
    static unsigned char chaining_vector[18];

    /* Set up initial values for Key_Generate call */
    return_code = 0;
    reason_code = 0;
    memcpy (key_form, KEY_FORM, 4); /* OPOP key pair */
    memcpy (key_length, KEY_LENGTH, 8); /* Single-length keys */
    memcpy (mac_key_type, KEY_TYPE_1, 8); /* 1st token, MAC key type */
    memcpy (macver_key_type, KEY_TYPE_2, 8); /* 2nd token, MACVER key type */
    memset (kek_key_id_1, 0x00, sizeof(kek_key_id_1)); /* 1st KEK not used */
    memset (kek_key_id_2, 0x00, sizeof(kek_key_id_2)); /* 2nd KEK not used */
    memset (mac_key_id, 0x00, sizeof(mac_key_id)); /* Init 1st key token */
    memset (macver_key_id, 0x00, sizeof(macver_key_id)); /* Init 2nd key token */

    /* Generate a MAC/MACVER operational key pair */
    CSNBKGN (&return_code, &reason_code,
            NULL, /* exit_data_length */
            NULL, /* exit_data */
            key_form, key_length, mac_key_type, macver_key_type, kek_key_id_1, kek_key_id_2, mac_key_id, macver_key_id);

    /* Check the return/reason codes. Terminate if there is an error. */
    if (return_code != 0 || reason_code != 0) {
        printf("Key_Generate failed: "); /* Print failing verb */
        printf("return_code = %ld,", return_code); /* Print return code */
        printf("reason_code = %ld.", reason_code); /* Print reason code */
        return;
    }
    else
        printf("Key_Generate successful.
");

    /* Set up initial values for MAC_Generate call */
    return_code = 0;
    reason_code = 0;
    text_length = sizeof (INPUT_TEXT) - 1; /* Length of MAC text */
    memcpy (text, INPUT_TEXT, text_length); /* Define MAC input text */
    rule_array_count = 3; /* 3 rule array elements */
    memset (rule_array, '*', sizeof(rule_array)); /* Clear rule array */
    memcpy (rule_array[0], MAC_PROCESSING_RULE, 8); /* 1st rule array element */
    memcpy (rule_array[1], SEGMENT_FLAG, 8); /* 2nd rule array element */
    memcpy (rule_array[2], MAC_LENGTH, 8); /* 3rd rule array element */
    memset (chaining_vector, 0x00, 18); /* Clear chaining vector */
memset (mac_value, 0x00, sizeof(mac_value)); /* Clear MAC value */

/* Generate a MAC based on input text */
CSNBMGN (&return_code, &reason_code, NULL, /* exit_data_length */ NULL, /* exit_data */ mac_key_id, /* Output from Key_Generate */ &text_length, text, &rule_array_count, &rule_array[0][0], chaining_vector, mac_value);

/* Check the return/reason codes. Terminate if there is an error. */
if (return_code != 0 || reason_code != 0) {
    printf ("MAC Generate Failed: "); /* Print failing verb */
    printf ("return_code = \%ld, ", return_code); /* Print return code */
    printf ("reason_code = \%ld.\n", reason_code); /* Print reason code */
    return;
}
else {
    printf ("MAC Generate successful.\n");
    printf ("MAC_value = %s\n", mac_value); /* Print MAC value (HEX-9) */
}

/* Set up initial values for MAC_Verify call */
return_code = 0;
reason_code = 0;
rule_array_count = 1; /* 1 rule array element */
memset (rule_array, ' ', sizeof(rule_array)); /* Clear rule array */
memcpy (rule_array[0], MAC_LENGTH, 8); /* Rule array element */
    /* (use default Ciphering */
    /* Method and Segmenting */
    /* Control) */
memset (chaining_vector, 0x00, 18); /* Clear the chaining vector */

/* Verify MAC value */
CSNBMVR (&return_code, &reason_code, NULL, /* exit_data_length */ NULL, /* exit_data */ macver_key_id, /* Output from Key_Generate */ &text_length, /* Same as for MAC_Generate */ text, /* Same as for MAC_Generate */ &rule_array_count, &rule_array[0][0], chaining_vector, mac_value); /* Output from MAC_Generate */

/* Check the return/reason codes. Terminate if there is an error. */
if (return_code != 0 || reason_code != 0) {
    printf ("MAC_Verify failed: "); /* Print failing verb */
    printf ("return_code = \%ld, ", return_code); /* Print return code */
    printf ("reason_code = \%ld.\n", reason_code); /* Print reason code */
    return;
}
else /* No error occurred */ {
}

Sample CCA JNI routine in C

The following sample program provides an example of how to write a CCA JNI routine in C.
Syntax, sample JNI routine

/**
 * Module Name: mac.java
 */
/**
 * DESCRIPTIVE NAME: Cryptographic Coprocessor Support Program
 */
/**
 * JNI example code
 */
/**
 * Licensed Materials - Property of IBM
 */
/**
 * Copyright IBM Corp. 2008, 2011 All Rights Reserved
 */
/**
 * US Government Users Restricted Rights - Use duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 */
/**
 * NOTICE TO USERS OF THE SOURCE CODE EXAMPLES
 */
/**
 * The source code examples provided by IBM are only intended to
 * assist in the development of a working software program. The
 * source code examples do not function as written: additional
 * code is required. In addition, the source code examples may
 * not compile and/or bind successfully as written.
 */
/**
 * International Business Machines Corporation provides the source
 * code examples, both individually and as one or more groups,
 * "as is" without warranty of any kind, either expressed or
 * implied, including, but not limited to the implied warranties of
 * merchantability and fitness for a particular purpose. The entire
 * risk as to the quality and performance of the source code
 * examples, both individually and as one or more groups, is with
 * you. Should any part of the source code examples prove defective,
 * you (and not IBM or an authorized dealer) assume the entire cost
 * of all necessary servicing, repair or correction.
 */
/**
 * IBM does not warrant that the contents of the source code
 * examples, whether individually or as one or more groups, will
 * meet your requirements or that the source code examples are
 * error-free.
 */
/**
 * IBM may make improvements and/or changes in the source code
 * examples at any time.
 */
/**
 * Changes may be made periodically to the information in the
 * source code examples; these changes may be reported, for the
 * sample code included herein, in new editions of the examples.
 */
/**
 * References in the source code examples to IBM products, programs,
 * or services do not imply that IBM intends to make these
 * available in all countries in which IBM operates. Any reference
 * to the IBM licensed program in the source code examples is not
 * intended to state or imply that IBM's licensed program must be
 * used. Any functionally equivalent program may be used.
 */
/**
 * This example program:
 */
/**
 * 1) Calls the Key_Generate verb (CSNBKGN) to create a MAC (message
 *  authentication code) key token and a MACVER key token.
 */
/**
 * 2) Calls the MAC_Generate verb (CSNBMGN) using the MAC key token
 *  from step 1 to generate a MAC on the supplied text string
 *  (INPUT_TEXT).
 */
/**
 * 3) Calls the MAC_Verify verb (CSNBMRV) to verify the MAC for the
 */
import java.io.*;
import com.ibm.crypto.cca.jni.*;

public class mac {
    static final String KEY_FORM = "OPOP";
    static final String KEY_LENGTH = "SINGLE ";
    static final String KEY_TYPE_1 = "MAC ";
    static final String KEY_TYPE_2 = "MACVER ";
    static final String INPUT_TEXT = "abcdefhgijklmnopqrstuvwx";
    static final String MAC_PROCESSING_RULE = "X9.9-1 ";
    static final String SEGMENT_FLAG = "ONLY ";
    static final String MAC_LENGTH = "HEX-9 ";

    public static void main(String args[]) {
        byte [] ByteExitData = new byte [4];
        byte [] Byte_key_form = new byte [4];
        byte [] Byte_key_length = new byte [8];
        byte [] Byte_mac_key_type = new byte [8];
        byte [] Byte_macver_key_type = new byte [8];
        byte [] Byte_mac_value = new byte [10];
        byte [] Byte_chaining_vector = new byte [18];
        byte [] Byte_rule_array = new byte [24];
        byte [] Byte_text = new byte [26];
        byte [] Byte_kek_key_id_1 = new byte [64];
        byte [] Byte_kek_key_id_2 = new byte [64];
        byte [] Byte_mac_key_id = new byte [64];
        byte [] Byte_macver_key_id = new byte [64];

        try {
            //setup to pause on non-zero return/reason code
            //and require enter key to continue
            BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
            hikmNativeInteger IntReturncode = new hikmNativeInteger(0);
            hikmNativeInteger IntReasoncode = new hikmNativeInteger(0);
            hikmNativeInteger IntExitDataLength = new hikmNativeInteger(0);

            /* Print beginning banner */
            System.out.println("\nCryptographic Coprocessor Support Program JAVA example program.\n");
            /* Set up initial values for Key_Generate call */
            Byte_key_form = new String(KEY_FORM).getBytes(); /* OPOP key pair */
            Byte_key_length = new String(KEY_LENGTH).getBytes(); /* Single-length keys */
            Byte_mac_key_type = new String(KEY_TYPE_1).getBytes(); /* 1st token, MAC key type */
            Byte_macver_key_type = new String(KEY_TYPE_2).getBytes(); /* 2nd token, MACVER key type */

            /* Generate a MAC/MACVER operational key pair */
            new HIKM().CSNBKGNJ(IntReturncode,
                IntReasoncode,
                IntExitDataLength,
                ByteExitData,
                Byte_key_form,
                Byte_key_length,
                Byte_mac_key_type,
                Byte_macver_key_type,
                Byte_kek_key_id_1,
                Byte_kek_key_id_2,
                Byte_mac_key_id,
                Byte_macver_key_id);

            if ( 0 != IntReturncode.getValue() || 0 != IntReasoncode.getValue() ) {
                System.out.println("\nKey Generate Failed"); /* Print failing verb. */
                System.out.println("Return_code = \" + IntReturncode.getValue()); /* Print return code. */
                System.out.println("Reason_code = \" + IntReasoncode.getValue()); /* Print reason code. */
                System.out.println("Press ENTER to continue...\""); /* Print Pause message */
            }
        } catch (IOException e) {
            System.out.println("\nInput/Output Error\n");
            System.out.println("Exiting Program\n");
        }
    }
}
stdin.readLine();
}
else{
    System.out.println("Key_Generate successful.");
}

/* Set up initial values for MAC_Generate call */
IntReturncode = new hikmNativeInteger(0);
IntReasoncode = new hikmNativeInteger(0);
IntExitDataLength = new hikmNativeInteger(0);

hikmNativeInteger Int_rule_array_count = new hikmNativeInteger(3);
hikmNativeInteger Int_text_length = new hikmNativeInteger(24);

Byte_text = new String(INPUT_TEXT).getBytes();           /* Define MAC input text */
byte[] temp_array = new String(MAC_PROCESSING_RULE).getBytes(); /* 1st rule array element */
System.arraycopy(temp_array, 0, Byte_rule_array, 0, temp_array.length); /* 1st rule array element */

temp_array = new String(SEGMENT_FLAG).getBytes();       /* 2nd rule array element */
System.arraycopy(temp_array, 0, Byte_rule_array, 8, temp_array.length); /* 2nd rule array element */

temp_array = new String(MAC_LENGTH).getBytes();       /* 3rd rule array element */
System.arraycopy(temp_array, 0, Byte_rule_array, 16, temp_array.length); /* 3rd rule array element */

/* Generate a MAC based on input text */
new HIKM().CSNBMGNJ(IntReturncode,
    IntReasoncode,
    IntExitDataLength,
    ByteExitData,
    Byte_mac_key_id,
    Int_text_length,
    Byte_text,
    Int_rule_array_count,
    Byte_rule_array,
    Byte_chaining_vector,
    Byte_mac_value);
   
if ( 0 != IntReturncode.getValue() || 0 != IntReasoncode.getValue() )
{
    System.out.println("\nMAC_Generate Failed");    /* Print failing verb. */
    System.out.println("Return_code = " + IntReturncode.getValue()); /* Print return code. */
    System.out.println("Reason_code = " + IntReasoncode.getValue()); /* Print reason code. */
    System.out.println("Press ENTER to continue..."); /* Print Pause message */
    stdin.readLine();
} else{
    System.out.println("MAC_Generate successful.");
    System.out.println("MAC_value = [" + new String(Byte_mac_value) + "]");
}

/* Set up initial values for MAC_Verify call */
IntReturncode = new hikmNativeInteger(0);
IntReasoncode = new hikmNativeInteger(0);
IntExitDataLength = new hikmNativeInteger(0);

Byte_rule_array = new String(MAC_LENGTH).getBytes(); /* Rule array element */
Int_rule_array_count = new hikmNativeInteger(1);
new HIKM().CSNBMMVRJ(IntReturncode,
    IntReasoncode,
    IntExitDataLength,
    ByteExitData,
    Byte_macver_key_id,
    Int_text_length,
    Byte_text,
    Int_rule_array_count,
    Byte_rule_array,
    Byte_chaining_vector,
    Byte_mac_value);
   
if ( 0 != IntReturncode.getValue() || 0 != IntReasoncode.getValue() )

System.out.println("MAC_Verify Failed"); /* Print failing verb. */
System.out.println("Return_code = " + IntReturncode.getValue()); /* Print return code. */
System.out.println("Reason_code = " + IntReasoncode.getValue()); /* Print reason code. */
System.out.println("Press ENTER to continue..."); /* Print Pause message */
stdin.readLine();

else
{
    System.out.println("MAC_Verify successful.");
}

catch (Exception anException)
{
    System.out.println(anException);
}

/* Print ending banner */
System.out.println("Cryptographic Coprocessor Support Program JAVA example program finished.");
//end main
}//end mac class
Appendix G. Coprocessor Load Utility reference

This appendix describes:
- The coprocessor memory segments into which you load the software
- The way in which the coprocessor validates software loads
- The syntax used to invoke the CLU utility
- The loader commands supported by the CLU
- A summary of shipped CLU files
- Return codes issued by the CLU utility return codes

Coprocessor memory segments

Coprocessor memory segments are organized as shown in Table 16.

<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>Software administration and cryptographic routines</td>
</tr>
<tr>
<td></td>
<td>Software in this segment:</td>
</tr>
<tr>
<td></td>
<td>• Administers the replacement of software already loaded to Segment 1.</td>
</tr>
<tr>
<td></td>
<td>• Administers the loading of data and software 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 software</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 software loads

When the coprocessor is shipped from the factory, it has within it the public key needed to validate replacement software 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 software, 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 software. Should another vendor intend to supply software 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. Use CLU to:

- New capability is to use the CLU ST and CLU SS commands 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.
- Obtain the release level and the status of software currently installed in the coprocessor memory segments.
- Confirm the validity of digitally signed messages returned by the coprocessor.
- Load and reload portions of the coprocessor software.
- The coprocessor is automatically reset.

Run CLU by entering this command at the command prompt:

```
/opt/IBM/4765/clu/csulclu
```

If you do not supply the necessary parameters, the utility will prompt you as information is required. Optional parameters are enclosed in brackets. The syntax for the parameters following the utility name is:

```
[ log_file cmd ] [coprocessor _#] [data_file] [-Q ] ]
```

where:

**log_file** Identifies the logfile name. The utility appends entries to this ASCII text file as it performs the operations requested. A second “machine readable” log file, with a file name of logfile_name.MRL, is also created. This log file can be processed by a program and contains the binary-encoded responses from the coprocessor. For information about the contents of this log file, see Appendix B, “Machine-readable-log contents,” on page 49.

---

3. 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.
cmd Specifies a two-letter abbreviation representing the loader command to be run. See “Coprocessor Load Utility commands.”

coprocessor_# Provides the coprocessor number as established by the device driver. This parameter defaults to 0. Coprocessors are designated to the device driver as numbers 0 - 7. You can use the serial number information that you obtain with the SS command and the serial number printed on the end-bracket of the coprocessor to correlate a particular coprocessor to the coprocessor_. The utility logically supports up to eight coprocessors per machine.

data_file Identifies the data file (drive, directory, and file name) used for the operation requested.
- For software loads and reloads, it is the file name of the software image that you are loading into the coprocessor.
- When obtaining coprocessor status with the VA command, this parameter is the class-key certificate file name used to validate the coprocessor response. See Table 5 on page 17. The product website (http://www.ibm.com/security/cryptocards) Frequently Asked Questions (FAQ) area contains a description of the procedure for validating the coprocessor and its code. This description also contains a list of all the current class-key certificate file names. You can download any required certificate file from the website.

-Q Suppresses (quiets) the CLU program output to the standard output device. The status information is still appended to the log files.

Example: To validate the coprocessor status with part number 45D7930 and save the results to the log file, enter:

csulclu #######.log va 45d7930v.clu

It is suggested that you make ####### the serial number of the coprocessor. It is not mandatory to use the serial number, but it can be of value to retain a history of all software changes made to each specific coprocessor. In the example shown, the coprocessor_# parameter defaults to 0.

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: 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 software.</td>
</tr>
<tr>
<td>RS: Reset the coprocessor</td>
<td>Resets the coprocessor. Generally you will not use this command. The command causes the coprocessor to perform a power-on reset. 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 with the coprocessor prior to issuing this command to enable the complete cryptographic subsystem to get to a reset state.</td>
</tr>
<tr>
<td>SS: Obtain system/box status</td>
<td>Obtains the part number, serial number, and a portion of the Segment 3 software image name for each of the installed coprocessors, provided that these are not being used by some application such as CCA. See Figure 7 on page 82</td>
</tr>
</tbody>
</table>
Table 17. CLU loader commands (continued)

<table>
<thead>
<tr>
<th>Loader command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST: Obtain coprocessor status</td>
<td>Obtains the status of loaded software 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 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 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>
</tbody>
</table>

In general, the utility can be invoked by a script file. When creating a script file to invoke the utility on an unattended system, add “quiet” syntax -q (or -Q, /q, or /Q) to request that nothing be output to the display. By default, the utility returns prompts and messages to the display.

Table 18. Summary of shipped CLU files

<table>
<thead>
<tr>
<th>Shipped CLU file name (short name)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reload_seg1_xipz_factory_keyswap_#.#.#.clu (cr1_#.#.#.clu)</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>reload_seg1_xipz_.#.#.#.clu (ce1_.#.#.#.clu)</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>establish_ownership_then_emergency_reload_segment2_seg3_xip_.#.#.#.clu (cnw_.#.#.#.clu)</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>reload_segment2_segment3_xip_.#.#.#.clu (ce2_.#.#.#.clu)</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>surrender_ownership_segment2_xipz_.#.#.#.clu (crs_.#.#.#.clu)</td>
<td>Use as data_file parameter of CLU PL command to zeroize any preexisting master keys, retained keys, roles, and profiles.</td>
</tr>
</tbody>
</table>
Table 18. Summary of shipped CLU files (continued)

<table>
<thead>
<tr>
<th>Shipped CLU file name (short name)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>45d7930v.clu</td>
<td>Use as data_file parameter of CLU VA command to validate the segment contents of a coprocessor with part number 45D7930.</td>
</tr>
<tr>
<td>41u8608v.clu</td>
<td>Use as data_file parameter of CLU VA command to validate the segment contents of a coprocessor with part number 41U8608.</td>
</tr>
</tbody>
</table>

Note: Substitute #.#.# above with the version.release.mod of the software.

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:

<table>
<thead>
<tr>
<th>CLU return code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK.</td>
</tr>
<tr>
<td>1</td>
<td>Command line parameters not valid.</td>
</tr>
<tr>
<td>2</td>
<td>Cannot access the coprocessor. Ensure that the coprocessor and its driver have been properly installed.</td>
</tr>
<tr>
<td>3</td>
<td>Check the utility log file for an abnormal condition report.</td>
</tr>
<tr>
<td>4</td>
<td>No coprocessor installed. Ensure that the coprocessor and its driver have been properly installed.</td>
</tr>
<tr>
<td>5</td>
<td>Invalid coprocessor number specified.</td>
</tr>
<tr>
<td>6</td>
<td>A data file is required with this command.</td>
</tr>
<tr>
<td>7</td>
<td>The data file specified with this command is incorrect or invalid.</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 [http://www.ibm.com/security/cryptocards](http://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>ANSI</td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASCII</td>
<td>American National 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>CDMF</td>
<td>Commercial Data Masking Facility</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>DMA</td>
<td>Direct Memory Access</td>
</tr>
<tr>
<td>ECB</td>
<td>Electronic Codebook</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>IBM</td>
<td>International Business Machines</td>
</tr>
<tr>
<td>IC</td>
<td>Integrated Circuit</td>
</tr>
<tr>
<td>ICSF</td>
<td>Integrated Cryptographic Services Facility</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>IPL</td>
<td>Initial Program Load</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>MDC</td>
<td>Modification Detection Code</td>
</tr>
<tr>
<td>OCSP</td>
<td>Online Certificate Status Protocol</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PCI</td>
<td>Peripheral Component Interconnect</td>
</tr>
<tr>
<td>PCIe</td>
<td>Peripheral Component Interconnect Express</td>
</tr>
<tr>
<td>PCI-X</td>
<td>Peripheral Component Interconnect Extended</td>
</tr>
<tr>
<td>PDD</td>
<td>Physical Device Driver</td>
</tr>
<tr>
<td>PDF</td>
<td>Portable Document Format</td>
</tr>
<tr>
<td>PIN</td>
<td>Personal Identification Number</td>
</tr>
<tr>
<td>PKA</td>
<td>Public Key Algorithm</td>
</tr>
<tr>
<td>POST</td>
<td>Power-On Self Test</td>
</tr>
<tr>
<td>RA</td>
<td>Registration Authority</td>
</tr>
<tr>
<td>RAM</td>
<td>Random access memory</td>
</tr>
<tr>
<td>RHEL</td>
<td>Red Hat Enterprise Linux</td>
</tr>
<tr>
<td>RSA</td>
<td>Rivest-Shamir-Adleman encryption algorithm.</td>
</tr>
<tr>
<td>SA</td>
<td>Share Administration</td>
</tr>
<tr>
<td>SCUP</td>
<td>Smart Card Utility Program</td>
</tr>
<tr>
<td>SHA</td>
<td>Secure Hashing Algorithm</td>
</tr>
<tr>
<td>SKA</td>
<td>Secret Key Authentication</td>
</tr>
<tr>
<td>SLES</td>
<td>SUSE Linux Enterprise Server</td>
</tr>
</tbody>
</table>
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, ANSI X3.172-1990, copyright 1990 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.


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 supplies 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.

CDMF algorithm
An algorithm for data confidentiality applications; it is based on the DES algorithm and possesses 40-bit key strength.

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 4765)
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.

Direct Memory Access (DMA)
The transfer of data between memory and input/output units without processor intervention.

driver A program that contains the code needed to attach and use a device.

electronic codebook (ECB)
A mode of operation used with block-cipher cryptographic algorithms in which plaintext or ciphertext is placed in the input to the algorithm and the result is contained in the output of the algorithm.

deciphered 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 4765 PCIe Cryptographic Coprocessor to yield a level of cryptographic service consistent with applicable export-and-import regulations.

host computer
In regard to the CCA Cryptographic Coprocessor Support Program, the server into which the IBM 4765 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 4765’s 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.

(T) 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.

secret key authentication (SKA) certificate
The SKA certificate contains enciphered values that could allow IBM to reinitialize a coprocessor after its tamper-sensors have been triggered. Without a copy of the certificate, there is no way to recover the coprocessor.

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.
IBM 4764 PCI-X Cryptographic Coprocessor.

IBM 4765 PCIe Cryptographic Coprocessor.
key storage (continued)
delete keys 38
initialization 37
key label, create 38
management 37
reencipher 37

L
link to CCA, application programs 42
Linux
file permissions 11
load command 79
load coprocessor software
commands 81
establish owner command 79
load command 79
owner identifier 79
reload command 80
surrender owner command 79
logon-attempt-failure count, reset 31

M
machine readable log 80
machine-readable log 49
makefile 42
management
cryptographic key 31
key storage 37
master key 32
master key
auto-set 33
create 33
description 31
management 32
new, set 33
registers 32
verification 33
master-key administration 32
master-key cloning 53
memory segments, coprocessor 79

N
node
setup, production-environment 22
setup, test 22
notices
notices statement 85

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 79

P
performance, enhancing 42
permit, access-control commands 28
placing orders 3
polling information, coprocessor 25
production-environment, node setup 22

R
reencipher stored keys 37
registers, master key 32
reload command 80
reset logon-attempt-failure count 31
restrict, access-control commands 31
return codes, CLU 83
role
create 27
define 27
delete 29
modify 29

S
sample routine, C programming language
makefile 42
source code 42
security-relevant data item (SRDI) 80
set new master-key 33
setup
production-environment node 22
test node 22
software validation, CLU 79
SRDI, security-relevant data item 80
status, coprocessor batteries 26
storage, KEKs 38
stored keys, reencipher 37
Support Program
components 7
coprocessor load 13
download 4
host install 7
host remove 7
overview, installation 1
surrender owner command 79
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 61
throughput, enhancing 42

U
user profile
create 29
define 29
delete 31
edit 30
reset logon-attempt-failure count 31
utilities
CLU 13
CNI 20, 39
CNM 20

V
validation, coprocessor software 79
verb calls, C programming language 41
verification, master key 33

Z
zeroization of the CCA node 24