

SafeNet ProtectToolkit-C

Administration Guide

Document Information

Product Version	5.3
Document Part Number	007-013682-001
Release Date	05 December 2016

Revision History

Revision	Date	Reason
Rev. A	05 December 2016	Initial release

Trademarks, Copyrights, and Third-Party Software

Copyright 2009-2016 Gemalto. All rights reserved. Gemalto and the Gemalto logo are trademarks and service marks of Gemalto and/or its subsidiaries and are registered in certain countries. All other trademarks and service marks, whether registered or not in specific countries, are the property of their respective owners.

Disclaimer

All information herein is either public information or is the property of and owned solely by Gemalto and/or its subsidiaries who shall have and keep the sole right to file patent applications or any other kind of intellectual property protection in connection with such information.

Nothing herein shall be construed as implying or granting to you any rights, by license, grant or otherwise, under any intellectual and/or industrial property rights of or concerning any of Gemalto's information.

This document can be used for informational, non-commercial, internal, and personal use only provided that:

- The copyright notice, the confidentiality and proprietary legend and this full warning notice appear in all copies.
- This document shall not be posted on any publicly accessible network computer or broadcast in any media, and no modification of any part of this document shall be made.

Use for any other purpose is expressly prohibited and may result in severe civil and criminal liabilities.

The information contained in this document is provided "AS IS" without any warranty of any kind. Unless otherwise expressly agreed in writing, Gemalto makes no warranty as to the value or accuracy of information contained herein.

The document could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Furthermore, Gemalto reserves the right to make any change or improvement in the specifications data, information, and the like described herein, at any time.

Gemalto hereby disclaims all warranties and conditions with regard to the information contained herein, including all implied warranties of merchantability, fitness for a particular purpose, title and non-infringement. In no event shall Gemalto be liable, whether in contract, tort or otherwise, for any indirect, special or consequential damages or any damages whatsoever including but not limited to damages resulting from loss of use, data, profits, revenues, or customers, arising out of or in connection with the use or performance of information contained in this document.

Gemalto does not and shall not warrant that this product will be resistant to all possible attacks and shall not incur, and disclaims, any liability in this respect. Even if each product is compliant with current security standards in force on the date of their design, security mechanisms' resistance necessarily evolves according to the state of the art in security and notably under the emergence of new attacks. Under no circumstances, shall Gemalto be held liable for any third

party actions and in particular in case of any successful attack against systems or equipment incorporating Gemalto products. Gemalto disclaims any liability with respect to security for direct, indirect, incidental or consequential damages that result from any use of its products. It is further stressed that independent testing and verification by the person using the product is particularly encouraged, especially in any application in which defective, incorrect or insecure functioning could result in damage to persons or property, denial of service, or loss of privacy.

All intellectual property is protected by copyright. All trademarks and product names used or referred to are the copyright of their respective owners. No part of this document may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, chemical, photocopy, recording or otherwise without the prior written permission of Gemalto.

CONTENTS

PREFACE	About the SafeNet ProtectToolkit-C Administration Guide	8
Customer Release Notes		8
Gemalto Rebranding		9
Audience		9
Document Conventions		9
Notes		9
Cautions		10
Warnings		10
Command Syntax and Typeface Conventions		10
Support Contacts		11
1	Introduction	12
Who Should Read This Manual?		12
Further Documentation		13
SafeNet Manuals		13
SafeNet Application Integration Guides		13
Utility Normal Mode vs. Work Load Distribution and HA Mode		13
2	Installation and Configuration	14
System Requirements		14
Supported Platforms		15
Operating Mode Setup		15
PCI and Network Operating Modes		16
Software-Only Mode		17
Basic Windows Installation		17
SafeNet ProtectToolkit-C Packages		17
Windows Install Preparation		18
Runtime Installation Procedure		18
SDK Installation Procedure		19
Basic Unix/Linux Installation		19
SafeNet ProtectToolkit-C Packages		20
Unix/Linux Install Preparation		20
Installation Procedure		21
Changing the Cryptoki Provider		21
Set Mode		22
Using the Unix Installation Utility		22
Utility Startup		23
Installing a package		23
Setting up your environment		24
Changing the Cryptoki provider		24
Uninstalling a package		25

Unix Installation Utility troubleshooting	25
Unix/Linux Command Reference	26
Manual installation	26
Changing the Cryptoki Provider manually	28
Manual uninstallation	28
Configuration Items	29
Windows	29
Unix/Linux	30
Secure Messaging	30
Configuring Session Protection	32
Specifying the Network Server(s)	33
Using IPv6 addressing	34
Software-Only Mode Configuration	34
Troubleshooting	35
3 Cryptoki Configuration	36
The SafeNet ProtectToolkit-C Model	36
Slots and Tokens	37
User Slots	37
Smart Card Slots	37
The Admin Slot	38
PKCS #11 Objects	38
Administration Objects	38
User Roles	38
PINs and Passwords	39
Initial Configuration	39
Trust Management	42
Establishing Trust Relationships	45
Token Replication	47
Work Load Distribution Model and High Availability	49
SafeNet ProtectToolkit-C Configuration	50
WLD Slots	50
Distribution Scheme	50
Token Replication	51
WLD System Setup	51
Configuration	53
Configuring WLD Slots	55
Operation in WLD Mode	56
Operation in HA Mode	57
HA Mode Logging	58
External Key Storage	59
External Key Storage Model	59
External Key Storage Configuration	63
Configuration for Application Development	63
Configuration for Runtime Operation	65
Creating Externally Stored Objects	66
Real Time Clock	66
4 Security Policies and User Roles	68

PKCS #11 Compliance and Security	69
Typical Security Policies	69
PKCS #11 Compatibility Mode	69
SafeNet Default Mode	70
FIPS Mode	70
Entrust Compliant Modes	71
Netscape Compliant Mode	71
Restricted Mode	71
Security Flags	72
Configuring Security Flags	72
Security Flag Descriptions	73
Security Policy Options	76
User Roles	77
Administration Security Officer (ASO)	78
Administrator	78
Security Officer (SO)	79
Token Owner (User)	79
Unauthenticated Users	79
5 Operational Tasks	80
Changing a User or Security Officer PIN	80
Secure Key Backup and Restoration	81
Adding and Removing Slots	85
Re-initializing a Token	86
Connecting and Removing Smart Card Readers	86
Using Transport Mode to Avoid a Board Removal Tamper	86
Adjusting the HSM Clock	87
Changing Secure Messaging Mode	87
Managing Session Key Rollover	87
Using the System Event Log	87
Updating Firmware	88
Tampering the HSM	89
Installing a Functionality Module	89
6 Command Line Utilities Reference	90
CTCERT	91
CTCHECK	100
CTCONF	104
CTFM	108
CTIDENT	111
CTKMU	114
CTLIMITS	122
CTPERF	125
CTSTAT	129
7 Administration Utility (gCTAdmin) Reference	130
Logging In and Out	130
Main gCTAdmin Interface	131
Slot and Token Management	132

HSM Management	135
8 Key Management Utility (KMU) Reference	139
Compatibility Issues	140
Main KMU Interface	140
Logging Into and Out From Tokens	142
Creating Keys	143
Available Keys	144
Key Attribute Types	144
Creating a Random Secret Key	145
Creating a Random Key Pair	146
Creating Key Components	147
Entering a Key from Components	149
Editing Key Attributes	150
Deleting a Key	151
Display Key Check Value	151
Importing and Exporting Keys	152
Key Backup Feature Tutorial	158
Key Definitions	158
Creation of Encrypted Key Set to Backup (Payload)	159
Back up to File	159
Backup to Smart Card – Single Custodian Mode	160
Backup to Smart Card – Multiple Custodian Mode	161
Error Messages and Warnings	163
APPENDIX A Event Log Error Types	165
APPENDIX B PKCS #11 Attributes	168
APPENDIX C KMU Key Check Value (KCV) Calculation	170
APPENDIX D Key Migration from SafeNet ProtectToolkit-C V4.1	172
APPENDIX E Sample EC Domain Parameter Files	173
C2tnB191v1	174
brainpoolP160r1	175
Hexadecimal to Decimal Conversion Table	176
APPENDIX F Glossary of terms	178

PREFACE

About the SafeNet ProtectToolkit-C Administration Guide

This document provides installation, configuration, security, and administration guidelines for the SafeNet ProtectToolkit-C cryptographic services suite. It contains the following chapters:

- ["Introduction" on page 12](#)
- ["Installation and Configuration" on page 14](#)
- ["Cryptoki Configuration" on page 36](#)
- ["Security Policies and User Roles" on page 68](#)
- ["Operational Tasks" on page 80](#)
- ["Command Line Utilities Reference" on page 90](#)
- ["Key Management Utility \(KMU\) Reference" on page 139](#)
- ["Administration Utility \(gCTAdmin\) Reference" on page 130](#)
- ["Event Log Error Types" on page 165](#)
- ["PKCS #11 Attributes" on page 168](#)
- ["KMU Key Check Value \(KCV\) Calculation" on page 170](#)
- ["Key Migration from SafeNet ProtectToolkit-C V4.1" on page 172](#)
- ["Sample EC Domain Parameter Files" on page 173](#)
- ["Glossary of terms" on page 178](#)

This preface also includes the following information about this document:

- ["Customer Release Notes" below](#)
- ["Gemalto Rebranding" on the next page](#)
- ["Audience" on the next page](#)
- ["Document Conventions" on the next page](#)
- ["Support Contacts" on page 11](#)

For information regarding the document status and revision history, see ["Document Information" on page 2](#)

Customer Release Notes

The customer release notes (CRN) provide important information about this release that is not included in the customer documentation. It is strongly recommended that you read the CRN to fully understand the capabilities, limitations, and known issues for this release. You can view or download the latest version of the CRN for this release at the following location:

http://www.securedby safenet.com/releasenotes/ptk/cn_ptk_5-3.pdf

Gemalto Rebranding

In early 2015, Gemalto completed its acquisition of SafeNet, Inc. As part of the process of rationalizing the product portfolios between the two organizations, the Luna name has been removed from the SafeNet HSM product line, with the SafeNet name being retained. As a result, the product names for SafeNet HSMs have changed as follows:

Old product name	New product name
ProtectServer External 2 (PSE2)	SafeNet ProtectServer Network HSM
ProtectServer Internal Express 2 (PSI-E2)	SafeNet ProtectServer PCIe HSM
ProtectServer HSM Access Provider	SafeNet ProtectServer HSM Access Provider
ProtectToolkit C (PTK-C)	SafeNet ProtectToolkit-C
ProtectToolkit J (PTK-J)	SafeNet ProtectToolkit-J
ProtectToolkit M (PTK-M)	SafeNet ProtectToolkit-M
ProtectToolkit FM SDK	SafeNet ProtectToolkit FM SDK



Note: These branding changes apply to the documentation only. The SafeNet HSM software and utilities continue to use the old names.

Audience

This document is intended for personnel responsible for maintaining your organization's security infrastructure. This includes SafeNet ProtectToolkit users and security officers, key manager administrators, and network administrators.

All products manufactured and distributed by Gemalto are designed to be installed, operated, and maintained by personnel who have the knowledge, training, and qualifications required to safely perform the tasks assigned to them. The information, processes, and procedures contained in this document are intended for use by trained and qualified personnel only.

It is assumed that the users of this document are proficient with security concepts.

Document Conventions

This document uses standard conventions for describing the user interface and for alerting you to important information.

Notes

Notes are used to alert you to important or helpful information. They use the following format:



Note: Take note. Contains important or helpful information.

Cautions

Cautions are used to alert you to important information that may help prevent unexpected results or data loss. They use the following format:



CAUTION: Exercise caution. Contains important information that may help prevent unexpected results or data loss.

Warnings

Warnings are used to alert you to the potential for catastrophic data loss or personal injury. They use the following format:



WARNING! Be extremely careful and obey all safety and security measures. In this situation you might do something that could result in catastrophic data loss or personal injury.

Command Syntax and Typeface Conventions

Format	Convention
bold	The bold attribute is used to indicate the following: <ul style="list-style-type: none"> Command-line commands and options (Type dir /p.) Button names (Click Save As.) Check box and radio button names (Select the Print Duplex check box.) Dialog box titles (On the Protect Document dialog box, click Yes.) Field names (User Name: Enter the name of the user.) Menu names (On the File menu, click Save.) (Click Menu > Go To > Folders.) User input (In the Date box, type April 1.)
<i>italics</i>	In type, the italic attribute is used for emphasis or to indicate a related document. (See the <i>Installation Guide</i> for more information.)
<variable>	In command descriptions, angle brackets represent variables. You must substitute a value for command line arguments that are enclosed in angle brackets.
[optional] [<optional>]	Represent optional keywords or <variables> in a command line description. Optionally enter the keyword or <variable> that is enclosed in square brackets, if it is necessary or desirable to complete the task.
{ a b c } {<a> <c>}	Represent required alternate keywords or <variables> in a command line description. You must choose one command line argument enclosed within the braces. Choices are separated by vertical (OR) bars.
[a b c] [<a> <c>]	Represent optional alternate keywords or variables in a command line description. Choose one command line argument enclosed within the braces, if desired. Choices are separated by vertical (OR) bars.

Support Contacts

If you encounter a problem while installing, registering or operating this product, please make sure that you have read the documentation. If you cannot resolve the issue, please contact your supplier or Gemalto support. Gemalto support operates 24 hours a day, 7 days a week. Your level of access to this service is governed by the support plan arrangements made between Gemalto and your organization. Please consult this support plan for further information about your entitlements, including the hours when telephone support is available to you.

Contact method	Contact	
Address	Gemalto 4690 Millennium Drive Belcamp, Maryland 21017 USA	
Phone	Global	+1 410-931-7520
	Australia	1800.020.183
	China	(86) 10 8851 9191
	France	0825 341000
	Germany	01803 7246269
	India	000.800.100.4290
	Netherlands	0800.022.2996
	New Zealand	0800.440.359
	Portugal	800.1302.029
	Singapore	800.863.499
	Spain	900.938.717
	Sweden	020.791.028
	Switzerland	0800.564.849
	United Kingdom	0800.056.3158
	United States	(800) 545-6608
Web	https://safenet.gemalto.com	
Support and Downloads	https://safenet.gemalto.com/technical-support Provides access to the Gemalto Knowledge Base and quick downloads for various products.	
Technical Support Customer Portal	https://serviceportal.safenet-inc.com Existing customers with a Technical Support Customer Portal account can log in to manage incidents, get the latest software upgrades, and access the Gemalto Knowledge Base.	

Introduction

SafeNet ProtectToolkit-C is a cryptographic service provider using the PKCS #11 application programming interface (API) standard, as specified by RSA Labs. It includes a lightweight, proprietary Java API to access these PKCS #11 functions from Java.

The PKCS #11 API, also known as Cryptoki, includes a suite of cryptographic services for encryption, decryption, signature generation, signature verification, and permanent key storage. The software found on the installation DVD is compliant with PKCS #11 v. 2.20. The latest versions of the client software and HSM firmware can be found on the Gemalto eService Support Portal at <https://serviceportal.safenet-inc.com>.

To provide the highest level of security, SafeNet ProtectToolkit-C interfaces with SafeNet access provider software and the SafeNet range of hardware security modules (HSMs):

- SafeNet ProtectServer Network HSM
- SafeNet ProtectServer PCIe HSM

HSMs include high-speed DES and RSA hardware acceleration, as well as generic security processing. Secure, persistent, tamper-resistant CMOS key storage is included. Multiple adapters may be used in a single host computer to improve throughput or to provide redundancy. HSMs may be installed locally, on the same host system as SafeNet ProtectToolkit-C or they may be located remotely across a network.

Two product packages are available:

- Runtime for operational use
- Software Development Kit (SDK) for developer use

With SafeNet ProtectToolkit-C SDK installed, the API may operate in Software-Only mode for testing and development. In this mode, access to an HSM is not required.

Who Should Read This Manual?

This manual is intended for the SafeNet ProtectToolkit-C Administrator, responsible for installation, configuration, security policy and number of applications (or users) of SafeNet ProtectToolkit-C. This configuration of SafeNet ProtectToolkit-C will determine the functionality and services available to the SafeNet ProtectToolkit-C applications. The Administrator is strongly encouraged to read this manual thoroughly before attempting any operations.

The manual also provides information on the structure and features of SafeNet ProtectToolkit-C, and therefore serves as a valuable reference for any user.

This manual also provides configuration details for some standard PKCS #11 applications compatible with SafeNet ProtectToolkit-C.

Further Documentation

SafeNet Manuals

In addition to this Administration Guide, the following manuals contain relevant information. They are referenced in this manual when applicable.

Hardware

- *SafeNet ProtectServer PCIe HSM Installation Guide*
- *SafeNet ProtectServer Network HSM Installation Guide*

Software

- *SafeNet HSM Access Provider Installation Guide*
- *SafeNet ProtectToolkit-C Programming Guide*

SafeNet Application Integration Guides

A number of integration guides are available, outlining the use of SafeNet products with third-party applications. For more information, contact your SafeNet representative (see ["Support Contacts" on page 11](#)).

Utility Normal Mode vs. Work Load Distribution and HA Mode

In this document, any references to the name of a utility without any further qualification refer to the utility operating in NORMAL mode. Any references to the name of a utility with the qualification (WLD/HA) refer to the utility operating in Work Load Distribution and High Availability Mode.

For example **ctkm** refers to the CTKMU utility operating in NORMAL mode, while, **ctkm** (WLD) refers to the CTKMU utility operating in WLD mode. Refer to section ["Operation in WLD Mode" on page 56](#) for details.

Installation and Configuration

This chapter covers key concepts for administrators who will install or uninstall the product on host computer systems.

The instructions are organized by the operating systems compatible with SafeNet ProtectToolkit-C. Users only need consult the sections applicable to their systems. See also ["System Requirements" below](#).

If you will be using SafeNet ProtectToolkit-C with a SafeNet ProtectServer HSM, access provider software *must be installed first*. See the *SafeNet HSM Access Provider Installation Guide* for instructions on setting up the Access Provider. If you plan to operate SafeNet ProtectToolkit-C in Software-Only Mode, no access provider software is required.

This chapter contains the following sections:

- ["System Requirements" below](#)
- ["Operating Mode Setup" on the next page](#)
- ["Basic Windows Installation" on page 17](#)
- ["Basic Unix/Linux Installation" on page 19](#)
- ["Changing the Cryptoki Provider" on page 21](#)
- ["Using the Unix Installation Utility" on page 22](#)
- ["Unix/Linux Command Reference" on page 26](#)
- ["Configuration Items" on page 29](#)
- ["Troubleshooting" on page 35](#)

System Requirements

- A PC with a Pentium-class processor or better and a spare PCI Express bus interface slot (if installing an adapter card)
- A SafeNet hardware security module (not required when using Software-Only operating mode for development and testing)
- Java runtime (required for graphical user interface utilities only). The product has been tested using Java runtime version 6.x, 7.x, and 8.x. It may also operate correctly using other versions of the runtime, but Gemalto does not support other versions.
- .NET versions 3.5 and 4.5 (Windows only). All required .NET versions are available for download from Microsoft.
- MSVC 2005, MSVC 2008, and MSVC 2010 (Windows only). All required MSVC versions are available for download from Microsoft.



Note: The Java runtime, .NET and MSVC must be installed first.

Supported Platforms

The supported platforms are listed in the following table.

C=SafeNet ProtectToolkit-C, PKCS #11 v2.10/2.20

M=SafeNet ProtectToolkit-M, MS CSP 2.0 with CNG

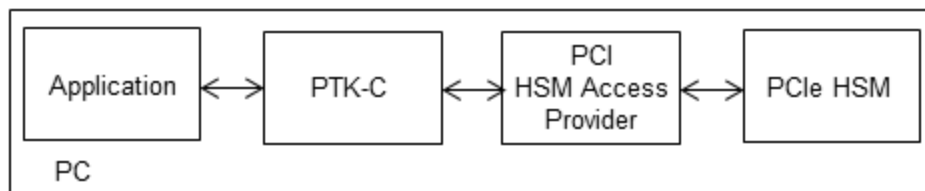
J=SafeNet ProtectToolkit-J, Java runtime 6.x/7.x/8.x

Operating System		OS type	64-bit PTK	64-bit PTK supported hardware	32-bit PTK	32-bit PTK supported hardware
Windows	Server 2008 (R1 and R2)	64-bit	C/M/J	All platforms	C/J	Network HSM, PSE
	Server 2012 R2	64-bit	C/M/J	All platforms	C/J	Network HSM, PSE
	7	32-bit	-	-	C/J (KSP supported)	All platforms
	7	64-bit	C/M/J	All platforms	C/J	Network HSM, PSE
Linux	RHEL 6	32-bit	-	-	C/J	All platforms
	RHEL 6	64-bit	C/J	All platforms	C/J	Network HSM, PSE
	RHEL 7	64-bit	C/J	All except PSI-E (K5)	C/J	Network HSM, PSE
	SUSE12	64-bit	C/J	All except PSI-E (K5)	C/J	Network HSM, PSE
AIX	6.1	64-bit	C/J	Network HSM, PSE	C/J	Network HSM, PSE
	7.1	64-bit	C/J	Network HSM, PSE	C/J	Network HSM, PSE
	7.2	64-bit	C/J	Network HSM, PSE	C/J	Network HSM, PSE
Solaris	10 (SPARC, x86) 11 (SPARC, x86)	64-bit	C/J	Network HSM, PSE	C/J	Network HSM, PSE
HP-UX	11	64-bit	C/J	Network HSM, PSE	C/J	Network HSM, PSE

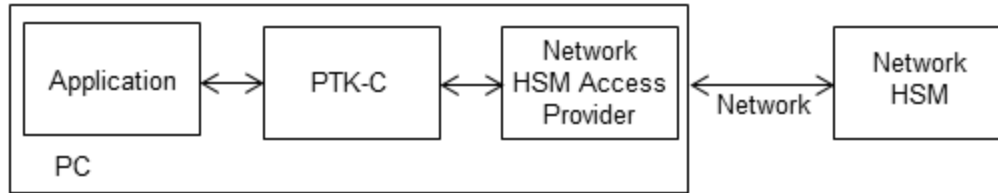
Operating Mode Setup

SafeNet ProtectToolkit-C can be used in one of three *operating modes*. These are:

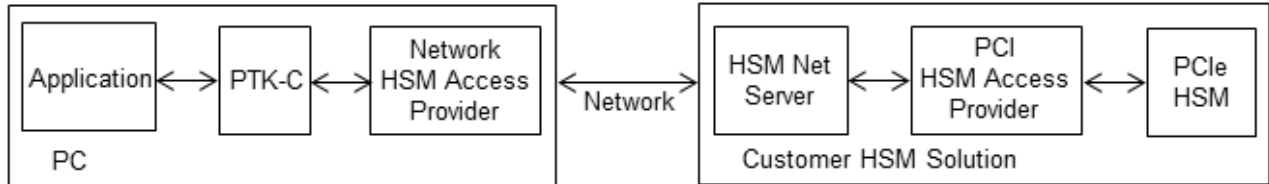
- **PCI mode** in conjunction with a locally-installed SafeNet cryptographic services adapter.



- **Network mode** over a TCP/IP network, in conjunction with a compatible product such as the SafeNet ProtectServer PCIe HSM.



A machine with a SafeNet ProtectServer PCIe HSM installed may also be used as a server in network mode.



- **Software-only mode**, on a local machine without access to a hardware security module.

Within the client/server runtime environment, the server performs cryptographic processing at the request of the client. The server itself will only operate in one of the hardware runtime modes.

The software-only version is available for a variety of platforms, including Windows NT and Solaris, and is typically used as a development and testing environment for applications that will eventually use the hardware variant of SafeNet ProtectToolkit-C.

Installation of SafeNet ProtectToolkit-C is part of an operating mode setup.

PCI and Network Operating Modes

1. Install the SafeNet HSM hardware.

This includes assigning an IP address, hostname, gateway, and access control. Consult the relevant installation manual:

- *SafeNet ProtectServer PCIe HSM Installation Guide*
- *SafeNet ProtectServer Network HSM Installation/Configuration Guide*

2. Install the necessary third-party software.

Install the Java runtime, .NET (Windows only) and MSCV (Windows only) software. See ["System Requirements" on page 14](#).

3. Install and configure access provider software.

Access provider software must be installed and configured to support the operating mode. See the *SafeNet HSM Access Provider Installation Guide*.

4. Install SafeNet ProtectToolkit-C (RT or SDK) and confirm correct operation of the hardware.

See the appropriate section for your system:

- ["Basic Windows Installation" on the next page](#)
- ["Basic Unix/Linux Installation" on page 19](#)

5. Configure the secure messaging system (SMS).

Refer to ["Secure Messaging" on page 30](#), ["Messaging Mode Configuration" on page 31](#), and ["Configuring Session Protection" on page 32](#).

6. Establish network communication (network operating mode only).

To establish network communication, the client must be configured to use one or more servers that are available on the same network. Refer to ["Specifying the Network Server\(s\)" on page 33](#).

Software-Only Mode

1. Install the necessary third-party software.

Install the Java runtime, .NET (Windows only) and MSCV (Windows only) software. See ["System Requirements" on page 14](#).

2. Install SafeNet ProtectToolkit-C SDK.

See the appropriate section for your system:

- ["Basic Windows Installation" below](#)
- ["Basic Unix/Linux Installation" on page 19](#)

3. Make configuration changes if required.

The installation can be customized to optimize performance. Refer to ["Software-Only Mode Configuration" on page 34](#).

Basic Windows Installation

This section provides instructions for installing the SafeNet ProtectToolkit-C Runtime and SDK packages on a Windows-based operating system. It contains the following subsections:

- ["SafeNet ProtectToolkit-C Packages" below](#)
- ["Windows Install Preparation" on the next page](#)
- ["Runtime Installation Procedure" on the next page](#)
- ["SDK Installation Procedure" on page 19](#)

SafeNet ProtectToolkit-C Packages

Two product packages are available for all supported Windows and Unix/Linux platforms. The latest versions of the client software and HSM firmware can be found on the Gemalto eService Support Portal at <https://serviceportal.safenet-inc.com>.

Runtime Package (PTKcprt)

The Runtime package provides all the necessary tools and interfaces for a SafeNet ProtectToolkit-C based Cryptoki service provider.

SDK Package (PTKcpsdk)

The SDK package is a software development platform. Header files are included, in addition to all the necessary tools and interfaces for a SafeNet ProtectToolkit-C based Cryptoki service provider.

Developers may work using an HSM installed locally (PCIe mode) or accessible across a network (network mode), to most closely approximate the operating environment. Software-only mode is also available.

Testing can easily be performed on any machine using software-only mode, without the need for an HSM. This may be useful when it is not feasible to make HSMs available to multiple developers.



CAUTION: Software-only mode is not secure, since key files are located on the hard drive of the host computer system.

Different SafeNet ProtectToolkit-C Cryptoki provider files are required for software-only mode and operation with an HSM. The developer must specify the required Cryptoki provider during installation. Both Cryptoki provider files are installed and the specified one is made active. This selection can be switched if the developer wishes to change operating modes. See ["Changing the Cryptoki Provider" on page 21](#) for details.

Windows Install Preparation

If you are planning to operate SafeNet ProtectToolkit-C in PCI or Network mode, you should already have installed the appropriate versions of the Java runtime, MSVC, and .NET prior to installing the access provider software (see ["System Requirements" on page 14](#)). If you have not done so already, please consult the *SafeNet HSM Access Provider Installation Guide* before continuing.

If you are planning to operate SafeNet ProtectToolkit-C in software-only mode, you must still install the Java runtime, MSVC, and .NET before SafeNet ProtectToolkit-C. No access provider software is required.

Upgrading/Uninstallation

If you are upgrading SafeNet ProtectToolkit-C from an earlier version, you must first uninstall any currently-installed version by using the Windows **Programs and Features** control panel.



Note: The Runtime and SDK packages cannot be installed concurrently. To switch from one package to the other, uninstall the package that is no longer required and then install the new one.

Preparation

1. Ensure that you have downloaded and installed the required versions of the Java runtime, MSVC, and .NET. See ["System Requirements" on page 14](#).
2. If you intend to operate SafeNet ProtectToolkit-C in PCI or Network mode, ensure you have installed the access provider software. See the *SafeNet HSM Access Provider Installation Guide*.
3. Ensure that you have Administrator privileges on the system.

Runtime Installation Procedure



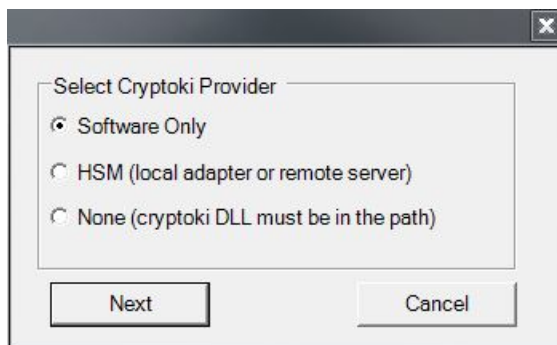
Note: You must install the access provider software before you can install the SafeNet ProtectToolkit-C Runtime. Otherwise the installation will fail. See the *SafeNet HSM Access Provider Installation Guide* for details.

To install the SafeNet ProtectToolkit-C Runtime package:

1. Locate the installer directory and execute the file **PTKcprt.msi**. This is the SafeNet ProtectToolkit-C Runtime package.
2. Work through the installation wizard to complete the installation.

SDK Installation Procedure**To install the SafeNet ProtectToolkit-C SDK package:**

1. Locate the installer directory and execute the file **PTKcpsdk.msi**. This is the SafeNet ProtectToolkit-C SDK package.
2. Work through the installation wizard to complete the installation.
3. A dialog is displayed during the installation process (shown below) that gives the option to update the **Path** to include the required Cryptoki provider. This will make the correct installed programs and libraries available from the command prompt.



If...	then select...
The SDK is to be used without access to a HSM (software only mode)	Software Only
An HSM will be available (PCI or network operating modes)	HSM
The Cryptoki provider required is already in the path (this might be the case if you are upgrading)	None

For more information about the available options see ["Operating Mode Setup" on page 15](#).



Note: The selection made here can be changed later using the **setmode** tool (see ["Changing the Cryptoki Provider" on page 21](#)).

Basic Unix/Linux Installation

This section provides instructions for installing the SafeNet ProtectToolkit-C Runtime and SDK packages on a Unix/Linux-based operating system. It contains the following subsections:

- ["SafeNet ProtectToolkit-C Packages" on the next page](#)
- ["Unix/Linux Install Preparation" on the next page](#)
- ["Installation Procedure" on page 21](#)

SafeNet ProtectToolkit-C Packages

Two product packages are available for all supported Windows and Unix/Linux platforms. The latest versions of the client software and HSM firmware can be found on the Gemalto eService Support Portal at <https://serviceportal.safenet-inc.com>.

Runtime Package (PTKcprt)

The Runtime package provides all the necessary tools and interfaces for a SafeNet ProtectToolkit-C based Cryptoki service provider.

SDK Package (PTKcpsdk)

The SDK package is a software development platform. Header files are included, in addition to all the necessary tools and interfaces for a SafeNet ProtectToolkit-C based Cryptoki service provider.

Developers may work using an HSM installed locally (PCIe mode) or accessible across a network (network mode), to most closely approximate the operating environment. Software-only mode is also available.

Testing can easily be performed on any machine using software-only mode, without the need for an HSM. This may be useful when it is not feasible to make HSMs available to multiple developers.



CAUTION: Software-only mode is not secure, since key files are located on the hard drive of the host computer system.

Different SafeNet ProtectToolkit-C Cryptoki provider files are required for software-only mode and operation with an HSM. The developer must specify the required Cryptoki provider during installation. Both Cryptoki provider files are installed and the specified one is made active. This selection can be switched if the developer wishes to change operating modes. See ["Changing the Cryptoki Provider" on the next page](#) for details.

Unix/Linux Install Preparation

Upgrading/Uninstallation

If you are upgrading SafeNet ProtectToolkit-C from an earlier version, you must first uninstall any currently-installed version.

The SafeNet ProtectToolkit-C uninstallation commands differ between the supported Unix/Linux platforms. The **Unix Installation Utility** is recommended for uninstalling previous versions, as it will automatically account for these differences. Simply select **uninstall a SafeNet package** from the **Main Menu**, and then select the appropriate package (see ["Using the Unix Installation Utility" on page 22](#)).

If you wish to enter platform-specific commands manually, use the commands given in ["Unix/Linux Command Reference" on page 26](#).



Note: The Runtime and SDK packages cannot be installed concurrently. To switch from one package to the other, uninstall the package that is no longer required and then install the new one.

Preparation

1. Ensure that you have downloaded and installed the required version of the Java runtime. See ["System](#)

[Requirements" on page 14.](#)

2. If you intend to operate SafeNet ProtectToolkit-C in PCI or Network mode, ensure you have installed the access provider software. See the *SafeNet HSM Access Provider Installation Guide*.
3. Ensure that you are the super-user on the host system.

Installation Procedure

The SafeNet ProtectToolkit-C installation commands differ between the supported Unix/Linux platforms. The **Unix Installation Utility** is recommended for installation, as it will automatically account for these differences. Simply select **Install a Package from this CD** from the **Main Menu**, and then select the appropriate package (see ["Using the Unix Installation Utility" on the next page](#)).

If you wish to enter platform-specific commands manually, use the commands given in ["Unix/Linux Command Reference" on page 26](#).

Setting Up Your Environment

After installing the software, you must run the SafeNet ProtectToolkit **setvars.sh** script to configure your environment to use the SafeNet ProtectToolkit-C software. You cannot run the script directly, but instead you must source it or add it to a startup file (for example, **.bashrc**).

To set up your environment:

1. Go to the SafeNet ProtectToolkit software installation directory:

```
cd /opt/safenet/protecttoolkit5/ptk
```

2. Source the **setvars.sh** script:

```
./setvars.sh
```

Once installed and configured, the software is ready to use under **/opt/safenet**.

Cryptoki Provider Selection (SDK Package Only)

The software-only Cryptoki provider is active by default on Unix/Linux systems. To change to the SafeNet HSM Cryptoki provider use the **Unix/Linux Installation Utility**. For more information, see ["Using the Unix Installation Utility" on the next page](#).

Changing the Cryptoki Provider

This section applies to the SDK package only.

Different SafeNet ProtectToolkit-C Cryptoki provider files are required if an HSM is present (PCI or network mode) or not (software-only mode).

Both Cryptoki provider files are installed with the SDK package. On Windows systems, the user is prompted during installation to choose which is made active. On Unix/Linux systems, the software-only Cryptoki provider is made active by default.



CAUTION: Software-only mode is not secure, as cryptographic material is stored on the host system and not a SafeNet ProtectServer HSM. See ["Changing the Cryptoki provider" on page 24](#) for information on changing the operating mode on Unix/Linux systems.

This section provides instructions for changing the Cryptoki provider on Windows systems.

Set Mode

In SafeNet ProtectToolkit SDK v 5.3, the software emulation batch files for **ctbrowse**, **KMU**, and **gCTAdmin** have been removed, and a new executable binary file called **setmode** has been added. **setmode** allows the user to toggle between software emulator and hardware modes.

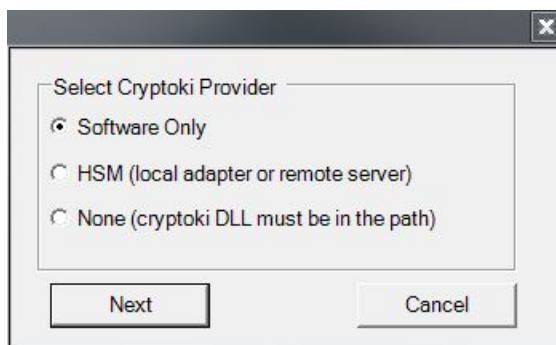
To change the active Cryptoki provider:

1. Execute **setmode** from the command line or open the **SetMode.cmd** file in the SafeNet install directory (default path: **C:\Program Files\SafeNet\Protect Toolkit 5\Protect Toolkit C SDK\bin\SetMode.cmd**).



Note: This tool edits the Windows registry, so you must have Administrator privileges on the client machine or an Unauthorized Access error will be returned. If you receive this error, open the command prompt or **SetMode.cmd** file by right-clicking and selecting **Run as Administrator**.

The **Select Cryptoki Provider** dialog appears.



2. Select your desired operating mode and click **Next** to complete the operation.

Using the Unix Installation Utility

Installation and uninstallation commands are different for each of the supported Unix platforms. To account for these differences, the package should be installed using the Unix Installation Utility. Manual commands specific to your operating system can be used, but this is not the recommended method. The Installation Utility is more likely to result in a problem-free installation or uninstallation. The latest versions of the client software and HSM firmware can be found on the Gemalto eService Support Portal at <https://serviceportal.safenet-inc.com>.

The utility provides a simple menu-driven interface. In addition to installing and uninstalling the access provider on Unix systems, it can also:

- List already-installed SafeNet packages
- List directory contents, for the current platform or all platforms
- Install a package from the directory (which also installs the utility in **/usr/bin**)
- Change the default operating mode (hardware or software-only).

Whenever the utility installs a SafeNet package, it also installs itself on the host system's hard disk (in **/usr/bin/safeNet-install.sh**). This copy can be used to uninstall or configure the software.

Utility Startup

Should you encounter any problems while following this procedure, please see ["Unix Installation Utility troubleshooting" on page 25](#). Options can be specified when executing the **safeNet-install.sh** command. These options are not normally required and are mainly useful for troubleshooting.

Syntax

safeNet-install.sh [-h] [-p] [-s <size>] [-v]

Option	Description
-h	Show help.
-p	Plain mode. In this mode the 'tput' is not used for video enhancements.
-s<size>	Override the screen size (default = 'tput lines/cols' or 24x80).
-v	Print the version of this script.

If you wish to enter platform-specific commands manually, use the commands given in ["Unix/Linux Command Reference" on page 26](#).

To start up the utility:

1. The Gemalto Unix Installation Utility is located in the installation DVD or image's root directory. Mount the DVD or unzip the image by following standard procedure for your platform and installation.
2. Change directory to the DVD or directory and start the utility. For example:

```
# cd /misc/cd
# ./safeNet-install.sh
```

The utility scans the system and the directory and displays the Main Menu.

Gemalto Unix Installation Utility (version 5.3.0):

Hostname: 66 (Linux 2.6.32-504.16.2.el6.i686)

Main menu

- 1 list Gemalto packages already installed
- 2 list packages on CD
- 3 install a package from this CD
- 4 uninstall a Gemalto package
- 5 Set the default cryptoki and/or hsm link
- q quit the utility

Choice (1 2 3 4 q) [Redraw]:



Note: Enter 'b' to go back to the previous menu and 'q' to quit the utility. You can also quit with the system **INTR** key (normally **^C**).

Installing a package

Should you encounter any problems, please see ["Unix Installation Utility troubleshooting" on page 25](#).

To install a package:

1. Select **install a package from this CD** from the utility's Main Menu.
A list of installable SafeNet packages is displayed.
2. Select the package required by typing the appropriate menu number followed by **Enter**.
The utility verifies the action and executes the appropriate command for your platform.
3. On some platforms, you may be prompted for additional installation options. On Linux, for example, you can add a **nodeps** option to suppress the checking of dependencies. These options should be selected with appropriate care.
4. You may now need to respond to any platform-specific messages (for example: to confirm you wish to proceed with the installation).
5. After installation, the utility will return **Success** or **Failure**, scan the system again, and display the current installation status. Press the **Enter** key to continue.

Setting up your environment

After installing the software, you must run the SafeNet ProtectToolkit **setvars.sh** script to configure your environment for the SafeNet ProtectToolkit software. You cannot run the script directly, but instead you must source it or add it to a startup file (for example, **.bashrc**). If you source the script, your environment will be set for the current session only. If you add it to your startup file, your environment will be set each time you log in.

To set up your environment:

1. Go to the SafeNet ProtectToolkit software installation directory:

```
cd /opt/safenet/protecttoolkit5/ptk
```

2. Source the **setvars.sh** script:

```
./setvars.sh
```

Once installed and configured, the software is ready to use under **/opt/safenet**.

Changing the Cryptoki provider

On Unix/Linux systems, the software-only Cryptoki provider is made active by default. If you plan to use this instance of SafeNet ProtectToolkit-C with a SafeNet ProtectServer HSM, you will need to change the Cryptoki provider. Software-only mode is not secure, as cryptographic material is stored on the host system. You can use the Unix Installation Utility to change modes.

To change the Cryptoki provider:

1. From the **Main** menu, select **Set the default cryptoki and/or HSM link**.

The **Cryptoki Selection** screen is displayed.

```
Gemalto Unix Installation Utility (version 5.3.0):
Hostname: 66 (Linux 2.6.32-504.16.2.el6.i686)
Main Menu >> Check/Set Default Cryptoki & HSM Menu
```

```
----- Cryptoki Selection -----
```

```
1 SafeNet ProtectToolkit C SDK Software (emulator)
2 * SafeNet ProtectToolkit C SDK Runtime (hardware)
```

```
3 * SafeNet Network HSM Access Provider
```

```
b back
```


q quit the utility

Choice (1 2 3 b q) [Redraw]:

2. Select **SafeNet ProtectToolkit C SDK Runtime (hardware)** and confirm your selection.

Uninstalling a package

Should you encounter any problems, please see ["Unix Installation Utility troubleshooting"](#) below.

To uninstall a package:

1. Select **Uninstall a SafeNet package** from the utility's **Main Menu**.
A list of installed SafeNet packages is displayed.
2. Select the required package by typing the appropriate menu number and pressing **Enter**.
The utility verifies the action and executes the appropriate command for your platform.
3. On some platforms, you may be prompted for additional uninstallation options. On Linux, for example, you can add a **-nodeps** option to suppress the checking of dependencies. These options should be selected with appropriate care.
4. After completing uninstallation, the utility will return **Success** or **Failure**, scan the system again, and display the current installation status.
5. You may now need to respond to any platform-specific messages to confirm that you wish to proceed with the uninstallation. Press the **Enter** key to continue.

Unix Installation Utility troubleshooting

Problem	Solution
Packages to install or uninstall are not visible	If no packages are shown to install or uninstall, close the utility, check that you are logged on as root , and ensure your current directory is on the DVD or directory before running the utility again.
The screen is confused or does not display correctly	<p>This utility relies on the TERM environment parameter when creating colors and measuring screen size, so make sure this is set correctly. The most common values are xterm or vt100. For example, to set TERM to vt100:</p> <pre># TERM=vt100# export TERM</pre> <ul style="list-style-type: none"> • If the screen is confused, run the utility in "plain" mode as follows: # ./safeNet-install.sh -p • If the size of the terminal is not correctly set by termcap (for example: the headings disappear off the top of the screen), override the screen size with the -s option: # ./safeNet-install.sh -s 24x80 • If using an X system terminal window, do not resize the window while running the utility, as it cannot sense the change.
The backspace key does not operate	On some terminals, the backspace key does not operate correctly. If, after typing a number and then backspace, the terminal returns "2^H" instead of an actual backspace:

Problem	Solution
correctly	<ul style="list-style-type: none"> Type the current KILL character (normally ^U) and then enter the desired number (you will need to do this each time a backspace is required) Exit the utility (perhaps with ^C) and use the stty(1) command to correct the erase character before restarting the utility: # stty erase ^H where ^H is the character created by pressing the backspace key. This will fix the problem semi-permanently, for the current session in that terminal.

Unix/Linux Command Reference

The simplest way to complete installation or uninstallation of SafeNet ProtectToolkit-C, or to change the Cryptoki provider on any of the Unix/Linux platforms, is to use the **Unix Installation Utility**. The utility ensures that the correct commands for your platform are executed automatically. See ["Using the Unix Installation Utility" on page 22](#) for more information.

If you wish to enter platform-specific commands manually, use the commands in this section.

Manual installation

Before adding or removing packages, you must become the super-user on the host system.



Note: The Runtime and SDK packages cannot be installed concurrently. To switch from one package to the other, uninstall the package that is no longer required and then install the new one.

The Runtime and SDK packages are packaged using the standard packaging software for each system. The latest versions of the client software and HSM firmware can be found on the Gemalto eService Support Portal at <https://serviceportal.safenet-inc.com>.

To install the software on your host system:

1. Refer to the following table for the name of the package to be installed on your operating system.

Linux Platforms	
Runtime package	PTKcprt
SDK package	PTKcpsdk

2. Refer to the following table for the package required and locate the row for the correct operating system, as installed on the host system. Use the program and commands listed to install the software.

Runtime Package		
Operating System	Program	Example Commands
Solaris (SPARC & Intel)	pkgadd(1M)	<code>pkgadd -d /cdrom/Solaris/PTKC_Runtime</code>

Runtime Package		
Operating System	Program	Example Commands
Linux	rpm(8)	cd <path to directory>/Linux/PTKC_Runtime rpm -i PTKcprt-x.xx-y.i386.rpm (where x.xx-y refers to the version of the software)
AIX	installp	cd <path to directory>/AIX/PTKC_Runtime installp -acgNQqWx -d . PTKcprt.rte
HP-UX	swinstall	cd <path to directory>/HP-UX/PTKcprt swinstall PTKcprt.depot

SDK Package		
Operating System	Program	Example Commands
Solaris (SPARC & Intel)	pkgadd(1M)	pkgadd -d /cdrom/Solaris/PTKC_SDK
Linux	rpm(8)	cd <path to directory>/Linux/PTKC_SDK rpm -i PTKcpsdk-x.xx-y.i386.rpm (where x.xx-y refers to the version of the software)
AIX	installp	cd <path to directory>/AIX/PTKC_SDK installp -acgNQqWx -d . PTKcpsdk.rte
HP-UX	swinstall	cd <path to directory>/HP-UX/PTKcpsdk swinstall PTKcpsdk.depot

3. Add the **/opt/safenet/protecttoolkit5/bin** directory to the execution path and the **/opt/safenet/protecttoolkit5/lib** directory to the library path. The following commands may be used to configure your paths for the **sh(1)** shell. Please consult your operating system manual for other shells.

Operating System	Commands
Solaris (SPARC & Intel) Linux	PATH=\$PATH:/opt/safenet/protecttoolkit5/bin export PATH LD_LIBRARY_PATH=\$LD_LIBRARY_PATH:/opt/safenet/protecttoolkit5/lib export LD_LIBRARY_PATH
Solaris (x86) 64-bit	PATH=\$PATH:/opt/safenet/protecttoolkit5/bin/amd64 export PATH LD_LIBRARY_PATH=\$LD_LIBRARY_PATH:/opt/safenet/protecttoolkit5/lib export LD_LIBRARY_PATH
AIX	PATH=\$PATH:/opt/safenet/protecttoolkit5/bin export PATH LIBPATH=\$LIBPATH:/opt/safenet/protecttoolkit5/lib export LIBPATH
HP-UX	PATH=\$PATH:/opt/safenet/protecttoolkit5/bin export PATH SHLIB_PATH=\$SHLIB_PATH:/opt/safenet/protecttoolkit5/lib export SHLIB_PATH

Once installed, the software is ready to use under **/opt/SafeNet/protecttoolkit5**.

Changing the Cryptoki Provider manually

This section applies to the SDK package only.

Different SafeNet ProtectToolkit-C Cryptoki provider files are required if an HSM is present (PCI or network mode) or not (software-only mode).

Both Cryptoki provider files are installed with the SDK package. On Unix/Linux systems, the software-only Cryptoki provider is made active by default.

To change the default Cryptoki provider selection:

Remove the soft-link:

```
/opt/safenet/protecttoolkit5/ptk/lib/libcryptoki.so or
/opt/safenet/protecttoolkit5/ptk/lib/libcryptoki.a (for AIX)
```

and recreate it to point to the SafeNet HSM Cryptoki provider. For example, the following shell commands may be used to enable the HSM (executed as the super-user):

```
# cd /opt/safenet/protecttoolkit5/ptk/lib
# rm libcryptoki.so
# ln -s libcthsm.so libcryptoki.so
```

The following shell commands may be used to enable the software emulation (executed as the super-user):

```
# cd /opt/safenet/protecttoolkit5/ptk/lib
# rm libcryptoki.so
# ln -s libctsw.so libcryptoki.so
```

Manual uninstallation

Before adding or removing any packages, you must become the super-user on the host system.

To uninstall the software from your host system:

1. Identify the name of the package to be uninstalled from your system.

Linux, Solaris and AIX Platforms	
Runtime package	PTKcprt
SDK package	PTKcpsdk

2. Use the program and command listed to remove the software, where <package name> is the correct name of the package to be uninstalled.

Operating System	Program	Command
Solaris (SPARC & Intel)	pkgrm(1M)	pkgrm <package name>
Linux	rpm(8)	rpm -e <package name>
AIX	installp*	installp -u <package name>

Operating System	Program	Command
HP-UX	swremove	swremove <package name>

* **smit** can also be used

** **sam** can also be used

Configuration Items

During installation, configuration items are created on the host system. Configuration changes are made by editing the values associated with these items. This chapter describes how to make such changes on your system.

Item values can exist at four configuration levels. When a configuration item is queried, item locations are searched in order of level precedence:

1. **Temporary:** Any changes made at the temporary configuration level override any corresponding entries at the user, system, and default levels.
2. **User:** Changes made at the user level override any corresponding entries at the system and default levels.
3. **System:** System changes override default-level entries.
4. **Default:** If no changes have been made at any other level, the default value for the configuration item is used. Default configuration values cannot be changed.

On Windows operating systems, user and system configuration information is stored in the Registry. On Unix-based systems, configuration files are used. Temporary configuration items are applied using environment variables on both Windows and Unix-based platforms.

Regardless of the platform, a common naming convention for configuration items has been followed. Understanding this naming convention will help you locate and change the appropriate configuration items when required.

Configuration items are hierarchical in structure, with the root node always being **PTK**. Child nodes of the root represent the class of the item, and are typically product abbreviations, such as **PTKC** (SafeNet ProtectToolkit-C) or **HSM** (Hardware Security Module). Nodes under class represent the component, such as **LOGGER** or **SMS**. Finally, nodes under component represent the configuration item, such as **FILE** or **MODE**. Configuration items therefore take the form:

PTK_<class>_<component>_<item>

Windows

Temporary

Temporary configuration changes are made using environment variables. Since environment variables are not hierarchical, the hierarchy is implicitly defined by the name of the variable.

User

User configuration changes are made in the registry tree starting from **HKEY_CURRENT_USER\SOFTWARE\SafeNet**.

System

System configuration changes are made in the registry tree starting from **HKEY_LOCAL_MACHINE\SOFTWARE\SafeNet**.

Example:

The name of the SafeNet ProtectToolkit-C file where the logger library writes log information (**ctlog.log**) is stored in the Windows registry as a string value for the entry:

PTK_PTKC_LOGGER_FILE

This is located in the key:

HKEY_LOCAL_MACHINE\SOFTWARE\SafeNet\PTKC\LOGGER

Unix/Linux

Temporary

Temporary configuration changes are made using environment variables. Since environment variables are not hierarchical in nature, the hierarchy is implicitly defined by the name of the variable.

User

User Configuration is a set of files located in the **\$HOME/.safenet** directory.

System

System Configuration is a set of files located in the **/etc/default** directory.

The User and System Configuration files are of the form: **PTK_<class>**. Entries in the file are of the form: **PTK_<class>_<component>_<item>=<value>**.

Example:

The name of the SafeNet ProtectToolkit-C file where the logger library writes log information (**ctlog.log**) is stored in the **/etc/default/et_ptkc** file as the entry:

PTK_PTKC_LOGGER_FILE=/ctlog.log

Secure Messaging

An optional *trusted channel* called the Secure Message System (SMS) may be enabled. It is disabled by default. This system enables applications to securely communicate with HSMs over the PCI bus interface, or across a network.

A trusted channel is created on-demand by the operator but may be terminated by either the HSM or the operator. Either the HSM or application may be configured to require a trusted channel to be created before cryptographically sensitive services can be provided. For the HSM to be compliant to FIPS 140-2 Level 3 operation the HSM must be configured in this way. However it is also possible for the application to request and use a trusted channel even though the HSM is not configured to require them.

The HSM can manage multiple simultaneous trusted channels, each of which will have its own set of randomly generated session keys for message encryption/decryption and message signing/verification. The negotiation of these session keys is based on a shared secret known by both the application and the HSM.

ProtectServer uses Anonymous Diffie Hellman (ADH) secure messaging. The shared secret is a triple length DES key derived from Anonymous Diffie Hellman key.

To configure and enable an SMS complete the following steps.

- 1. Configure secure messaging mode.**

You may need to change the session key rollover default configuration. See the section ["Configuring Session Protection" on the next page](#) for further information.

2. Configure session protection and enable SMS.

The SMS is enabled by setting one or more security flags that control how the SMS functions. By default these flags are cleared so SMS is disabled. To enable and configure SMS, see the section ["Configuring Session Protection" on the next page](#).

Messaging Mode Configuration

Configuring Session Key Rollover

Session key rollover involves dynamically changing the keys used to perform encryption/decryption between the application and the hardware security module (HSM).

Two mechanisms can be used to trigger session key rollover.

1. The first mechanism triggers session key rollover once a preset number of blocks have been encrypted or decrypted by the application.
2. The second mechanism triggers session key rollover after a preset number of hours have elapsed since a connection was established with the HSM.

Each of these mechanisms is covered in more detail in the following sections.

Preset Number of Blocks Trigger

This mechanism is used to trigger session key rollover once a preset number of blocks have been encrypted or decrypted by the application. The default value for the number of blocks is 2^{32} . This default value can be overridden by setting the configuration item ET_PTKC_SMS_BLOCKS to the desired value.

For example, on a UNIX machine, to temporarily change the key rollover trigger so that key rollover occurs after 10,000 blocks have been encrypted or decrypted the following shell commands would be used:

```
$ ET_PTKC_SMS_BLOCKS=10000
$ export ET_PTKC_SMS_BLOCKS
```

This change can be made at the temporary, user or system levels on both UNIX and Windows platforms. Refer to ["Configuration Items" on page 29](#) for further details on how to go about this if required.

Preset Number of Hours Trigger

This mechanism is used to trigger session key rollover after a preset number of hours have elapsed since a connection was established with the HSM. The default value for the number of hours is 24. This default value can be overridden by setting the configuration item ET_PTKC_SMS_HOURS to the desired value.

For example, on a UNIX machine, to temporarily change the key rollover trigger to occur after 4 hours have elapsed, the following shell commands would be used:

```
$ ET_PTKC_SMS_HOURS=4
$ export ET_PTKC_SMS_HOURS
```

This change can be made at the temporary, user or system levels on both UNIX and Windows platforms. Refer to ["Configuration Items" on page 29](#) for further details on how to go about this if required.

Configuring Session Protection

When applications establish a session with a hardware security module (HSM) using SafeNet ProtectToolkit-C, secure messaging layer activation depends upon:

- Security flag settings (the security policy) stored in tamperable memory inside the HSM by the administrator
- Any additional security flag settings specified by users where they wish to increase the level of security used. These user specified security flag settings are stored in the Secure Messaging Policy Register (SMPR) on the client machine.

Generally, the HSM-stored security flag settings are sufficient so the Secure Messaging Policy Register is rarely used.



Note: Session protection is only applied to Cryptoki functions that use a session handle returned from a previous call to **C_OpenSession()**.

HSM Stored Security Flags

HSM stored security flags can be set at the local machine regardless of whether the HSM is located in the same machine as the application (PCI mode) or remotely (network mode). In the latter case it will be necessary to know the administrator's password for the server machine as this must be entered before any server side changes can be made.

The following table lists those flags that, when set for HSM storage, effect secure messaging. For further information about these flags please see "[Security Policies and User Roles](#)" on page 68.

Flag	Secure Messaging Effect
No clear PINs	Only messages sent to the HSM that contain sensitive data are encrypted
Auth Protection	Only messages sent to the HSM are signed
Full Secure Message Encryption	All messages sent to and from the HSM are encrypted
Full Secure Message Signing	All messages sent to and from the HSM are signed

To Set HSM-Stored Security Flags:

These flags can be set using the SafeNet ProtectToolkit-C **ctconf** utility command, **ctconf -fflags**. Refer to "[Security Policies and User Roles](#)" on page 68 for full details on security policies, setting flags and the use of this command.

SMPR Security Flags

The Secure Messaging Policy Register (SMPR) flag settings augment the HSM settings discussed above and are stored on the client machine by assigning configuration item values.

As the client may access more than one HSM the SMPR can store a unique set of settings for each accessible HSM if required. Each HSM is identified by its serial number for SMPR storage purposes.

The following table lists the SMPR security mode flags, their effect on secure messaging and the configuration item values that must be assigned in order to set them.

Flag	Secure Messaging Effect	Configuration Item Value
No clear PINs	Only messages sent to the HSM that contain sensitive data are encrypted	E
Auth Protection	Only messages sent to the HSM are signed	S
Auth Replies	Only messages received from the HSM are signed	R

To Set SMPR Security Flags

- Obtain the serial number of the HSM.
This can be done by executing the command **ctconf -a<device>** from a command line. Replace **<device>** with the number of the HSM required.
- Create the following configuration item:
`ET_PTKC_<serial>_SMPR`

Replace **<serial>** with the serial number of the HSM found in step 1.
This change can be made at the temporary, user or system levels on both UNIX and Windows platforms. Refer to ["Configuration Items" on page 29](#) for further details on how to go about this if required.
- Set one or more flags by assigning a value to the configuration item using one or more of the Configuration Item Value letters given in the table above. For example, if both Auth Protection and Auth Replies are required assign the value **SR**.

Specifying the Network Server(s)

By default, the net client will attempt to use the local machine as its server. Default values are:

- Server Name = **127.0.0.1**
- Server Port = **12396**

It is necessary to configure the client to use a different host by using the `ET_HSM_NETCLIENT_SERVERLIST` configuration item. Several servers may also be specified using this configuration item in which case the services from each server will be available seamlessly to the client.

You can use hostnames, IPv4 addresses, or IPv6 addresses to specify your network servers.

The full syntax for the `ET_HSM_NETCLIENT_SERVERLIST` configuration item is:

`ET_HSM_NETCLIENT_SERVERLIST=server1[:port1][server2[:port2]]`

UNIX Example

To set the net server to the hostname `ptkc.mydomain.com` at the system level:

- Open the file: `/etc/default/et_hsm`
- Make the entry: `et_hsm_netclient_serverlist=ptkc.mydomain.com`

Windows Example

To set the net server to the hostname **ptkc.mydomain.com** at the system level:

1. Locate the registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\SafeNet\HSM\NETCLIENT

2. Assign the value **ptkc.mydomain.com** to the entry:

ET_HSM_NETCLIENT_SERVERLIST

Using IPv6 addressing

When specifying a host by its IPv6 address, you must enclose the IPv6 address in square brackets.. All other rules which apply to IPv4 addressing also apply for IPv6 addressing. For example, the following command is valid for a server with an IPv6 address of 2001:db8::221:5eff:fe46:f17e:

```
export ET_HSM_NETCLIENT_SERVERLIST=[2001:db8::221:5eff:fe46:f17e]
```

Symbolic server names are also supported and they must be declared in the /etc/hosts and /etc/networks files. For example, if the /etc/hosts file contains the following entry:

```
2001:db8::221:5eff:fe46:f17e  ServerV6
```

then you can run the following command:

```
export ET_HSM_NETCLIENT_SERVERLIST=[ServerV6]
```

Since the interface ports listen for both IPv6 and IPv4, you can specify both IPv4 and IPv6 addresses in the ET_HSM_NETCLIENT_SERVERLIST configuration item, as follows:

```
export ET_HSM_NETCLIENT_SERVERLIST=[<IPv6_address>] <IPv4_address>...
```

Software-Only Mode Configuration

After installing the SafeNet ProtectToolkit-C Software Development Kit (SDK) on your computer system further changes, as detailed in this section, may be made to customize the installation and optimize its performance.

Storage Location Assignment

The software only variant of SafeNet ProtectToolkit-C uses the local file system for storing keys and configuration information. By default, the directory c:\cryptoki is used under Windows and **\$HOME/.cryptoki/cryptoki** under UNIX. It is possible to use a storage location other than the default location for your system by setting the value of the ET_PTKC_SW_DATAPATH configuration item to that of the path required.

For example, on a UNIX machine, to temporarily set the location to **/usr/local/cryptoki** the following **/bin/sh** shell commands would be used:

```
# ET_PTKC_SW_DATAPATH=/usr/local/cryptoki
# export ET_PTKC_SW_DATAPATH
```

This change can be made at the temporary, user or system levels on both UNIX and Windows platforms. Refer to ["Configuration Items" on page 29](#) for further details on how to go about this if required.

Fixing Command Line Utility Low Performance

In software only mode the time taken to detect peripherals, such as attached smart card terminals, can significantly slow the execution of command line utility commands. If this proves to be an annoyance then peripheral detection can be disabled by creating the configuration item below and setting its value equal to **FALSE**.

```
ET_PTKC_SW_DETECTPERIPHERALS
```

This change can be made at the temporary, user or system levels on both UNIX and Windows platforms. Refer to ["Configuration Items" on page 29](#) for further details on how to go about this if required.

Enabling Smart Card Access under UNIX

When attempting to access a smart card reader while operating under any of the supported UNIX platforms in software only mode, ensure that the serial port permissions have been set to allow access to the required port. If this is not done, the logged on user will be unable to see the attached reader.

Troubleshooting

When installing SafeNet ProtectToolkit-C with an encryption adapter card such as the ProtectServer, the most common problems encountered are due to the access provider software being loaded incorrectly.

When installing a Windows driver, you may receive a warning that the SafeNet driver is not signed. This message can safely be ignored. If you have received this message, the version you have received was released before it completed the Microsoft WHQL process. While we do submit the Windows versions of our drivers for Windows Hardware Quality Labs (WHQL) certification, we do not normally hold back a product release or an important update while validation is pending. Note, however, that this assurance applies only to software that you have received directly from SafeNet or via a trusted third-party seller.

If you encounter any difficulties:

- Check that you have followed all the installation instructions in this Guide and any associated manuals for the hardware and access provider software
- Follow any troubleshooting guidance given in the hardware and access provider manuals

Should you encounter any difficulties with the **gctadmin** and **kmu** utilities, the Java runtime might not have been installed prior to SafeNet ProtectToolkit-C. Uninstall both packages and reinstall them in the correct order.

If the issue is still not resolved please see ["Support Contacts" on page 11](#) to contact SafeNet Technical Support.

Cryptoki Configuration

A number of steps must be taken in order for applications to operate correctly with SafeNet ProtectToolkit-C. The SafeNet ProtectToolkit-C environment can be extensively configured in order to allow for the wide range of security requirements that various applications may have. It is important therefore that these requirements be known when configuring SafeNet ProtectToolkit so that the most suitable security settings and functionality for the particular applications can be chosen.

This chapter begins with an introduction to the application and security model used by SafeNet ProtectToolkit-C. The chapter then covers the steps required to configure a system utilizing SafeNet ProtectToolkit-C for the first time. The concepts of Trust Management and Token Replication are discussed and illustrated with examples. Finally, the Work Load Distribution Model is explained and a configuration example is provided.

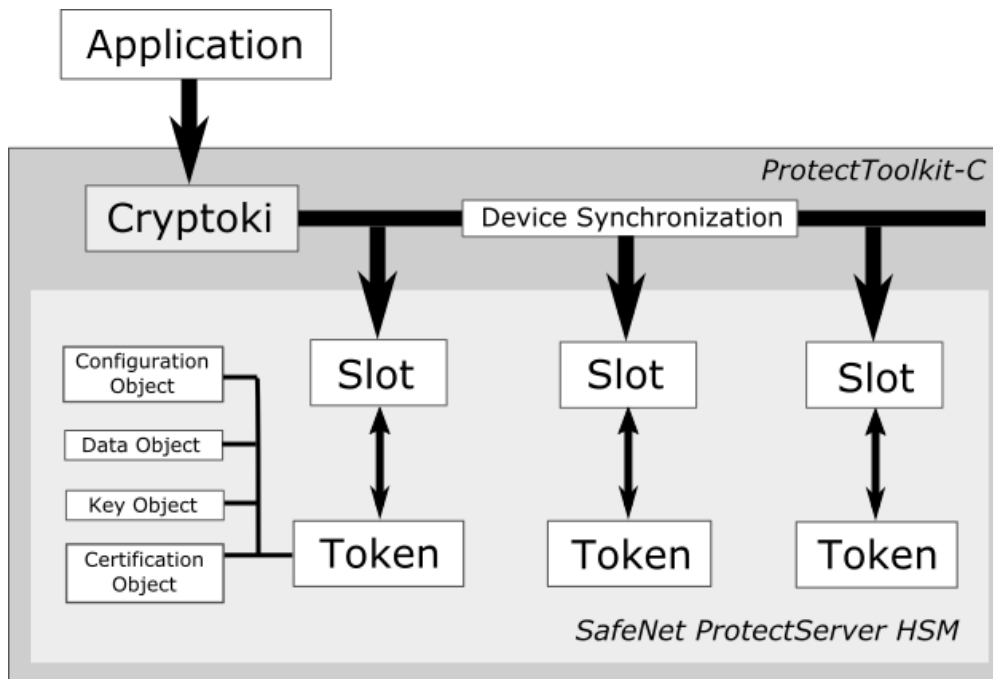
This chapter contains the following sections:

- ["The SafeNet ProtectToolkit-C Model" below](#)
- ["Initial Configuration" on page 39](#)
- ["Work Load Distribution Model and High Availability" on page 49](#)
- ["External Key Storage" on page 59](#)
- ["Real Time Clock" on page 66](#)

The SafeNet ProtectToolkit-C Model

The model for SafeNet ProtectToolkit-C is based on standard PKCS #11 processing. SafeNet ProtectToolkit-C running in hardware mode is depicted below to show how an application sends requests to a token via the PKCS #11 interface.

Figure 1: SafeNet ProtectToolkit-C Model



Slots and Tokens

In the PKCS #11 model, a *slot* represents a device interface and a *token* represents the actual cryptographic device. For example, a smart card reader would represent a slot and the smart card would represent the token.

SafeNet ProtectToolkit-C supports three different slot types: user slots, smart card slots and admin slots. These are described below.

User Slots

User slots are created by the Administrator for use with applications. Each slot automatically holds a User token. All cryptographic mechanisms are supported with these tokens. The system is configurable such that any number of User slots may be created. It is also possible to specify the security policy setting for these slots.

In the default configuration, a single User slot is available. The Administrator can add more slots as required for the local configuration. HSM performance degrades as the number of slots increases. Creating too many slots may cause unacceptable performance. To ensure reasonable performance, it is recommended that you create no more than 200 slots.

Smart Card Slots

Smart card slots are automatically created and configured for each smart card reader attached to the HSM's external serial ports. The smart card tokens can be used for storage of data objects. Their primary purpose is key backup and restoration. To protect objects stored on the token from unauthenticated access, these objects may be PIN-protected. The smart card slots do not support cryptographic operations.

When a supported smart card token is inserted into a configured smart card slot, it will become available to the SafeNet ProtectToolkit-C system. New smart card tokens are blank and require initialization before use. The storage format and layout of files on the tokens is proprietary and can store a maximum of 5 objects (up to the storage capacity of the

actual token). Objects may be deleted; however, the storage allocated to the object is not reclaimed until the token is re-initialized by the Security Officer or Administrator.

The Admin Slot

The Admin slot is designated for the administrator and is used for configuration and administration of the HSM. There is only one Admin slot for each HSM.

The Admin slot holds the *Admin token* and it is on this token that the administration objects reside (See the discussion on "[Administration Objects](#)" below).

PKCS #11 Objects

As shown in "[SafeNet ProtectToolkit-C Model](#)" on the previous page, each token may contain a number of *objects*. The PKCS #11 standard allows for these different types of objects:

- Data objects, which are defined by an application
- Certificate objects, which represent digital certificates such as X.509
- Key objects, which can be public, private or secret cryptographic keys

Each object in the system is comprised of a number of *attributes*. These attributes describe the actual object as well as the *access policy* for that object. For example, each object may be classified as *public* or *private*; this classification determines who may access the object. A *public object* is visible to any user (or application), whereas a *private object* is only visible once the user is authenticated to the token where that object is stored.

For a complete description of the object attributes, see "[PKCS #11 Attributes](#)" on page 168.



Note: It is recommended that the number of objects stored in any single token be less than 1000, and that the number of objects stored on the entire HSM be less than 2000.

Administration Objects

In addition to the object classes defined within PKCS #11, SafeNet ProtectToolkit-C introduces a new set of objects known as *administration objects*.

The administration objects represent the hardware and contain HSM configuration settings. They can be queried by the application and some can be modified by an administrator. The default administration objects are automatically created when SafeNet ProtectToolkit-C initializes.

The administration objects reside on a special token referred to as the *Admin token*. This token has a fixed security policy. The *Admin token* resides only in the *Admin slot* on the HSM.

User Roles

As part of the SafeNet ProtectToolkit-C configuration process, different *user roles* are assigned to those responsible for the application's administration and use.

For SafeNet ProtectToolkit-C there are four defined roles available. These are:

- Security Officer (SO)
- Token Owner or User
- Administration Security Officer (ASO)

- Administrator

Standard PKCS #11 defines the first two of these, the *Security Officer* (SO) and the *Token Owner* or *User*. Each slot and its associated token will have an SO and a User, each with their own respective PINs.

- A Security Officer grants and revokes access to a token and assists with key backups
- A Token Owner uses the token for the application

Two additional roles are defined that are only available on the Admin token. The holders of these roles handle HSM-level administration and management. These are the *Administration Security Officer* (ASO) and the *Administrator*. These roles effectively mirror their standard PKCS #11 counterparts.



Note: The services available to the various roles depend on the security policy set for the HSM. For a complete description of these roles and the services available to each of them, please see ["Security Policies and User Roles" on page 68](#).

PINs and Passwords

PINs and passwords are used to authenticate users and to provide access to secured computer systems. In Cryptoki and SafeNet ProtectToolkit-C, they are defined as variable-length strings of characters selected from the ANSI C character set. They are case sensitive, and must be between 1 and 32 characters in length.



Note: The term *password* is not defined as something distinct from a *PIN* in Cryptoki environments. You will find the terms used interchangeably in Cryptoki-related documentation.

PIN Retry Delay

A brute-force search of PINs can be stopped using two approaches:

1. Prevent logging in after a certain number of PIN failures.
2. Enforce a time-delay between login attempts after a certain number of PIN failures.

The time-delay approach is used for SafeNet ProtectToolkit-C implementations utilizing the SafeNet ProtectServer Network HSM.

After the third failed PIN presentation, the device imposes a delay (lengthening in increments of 5 seconds) before the next presented PIN is checked:

- third failed attempt = delay of 5 seconds
- fourth failed attempt = delay of 10 seconds
- fifth failed attempt = delay of 15 seconds
- etc.

If a PIN presentation occurs before the delay period has expired, the attempt fails with an error indicating that the PIN is locked.

Initial Configuration

In this section, it is assumed that:

- SafeNet ProtectToolkit-C has been successfully installed on your system

- you can access the SafeNet ProtectToolkit-C utilities used to carry out configuration tasks, as described in ["Installation and Configuration" on page 14](#).

Setting the Admin Token PINs

Following an initial installation or tamper event, it is necessary to introduce the Administrator and Administrator SO user roles by setting their initial PINs. This is done using the **ctconf** utility.

1. From a command prompt, type **ctconf** and press **ENTER**.
You are prompted to set the Administration Security Officer (ASO) PIN.
2. Enter the ASO PIN and press **Enter**. Enter this same PIN again for confirmation.



Note: PIN characters or asterisks (*) do not appear on screen while the PIN is being typed. For details of what constitutes a valid PIN see ["PINs and Passwords" on the previous page](#).

You are prompted to set the Administrator PIN.

3. Enter the Administrator PIN and press **Enter**. Enter this same PIN again for confirmation.
4. On board each HSM is a Real Time Clock (RTC). If the RTC is out of synchronization with the host system clock, you are then prompted to allow synchronization. To synchronize the RTC to the host system clock type **Y** and then **Enter**. Otherwise, type **N** to abort.

After successful completion of the above, HSM configuration details are displayed. For example:

```
Current Adapter Configuration for Device 0:
Model           : PSI-E2:PL220
Serial Number   : 518687
Adapter Clock    : 30/08/2016 15:07:04 (-5:00+DST)
Battery Status  : GOOD
Security Mode    : Default (No flags set)
Transport Mode   : None
FM Support       : Enabled
FM Status        : No FM downloaded yet
Open Session Count: 0
Number of Slots  : 1
RTC Adjustment Access Control: Disabled
```

Following this, this message is displayed:

```
PLEASE NOTE that the firmware allows FMs to be downloaded; but the "Tamper
before upgrade" security flag is not set. To protect existing keys against a
possible threat of a rogue FM, this flag should be set (using 'ctconf -ft')
```



Note: An *FM* is a *functionality module*. For more information see ["Installing a Functionality Module" on page 89](#).

Finally, the utility closes and the operating system command prompt returns.

Selecting and Setting a Security Policy

A security policy is a set of security settings that control how SafeNet ProtectToolkit-C is allowed to function. Setting the security policy is the most important part of SafeNet ProtectToolkit-C initial configuration.

A number of security settings offered by SafeNet ProtectToolkit-C can be used to implement *typical security policies* that meet certain standards or satisfy application integration requirements. Alternatively, custom security policies can be implemented.

Refer to ["Security Policies and User Roles" on page 68](#) for full details and instructions.

Setting up Slots

The Administrator will have to decide on the number of slots required for the particular environment. In its default initial configuration, SafeNet ProtectToolkit-C will have one User slot, one Admin slot and one slot for each connected smart card reader.

As a general guide, the Administrator should create as many slots as there are applications, or users, that will want to perform PKCS #11 processing. This configuration allows for individual applications to be completely separate from each other.

For more information on the types of slots and tokens, see ["Slots and Tokens" on page 37](#).

To create new user slots, use the **ctconf** utility with the **-c** switch.

Example:

```
ctconf -c2
```

Since only the Administrator is authorized to create new slots, you will be prompted for the Administrator PIN.

This command will create two new User slots, each with an associated token. To check that the slots were created, use the **ctstat** utility, which will report information on all current slots and tokens.

Slots are numbered consecutively, with the last or highest slot number always being the Admin slot.

Example:

The current configuration is:

User	User	Admin
Slot 0	Slot 1	Slot 2

If two slots are added, the configuration becomes:

User	User	User	User	Admin
Slot 0	Slot 1	Slot 2	Slot 3	Slot 4

Multiple Adapter HSMs

When multiple HSM adapters (such as the SafeNet ProtectServer PCIe HSM) are installed in a single machine, there will be multiple Admin slots - one per HSM. The slots for the second HSM will appear directly following the slots for the first HSM. Thus if two HSMs were installed with their default configuration, slot 0 and slot 2 would be user slots, slots 1 and 3 would be the Admin slots for the first and second HSM respectively.

Example:

HSM 1		HSM 2	
User	Admin	User	Admin
Slot 0	Slot 1	Slot 2	Slot 3

Token Initialization

After creating a slot within SafeNet ProtectToolkit-C, the token within that slot must be initialized so it may be used by an application. This initialization will assign a label and set up the Security Officer and User PINs for that token. In addition to initialization of User slots, this procedure is also applicable to any smart card tokens used with SafeNet ProtectToolkit-C.

The party responsible for token initialization depends on the Security Policy that has been set for the adapter. In the case where 'clear PINs' are allowed, any user taking on the role as that token's Security Officer can perform the token initialization. In the case where 'clear PINs' are not allowed, only the Administrator can perform the token initialization. For more information on Security Policies, see ["Security Policies and User Roles" on page 68](#).

The **ctconf** utility is used by the Administrator to initialize a token on a particular slot. Once a token is initialized, it may only be reinitialized, or reset, by the token SO, using the **ctkmu** or **ctconf** utility.

Example:

```
ctconf -n1
```

This example will initialize token 1 in slot 1.

ctconf will prompt for the token label to be entered, followed by the token SO PIN.



Note: A token initialization will destroy all objects on that token. This is an important consideration when reinitializing a token that has already been used.

Following initialization of a token, the token SO should change the PIN set by the Administrator with the **ctkmu** utility. Instructions are provided in ["Changing a User or Security Officer PIN " on page 80](#).

Next, the token SO must initialize the token user PIN using the **ctkmu** utility.

Example:

```
ctkmu p -s2
```

This example will initialize the user PIN on token 2. The SO PIN will be prompted for, followed by a prompt for the new User PIN.

Once both the SO and User PINs have been selected, the token is ready for use with an application. The User is advised to change his/her PIN from the one the SO assigned by repeating the above command.

Trust Management

When secure data or keys must be transferred from one HSM to another through the process of token replication, trust management is required. Environments using Work Load Distribution (WLD) and High Availability (HA) are one

example. Refer to ["Work Load Distribution Model and High Availability" on page 49](#) for details.

Currently, trust management is supported by SafeNet ProtectServer PCIe HSMs and SafeNet ProtectServer Network HSMs.

When a WLD system is configured, tokens must be replicated across all the HSM User slots associated with a common WLD virtual slot. It is essential that the token is deemed trustworthy before it is imported by the HSM; the token must come from a trustworthy source, and remain unaltered during transmission.

Public-key cryptography establishes trust between HSMs. Private keys are used for signing extracted information and unwrapping tokens. Public keys are used for wrapping tokens and verifying signed information. An RSA key-pair must be generated on the administrative token of each device. This key-pair is referred to as the *local HSM Identity Key-Pair*. The public half of the key-pair is termed the *HSM Identity Public-Key*, while the private portion is called the *HSM Identity Private-Key*. An HSM trusts another HSM when it holds the other's HSM Identity Public-Key in its administrative token. This is referred to as the *peer HSM Identity Public-Key*. ["Simple trust relationships" below](#) shows an example of a system where simple trust relationships have been established between HSMs.

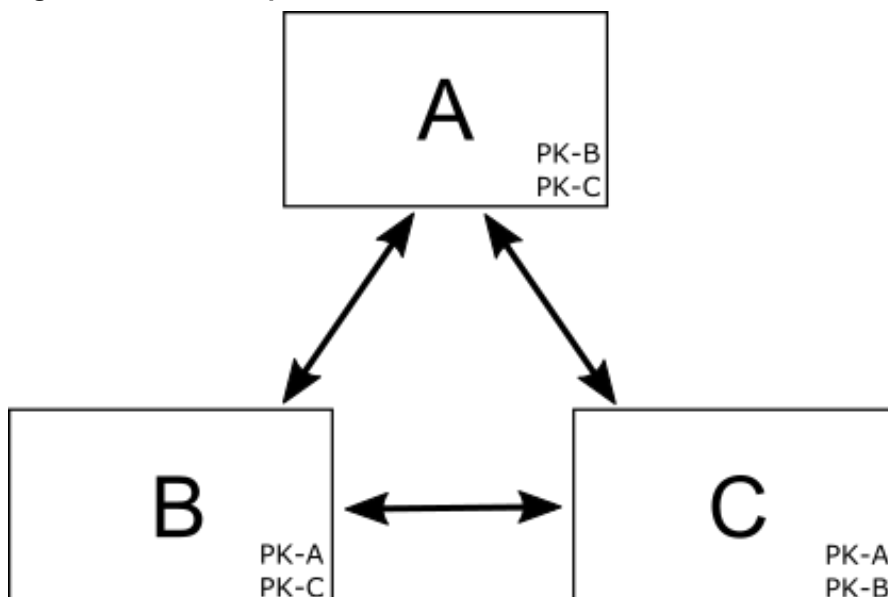
The arrows indicate the trust relationship. In this system, HSM A trusts HSM B. That is, HSM A holds the HSM Identity Public-Key of HSM B in its administrative token. However, HSM B does not trust HSM A. HSM B and HSM C share a relationship of mutual trust. In this system, token replication could only be performed between HSM B and HSM C (with either device originating the tokens), as token replication requires a relationship of mutual trust.

Figure 1: Simple trust relationships



["Relationships of mutual trust" below](#) shows a system where every HSM shares a relationship of mutual trust with every other HSM. In this scenario, token replication can be performed from any HSM to any other HSM on the system.

Figure 2: Relationships of mutual trust

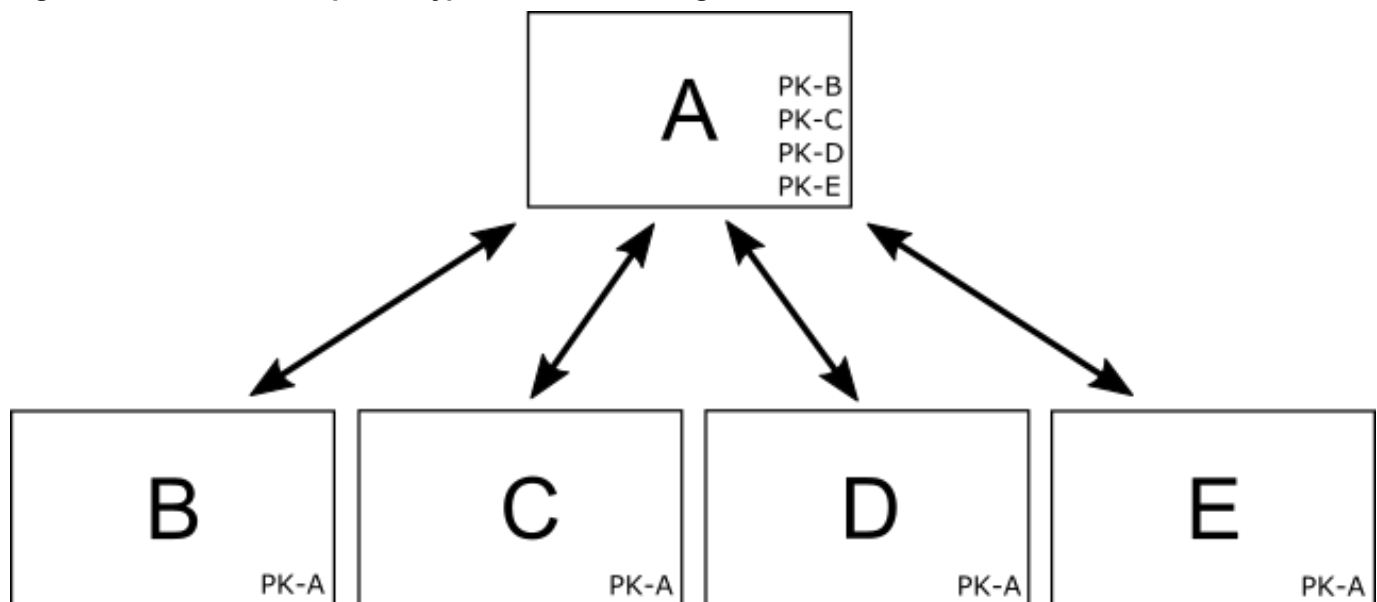


Typically, when token replication is performed in a WLD configuration, an HSM is selected to hold the master tokens and tokens are then replicated to the other HSMs.

"Trust relationships in a typical WLD/HA configuration" below illustrates a system in a typical WLD configuration. In this system, HSM A has been selected to hold the master tokens.

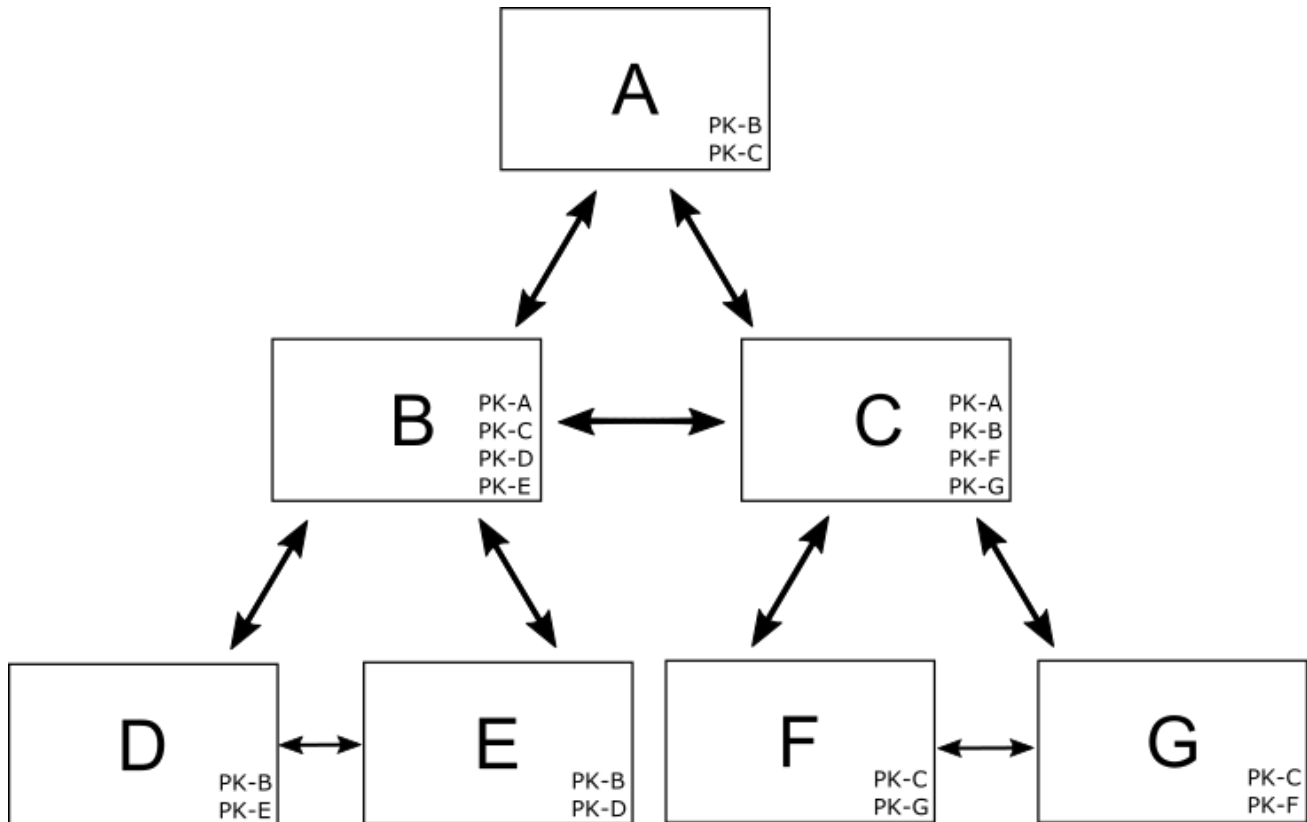
The arrows indicate the relationships of mutual trust between HSM A and the other HSMs that are necessary for token replication to be performed. The figure also illustrates that it is not necessary to establish trust among the HSMs that the tokens are replicated to, in other words, no trust need be established among HSM B, HSM C, HSM D and HSM E.

Figure 3: Trust relationships in a typical WLD/HA configuration



Complex trust topologies can be configured depending upon system and administrative requirements. "Complex trust topology" on the next page illustrates an example of a complex trust topology.

Figure 4: Complex trust topology



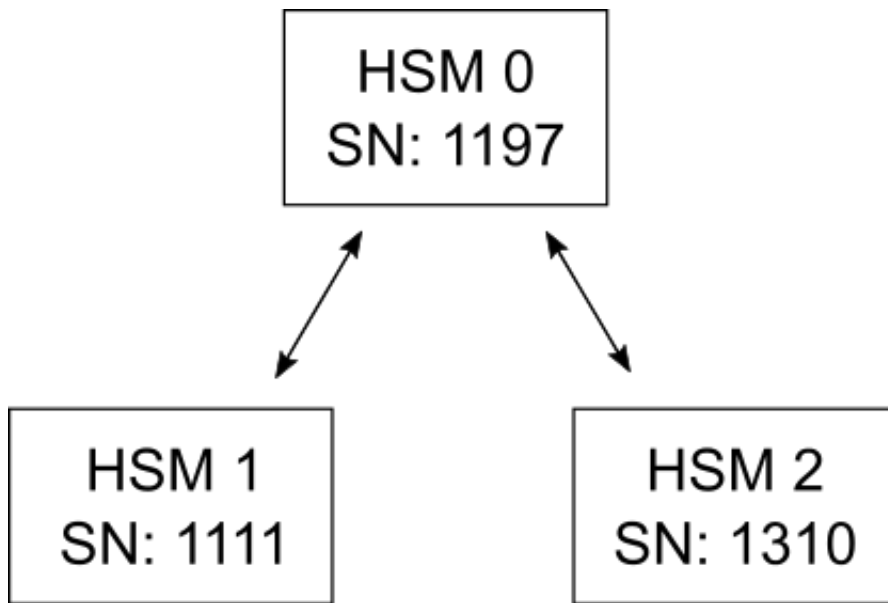
The **ctident** utility provides the mechanism for establishing, maintaining and removing trust relationships on HSMs. In an offline environment, the **ctkm** utility can be used to import and export the HSM Identity Public-Keys.

The example in ["Establishing Trust Relationships" below](#) shows how to establish trust among HSMs. The **ctident** utility can be used to display, check, and remove trust relationships. It can also be used to rollover the HSM identity keys used in trust management.

Establishing Trust Relationships

The following example describes how to set-up the trust relationships illustrated in ["Establishing trust relationships example configuration" on the next page](#). In this system HSM 0 shares a mutual trust relationship with both HSM 1 and HSM 2. No trust is established between HSM 1 and HSM 2. This is a typical configuration used for token replication, where the master tokens are located on HSM 0. The abbreviation SN in the figure refers to the serial number of the admin token on each device. The serial numbers are used in the example to identify the HSM device.

Figure 1: Establishing trust relationships example configuration



Configuration

1. Generate a list of all the slots on the system to find the admin tokens' serial numbers.

Use the **ctkm** utility. Each device on the system is assigned a slot number in the order: User slots, Smart card slots, Administration slot. The admin token's serial number is listed in brackets after "AdminToken". For example:

```

C:\>ctkm u 1
ProtectToolkit C Key Management Utility 5.3.0
Copyright (c) SafeNet, Inc. 2009-2016

Cryptoki Version = 2.20
Manufacturer      = SafeNet, Inc.
WLD_Slot_11      (Slot 0)
WLD_Slot_22      (Slot 1)
WLD_Slot_33      (Slot 2)
AdminToken (1197) (Slot 3)
<uninitialized token> (Slot 4)
<uninitialized token> (Slot 5)
<uninitialized token> (Slot 6)
AdminToken (1111) (Slot 7)
<uninitialized token> (Slot 8)
<uninitialized token> (Slot 9)
<uninitialized token> (Slot 10)
AdminToken (1310) (Slot 11)
  
```

2. Generate the HSM Identity Key-Pair on each device.

You must generate an HSM Identity Key-Pair on each participating device in a trust relationship. Use the **ctident** utility with the **gen** command and appropriate parameters. The Administration Token SO pin for each device will be prompted for. In a system where all HSMs are to participate in a trust relationship, use the **ctident gen all** command. Alternatively, specify the devices participating in token replication by their serial number.

Example:

```
C:\>ctident gen sn:1197,sn:1111,sn:1310
```

The **ctident gen** command also allows devices to be specified by device positional number. The device positional numbers are dynamically assigned when the command is invoked. If a device goes offline at the moment the command is invoked, the positional device number will move. This could result in incorrect trust relationships being established. The use of device serial numbers is **STRONGLY** recommended to avoid problems with positional device number reassignment.

3. Command the destination devices to trust the master device.

The HSM Identity public key of HSM 0 must be shared to HSM 1 and HSM 2 by using the **ctident trust** command. The first parameter specifies the device to be trusted, while the second parameter is the list of devices that are to trust the first. The Administration Token SO pin for each device must be entered.

Example:

```
C:\>ctident trust sn:1197 sn:1111,sn:1310
```

4. Command the master device to trust the destination devices.

The HSM Identity public-keys of HSM 1 and HSM 2 must be shared to HSM 0 by using the **ctident trust** command again. In the example below, the first command line shares HSM 1's public key with HSM 0. The second command line shares HSM 2's public key with HSM 0. The Administration Token SO PIN for each device must be entered.

Example:

```
C:\>ctident trust sn:1111 sn:1197
C:\>ctident trust sn:1310 sn:1197
```

Token Replication

This process replicates tokens across one or more HSMs, required for a system operating in WLD mode. Token replication is not a suitable mechanism to use in place of token export.

Token replication can only be performed on User Tokens (Smart card and Administration Slots are not supported). Refer to "[The SafeNet ProtectToolkit-C Model](#)" on page 36 for a description of slot types. Tokens on any User slot can be replicated to any other User slot, on the same HSM or any other in the system. During token replication, all the objects contained within the master token and the master token label are replicated. When a system is operating in WLD mode, the token label identifies its associated virtual WLD slot. Refer to "[Work Load Distribution Model and High Availability](#)" on page 49 for more information.

Once a token has been replicated, any objects created or modified on that token *will not* be automatically transferred to the replicated tokens. If a token is modified, and token consistency is required, the token replication process must be repeated.



Note: WLD requires token consistency, so whenever a token is modified, manual replication to all participating WLD tokens is mandatory.

Replicate tokens by using the **ctkm** utility with the **rt** command. Refer to "[CTKM](#)" on page 114 for more information. The token in the master slot and the replicated tokens must have the same SO and User PINs. When replicating to an uninitialized token, the token's SO PIN is required. If the **No Clear PINs** flag is set, the User PIN for the importing device's Administration token is also required. Refer to "[Security Flag Descriptions](#)" on page 73 for more information.

The **ctkm rt** command uses slot positional numbers to identify the master and destination slots. The slot positional numbers are dynamically assigned when the command is invoked. If a device goes offline at the moment the command is invoked, the positional device number will be reassigned. This could result in the token being replicated to an incorrect slot. The system should be stable when using this command.

The following examples show how to replicate a token from:

1. the first slot on HSM 0 (slot 0) to the second slot on HSM 1 (slot 5) and the second slot on HSM 2 (slot 9)
2. the second slot on HSM 0 (slot 1) to the first slot on HSM 1 (slot 4) and the third slot on HSM 2 (slot 10)

Example 1: Master Tokens Replicated to a Single Slot or List of Slots

This example illustrates replication to a single token and to a list of tokens. This method is recommended for initial configuration.

1. Generate a list of all the slots on the system to find their positional numbers.

Use the **ctkm** utility. Refer to "[CTKM](#)" on page 114 for more information. For each device, slot positions are assigned in the following order: User slots, Smart card slots, Administration slot. For each slot, the token label is displayed followed by the slot positional number. In the example below, HSM 0 contains 3 User slots, configured with the token labels "WLD_Slot_11" (Slot 0), "WLD_Slot_22" (Slot 1), and "WLD_Slot_33" (Slot 2). These are followed by the Administration slot (Slot 3) with serial number 1197. HSM 1 and HSM 2 each contain 3 slots with uninitialized tokens, followed by the Administration slot. The slot positional number is used to identify the tokens during replication in the next step.

Example:

```
C:\>ctkm 1
ProtectToolkit C Key Management Utility 5.3.0
Copyright (c) SafeNet, Inc. 2009-2016

Cryptoki Version   = 2.20
Manufacturer       = SafeNet, Inc.
WLD_Slot_11        (Slot 0)
WLD_Slot_22        (Slot 1)
WLD_Slot_33        (Slot 2)
AdminToken (1197)   (Slot 3)
<uninitialized token> (Slot 4)
<uninitialized token> (Slot 5)
<uninitialized token> (Slot 6)
AdminToken (1111)   (Slot 7)
<uninitialized token> (Slot 8)
<uninitialized token> (Slot 9)
<uninitialized token> (Slot 10)
AdminToken (1310)   (Slot 11)
```

2. Replicate the token.

Use the **ctkm** utility with the **rt** command with two parameters: the slot exporting the token and the list of slots receiving the token (see "[CTKM](#)" on page 114). The exporting slot must have the same SO PIN and User PIN as the receiving slots. When replicating to an uninitialized token, the exporting slot's SO PIN must be entered. If the **No Clear PINs** flag is set, the User PIN for the receiving device's Administration token is also required. Refer to "[Security Flag Descriptions](#)" on page 73 for more information.

Examples:

Replicate token from slot 0 to slot 5

```
C:\>ctkm rt -s 0 -d 5
```

Replicate token from slot 0 to slot 9

```
C:\>ctkm rt -s 0 -d 9
```


Replicate token from slot 1 to slot 4 and slot 10

```
C:\>ctkmu rt -s 1 -d 4,10
```

Alternative 2 – Token Replicated to Many Tokens

The following example illustrates token replication from a master token to many tokens. This method permits tokens to be replicated to other tokens that share the same token label. This method can be used to update token after the master token has been modified. This example illustrates the same configuration as in the example above.

1. Generate a list of all the slots on the system to find their positional numbers.

For this method, the receiving slot's token label must be the same as the exporting token. In this example, the tokens in HSM 1 and HSM 2 must be initialized with the appropriate token labels. That is, slot 5 and slot 9 must be initialized with the same token label as slot 0 and slot 4 and slot 10 must be initialized with the same token label as slot 1. Refer to ["Token Initialization" on page 42](#) for further details.

Example:

```
C:\>ctkmu l
ProtectToolkit C Key Management Utility 5.3.0
Copyright (c) SafeNet, Inc. 2009-2016

Cryptoki Version   = 2.20
Manufacturer       = SafeNet, Inc.
WLD_Slot_11        (Slot 0)
WLD_Slot_22        (Slot 1)
WLD_Slot_33        (Slot 2)
AdminToken (1197)   (Slot 3)
WLD_Slot_22        (Slot 4)
WLD_Slot_11        (Slot 5)
<uninitialized token> (Slot 6)
AdminToken (1111)   (Slot 7)
<uninitialized token> (Slot 8)
WLD_Slot_11        (Slot 9)
WLD_Slot_22        (Slot 10)
AdminToken (1310)   (Slot 11)
```

2. Replicate the tokens.

Use the **ctkmu** utility with the **rt** command to replicate tokens. When using the **all** command line parameter, the master token is replicated to all tokens on the system that share the same token label.

Example:

```
C:\>ctkmu rt -s0 -d all
C:\>ctkmu rt -s1 -d all
```

Work Load Distribution Model and High Availability

There is no restriction on the number of HSMs working together in a system. High scalability, availability, reliability and increased throughput are the result. The built-in configurable WLD mode can relieve the application of its own load sharing processing, allowing it to focus on its primary tasks. A high availability/load balancing setup reliably boosts overall performance.

Work Load Distribution (WLD)

In a Load Distribution design approach, work is balanced across a system by transferring units of work between processing modules. The demand placed on any particular module is thereby reduced. A well-balanced system results in an increase in the overall throughput of processing tasks.

There are a number of integral components within a system which deploys load distribution. In a SafeNet system, the load distribution scheme is called Work Load Distribution. Within SafeNet ProtectToolkit-C, a distribution engine portions work requests and distributes them among HSMs according to a distribution scheme. The tokens used within the scheme must be replicated across the HSMs, according to the system design. A good system design should address throughput requirements, resource portioning and fault tolerance/disaster recovery. The **ctident** utility establishes trust between HSMs that share tokens. The **ctkmu** utility replicates a token once trust has been established.

High Availability (HA)

Enterprises must maintain their services and keep them reliably up and running. By providing redundancy and availability in services, High Availability (HA) is critical to security.

The HA feature keeps track of the commands sent to a session. In case of session failure, SafeNet ProtectToolkit-C will re-establish a new session by replaying these commands. This is the best approach to achieve transparent fail-over. The HA feature requires the support of the WLD system to manage failed HSMs and allocate new sessions to them.

SafeNet ProtectToolkit-C Configuration

To enable Work Load Distribution/High Availability, SafeNet ProtectToolkit-C must be configured to operate in WLD or HA mode. Refer to ["Operation in WLD Mode" on page 56](#) and ["Operation in HA Mode" on page 57](#) for details on each.

When applications use the SafeNet ProtectToolkit-C interface in WLD/HA mode, the system of physical HSMs appears as a single virtual HSM. SafeNet ProtectToolkit-C uses virtual WLD slots to achieve this. To use a WLD slot, applications use the standard PKCS #11 function calls. The distribution engine distributes the session over the physical HSM slots associated with the WLD slot (see ["WLD System Setup" on the next page](#)).

WLD Slots

A WLD Slot is a virtual PKCS #11 slot. Associated with this slot may be several (but at least one) 'real' HSM slots, possibly located across multiple devices. Each WLD slot must be configured by the user (see ["Configuring WLD Slots" on page 55](#)). *For a physical HSM slot to be associated with a WLD slot, it must share the same token label as the WLD slot. Each WLD slot token label must be unique.* The distribution engine uses the token label for determining the underlying physical HSM slots on which to share workload.



Note: The HA system cannot support more than 16 slots. The Administrator must limit the WLD slot numbers to be 16 or fewer (from 00 to 15 inclusive).

Distribution Scheme

The distribution of application requests is performed on a per-session basis. When an application opens a session to a WLD slot, the distribution engine selects the initial physical HSM slot to service the open session request, according to the distribution scheme. Once the session has been opened, all other requests performed on that session are routed to

the initial physical HSM slot. When an application opens subsequent sessions, the distribution engine randomly selects a physical HSM slot from those with the least number of sessions.

As multiple applications may be using the distribution engine, the scheme ensures that slots are not 'victimized' because of their position in the scheme. For example, if multiple applications are started one at a time, and each application requests a single session on the same WLD slot, randomization will ensure an even distribution of sessions across the available physical HSM slots.

Token Replication

SafeNet ProtectToolkit-C supports replication of token information in a protected form to other SafeNet HSMs. The **ctident** utility is used to establish trust between HSMs that share tokens. The **ctkmu** utility is used to replicate a token once trust has been established. See ["Trust Management" on page 42](#) and ["Token Replication" on page 47](#) for more information.

Token replication must be performed by the user at configuration time. The WLD model works on a static configuration.



CAUTION: The tokens in WLD must always be consistent. The distribution engine does not check or ensure that the physical HSM tokens associated with a particular WLD token are consistent. If the state of the tokens is inconsistent or incorrect, inappropriate keys could be used. This could occur without notice and without incident.

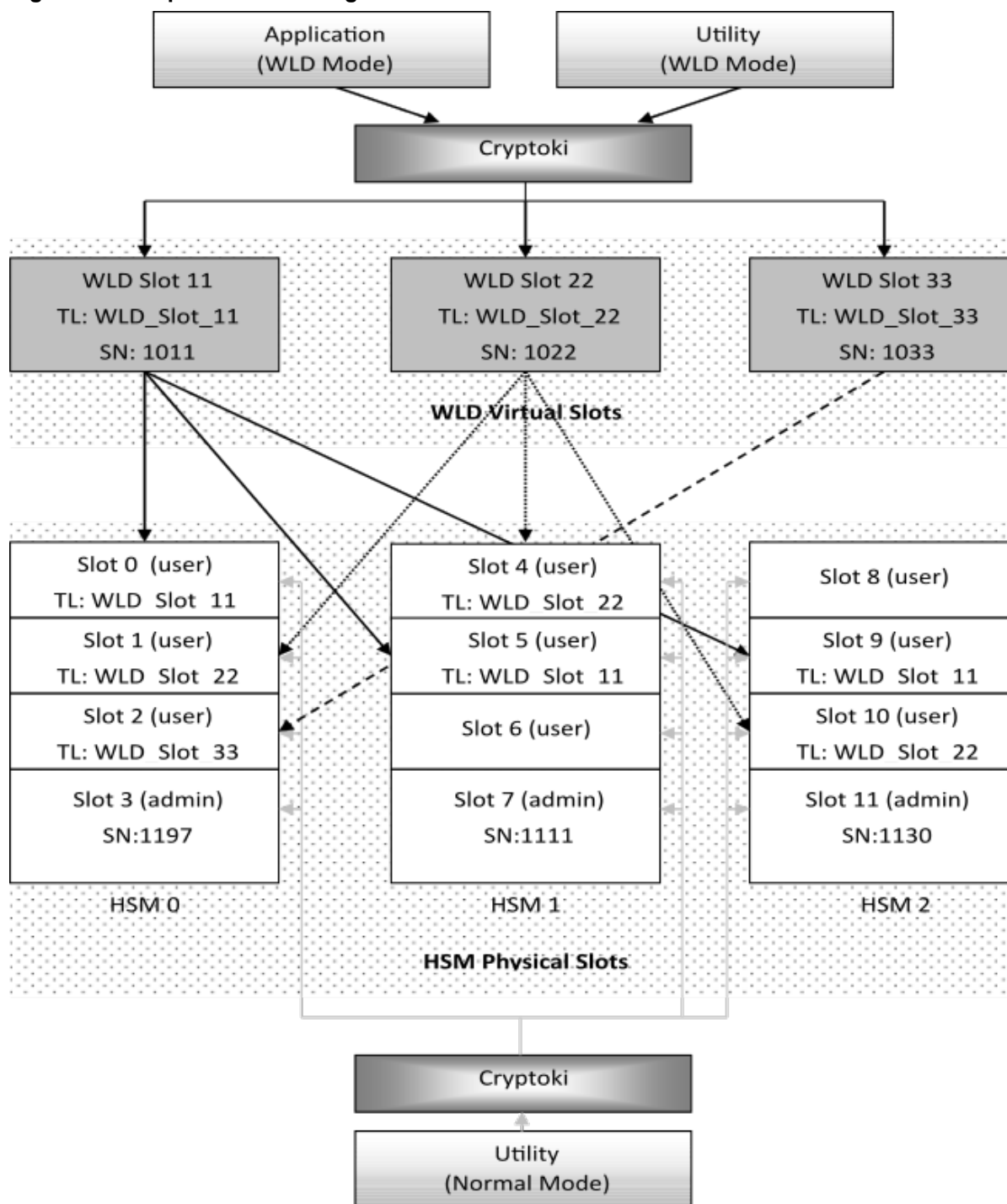
WLD System Setup

This section provides instructions on how to set up a system for Work Load Distribution. The example system contains 3 remote HSMs and 3 virtual WLD slots with SafeNet ProtectToolkit-C running on a Windows platform.

A diagram of the resulting configuration is shown in ["Example of WLD configuration" on the next page](#). To any application or utility operating in WLD mode, the system of physical HSMs appears as a single virtual HSM that is accessible via virtual WLD slots. Any application or utility that accesses the system does so through the Cryptoki library. When an application or utility is configured to operate in WLD mode, the WLD virtual slots are the only slots made accessible by the Cryptoki Library. An application or utility configured to operate in WLD mode cannot access the HSM slots directly.

The arrows represent associations between the virtual WLD slots and the physical HSM slots in this configuration. For example, WLD Slot 11 is associated with User Slot 0 on HSM 0, User Slot 5 on HSM 1 and User Slot 9 on HSM 2.

Figure 1: Example of WLD configuration



WLD Slot	Associated HSM User Slots	Token Label
WLD Slot 11	Slot 0 (HSM 0) Slot 5 (HSM 1) Slot 9 (HSM 2)	WLD_Slot_11

WLD Slot	Associated HSM User Slots	Token Label
WLD Slot 22	Slot 1 (HSM 0) Slot 4 (HSM 1) Slot 10 (HSM 2)	WLD_Slot_22
WLD Slot 33	Slot 2 (HSM 0)	WLD_Slot_33

As illustrated in ["Example of WLD configuration" on the previous page](#), each WLD slot shares the same token label (TL) as the HSM slots that are associated with it. For example, WLD Slot 22 shares the token label WLD_Slot_22 with its associated HSM User slots (1, 4, and 10).

You must know the Admin Token serial numbers (SN) when configuring the system for WLD operation. Each WLD slot must be configured with a unique serial number allocated by the user.

During configuration, the utilities must be able to access the HSM slots directly. They are initially configured to operate in NORMAL mode, as shown by the boxes at the bottom of the figure. After configuration, applications and utilities that need to access the system in WLD mode must be configured to operate in WLD mode.

Configuration

1. Establish Network Communication.

Set the environment variable ET_HSM_NETCLIENT_SERVERLIST with a list of the IP addresses of the HSMs in the order HSM0, HSM1, HSM2. IPv6 addresses must be enclosed in square brackets. See ["Specifying the Network Server\(s\)" on page 33](#) for more information.

2. Set the Library Mode to NORMAL.

The HSM slots must be accessible to set up the system, so the utilities which access them must operate in NORMAL mode. See ["Operation in WLD Mode" on page 56](#) for more on setting the Cryptoki Library to NORMAL mode.

3. Initialize Admin Tokens and Security Policy.

If an HSM has not been initialized, the Admin Token and Security Policy for each HSM must be configured. Refer to ["Initial Configuration" on page 39](#) for further details.

4. Create User Slots.

Create User slots for each HSM, as described below. Refer to ["Initial Configuration" on page 39](#) for further details.

User Slots	HSM
Slot 0 Slot 1 Slot 2	0
Slot 4 Slot 5 Slot 6	1
Slot 8 Slot 9 Slot 10	2

5. Create Master Tokens.

In this example, the master tokens are created on HSM 0 and replicated to HSM 1 and HSM 2. The master tokens could be created on any HSM User slot that is associated with the WLD slot and then replicated to the other HSMs. As HSM 0 has slots associated with all the WLD slots used in this example, it was selected as the HSM to hold the master tokens.

Configure the tokens for each of the slots, according to the following table. Refer to ["Configuring WLD Slots" on the next page](#) for further details.

HSM 0 User Slot	Token Label
Slot 0	WLD_Slot_11
Slot 1	WLD_Slot_22
Slot 2	WLD_Slot_33

6. Create Keys, Certificates, Data, HW Objects on Master Tokens.

It is necessary to create any objects that are contained within the master tokens before the token is replicated. Refer to ["Token Replication" on page 47](#) for further details.

7. Establish Trust.

For token replication to be performed from the HSM holding the master tokens to another HSM, the HSMs must have a mutual trust relationship. Refer to ["Trust Management" on page 42](#) for further details.

As the master tokens are located on HSM 0 and are to be duplicated to HSM 1 and HSM 2, establish mutual trust relationships between

- HSM 0 and HSM 1
- HSM 0 and HSM 2

8. Replicate Tokens.

Once trust is established the tokens can be replicated. Refer to ["Token Replication" on page 47](#) for further details. Replicate the master tokens from HSM 0 to HSM 1 and HSM 2 as follows:

Master Token	Replication
WLD_Slot_11	Replicate token from User slot 0 (HSM 0) to User slot 5 (HSM 1)
	Replicate token from User slot 0 (HSM 0) to User slot 9 (HSM 2)
WLD_Slot_22	Replicate token from User slot 1 (HSM 0) to User slot 4 (HSM 1)
	Replicate token from User slot 1 (HSM 0) to User slot 10 (HSM 2)

9. Configure WLD Slots.

WLD slots are configured via environment variables at either the temporary, user or system level. Refer to ["Configuring WLD Slots" on the next page](#) for further details. In this example, WLD slots are configured at the system level:

- a. Locate the registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\SafeNet\PTKC\WLD

- b. Make the following assignments:

Variable	Assignment
ET_PTKC_WLD_SLOT_11	WLD_Slot_11,1011,WLD Slot: 11
ET_PTKC_WLD_SLOT_22	WLD_Slot_22,1022,WLD Slot: 22
ET_PTKC_WLD_SLOT_33	WLD_Slot_33,1033,WLD Slot :33

10. Set the Library Mode to WLD.

WLD mode is configured via an environment variable at either the temporary, user or system level. To any application or utility operating in WLD mode, the HSM system appears as a single virtual HSM with a collection of WLD virtual slots. The HSM physical slots are not accessible to applications or utilities operating in WLD mode (see ["Operation in WLD Mode" on the next page](#)).

11. Check the WLD Slot Configuration.

Run the **ctkm** (*WLD mode*) utility to view the slots available on the system. Only the WLD virtual slots should be visible. Any HSM physical slot on the system which has not been associated to a WLD virtual slot will no longer be accessible.

Example:

```
ProtectToolkit C Key Management Utility 5.3.0
Copyright (c) Safenet, Inc. 2009-2016
```

```
Cryptoki Version   = 2.20
Manufacturer       = Safenet, Inc.
WLD_Slot_11        (Slot 11)
WLD_Slot_22        (Slot 22)
WLD_Slot_33        (Slot 33)
```

Configuring WLD Slots

To operate SafeNet ProtectToolkit-C in WLD Mode, virtual WLD slots must be configured.

Configuration parameters for the WLD slots are specified by environment variables in the format ET_PTKC_WLD_SLOT_*n*. An environment variable must be configured for each WLD slot.

In the ET_PTKC_WLD_SLOT_*n* environment variable, *n* defines the Slot Number, an integer in the range 0 to 99. Slot Numbers allocated within an application must be unique.

The format of these variables is:

```
<WLDTokenLabel>[, [<WLDTokenSerial#>][, <WLDSlotDescription>]]
```

Where:

<WLDTokenLabel>	is mandatory. The PKCS #11 Token Label for this WLD Token identifies the HSM Tokens to be used for WLD. The <WLDTokenLabel> should be unique in the complete list of WLD Slot Configurations.
<WLDTokenSerial#>	is optional. You can assign any PKCS #11 Token Serial Number you wish to this WLD Token. The default value is the same as the value of <i>n</i> in the configuration variable name.
<WLDSlotDescription>	is optional. You can assign any PKCS #11 Slot Description you wish for this WLD Slot. The default value is "WLD Slot: <i>n</i> ", where <i>n</i> is the same as the value of <i>n</i> in the configuration variable name.

The example below shows a conceptual configuration for three virtual slots. The entire list of WLD Slots will be visible by any application that is using this WLD configuration.

UNIX

Under UNIX variants, the variable name and value are stored in the file **et_ptkc** in the directory **/etc/default** (for system configuration) and/or **\$HOME/.safenet** (for user configuration).

Example:

To configure WLD slots at the system level:

- Open the file: **/etc/default/et_ptkc**
- Make the following entries:
 ET_PTKC_WLD_SLOT_0=WLD Token 0,1002,PIN generation slot
 ET_PTKC_WLD_SLOT_5=WLD Token 5
 ET_PTKC_WLD_SLOT_6= WLD Token 6,,Password generation slot



Note: For WLD Slot 5, SafeNet ProtectToolkit-C will assign the default PKCS #11 Token Serial Number of 5, and the PKCS #11 Slot Description "WLD Slot:5". For WLD Slot 6, the default PKCS #11 Token Serial Number of 6 will be assigned.

Windows

Under Win32 and Win64, the variable name and value are stored in the HKLM (for system configuration) and/or HKCU (for user configuration) registry, in the key **SOFTWARE\SafeNet\PTKC\WLD**.

Example:

To configure WLD slots at the system level:

1. Locate the registry key:
HKEY_LOCAL_MACHINE\SOFTWARE\SafeNet\PTKC\WLD
2. Assign the ET_PTKC_WLD_SLOT_*n* variables the values shown in the UNIX example above.

Operation in WLD Mode

You must configure the Cryptoki Library to operate SafeNet ProtectToolkit-C in WLD mode.

The environment variable ET_PTKC_GENERAL_LIBRARY_MODE specifies the Cryptoki Library operating mode. This variable controls which PKCS #11 model is applied to slot and token usage (see ["Work Load Distribution Model and High Availability" on page 49](#)).

Valid values for this variable are NORMAL or WLD or HA. If this variable is not defined, or contains an invalid value, then SafeNet ProtectToolkit-C will operate in NORMAL PKCS #11 mode.

The HSM system appears to any application or utility operating in WLD mode as a collection of WLD virtual slots. The HSM physical slots are not accessible to applications or utilities operating in WLD mode.

While configuring the system, it is useful to configure WLD mode with a temporary configuration parameter first by entering **set ET_PTKC_GENERAL_LIBRARY_MODE=WLD** into a command prompt. Then, when configuration is stable, set the environment variable at the user or system configuration level.

It is possible to have some applications running in WLD mode and others running in NORMAL mode on the same platform. In this case, WLD mode will need to be set in both temporary environment variables and at either the user or system level appropriately. For example, if three applications are to operate in WLD mode and one application is to operate in NORMAL mode, then WLD mode should be set at the user or system level and NORMAL mode should be set in an environment variable operating in the context of the application using it.

If any changes need to be made to the system after configuration, the Library mode must be set to NORMAL so that the utilities can access the HSM slots directly.

To configure a basic WLD system across two SafeNet ProtectServer Network HSMs with IP addresses 192.168.1.100 and 192.168.1.101, where the participating tokens are labeled "TokName", set these configuration items (see ["Configuration Items" on page 29](#)):

```
ET_HSM_NETWORK_SERVERLIST=192.168.1.100 192.168.1.101
ET_PTKC_WLD_SLOT_0=TokName
ET_PTKC_GENERAL_LIBRARY_MODE=WLD
```

Operation in HA Mode

To operate SafeNet ProtectToolkit-C in HA Mode, the Cryptoki Library keeps track of the commands sent to a session. In case of session failure, SafeNet ProtectToolkit-C will re-establish a new session by replaying these commands.

SafeNet ProtectToolkit-C provides the following functions in HA mode:

- Detects that a session has terminated because of HSM failure and automatically establishes a new session on a functioning HSM
- After an HSM failure is detected, periodically attempts to bring the affected HSM back online
- Restarts an object search at the point of failure
- Restarts an Encrypt, Decrypt, Sign, Verify, SignRecover, VerifyRecover and Digest operation and replays the Update operations (up to a certain data length limit)
- Creates a log entry to note significant events
- Recovers session objects created by:
 - C_CopyObject
 - C_DeriveKey
 - C_UnwrapKey
 - C_GenerateKey *
 - C_GenerateKeyPair *



Note: Randomly-generated keys cannot be recovered if they are lost after they have been used in a cryptographic operation (otherwise, inconsistent results may be generated).

The environment variable ET_PTKC_GENERAL_LIBRARY_MODE specifies the Cryptoki Library operating mode. This variable controls which PKCS #11 model is applied to slot and token usage (see ["Work Load Distribution Model and High Availability" on page 49](#)).

Valid values for this variable are NORMAL or WLD or HA. If this variable is not defined, or contains an invalid value, then SafeNet ProtectToolkit-C will operate in NORMAL PKCS #11 mode.

The environment variable `ET_PTKC_HA_RECOVER_DELAY` defines the time (in minutes) the system will wait after an HSM failure before attempting reconnection to the failed HSM. If the value is zero, reconnection is not attempted.

The environment variable `ET_PTKC_HA_RECOVER_WAIT` allows the system to poll and attempt recovery if an HSM has failed. Valid values for this variable are YES or NO, valid only if the HA feature is enabled (`ET_PTKC_GENERAL_LIBRARY_MODE=HA`).

Example

To configure a basic HA system across two SafeNet ProtectServer Network HSMs with IP addresses 192.168.1.100 and 192.168.1.101, where the participating tokens are labeled "TokName", set these configuration items (see ["Configuration Items" on page 29](#)):

```
ET_HSM_NETWORK_SERVERLIST=192.168.1.100 192.168.1.101
ET_PTKC_WLD_SLOT_0=TokName
ET_PTKC_GENERAL_LIBRARY_MODE=HA
ET_PTKC_HA_RECOVER_DELAY=120
ET_PTKC_HA_RECOVER_WAIT=YES
```

HA Mode Logging

When the library is operating in HA mode it will generate log messages on certain events.

Configuration Name	Possible Values
ET_PTKC_HA_LOG_FILE	Log filename and location – default <code>/ptk_cryptoki</code> For example, <code>ET_PTKC_HA_LOG_FILE=C:\temp\ha_log.log</code> (Windows) or <code>ET_PTKC_HA_LOG_FILE=/tmp/hsm_log.log</code> (Unix)
ET_PTKC_HA_LOG_NAME	Application name – default <code>ptk_cryptoki</code>

The HA feature will generate the following log messages.

Message	Type	Meaning
Session potentially not recoverable: <desc>	Warning	Application has performed an operation that makes the session unrecoverable. The <desc> field will describe the type of operation. Only one message of this type is generated per <code>C_Initialize/C_Finalize</code> session.
HSM Failure detected hsmIdx=<>, hsmSlotId=<>	Error	A session has failed due to an HSM failure and the HA has attempted a session recovery. The hsmIdx is the zero-based index of the failing HSM, as specified by the <code>ET_HSM_NETWORKCLIENT_SERVERLIST</code> or in the order the SafeNet ProtectServer Network HSMs are detected. This is the same order reported by hsmstate utility.
Found HSM Dead:HSM Failed	Error	This message is generated only when <code>ET_PTKC_HA_RECOVER_DELAY</code> and <code>ET_PTKC_</code>

Message	Type	Meaning
		HA_RECOVER_WAIT are enabled. It indicates that the library has seen an HSM fail and is holding off all application threads while it attempts to recover the lost HSM.

External Key Storage

SafeNet ProtectServer HSMs have 4 MB of available secure memory. This is the only limit to the number of keys (by type and size) that can be stored.

Applications whose secure memory requirements exceed this limitation can use the External Token Support Library (ExtToken). ExtToken manages secure, external token object storage to host applications transparently. Host applications can use standard PKCS#11 function calls to access and manipulate token objects as though the token objects were stored on the HSM.

The ExtToken library is available with SafeNet ProtectToolkit-C and is a part of the standard **PTKcprt** package installation. The ExtToken library is supported on Windows only.

Using ExtToken, externally stored token objects can be used for RSA signing, certificate checking, DES key exchange, DES encryption of transaction messages, and more. To reduce processing overhead, the HSM stores the most recently used token objects in its internal cache memory. The number of token objects stored in cache is configurable by the user.

SafeNet ProtectServer HSMs support the storage of token objects in secure external locations and user slots simultaneously.

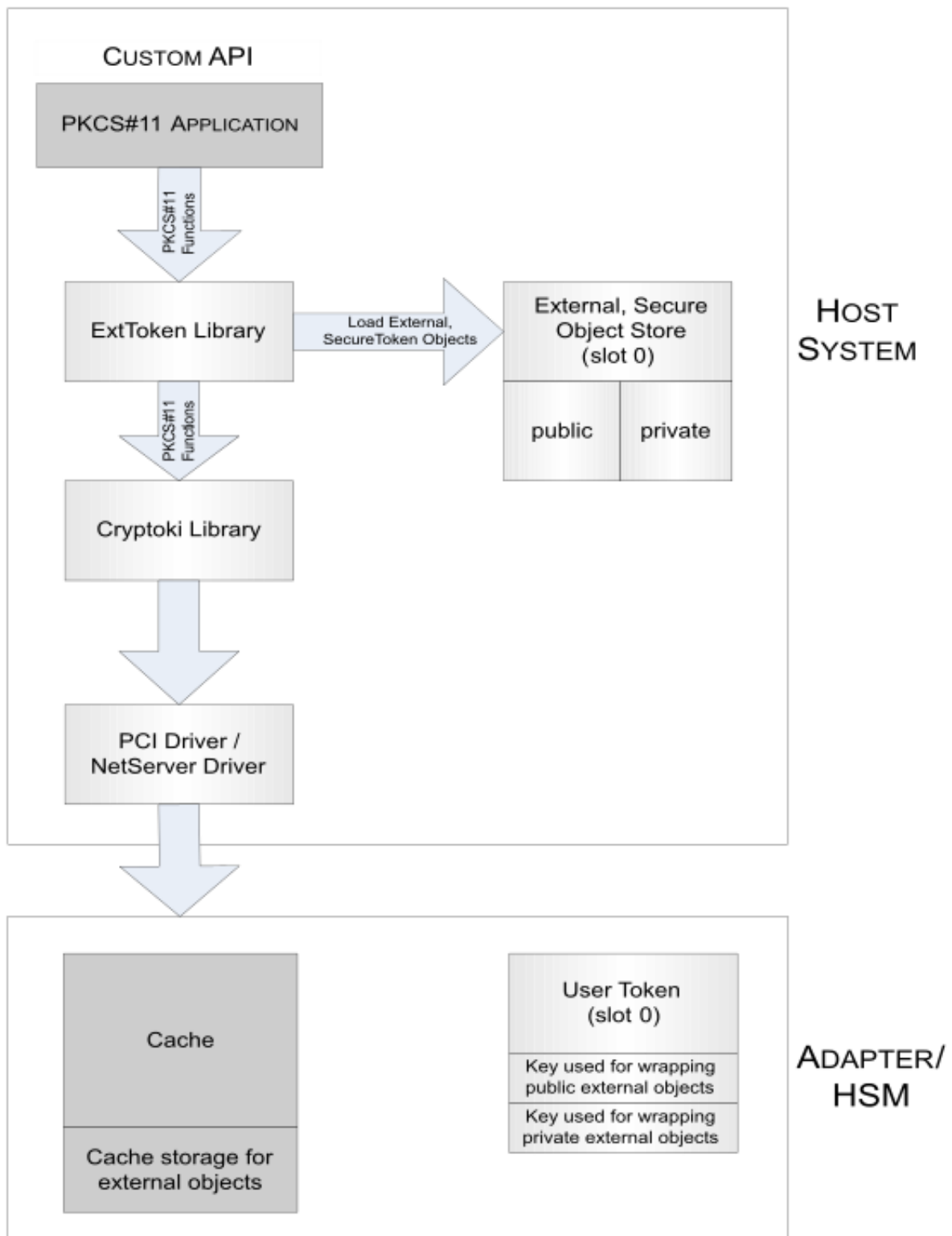
This section contains the following subsections:

- ["External Key Storage Model" below](#)
- ["External Key Storage Configuration" on page 63](#)
- ["Creating Externally Stored Objects" on page 66](#)

External Key Storage Model

["External Key Storage" on the next page](#) shows how secure external key storage is achieved on the host system and HSM.

Figure 1: External Key Storage



PKCS#11 applications interface with the ExtToken library via standard PKCS#11 function calls. The ExtToken library uses the Cryptoki library to enforce security policies and to store all data unrelated to external token objects. All cryptographic processing is performed on the standard Cryptoki library. The Cryptoki Library uses the PCI driver to interface with local HSMS, and the Netserver Driver for remote HSMS.

ExtToken achieves secure external storage by using two master DES2 keys to transparently wrap and unwrap the external objects. One key is for protecting public objects, and the other for private objects. These master keys are stored in slot 0. The token in slot 0 is automatically treated as an external token. If the relevant objects (the external token data object and the two master keys) are missing, they are automatically generated.

Token objects created by the ExtToken library are stored in the External, Secure Object Store residing on the host system. The External Secure Object Store is divided so that objects wrapped by public keys are stored separately from objects wrapped by private keys. When these external objects are referenced by an application, the ExtToken library automatically loads them into the standard Cryptoki library. Any operations on a non-external token are passed on to the standard Cryptoki library for processing. All externally stored objects reside in the token in slot 0. The externally stored objects share the same logical slot (slot 0) as the master keys, although they are physically stored in separate locations.

The HSM utilizes internal cache memory to store the most often-used token objects. The number of token objects stored in cache is configurable by the user. During operation, the token objects are loaded into cache one at a time. If the user-configured limit is reached, the least-used object is unloaded from cache.

Key backup is accomplished by storing the master keys on a smart card and compressing the files used by the Secure Object Store.

Performance

When storing objects externally, the need to unwrap objects introduces processing overhead. SafeNet ProtectToolkit-C's **CTPERF** utility can be used to gauge performance on individual systems (see "[CTPERF](#)" on page 125). For example: 109 keys per second can be unwrapped using a DES3 key. As mentioned above, the HSM stores the most recently-used token objects in cache memory to reduce overhead. The user can choose the maximum number of token objects stored in cache by configuring the environment variable `ET_PTKC_EXTTOKEN_MAXLOADED`. Managing the keys stored in cache, however, also creates processing overhead that increases linearly as the number of items stored in cache increases. Systems must be individually tuned for maximum performance depending on patterns of key usage by the host application. The processing overhead tradeoff between unwrapping keys and managing the cache must be taken into consideration.

Mechanisms Underlying ExtToken

ExtToken library treats an underlying token as an external token if it contains a data object with the label "ExtToken". To be functional, the underlying token must also contain two DES2 keys (one public and one private) with the label "ExtToken". Both will have the `CKA_WRAP` and `CKA_UNWRAP` attribute set to `TRUE`. For security reasons, `CKA_ENCRYPT` and `CKA_DECRYPT` are set to `FALSE`.

The `CKA_VALUE` attribute of the ExtToken data object is of the form "file:<file_name>", where <file_name> is the base name of the files that manage the token objects of the external token.

Two files exist for each external token:

- The Object Data Store (ODS) contains the token objects of the external token, wrapped under its corresponding master key (public objects using the public master key; private objects with the private master key) using the SafeNet vendor-defined mechanism `CKM_WRAPKEY_DES3_CBC`. This mechanism wraps both the object value and attributes in the created cryptogram.

- The Object Reference Table (ORT) contains an index of the token objects stored in the ODS and the KVCs of the master keys of the external token.

PKCS#11 requires that the CKA_EXTRACTABLE attribute be set to TRUE for any object to be wrapped using a key which has CKA_WRAP set to TRUE. As a result, the ExtToken library transparently sets the CKA_EXTRACTABLE attribute to TRUE for all token objects on an external token.

When an application acquires an object handle to a token object on an external token, the related cryptogram is read from the ODS file, and unwrapped into the underlying token as a session object.

If allowed, the token in slot ID 0 is automatically treated as an external token. The relevant objects (the external token data object and the two master keys) are automatically generated, if they are missing.

Known Limitations

The ExtToken library does not protect against multiple processes updating the external token files concurrently. When an application starts, the ORT is cached. If a second application modifies the ORT by manipulating token objects on the external token, the cache of the first application will be inconsistent. The results are undefined.

For performance reasons, the attributes in the template passed to **C_FindObjectsInit()** function should be limited to:

- CKA_TOKEN (If present, must be TRUE. If missing, assumed to be TRUE - can only find token objects).
- CKA_LABEL
- CKA_CLASS
- CKA_KEY_TYPE
- CKA_PRIVATE

Session objects can be used in an external token, so long as they are generated or created. Other attributes in the template are supported, but they may have a negative effect on application performance. This negative effect can be mitigated by using as many attributes as possible from the list above, and limiting such operations to application initialization.

Only objects with the CKA_EXTRACTABLE attribute set to TRUE can be imported to an external token.

It is not possible to set the SafeNet vendor-defined CKA_EXPORT attribute to TRUE on an external token object.

It is not possible to set the CKA_TRUSTED attribute to TRUE on an external token object.

The ORT and ODS files are susceptible to growth. The space associated with the cryptogram of deleted objects in the ODS is not reused. One way to reclaim this space is to use the CTKMU utility to backup all the objects to a file, rename/delete the existing ORT and ODS files, then restore from the backup.

If an application uses one session to access all objects on an external token, the HSM may run out of resources. As this is related to the size and number of objects, it is not possible to state exactly the upper limit supported by SafeNet HSMs. One example of such an application is **ctkm**. If there are too many objects on an external token to back them up, adjust the value of ET_PTKC_EXTTOKEN_MAXLOADED to a value that better suits your application/environment.

The SafeNet implementation of JCA/JCE (SafeNet ProtectToolkit-J) uses one session per KeyStore. An application using the same KeyStore to access a large number of keys runs the risk of consuming all HSM resources. To work around this risk, use a new KeyStore object when locating keysto avoid introducing a significant performance overhead.

Smart cards and the Admin Token cannot be used as external tokens.

External Key Storage cannot be used in conjunction with WLD.

External Key Storage Configuration

There are a number of files named **cryptoki.dll** provided as part of the SafeNet ProtectToolkit-C installation. This design ensures that PKCS #11 applications always link to a library called **cryptoki.dll**. The table below gives the location of **cryptoki.dll** files and their purpose. The ID field identifies the Cryptoki library and is used in discussions that follow.

Path	Purpose	ID
C:\Program Files\SafeNet\Protect Toolkit 5\Protect Toolkit C RT	Cryptoki library used for runtime applications	1
C:\Program Files\SafeNet\Protect Toolkit 5\ProtectToolkit C RT\extToken	ExtToken library used for runtime applications	2
C:\Program Files\SafeNet\Protect Toolkit 5\ProtectToolkit C SDK\bin\logger	Logger library used during application development	3
C:\Program Files\SafeNet\Protect Toolkit 5\ProtectToolkit C SDK\bin\hsm	Cryptoki library used during application development when communicating to HSMs	4
C:\Program Files\SafeNet\Protect Toolkit 5\ProtectToolkit C SDK\bin\ExtToken	ExtToken library used for application development	5
C:\Program Files\SafeNet\Protect Toolkit 5\ProtectToolkit C SDK\bin\sw	Cryptoki library used during application development in software-only mode	6

As both the ExtToken library and a Cryptoki library are named **cryptoki.dll** and used in a configuration that requires external key storage, environment variables indicate which library is linked to which software component. As indicated in ["External Key Storage Model" on page 59](#), PKCS#11 applications link to the ExtToken library, which in turn links to a Cryptoki library. The following subsections detail how this can be achieved:

- ["Configuration for Application Development" below](#)
- ["Configuration for Runtime Operation" on page 65](#)



Note: The following processes should only be followed if the ExtToken library is to be used. To switch between one of the hardware modes and software-only mode, use the **setmode** utility. See ["Changing the Cryptoki Provider" on page 21](#) for details.

Configuration for Application Development

1. Locate the current **cryptoki.dll** in use.

The current Cryptoki library in use is determined by the **Path** environment variable. The first folder named in the **Path** environment variable that contains a **cryptoki.dll** file indicates the path to the current Cryptoki library. Refer to the table above for folder locations.



Note: Record for use in step 3 the path of the folder containing the current **cryptoki.dll**.

2. In the **Path** environment variable, insert the path to the ExtToken library, so that this folder appears before any other folders containing **cryptoki.dll** files. This is typically **C:\Program Files\SafeNet\Protect Toolkit**

5\ProtectToolkit C SDK\bin\ExtToken

3. Configure the ET_PTKC_EXTTOKEN_PKCS11LIB environment variable.

ET_PTKC_EXTTOKEN_PKCS11LIB is the fully qualified file path to the original **cryptoki.dll** located in step 1.

Typically this should be achieved by:

- a. Creating or editing the registry key
HLKM (or HKCU)\SOFTWARE\SafeNet\PTKC\EXTTOKEN.
- b. Creating a string value named ET_PTKC_EXTTOKEN_PKCS11LIB.
- c. Setting the string to the fully qualified path of the original **cryptoki.dll**.

To specify the SafeNet ProtectToolkit-C SDK (PCI and network modes) Cryptoki library (Id 4), typically, ET_PTKC_EXTTOKEN_PKCS11LIB should be set to

C:\Program Files\SafeNet\Protect Toolkit 5\ProtectToolkit C SDK\bin\hsm.

4. Configure the ET_PTKC_EXTTOKEN_PATH environment variable in the registry key HLKM(or HKCU) \SOFTWARE\SafeNet\PTKC\EXTTOKEN. ET_PTKC_EXTTOKEN_PATH is the fully qualified directory path that determines where ExtToken stores its data files. These data files will contain the encrypted key material. The default value is **C:\ETExtToken**.
5. Configure the ET_PTKC_EXTTOKEN_MAXLOADED environment variable in the registry key HLKM(or HKCU) \SOFTWARE\SafeNet\PTKC\EXTTOKEN. ET_PTKC_EXTTOKEN_MAXLOADED is the maximum number of objects which will be loaded to the underlying token at one time. If this limit is reached, then the least used object is unloaded from the underlying token. The default value is 100.

Checking the Configuration

The steps listed previously should result in a configuration where the utilities provided in the SDK folders provide a view of the HSM deploying the ExtToken functionality. These utilities should be utilized for the management of external key storage.

In this configuration, the utilities installed under the Runtime folder do not utilize the ExtToken library and can be used to verify correct operation.

1. Open a command prompt in the SDK bin folder. This should typically be **C:\Program Files\SafeNet\Protect Toolkit 5\ProtectToolkit C SDK\bin\hsm**. This Window is referred to as **Command Prompt(1)** in later steps.
2. Open a command prompt in the Runtime folder. This should typically be **C:\Program Files\SafeNet\Protect Toolkit 5\Protect Toolkit C RT**. This Window is referred to as **Command Prompt(2)** in later steps.
3. At **Command Prompt(1)** enter the command **ctkmu I -s0**.

This command is generally utilized to display a list of the objects contained in slot 0. When used with the ExtToken functionality, this command additionally serves to initialize the mechanism that provides the external key storage functionality in the HSM. In a HSM where the ExtToken mechanism has not been initialized, this results in the creation of a number of objects on the HSM and the creation of the secure storage files on the Host.

4. Verify the creation of ExtToken mechanism on the HSM.
5. At **Command Prompt(2)** enter the command **ctkmu I -s0**.

Three additional objects should have been created with the label ExtToken. The first object is a Data object containing information relating to the configuration of ExtToken. Two secret keys are created; one key is for private objects and the other key is for public objects. These keys are only visible when the utility utilizes the Cryptoki Library. When the utility utilizes the ExtToken Library only the externally stored keys are visible.

6. Verify the creation of the Storage Files on the Host computer.

The ET_PTKC_EXTTOKEN_PATH determines the location of the storage files. This is typically **C:\ETExtToken**. Refer to step 4 above. Check that the folder has been created and contains an .ord file and an .ort file.

Configuration for Runtime Operation

In the standard installation folder hierarchy for the runtime software components, the utilities and the cryptoki.dll file are located in the same folder.

If a **cryptoki.dll** file is located in the same directory as the utilities, the utilities make use of this library, otherwise a utility located via the path environment variable is used. As this configuration requires the utilities to use the ExtToken library (Id 2) the Runtime Cryptoki library (Id 1) must be removed from the folder containing the utilities.

1. Move the Runtime Cryptoki Library from its current location (typically in folder **C:\Program Files\SafeNet\Protect Toolkit 5\Protect Toolkit C RT** (Id 1)) into a new sub-folder in this folder called **hsm** (typically **C:\Program Files\SafeNet\Protect Toolkit 5\Protect Toolkit C RT\hsm**).
2. In the **Path** environment variable, insert the path to the ExtToken library, so that this folder appears before any other folders containing **cryptoki.dll** files. This is typically **C:\Program Files\SafeNet\Protect Toolkit 5\Protect Toolkit C RT\ExtToken**.
3. Configure the ET_PTKC_EXTTOKEN_PKCS11LIB environment variable. ET_PTKC_EXTTOKEN_PKCS11LIB is the fully qualified file path to the original **cryptoki.dll** located in step 1. Typically this should be achieved by creating or editing the registry key **HLKM(or HKCU)\SOFTWARE\SafeNet\PTKC\EXTTOKEN**, creating a string value named **ET_PTKC_EXTTOKEN_PKCS11LIB** and setting it to the fully qualified path to the original **cryptoki.dll** (typically **C:\Program Files\SafeNet\Protect Toolkit 5\Protect Toolkit C RT\hsm\cryptoki.dll**).
4. Configure the ET_PTKC_EXTTOKEN_PATH environment variable in the registry key **HLKM(or HKCU)\SOFTWARE\SafeNet\PTKC\EXTTOKEN**. ET_PTKC_EXTTOKEN_PATH is the fully qualified directory path that determines where ExtToken stores its data files. These data files will contain the encrypted key material. The default value is **C:\ETExtToken**.
5. Configure the ET_PTKC_EXTTOKEN_MAXLOADED environment variable in the registry key **HLKM(or HKCU)\SOFTWARE\SafeNet\PTKC\EXTTOKEN**. ET_PTKC_EXTTOKEN_MAXLOADED is the maximum number of objects which will be loaded to the underlying token at one time. If this limit is reached, then the least used object is unloaded from the underlying token. The default value is 100.

Checking the Configuration

1. Open a command prompt in the **Runtime** folder. This should typically be **C:\Program Files\SafeNet\Protect Toolkit 5\Protect Toolkit C RT**.
2. Set the ET_PTKC_EXTTOKEN_PKCS11LIB environment variable to point to the directory containing the original **cryptoki.dll** file (**C:\Program Files\SafeNet\Protect Toolkit 5\Protect Toolkit C RT\hsm** in this example).
3. Enter the command **ctkmui -s0**. This command is generally used to display a list of the objects contained in slot 0. When used with the ExtToken functionality, this command additionally serves to initialize the mechanism that provides the external key storage functionality in the HSM. In a HSM where the ExtToken mechanism has not been initialized, this results in the creation of a number of objects on the HSM and the creation of the secure storage files on the Host.
4. Verify the creation of the Storage Files on the Host computer. The ET_PTKC_EXTTOKEN_PATH determines the location of the storage files. This is typically **C:\ETExtToken**. Refer to step 4 above. Check that the folder has been created and contains an .ord file and an .ort file.

Creating Externally Stored Objects

The utilities typically located in either **C:\Program Files\SafeNet\Protect Toolkit 5\Protect Toolkit C RT** (for runtime installation) or **C:\Program Files\SafeNet\Protect Toolkit 5\ProtectToolkit C SDK\bin\hsm** (for SDK installation) can now be used to create externally-stored keys. Slot 0 is used for externally stored keys. For example, the command **ctkmu c -s0 -z1024 -nexternal1 -aX -trsa** creates an RSA key pair in external storage.

Backup and Restore

1. The simplest way to back up keys is to zip the secure external storage files and to export the object keys used in the ExtToken mechanism. As the files are already encrypted, format encryption need not be applied when compressing the files. These files are located in the folder indicated by the **ET_PTKC_EXTTOKEN_PATH** environment variable and have the extensions **.ort** and **.ods**.
2. To access the objects in the ExtToken mechanism, the Cryptoki Library (ID 1 or 4) must be used. When the utilities are used with the Cryptoki Library, the physical slots are made accessible and therefore the objects that underlie the ExtToken mechanism are accessible. To enable the utilities to use the Cryptoki Library, in the **Path** environment variable, insert the path to the Cryptoki library, so that this folder appears before any other folders containing cryptoki.dll files. For Runtime operation, this is typically **C:\Program Files\SafeNet\Protect Toolkit 5\Protect Toolkit C RT** (if the previous configuration steps were followed). For SDK, this is typically **C:\Program Files\SafeNet\Protect Toolkit 5\ProtectToolkit C SDK\bin\hsm**.
3. At a command prompt, enter the command **ctkmu l -s0**. Three objects should be listed with the label ExtToken. The first object is a Data object containing information relating to the configuration of ExtToken. Two secret keys are created; one key is for private objects and the other key is for public objects.
4. Export the objects in slot 0 onto Multiple Custodian smart cards. The following example illustrates how to do this with two smart cards, where the smart card reader is located in slot 1.

```
ctkmu x -s0 -c1
```

5. When restoring keys, the Cryptoki Library (ID 1 or 4) must be used (see step 2). To restore keys after tampering the HSM, uncompress the secure external storage files into the folder indicated by the **ET_PTKC_EXTTOKEN_PATH** environment variable. Import the secret keys from the smart cards. The following example illustrates how to import the keys if exported in the manner described above.

```
ctkmu i -s0 -c1
```

6. To make use of the ExtToken Library the system must be reconfigured to use the ExtToken Library for Application Development or for Runtime Operation.

Real Time Clock

The HSMAdmin API allows applications to access and adjust the Real Time Clock (RTC). Information about the RTC status, and how many times it has been adjusted, is also available.

The **CTCONF** utility allows an administrator to configure adjustment access control for the RTC. The administrator can control the delta amount and the number of times the RTC can be adjusted within a configurable period of time.

CTCONF has two applicable command line options: one that sets the rule for adjustment access control and one that enables/disables adjustment access control. See "**CTCONF**" on page 104 for details regarding the use of these command line options.

Setting the Rule for RTC Adjustment Access Control

The RTC Adjustment Access Control Rule specifies the guard parameters which control modification of the RTC. If modification of the RTC is attempted outside of these guard parameters, it will fail.

The table below describes the guard parameters:

Parameter	Meaning
secs	Total amount of deviation (in seconds) within a guard duration. Range: 1-120
count	Total number of adjustments that can be made within the guard duration. Range: any integer. Setting this variable to 0 allows an unlimited number of adjustments
days	The guard duration in days. Range: 1-12

The following examples show how to set guard parameters:

Example 1:

If applications accessing the RTC do not need to alter the RTC by more than 12 seconds, but can make as many adjustments as needed within a period of 1 day, the following command would set the rule for RTC Adjustment Access Control.

```
ctconf --rtc-adj-access-control-rule=12:0:1
```

Example 2:

If the guard duration is extended to 4 days, the following command would ensure the other access control rule parameters are not modified:

```
ctconf --rtc-adj-access-control-rule=:4
```

The current settings for the access control rule are displayed via the **ctconf -v** command.

Enabling/Disabling RTC Adjustment Access Control

Once the RTC Adjustment Access Control Rule has been set, RTC Adjustment Access Control can be enabled. When RTC Adjustment Access Control is enabled, the functions provided by the HSMAdmin API (refer to the *SafeNet ProtectToolkit-C Programmers Guide*) are governed by the RTC Adjustment Access Control Rule. By disabling RTC Adjustment Access Control, unlimited adjustments to the RTC may be performed.

To enable RTC access control:

```
ctconf --rtc-adj-access-control=1
```

When access control is disabled, the parameters passed via the HSMADM_GetRtcAdjustAmount and HSMADM_GetRtcAdjustCount function calls are not valid. **CTCONF** may be specified with both the **--rtc-adj-access-control-rule** and **--rtc-adj-access-control** command line parameters simultaneously. The RTC Adjustment Access Control Rule is given precedence over the RTC Access Control command.

Security Policies and User Roles

This chapter covers considerations administrators should make when selecting and setting a security policy for the SafeNet ProtectToolkit-C environment. Many factors can affect operational security, and the various security features provided may affect SafeNet ProtectToolkit-C performance and security during runtime operations.

A security policy is a set of security settings that control how SafeNet ProtectToolkit-C is allowed to function. For example:

- whether PINs may be passed across the host interface in an unencrypted form
- whether a soft tamper (erase all internal secure memory) should occur as part of a firmware upgrade

Organizations can create unique security policies to satisfy their own needs, or they may adopt policies defined by standards bodies or other organizations.

A number of security settings offered as a part of SafeNet ProtectToolkit-C implement *typical security policies* that meet certain standards or satisfy application integration requirements. These or any other custom security policy can be activated. The available options are fully described in ["Typical Security Policies" on the next page](#).

If you are implementing a security policy to satisfy application integration requirements, other information may be available. See the documentation for your specific application.

Compliance with the PKCS #11 standard will vary from policy to policy. Generally, stricter compliance results in lowered security. See ["PKCS #11 Compliance and Security" on the next page](#) for further information.

Some security policy settings have effects that are specific to different user roles. See ["User Roles" on page 77](#).

SafeNet ProtectToolkit-C security policies are implemented by setting or clearing *security flags* to switch different functions on or off. A policy might be implemented by setting a single security flag. In other cases, more than one flag must be set. See ["Security Flags" on page 72](#) for specific procedures.

PKCS #11 Compliance and Security

SafeNet ProtectToolkit-C can be configured for strict compliance with the PKCS #11 standard by using the security policy PKCS #11 Compatibility Mode. If a greater level of security is required, an alternate standard or custom security policy may be adopted. These and all other typical security policies are discussed in ["Typical Security Policies" below](#).

By default (after initial HSM installation or following a tamper event) the SafeNet Default Mode security policy is applied. This mode offers a greater level of security than PKCS #11 Compatibility Mode, while offering more PKCS#11 functions than other possible security policies.

For more about how SafeNet Default Mode differs from PKCS #11 Compatibility Mode, and the related security issues, see ["PKCS #11 Compatibility Mode" below](#).

Typical Security Policies

A number of *typical security policies* designed to meet standards or satisfy application integration requirements are offered as a part of SafeNet ProtectToolkit-C. The following typical security policies are described in this section:

- ["PKCS #11 Compatibility Mode" below](#)
- ["SafeNet Default Mode" on the next page](#)
- ["FIPS Mode" on the next page](#)
- ["Entrust Compliant Modes" on page 71](#)
- ["Netscape Compliant Mode" on page 71 and](#)
- ["Restricted Mode" on page 71](#)

The **ctconf** command line utility is used to implement the policies by setting *security flags*. The specific commands for each are provided.

Security flags are discussed in detail in ["Security Flags" on page 72](#).

For some policies, security flags may be available that alter security behavior without invalidating the policy. See ["Security Policy Options" on page 76](#).

For the complete **ctconf** command reference, see ["CTCONF" on page 104](#) in ["Command Line Utilities Reference" on page 90](#).

PKCS #11 Compatibility Mode

This mode allows full compatibility with all cryptographic mechanisms provided by the PKCS#11 v2.20 standard, including those mechanisms subsequently found to have security flaws. The following affected mechanisms are available when this policy is set:

```
CKM_CONCATENATE_BASE_AND_KEY
CKM_CONCATENATE_BASE_AND_DATA
CKM_CONCATENATE_DATA_AND_BASE
CKM_EXTRACT_KEY_FROM_KEY
```



WARNING! Use of this security policy compromises security. A skilled attacker may be able to exploit vulnerabilities in certain mechanisms when this policy is set.

Command:

```
ctconf -fp
```

SafeNet Default Mode

By default (after initial HSM installation or following a tamper event), SafeNet Default Mode is applied to SafeNet ProtectToolkit-C. This mode provides better security than PKCS #11 Compatibility Mode, while offering more of the PKCS #11 standard mechanisms than other, more restrictive security policies.

For more about how SafeNet Default Mode differs from PKCS #11 Compatibility Mode, and the related security issues, see ["PKCS #11 Compatibility Mode" on the previous page](#).

Command:

```
ctconf -f0
```

FIPS Mode

SafeNet ProtectToolkit-C and the ProtectServer HSM have been certified to Federal Information Processing Standard (FIPS) 140-2 level 3. The FIPS certification assures users that an independent third party has verified that the product meets the high level of security demanded.



Note: SafeNet ProtectToolkit-C and the HSM can function outside the scope of this accreditation. Therefore, to guarantee that the HSM functions in FIPS mode, ensure that the correct configuration is set using the **ctconf** command given below.

The attributes of the FIPS Mode security policy are:

- No public cryptographic operations.



Note: RSA and other public key processing can still occur. The setting restricts cryptographic services from being performed by unauthenticated users.

- No clear PINs allowed
- Authentication protection turned on
- Security policy locked to prevent any change
- Tamper before upgrade
- Only allow FIPS-approved algorithms

FIPS Mode Operational Restrictions

In FIPS mode, operations of certain cryptographic algorithms are restricted to keys with a minimum modulus. Any attempt to use or create a key smaller than the specified minimum will result in a CKR_KEY_SIZE_RANGE error. The minimum key size for verify operations may be smaller, to verify legacy keys created in earlier versions of FIPS mode. The key sizes are restricted as follows:

- **RSA:** must be 2048, 3072, or 4096 bits (verify - 1024 or 1536 bits)
- **DSA:** must be 2048, 3072, or 4096 bits (verify - 1024 or 1536 bits)

- **EC:** must be 224 bits at minimum (verify - 160 bits)

Command:

```
ctconf -fF
```

equivalent to:

```
ctconf -faclntu
```

Entrust Compliant Modes

Entrust Compliant Mode 1

The Entrust Compliant Mode 1 uses the specific security profile required by Entrust Authority version 5.x software.

Command:

```
ctconf -fe
```

Entrust Compliant Mode 2

The Entrust Compliant Mode 2 uses the specific security profile required by Entrust Authority version 6.x and Entrust Security Manager version 7.x software.

Command:

```
ctconf -fc
```

Netscape Compliant Mode

SafeNet ProtectToolkit-C is compatible with the Netscape/iPlanet range of products. The HSM has been tested with the following products:

- iPlanet Certificate Management System 4.1/4.2
- Netscape Enterprise Server 4.1
- Netscape Communicator 4.5 or later

Place the HSM in this mode by enabling the *No Public Cryptography* flag.

Command:

```
ctconf -fc
```

Restricted Mode

In Restricted Mode, the HSM requires users to identify themselves before cryptographic services are made available. This security policy will also prevent any clear PINs or sensitive key material from passing through the PCI bus interface of the HSM. It does not, however, require each individual request to the HSM to be signed.

Command:`ctconf -fcnl`

Security Flags

Policies are implemented in SafeNet ProtectToolkit-C by configuring *security flags*.

Setting a security flag activates its particular security settings. One or more of these flags can be set to create custom security policies or to implement the typical security policies described in the previous section.

Configuring Security Flags

Security flags are configured using the **ctconf** command line utility.

The command syntax is as follows:

ctconf -f<flags>

Multiple flags may be set simultaneously. For example, the command: **ctconf -ftu** would set both the **t** and the **u** flags.

When flags are set, any flags set previously are cleared.

Set **flags = 0** to clear all the flags. This places the device in *SafeNet Default Mode* (Default <No flags set>). See the *Typical Security Policies* section "[SafeNet Default Mode](#)" on page 70, for more information about this security policy.

Use other **flags** values to set flags as follows:

To set flag:	Use flags value:
Auth Protection	u
DES Keys Even Parity Allowed	d
Entrust Ready	e
FIPS Algorithms Only	a
FIPS Mode	F
Full Secure Messaging Encryption	N
Full Secure Messaging Signing	U
Increased Security Level	i
Mode Locked	l
No Clear PINs	n
No Public Crypto	c
Pure PKCS11 (PKCS#11 Compatibility Mode)	p
Tamper Before Upgrade	t
User-specified ECC DomainParameters Allowed	E

To set flag:	Use flags value:
Weak PKCS#11 Mechanisms	w

Each of these flags is fully described below.

For the complete **ctconf** command reference, see ["CTCONF" on page 104](#).

Security Flag Descriptions

The security settings configured by each of the security flags are described below. A mapping of security flags to the typical security policies described in this manual is provided in ["Security Policy Options" on page 76](#).

Auth Protection

The *Auth Protection* (Authentication/Session Protection) flag, when set, ensures *secure messaging authentication* between applications and the HSM is enforced for certain messages sent from applications to the HSM. Critical messages or messages that might otherwise contain sensitive information are affected. These messages must be digitally signed so they can be verified by the HSM.

With this setting applied, applications will operate more securely. HSM performance, however, may suffer due to the additional operations required to sign and verify each message request.

DES Keys Even Parity Allowed

The *Des Keys Even Parity Allowed* flag permits creation of DES, DES2 and DES3 keys and components with even parity.

Entrust Ready

The *Entrust Ready* (Entrust Compliant) flag, when set, establishes the following rules:

- When a nonexistent mechanism is queried, an empty mechanism structure is returned.
- When a token is initialized with the **C_InitToken** command, the SO PIN is not required.
- A user who is already logged in is permitted to log in again.
- When using the **C_SignFinal** command, the size of the message authentication code (MAC) returned can be controlled, even if the mechanism is not one of the general-length MAC mechanisms specified in the PKCS #11 standard.
- When using the **C_WrapKey** function, if the **CKA_extractable** attribute is not specified, it defaults to **true** so that wrapping is allowed.

FIPS Algorithms Only

The *FIPS Algorithms Only* (Only Allow FIPS-Approved Algorithms) flag, when set, disables non-FIPS approved algorithms.

The algorithms approved by FIPS are: AES, Triple-DES, DSA, RSA, ECDSA, HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, SHA-1, SHA-256, SHA-384, SHA-512, Triple-DES MAC.

Refer to the *Typical Security Policies* section ["FIPS Mode" on page 70](#) for more.



Note: For FIPS-approved algorithms for individual products, please check the FIPS product certification.

FIPS Mode

The *FIPS Mode* (FIPS 140-1 Mode or FIPS 140-2 Mode) flag, when set, sets the following composite flags:

- FIPS Algorithms Only
- No Public Crypto
- Mode Locked
- No Clear PINs
- Tamper Before Upgrade
- Auth Protection

Instead of specifying each of these flags separately with **ctconf**, the *FIPS Mode* flag can be set as a shortcut.

Refer to the entries for the individual flags and the *Typical Security Policies* section "[FIPS Mode](#)" on page 70.

Full Secure Messaging Encryption

The *Full Secure Messaging Encryption* flag, when set, ensures that:

- User PINs or other sensitive information cannot be passed across the host interface unencrypted.
- Secure messaging encryption is enabled, so every message between the application and the HSM is encrypted in both directions.
- Certain functions that would otherwise result in the clear transmission of sensitive data are disabled
- The creation of any keys with the **CKA_SENSITIVE** attribute set to **false** is not permitted.

Note that the *Full Secure Messaging Encryption* flag is similar to the *No Clear PINs Allowed* flag, except every message between the application and the HSM is encrypted in both directions. The key used for the message encryption is generated using the PKCS #3 Diffie-Hellman Key Agreement Standard.

By enabling this setting, applications will operate more securely. however this will also have the effect of decreasing HSM performance. This is due to the increased operations required to encrypt and decrypt each request and response message.

Full Secure Messaging Signing

The *Full Secure Messaging Signing* flag, when set, indicates that secure messaging authentication between applications and the HSM is being enforced for every message, in both directions, between the application and the HSM. All messages must be digitally signed so that they can be verified by the HSM.

Note that the *Full Secure Messaging Signing* flag is similar to the *Auth Protection* flag except that every message, in both directions, between the application and the HSM is digitally signed and verified. The key used for the message signing is generated using the PKCS #3 Diffie-Hellman Key Agreement Standard.

With this setting applied, applications will operate more securely. HSM performance, however, may suffer due to the additional operations required to sign and verify each message request.

Increased Security Level

The *Increased Security Level* flag, when set, ensures that:

- The mechanism CKM_EXTRACT_KEY_FROM_KEY is disabled.
- Changing the CKA_MODIFIABLE attribute from **false** to **true** while using the **C_CopyObject** command is not permitted.

Mode Locked

The *Mode Locked* (Lock Security Mode) flag, when set, prevents any further security flag modification. A new security policy can only be implemented after performing a tamper operation.

No Clear PINs

The *No Clear PINs* (No Clear PINs Allowed) flag, when set, ensures that:

- User PINs or other sensitive information cannot be passed across the host interface unencrypted.
- Secure messaging encryption is enabled for critical requests to the HSM, or for those requests that might otherwise contain sensitive information.
- Certain functions that would otherwise result in the clear transmission of sensitive data are disabled.
- The creation of any keys with the CKA_SENSITIVE attribute set to **false** is not permitted.

No Public Crypto

The *No Public Crypto* flag, when set, ensures that no user can perform a cryptographic operation without having first authenticated themselves.

When this flag is set, each token in the system will have the PKCS #11 CKF_LOGIN_REQUIRED flag set so that applications must authenticate before operations are allowed. Note that this security flag does not affect the Admin token, which always requires authentication for access.



Note: The name of this flag does not imply that public key cryptography is not allowed. Setting this flag will not prevent RSA processing.

Pure PKCS11 (PKCS#11 Compatibility Mode)



WARNING! Setting this flag compromises security. A skilled attacker may be able to exploit vulnerabilities in certain mechanisms when this flag is set.

The *Pure PKCS11* flag, when set, allows that the following mechanisms to function as the PKCS #11 v2.20 standard requires.

- CKM_CONCATENATE_BASE_AND_KEY
- CKM_CONCATENATE_BASE_AND_DATA
- CKM_CONCATENATE_DATA_AND_BASE
- CKM_EXTRACT_KEY_FROM_KEY

Tamper Before Upgrade

The *Tamper Before Upgrade* flag, when set, ensures that a soft tamper (erasure of all HSM internal secure memory) will occur when any of the following operations are undertaken.

- Firmware upgrade
- FM download
- FM disable operation

User Specified ECC DomainParameters Allowed

The *User Specified ECC DomainParameters Allowed* flag, when set, allows ECC Public and Private keys with Domain Parameters other than the set of named curves built into the HSM to be generated and stored on the HSM.

Weak PKCS#11 Mechanisms



WARNING! Setting this flag compromises security. A skilled attacker may be able to exploit vulnerabilities in certain mechanisms when this flag is set.

Newly-discovered key extraction techniques have revealed vulnerabilities in some mechanisms. These mechanisms are now restricted by default in the factory settings of all new HSMs, or when flags are set to "0" (all flags cleared). Also, these mechanisms cannot be enabled when flags are set to "F" (FIPS 140-2 Mode) or "a" (Only Allow FIPS-Approved Algorithms). The *Weak PKCS#11 Mechanisms* flag, when set, allows the use of these less-secure mechanisms. It can be used with any combination of flags except "F" and "a".

The following mechanisms are affected:

- CKM_CONCATENATE_BASE_AND_DATA
- CKM_CONCATENATE_BASE_AND_KEY
- CKM_CONCATENATE_DATA_AND_BASE
- CKM_XOR_BASE_AND_DATA
- CKM_XOR_BASE_AND_KEY
- CKM_EXTRACT_KEY_FROM_KEY

Security Policy Options

Optionally with some of the typical security policies, security flags may be changed to change security behavior without invalidating the policy.

The following table details the mandatory and optional security flag settings for each of the typical security policies.

Security Policies	Impact of Security Flags on Policies												
	a	c	d	e	i	l	n	N	p	t	u	U	E
PKCS #11 Compatibility Mode	x		x	x	x				✓				
SafeNet Default Mode	x	x	x	x	x	x	x	x	x	x	x	x	
FIPS Mode	✓	✓	x	x		✓	✓			✓	✓		
Entrust Compliant Mode 1 ¹	x	x	x	✓			x	x			x	x	
Entrust Compliant Mode 2 ²	x	✓	x	x			x	x			x	x	

Security Policies	Impact of Security Flags on Policies												
	a	c	d	e	i	l	n	N	p	t	u	U	E
Netscape Compliant Mode	x	✓	x	x			x	x			x	x	
Restricted Mode	x	✓	x	x		✓	✓				x	x	

¹ When using Entrust Authority version 5.x

² When using Entrust Authority version 6.x and Entrust Security Manager version 7.x

Key

a	FIPS Algorithms Only	✓	The security flag must be set. If cleared the security policy is invalidated.
c	No Public Crypto		
d	DES Keys Even Parity Allowed	x	The security flag must be cleared. If set the security policy is invalidated.
e	Entrust Ready		
i	Increased Security Level		Optional. Setting or clearing the security flag will not invalidate the security policy.
l	Mode Locked		
n	No Clear PINs		
N	Full Secure Messaging Encryption		
p	Pure PKCS11		
t	Tamper Before Upgrade		
u	Auth Protection		
U	Full Secure Messaging Signing		
E	User Specified ECC Parameters		

User Roles

As part of the SafeNet ProtectToolkit-C configuration process, different *user roles* are assigned to those responsible for application administration and use.

For SafeNet ProtectToolkit-C, there are four defined roles available. These are:

- ["Security Officer \(SO\)" on page 79](#)
- ["Token Owner \(User\)" on page 79](#)
- ["Administration Security Officer \(ASO\)" on the next page](#) and
- ["Administrator" on the next page](#)

For public access roles, see ["Unauthenticated Users" on page 79](#).

Standard PKCS #11 defines the *Security Officer* (SO) and the *Token Owner* or *User* roles. Each slot and its associated token will have an SO and a User, each with their own respective PINs. A Security Officer grants and revokes access to a token and assists with key backups. A Token Owner uses the token for the application.

Two additional roles are only available on the Admin token. The holders of these roles handle HSM-level administration and management. These are the *Administration Security Officer* (ASO) and the *Administrator*. These roles effectively mirror their standard PKCS #11 counterparts.

It should be noted that the services available to the various roles are highly dependent upon the security policy set for the HSM. The following sections give a complete description of these roles and the services available to each of them.

Administration Security Officer (ASO)

This user knows and can present the Admin Token SO PIN. The ASO's main role is to introduce the Administrator to the module. The following services are available to the ASO:

- Set the initial Administrator PIN value (ASO cannot change it later)
- Set the CKA_TRUSTED attribute on a Public object
- Set the CKA_EXPORT attribute on a Public object
- Exercise cryptographic services with Public objects
- Create, destroy, import, export, generate and derive Public objects
- Can change his/her own PIN

Administrator

This user knows and can present the Admin Token User PIN. The following services are available to the Administrator:

- Set or change Real Time Clock (RTC) value
- Read the System Event Log
- Purge a full System Event Log
- Configure the Transport Mode feature
- Specify the security policy of the HSM
- Create new SafeNet ProtectToolkit-C slots/tokens and specify their labels, SO PINs, and minimum PIN Length
- Initialize smart cards and specify their labels and SO PINs
- Destroy individual SafeNet ProtectToolkit-C slots/tokens
- Erase all HSM secure memory, including all PINs and User Keys
- Perform firmware upgrade operations
- Manage Host Interface Master Keys
- Exercise cryptographic services with Public objects on the Admin Token
- Exercise cryptographic services with Private objects on the Admin Token
- Create, destroy, import, export, generate and derive Public objects on the Admin Token
- Create, destroy, import, export, generate and derive Private objects on the Admin Token
- May change his/her own PIN

Security Officer (SO)

Many users may be assigned this role. There will be one per user slot. The SO has the following abilities:

- Set the initial User PIN value (SO cannot change it later)
- Reset (re-initialize) the Token (destroys all keys and the User PIN on the Token) and set a new label
- Set the CKA_TRUSTED attribute on a Public object
- Set the CKA_EXPORT attribute on a Public object
- Exercise cryptographic services with Public objects
- Create, destroy, import, export, generate and derive Public objects
- May change his/her own PIN

Token Owner (User)

Many users may be assigned this role. There will be one per user slot. The user has these abilities:

- Exercise cryptographic services with Public objects
- Exercise cryptographic services with Private objects
- Create, destroy, import, export, generate and derive Public objects
- Create, destroy, import, export, generate and derive Private objects
- May change his/her own PIN

Unauthenticated Users

Public (unauthenticated) access to HSMs is allowed. Because authentication applies to tokens, a user may be simultaneously authenticated to one token while accessing another token without authentication.



Note: The services available to unauthenticated users are heavily dependent on the active security policy.

Unauthenticated users have these abilities:

- Exercise status querying services
- Authenticate to a token
- If 'No Clear PINs' is not set, they may initialize User or Smart Card Tokens and specify their labels and SO PINs
- If token flag CKF_LOGIN_REQUIRED is FALSE, they can create, destroy, import, export, generate, derive and use Public objects on the token
- If token flag CKF_LOGIN_REQUIRED is FALSE, they can exercise cryptographic services with Public objects
- If 'Authentication Protection' is not set, they can exercise the digesting services
- Force session terminate, restart HSM by running the **hsmreset** utility.

Operational Tasks

This chapter describes some of the most common operational procedures a User, Administrator or Security Officer may perform during normal SafeNet ProtectToolkit-C operation.

Before attempting any of the procedures in this chapter, ensure that SafeNet ProtectToolkit-C has been installed and configured for normal runtime use. You can find more information about the frequently-referenced command line utilities in ["Command Line Utilities Reference" on page 90](#). Many of these functions can also be achieved with the GUI-based tools, which are described in ["Key Management Utility \(KMU\) Reference" on page 139](#) and ["Administration Utility \(gCTAdmin\) Reference" on page 130](#).

This chapter contains the following sections:

- ["Changing a User or Security Officer PIN " below](#)
- ["Secure Key Backup and Restoration" on the next page](#)
- ["Adding and Removing Slots" on page 85](#)
- ["Re-initializing a Token" on page 86](#)
- ["Connecting and Removing Smart Card Readers" on page 86](#)
- ["Using Transport Mode to Avoid a Board Removal Tamper" on page 86](#)
- ["Adjusting the HSM Clock" on page 87](#)
- ["Changing Secure Messaging Mode" on page 87](#)
- ["Managing Session Key Rollover" on page 87](#)
- ["Using the System Event Log" on page 87](#)
- ["Updating Firmware" on page 88](#)
- ["Tampering the HSM" on page 89](#)
- ["Installing a Functionality Module" on page 89](#)

Changing a User or Security Officer PIN

It may sometimes be necessary to change the User or SO PIN on a particular token. The appropriate user may perform a PIN change at any stage and on any token.

To perform a PIN change, use the command line utility **ctkmu**.

To perform an SO PIN change on slot 1:

```
ctkmu p -s1 -O
```

The utility will prompt the user for the current SO PIN and the new PIN must be entered and confirmed.

To perform a token user PIN change on slot 2:

```
ctkmu p -s2
```


The utility will prompt the user for the current PIN and the new PIN must be entered and confirmed.



Note: This command is also used to initialize the User PIN. When this command is executed for an uninitialized token, the utility will prompt the user for the SO PIN and to set the initial user PIN.

Secure Key Backup and Restoration

SafeNet ProtectToolkit-C allows for keys to be backed up to disk files or smart cards, for convenient transfer of sensitive keys to other machines or off-site storage and subsequent recovery. Encrypted parts may also be displayed on the screen.

Determining Backup Requirements

There are no set rules within SafeNet ProtectToolkit-C dictating which keys should be backed up. The individual key owner decides which keys require backup protection.

As a guideline, keys that cannot be recreated or easily reconstructed by other means should generally be backed up. These may include generated key values or long keys manually entered by multiple custodians.

Not all keys can be backed up, since certain key attribute values have to be set in order to allow the backup. The setting of key attributes is therefore an important consideration when creating keys suitable for backup operations and is covered in ["Key Attributes" on the next page](#).

Available Backup and Recovery Methods

There are two methods of backing up a key:

- The *multiple custodians method*, where a key is split into shares and distributed among multiple custodians. The shares are encrypted (wrapped) by a second, randomly-selected key called the wrapping key.
- The *single custodian method*, where the key is encrypted (wrapped) by a specifically-selected wrapping key.

The encrypted key is then stored on the backup medium.

For key backup and restore procedures, see:

- ["Key Backup Procedure" on page 84](#)
- ["Key Restore Procedure" on page 85](#)

Key Splitting Scheme Selection

If a key is to be split into multiple shares, first select the scheme to split the key. It is possible to split the key so that the original key may only be recovered with the co-operation of either:

- all the custodians using the *standard scheme*, or
- a user-specified minimum number of the custodians using the *N of M scheme**

In the N of M scheme, no single custodian is required to recover the key. This is particularly advantageous if a smart card should become corrupted. Corrupted cards will be rejected during an import operation. The custodians will be prompted for another card to continue.

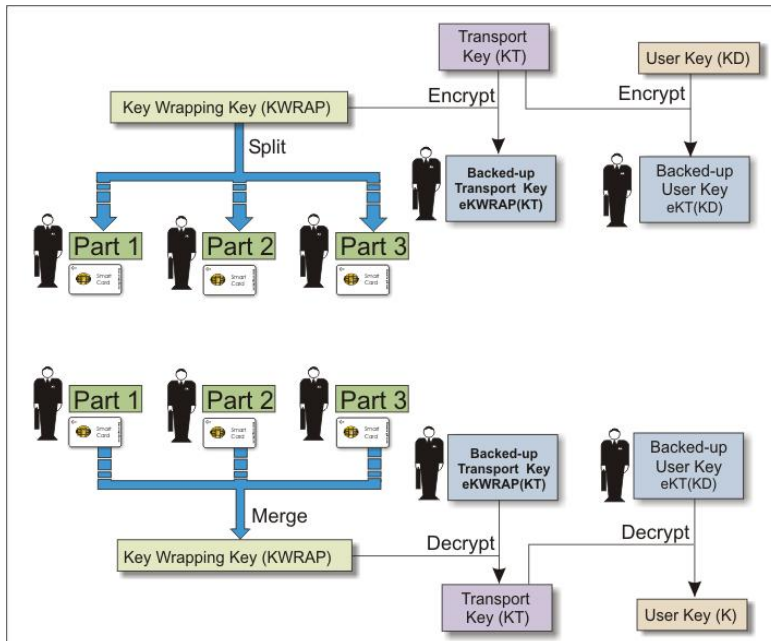
*As defined in A. Shamir – *How to Share a Secret*, Communications of the ACM, Vol 22, no. 11, November 1979, pp 612-613.

A Typical Key Backup and Recovery Scheme

An overall backup scheme typically combines the multiple custodians and single custodian methods. In this scenario, a wrapping key (KWRAP) is generated and backed up using the multiple custodians method. The KWRAP key would then be used to back up other keys using the single custodian method.

This key backup scheme is illustrated in "A typical key backup and recovery scheme" below.

Figure 1: A typical key backup and recovery scheme



Key Attributes

It is important to select key attributes appropriate for the intended purpose while ensuring compatibility with the intended backup scheme.

The standard PKCS #11 key backup method requires that the *wrapping key* (a key used to back up another key) has the CKA_WRAP attribute set to **true**. It also requires that the *extractable key* (the key to be backed up) has the CKA_EXTRACTABLE attribute set to **true**. These attributes may be chosen arbitrarily by the application. Therefore, once a key is marked as extractable, any wrapping key may be used to back it up. It is thus possible for an attacker to introduce a known-value wrapping key, back up the target extractable key and then decrypt it offline.

To defend against this type of attack, SafeNet ProtectToolkit-C uses an alternate key backup method. This extension allows the wrapping key to have its CKA_EXPORT attribute set to **true** and the extractable key to have its CKA_EXPORTABLE attribute set to **true**. In this case, the wrapping key is known as the *export key* and the key to be backed up is the *exportable key*, and only the Security Officer may set the CKA_EXPORT attribute to **true**.

The key backup procedures described below are valid for both the PKCS #11 standard attributes and the extended SafeNet ProtectToolkit-C attributes. The extension method is recommended to mitigate the threat described above.



Note: When the export/exportable procedure is used with the multiple custodians method, the token Security Officer must be present to create the custodian export keys.

The CKA_EXPORT attribute of any key is set to **false** when the key is imported. Therefore, after it is restored, it cannot be used again. The user should create a new export key to create a new backup batch of exportable keys.

Prior to key backup, the extractable keys (selected for backup) must have the correct attributes set to allow the export. The table below details the key attribute settings. Also shown are the attribute settings required to enable use of a key as a wrapping key.

Attribute	Wrapping/Export Key	Extractable/Exportable Key
CKA_MODIFIABLE	FALSE	-
CKA_SENSITIVE	TRUE	-
CKA_WRAP	TRUE	-
CKA_EXPORT	TRUE ¹	-
CKA_UNWRAP ³	TRUE ¹	-
CKA_EXTRACTABLE	-	TRUE ¹
CKA_EXPORTABLE	-	TRUE ¹
CKA_DERIVE	FALSE	-
CKA_ENCRYPT	FALSE	-
CKA_DECRYPT	FALSE ²	-
CKA_SIGN	FALSE	-
CKA_VERIFY	FALSE	-

¹ The user should choose only one of these two attributes. There are two pairs of attributes that must match (be set to **true**):

- CKA_EXPORT for the export key and CKA_EXPORTABLE for the exportable key
- CKA_WRAP for the wrapping key and CKA_EXTRACTABLE for the extractable key

² Wrapping/export keys should not be available for decryption; otherwise the extractable/exportable key may be decrypted directly.

³ The CKA_IMPORT attribute can be used in place of the CKA_UNWRAP attribute. CKA_IMPORT determines whether a key can be used to unwrap encrypted key material. The important difference is that if CKA_IMPORT is set to **true** and CKA_UNWRAP attribute is set to **false**, the only unwrapping mechanism available is CKM_WRAPKEY_DES3_CBC. The error code CKR_MECHANISM_INVALID will be returned for all other mechanisms.

Both the command line tool **ctkmu** and the GUI tool **km** can be used to create keys and change their attributes. See the "[Command Line Utilities Reference](#)" on page 90 and "[Key Management Utility \(KMU\) Reference](#)" on page 139 respectively. More details about key attributes can be found in "[PKCS #11 Attributes](#)" on page 168.

Key Backup Procedure

Prior to attempting a key backup, please ensure that you have:

- a valid key that can be backed up
- a connected smart card reader (if backing up to smart cards)
- sufficient initialized and erased smart cards or disk space to back up the keys

The rules applying to key backup are:

- Attempting a key backup without specifying a wrapping key will result in a multiple custodian backup using a random key (smart cards only).
- When a wrapping key is specified, the unwrapping key used to import a key must be the same as the wrapping key used to export it.
- When using the **ctkmu** command line utility, the *standard scheme* will be used by default for multiple custodian key backups, unless the **-M** parameter is specified. When **-M** is specified, the *N of M scheme* is used.

See ["Available Backup and Recovery Methods" on page 81](#) for more about these methods.

Either the command line utility **ctkmu** or the GUI utility **kmu** can be used for key backup and restoration. These utilities can back up and restore keys from either a disk file or one or more smart cards. Please refer to ["Command Line Utilities Reference" on page 90](#) and ["Key Management Utility \(KMU\) Reference" on page 139](#) for the complete **ctkmu** and **kmu** references respectively.

The following examples use the **ctkmu** command line utility. See ["Importing and Exporting Keys" on page 152](#) for the procedure when using the KMU GUI utility.

Example 1: Using a Wrapping Key

In this example, a key with the label **MyDES2** on slot 2 will be encrypted (wrapped) with the key labeled **MyWRAP1**. The backup data will be written to the disk file named **wrapkey.bin**. This operation will prompt for the User PIN.

```
ctkmu x -s2 -nMyDES2 -wMyWRAP1 fwrapkey.bin
```

Example 2: Using Multiple Custodians and the Standard Key Splitting Scheme

In this example, the key labeled **MyWRAP1** on slot 2 will be backed up to smart cards in slot 4 using the multiple custodians method and the standard scheme. The original key can only be recovered with the co-operation of all custodians.

```
ctkmu x -s2 -nMyWRAP1 -c4
```

The operation will prompt for the User PIN and the number of custodians required (minimum of 2). Each custodian will be prompted to enter and confirm a PIN. The PIN is then used to protect the key component on the smart card.

Example 3: Using Multiple Custodians and the N of M Scheme

Specify the **-M** parameter to use the *N of M scheme*. The original key can be recovered with the co-operation of a user-specified minimum number of custodians N out of the total M.

```
ctkmu x -s2 -nMyWRAP1 -c4 -M
```

After the prompts from Example 2, the utility will prompt for the minimum number of custodians required to recover the key N (minimum of 2, maximum equal to the total number of custodians specified M). Note that N cannot be set equal to M.

See ["Available Backup and Recovery Methods" on page 81](#) for more about the N of M scheme.

Key Restore Procedure

Key restoration is essentially a reversal of the procedures above. When restoring or importing key data, remember:

- When restoring a key held by multiple custodians, all custodians (standard scheme) or the minimum number of custodians (N of M scheme) will have to present their smart card so that the individual key shares can be recombined to form the original key.
- When restoring a key held by a single custodian, the same wrapping key used to encrypt the key must first be available on the token.

Example 1: Single Custodian Key Recovery

In this example, a key will be imported to the token in slot 2 from a disk file named **wrapkey.bin**. It will be decrypted (unwrapped) with the wrapping key **MyWRAP1**. This operation will prompt for the User PIN.

```
ctkmu i -s2 -wMyWRAP1 fwrapkey.bin
```

Example 2: Multiple Custodian Key Recovery

This example will import a key to slot 2 from smart cards held by multiple custodians. When prompted, each custodian in turn must insert their smart card in the card reader designated as slot 4. Custodians will also be prompted for their PIN. This process continues until enough shares have been assembled to enable reconstruction of the key. This operation will prompt for the User PIN.

```
ctkmu i -s2 -c4
```



Note: The command used to recover keys shared between multiple custodians is the same, regardless of which scheme was used (standard or N of M) to split the key. See ["Available Backup and Recovery Methods" on page 81](#) for further information regarding the schemes.

Adding and Removing Slots

The Administrator can use the **ctconf** utility to add or remove slots in SafeNet ProtectToolkit-C



Note: It is not possible to add slots using **ctconf** while other SafeNet ProtectToolkit-C applications are running.

Adding Slots

When adding slots, new slots will have no effect on existing slots.

To add slots:

This example will add 2 slots to the current configuration. The user will be prompted for the Administrator's PIN.

```
ctconf -c2
```

After adding slots, smart card and admin slot numbers will be readjusted automatically. Each token in the newly-created slots must be initialized, as described in ["Token Initialization" on page 42](#).

Removing Slots

Before removing slots from SafeNet ProtectToolkit-C, ensure that the user or an application is not currently accessing a token within that slot. Removal of a slot should only be undertaken after ensuring that the contained token and objects are no longer in use.

To remove slots:

This example will permanently remove slot 2 from SafeNet ProtectToolkit-C. The user will be prompted for the Administrator's PIN.

```
ctconf -d2
```

Re-initializing a Token

Re-initialization of a token is generally performed when the objects contained on that token are no longer being used or the owner of those objects is no longer available to access them. After re-initialization the token may be reused for a different application.

Re-initialization of a token can only be performed by the slot Security Officer.



Note: Re-initialization of a token will erase all objects and user data contained on that token and set a new user PIN.

To re-initialize a token:

```
ctkmu t -s1
```

This example will erase the token on slot 1 and initialize it with a new User PIN and a new label. The user will be prompted for the slot's SO PIN.

Connecting and Removing Smart Card Readers

To speed application startup, the last detected smart card reader information is cached. If the configuration of the attached smart card readers changes, the user must explicitly tell the system to query for changes in peripheral devices. This applies both to connecting and removing smart card readers.

Use the command **ctconf -q** or **ctconf --query** to perform a full device scan on all available serial ports. Alternatively, a system reboot or an HSM reset will initiate a full detection cycle on the next application startup.

Using Transport Mode to Avoid a Board Removal Tamper

Transport mode allows the HSM hardware to be removed from the host system PCI bus without causing a board removal tamper condition. A board removal tamper will remove all sensitive material from the HSM, including the HSM configuration, keys and certificates.

Only the Administrator can set the required transport mode on the HSM.

Use the command line utility **ctconf** with the **-m** option.

To set the Transport Mode:**ctconf -m2**

The numeric value following the **-m** switch will set the transport mode to one of the following:

0	No Transport Mode (Default) – to be applied when HSM is installed and configured. This mode will tamper the HSM if removed from the PCI bus.
1	Single Transport Mode – HSM will not be tampered after removal from the PCI bus. HSM will automatically change to No Transport Mode the next time the HSM is reset or power is removed and restored.
2	Continuous Transport Mode – HSM will not be tampered by being removed from the PCI bus.



Note: Transport Mode does not entirely disable the tamper response mechanism. Any attempt to physically attack the HSM will still result in a tamper response.

Adjusting the HSM Clock

The HSM hardware's internal clock may occasionally need to be adjusted, due to clock drifts and other timing differences between the HSM and the host system. Only the Administrator can perform this task.



Note: The HSM clock value cannot be specified directly. It is only possible to synchronize the HSM clock with the host system clock.

To adjust the HSM clock:

1. Verify or set the correct time on the host system.
2. From a command prompt, type:

ctconf -t

Changing Secure Messaging Mode

See ["Secure Messaging" on page 30](#).

Managing Session Key Rollover

See ["Messaging Mode Configuration" on page 31](#).

Using the System Event Log

SafeNet ProtectToolkit-C maintains a system event log as a means of tracking serious hardware or operational faults, tamper events, and self-test error information.

Viewing and Interpreting the Event Log

Each time a self-test fails, an unexpected event occurs at run-time, or a tamper occurs, information about the event is recorded to the event log. There can be up to 1024 events in the event log.

Event records are written sequentially and labeled chronologically. If the date and time of a later entry is stating earlier than the entry preceding it, the real-time clock or audit information has likely been altered.

See ["Event Log Error Types" on page 165](#) for a complete list of possible error code values that may be recorded in the event log.

To view the event log:

From a command prompt, type:

```
ctconf -e
```

Purging the Event Log

When the event log is full, the HSM will no longer store new event records. The event log will then need to be purged.

The event log cannot be purged until it is full.

To purge the event log:

From a command prompt, type the following:

```
ctconf -p
```

Updating Firmware

The SafeNet ProtectToolkit-C firmware that operates on the HSM hardware can be upgraded to newer versions via a secure upgrade facility. The *firmware upgrade* can only be performed by the SafeNet ProtectToolkit-C administrator using the **ctconf** command line utility. This facility will only allow firmware versions that have been digitally signed by SafeNet.



Note: Depending on the security policy in place, the HSM may perform a soft-tamper before the upgrade process is executed. This tamper will erase all key and configuration data on the HSM. Please see ["Security Policies and User Roles" on page 68](#) for more information on security policies.

Firmware upgrades are distributed in the form of a digitally-signed file.

Prior to performing a firmware upgrade, ensure that:

- All important user data and keys have been backed up
- The current HSM configuration has been noted
- All applications using the HSM have been closed

The HSM firmware is upgraded by entering the following at a command prompt:

```
ctconf -g<filename>
```

where *<filename>* refers to the name of the firmware upgrade file. The user will be prompted for the Administrator password.

Notification of the firmware upgrade's success or failure will be displayed.

Following an upgrade, normal operation of SafeNet ProtectToolkit-C may be resumed.

Tampering the HSM

It may be necessary to tamper the HSM at the end of its lifecycle, or after any other security-sensitive event requiring all stored data to be immediately destroyed.

A tamper formats the secure memory of the HSM, erasing all configuration and user data.

Due to the highly destructive nature of this action, only the Administrator may tamper the HSM. It also requires that all sessions have been closed and that no user is currently accessing the HSM.

To tamper the HSM:

From a command prompt, type:

ctconf -x

The Administrator will then be prompted for their PIN and to confirm the action. Notification of success or failure will be displayed.

Installing a Functionality Module

SafeNet ProtectServer HSMs support development of custom functionality, which affects the hardware's internal processing.

Functionality modules can be developed with the aid of a Functionality Module Software Development Kit (FM-SDK), which can be purchased separately from Gemalto.

Each functionality module is distributed with a certificate so its identity can be verified. This certificate will need to be placed in the admin token by the administrator.



Note: Before proceeding, please ensure that your firmware supports FM functionality. You can check this by typing **ctconf** from a command prompt. If you have an older version of the firmware, contact Gemalto about upgrading it.

To install a functionality module:

To install an FM named **fmFile.fm**, using a verification certificate named **certname.cert**, enter in a command prompt:

ctfm i -lcertname -ffmFile.fm

Command Line Utilities Reference

This chapter contains reference material for the command line utilities provided with SafeNet ProtectToolkit-C and is applicable to both the *Administrator* and normal *User*. The following utilities are defined:

- ["CTCERT" on the next page](#)
- ["CTCHECK" on page 100](#)
- ["CTCONF" on page 104](#)
- ["CTFM" on page 108](#)
- ["CTIDENT" on page 111](#)
- ["CTLIMITS" on page 122](#)
- ["CTKMU" on page 114](#)
- ["CTPERF" on page 125](#)
- ["CTSTAT" on page 129](#)

CTCERT

Certificate Management Utility for the SafeNet ProtectToolkit-C environment.

Synopsis

ctcert	c	Generate certificate with existing keys and self-sign -l<label> [-s<slot>] [-b<YYYYMMDDhhmmss[Z]>] [-d<duration<h d m y>>] [-e<YYYYMMDDhhmmss[Z]>] [-x<name>] [-S<mechanism>]
		Generate certificate with existing keys and sign with existing key -l<label> -c<label> [-s<slot>] [-b<YYYYMMDDhhmmss[Z]>] [-d<duration<h d m y>>] [-e<YYYYMMDDhhmmss[Z]>] [-x<name>] [-S<mechanism>] [-i<slot>]
		Generate certificate with new keys and self-sign -l<label> [-s<slot>] [-b<YYYYMMDDhhmmss[Z]>] [-d<duration<h d m y>>] [-e<YYYYMMDDhhmmss[Z]>] [-x<name>] [-S<mechanism>] -k [-t<type>] [-C<curve_name>] [-z<bits>]
		Generate certificate with new keys and sign with existing key -l<label> -k -c<label> [-s<slot>] [-b<YYYYMMDDhhmmss[Z]>] [-d<duration<h d m y>>] [-e<YYYYMMDDhhmmss[Z]>] [-x<name>] [-S<mechanism>] [-i<slot>] [-t<type>] [-C<curve_name>] [-z<bits>]
	i	Import certificate -f<file> [-s<slot>] -l<label>
	l	List certificate [-s<slot>]
	r	Generate certificate request with new keys -l<label> [-s<slot>] [-S<mechanism>] -k [-t<type>] [-C<curve_name>] [-z<bits>]
		Generate certificate request with existing keys -l<label> [-s<slot>] [-S<mechanism>]
	t	Set trusted certificate -l<label> [-s<slot>]
	x	Export certificate or certificate request to file -l<label> -f<name> [-s<slot>]
		Export certificate or certificate request to screen -l<label> [-s<slot>]

Description

The **ctcert** utility provides basic support for the creation of X.509v3 certificates using SafeNet ProtectToolkit-C. The tool's functions include:

- Generation of self-signed certificates and certificates signed with a specified CA key.
- Generation of PKCS #10 certificate requests.

- Listing certificates, certificate requests, and key objects that exist in a specified slot.
- Importing certificates (PEM format).
- Exporting certificates (PEM format).
- Marking certificates as trusted



Note: When operating in WLD/HA mode, this utility should only be used to view the configuration. Any changes to the configuration should be made in NORMAL mode. See ["Operation in WLD Mode" on page 56](#) and ["Operation in HA Mode" on page 57](#).

Commands

Command	Description
c	<p>Generate Certificate</p> <p>This command is used to generate X.509v3 certificates. A number of approaches are available using ctcert. These approaches and the minimum options required are listed below.</p> <p>To generate new keys and self sign:</p> <pre>ctcert c -k -l<label> [options ...]</pre> <p>To generate new keys and sign with a CA key:</p> <pre>ctcert c -c<label> -k -l<label> [options ...]</pre> <p>To use existing keys and self sign:</p> <pre>ctcert c -l<label> [options ...]</pre> <p>To use existing keys and sign with a CA key:</p> <pre>ctcert c -c<label> -l<label> [options ...]</pre> <p>One of the above combinations of options -c, -k, -l is mandatory. All other options allow finer control over ctcert's default actions. A detailed description of each option is provided below.</p> <p>When the -l<label> option is present without the -k (generate new key pair) option, <label> is the label for an existing PKCS #10 certificate request or an existing public key. ctcert first searches for a certificate request with a matching label and uses it to generate the certificate. If a certificate request does not exist, ctcert searches for a public key with a matching label and uses it to generate a certificate. Otherwise, ctcert reports an error.</p>
i	<p>Import Certificate or Certificate Request</p> <p>This command is used to import a new certificate or certificate request object onto the HSM. The object to be imported is PEM-encoded and contained in a text file. To verify the object is PEM-encoded, the first line in the text file should contain one of the following strings.</p> <pre>"-----BEGIN CERTIFICATE-----" "-----BEGIN CERTIFICATE REQUEST-----"</pre> <p>The -l<label> option specifies the label for the newly-imported certificate or certificate request object.</p>
l	<p>List Certificates, Certificate Requests, and Keys</p> <p>This command will list the existing certificates, certificate requests and keys on the specified token.</p>


Command	Description
r	<p>Generate Certificate Request</p> <p>This command generates a certificate request from an existing or newly-generated key pair. For an existing key pair, the -l<label> option specifies the label of the public key. The private key is identified by CKA_ID attribute, which should be the same for key pairs. This is the default behavior for keys generated by SafeNet ProtectToolkit-C. If the public key contains a value for the CKA_SUBJECT attribute, then it will be used for the certificate request object's subject-distinguished name. If this attribute does not exist, ctcert will prompt the user for this information.</p> <p>If a new key pair is being generated, the -l<label> option specifies the label for the new key pair and ctcert will prompt for the certificate request object's subject-distinguished name. The new certificate request object's label will also be set to <label>.</p>
t	<p>Set Trusted Certificate</p> <p>This command will set the CKA_TRUSTED attribute on the specified certificate on the token. The SO is the only user who can set this attribute. The user will be prompted for the token SO PIN.</p>
x	<p>Export Certificate or Certificate Request</p> <p>This command exports a certificate or certificate request object in a PEM-encoded format. The PEM encoding is written to standard output, or the file specified with the -f<file> option.</p> <p>The -l<label> option specifies the object to export. If a certificate object with a matching label exists, it will be exported. Otherwise, ctcert will search for a certificate request with a matching label.</p>

Options

The following options may be used with various commands, as indicated in the Synopsis.

Option	Description
-b<YYYYMMDDhhmmss[Z]>	<p>--cert-begin=<YYYYMMDDhhmmss[Z]></p> <p>Specifies the begin time (notBefore) for a Certificate. Including the Z sets the time as GMT. Otherwise, local time is assumed and is converted to GMT.</p> <p>This option is only valid for the Generate Certificate Command (c), and must be used with either the -b<YYYYMMDDhhmmss[Z]> or -d<integer[h d m y]> option.</p>
-c<label>	<p>--ca-label=<label></p> <p>Specifies a label identifying a CA (private) key used to sign a newly-generated certificate.</p> <p>The <label> can be a label for a certificate associated with the CA key, or the label of the private key itself.</p> <p>This option is only valid for the Generate Certificate Command (c).</p>
-C<curve_name>	<p>--curve-name=<label></p> <p>Specifies which curve to use. Valid values:</p> <p>P-192 (also known as prime192v1 and secp192r1)</p> <p>P-224 (also known as secp224r1)</p> <p>P-256 (also known as prime256v1 and secp256r1)</p> <p>P-384 (also known as secp384r1)</p>

Option	Description
	<p>P-521 (also known as secp521r1) c2nb191v1 c2tnb191v1e or any valid Domain Parameters object label. If -tec is specified, the -C parameter must be included in the command, or ctcert will exit with an error message.</p>
-d <integer[h d m y]>	<p>-cert-duration=<integer[h d m y]> Specifies the duration of a Certificate. Must specify one of: h - hours, d - days, m - months, y - years. May be used with the -b<YYYYMMDDhhmmss[Z]> option. If the -b option is not included, the duration will begin at the moment of creation. This option is only valid for the Generate Certificate Command (c).</p>
-e <YYYYMMDDhhmmss[Z]>	<p>-cert-end=<YYYYMMDDhhmmss[Z]> Specifies the end time (notAfter) for a Certificate. Including the Z sets the time as GMT. Otherwise, local time is assumed and is converted to GMT. This option is only valid for the Generate Certificate Command (c), and must be used with the -b<YYYYMMDDhhmmss[Z]> option.</p>
-f <name>	<p>-import-file=<name> Specifies a text file containing a PEM encoding of a certificate or certificate request object. This option is only valid with the Import Command (i), Export Command (x) or Generate Certificate Request Command (r).</p>
-h, -?	<p>-help Display usage information</p>
-i <slot>	<p>-ca-slot=<slot> Specifies the slot containing the CA signing key identified by the -c<label> option. If the -i<slot> option is not present, the CA key is assumed to be in the slot identified by the -s<slot> option. If -s is not present then the CA key is assumed to exist in slot 0. If the CA signing key has the CKA_SIGN attribute set to FALSE and the CKA_SIGN_LOCAL_ATTRIBUTE set to TRUE, the CA signing key must reside in the same slot as the certificate it is signing. This option is only valid with the -c option.</p>
-k	<p>-key-gen Specifies that a new key pair be generated. The l<label> option specifies the label for the new keys. A key pair with the same label must not already exist.</p>
-l <label>	<p>-label=<label> This option specifies a label for a new or existing certificate request or public key</p>

Option	Description
	object. Refer to the description of each command for relevant details.
-s <slot>	--slot=<slot> Specifies the slot: <ul style="list-style-type: none"> in which a new key pair and a certificate or certificate request will be generated into which a certificate or certificate request will be imported from which keys, certificates, and certificate requests will be listed that contains the certificate or certificate request to be exported
-S <mechanism>	--sig-hash-alg=<rsa_sign_alg> Specifies the RSA signing algorithm for certificate request or certificate generation. Valid mechanisms are: <ul style="list-style-type: none"> SHA1 SHA224 SHA256 SHA384 SHA512 The default is SHA1 . If this option is applied to a DSA or EC key pair and is not SHA1, ctcert will exit with an error message. <div style="border: 1px solid black; padding: 5px; margin-top: 10px;">  Note: ECDSA is used for a certificate and EC is used for a key pair. </div>
-t <type>	--type=<type> Use with the -k option to specify the key type that should be generated. The valid key types are rsa , rsax931 , ec , and dsa . The default is rsa . If -tec is specified, the -C parameter must be included in the command. Otherwise, ctcert will exit with an error message.
-x <name>	--attribute-file=<name> Specifies a text file containing certain certificate attributes and extensions. For details on supported attributes and extensions and the format of this file refer to "Certificate Attribute File" below . This option is only valid with the Generate Certificate command (c).
-z <bits>	--size=<bits> Use with the -k option to specify the new key size in bits. The default key size is 1024 bits.

Certificate Attribute File

The certificate attribute file allows the user to specify certain certificate attributes, including extensions, that should be used when generating a new certificate. The supported attributes and extensions are:

- Certificate label

- Certificate serial number
- Certificate issuer-distinguished name
- Certificate subject-distinguished name
- Certificate policies extension with support for a certification practice statement (CPS) or user notice
- Certificate key usage extension

The format for specifying an attribute or extension is:

```
<tag> { <value> , <value> , ... }
```


White space is ignored throughout the file, except where it occurs within a <value> string.

The valid <tags> are:

- **label**
- **serialnumber**
- **issuer**
- **subject**
- **certificatepolicies**
- **keyusage**

The following sections describe the allowed values for each of these tags.


Tag	Description
label	<p>This tag defines the certificate's label and is different from the label specified by the I<label> option. This latter label refers to the key pair for which the certificate is being generated. If this tag is missing in the certificate attribute file, then the certificate label will default to the one specified with the -I<label> option.</p> <p>The label can be any string of ASCII characters. If the label contains multiple words, white space between the words is maintained. If a new line is encountered between words it is replaced by a space. The following example demonstrates how to use this tag:</p> <pre>label { Test Certificate }</pre>
serialnumber	<p>This tag defines the certificate's serial number. To ensure uniqueness, it is only used if the signing key does not have the usage count attribute defined. If this attribute is defined, the current value of the usage count is used as the certificate's serial number. If the usage count attribute is not defined and the serial number is not defined in the certificate attribute file, then ctcert will prompt the user for this information.</p> <p>The following example illustrates the correct use of this tag:</p> <pre>serialnumber { 999999 }</pre>
issuer subject	<p>These tags define the issuer and subject distinguished names and are defined by a set of name/value pairs. The format for an issuer or subject distinguished name is:</p> <pre>issuer subject { CN=<string> , OU=<string> , O=<string> ,</pre>

Tag	Description
	<p>C= <string> }</p> <p>The meaning of each name component in each name/value pair is as follows:</p> <p>CN – Common Name OU – Organizational Unit Name O – Organization Name C – Country Name</p> <p>The following example illustrates a well-formed issuer-distinguished name:</p> <pre>issuer { CN=any string , OU=Testing , O=SafeNet , C=CA }</pre> <hr/> <p> Note: White space is ignored except when it appears between multiple words that constitute the value component of a name/value pair. In the above example, the space between any string in the common name component is preserved.</p> <hr/> <p>For the subject-distinguished name tag, two special strings can be assigned to the CN (Common Name) component. These special strings are serialno and unique.</p> <p>When the serialno string is used, ctcert will replace the serialno string with the HSM's serial number. This can be used to distinguish certificates that belong to specific HSMs.</p> <p>When the unique string is used, ctcert appends the current value of the signing key's usage count to the HSM serial number and replaces the unique string with this value. Thus the unique string will be replaced with a string of the form nnnn-xx, where nnnn is the HSM serial number and xx is the signing key's usage count.</p>
certificatepolicies	<p>This tag identifies a certificate policies extension that defines the policy under which this certificate was issued. The format of a certificate policy extension entry is:</p> <pre>certificatepolicies { oid=oid_string , [critical noncritical ,] [unnotice="<string>" ,] [cps="<string>"] }</pre> <p>The certificate policy is identified by an object identifier (OID) and may contain one of the policy qualifiers cps or unnotice. The cps qualifier string is the URI of the Certification Practice Statement that relates to this policy, and the unnotice qualifier is a string that is included in the certificate as a user notice that relates to the certificate policy. Both the cps and unnotice strings are composed of printable ASCII characters. An object identifier (OID) is defined by a series of numerical labels separated by periods. For example, the OID that identifies a key usage extension within an X.509v3 certificate is written as:</p> <pre>id-ce-keyusage OBJECT IDENTIFIER ::= { 2.5.29.15 }</pre>

Tag	Description
	<p>The critical / noncritical keywords indicate whether this certificate policy extension is critical or not. By default, the certificate policy extension is marked noncritical. Multiple certificate policy extensions may be defined in the certificate attribute/extension file.</p> <p>The following example illustrates a well-formed certificatepolicies extension:</p> <pre>certificatepolicies { oid=1.2.3.45.6.8 , unotice=Test string, critical }</pre>
keyusage	<p>This extension restricts the usage of the public key in the certificate. The format of the keyusage entry is:</p> <pre>keyusage{ <key usage string> , [<key usage string> ,] [critical noncritical ,] }</pre> <p>The <key usage strings> conform to those defined in RFC2459 and acceptable values are:</p> <ul style="list-style-type: none"> • digitalSignature • nonRepudiation • keyEncipherment • dataEncipherment • keyAgreement • keyCertSign • cRLSign • encipherOnly • decipherOnly <p>The critical / noncritical keywords indicate whether this key usage extension is critical or not. By default the key usage extension is marked critical.</p> <p>An example of a well formed keyusage extension is:</p> <pre>keyusage{digitalSignature ,keyCertSign}</pre>

Examples

Example 1	<p>Generate new DSA keys with label "Test" and self sign. This command will prompt for the subject distinguished name.</p> <pre>ctcert c -k -lTest -tdsa</pre>
Example 2	<p>Generate new RSA keys with label "Test" and key size 512 bites, sign with a key that has the label "CA Key".</p> <pre>ctcert c -c"CA Key" -k -lTest -z512</pre>
Example 3	<p>Generate a new ECDSA certificate, a EC key pair, and self-sign using the P-192 curve:</p>

	<pre>ctcert c -tec -CP-192 -lecDSA Cert1 -k</pre> <div>  Note: Use ECDSA for a certificate and EC for a key pair. </div>
Example 4	<p>Use existing keys with label "Test" and use certificate attribute file.</p> <pre>ctcert c -lTest -x certificate_file.txt</pre>
Example 5	<p>Use existing keys with label "Test" and sign with a key with that has the label "CA Key".</p> <pre>ctcert c -c"CA Key" -lTest</pre>
Example 6	<p>Use existing certificate request with a label "Test Cert" and sign with a key that has the label "CA Key".</p> <pre>ctcert c -c"CA Key" -l"Test Cert"</pre>
Example 7	<p>To create a new certificate request with the label "User" and generate new keys (RSA is default)</p> <pre>ctcert r -k -lUser</pre>
Example 8	<p>To export a previously generated certificate request as a PEM object and store this in the file name mycert.txt.</p> <pre>ctcert x -lUser -fmycert.txt</pre>

CTCHECK

SafeNet Cryptoki provider status enquiry utility.

Synopsis

ctcheck	[-a] [-b<string>] [-d<device>] [-f<x s>] [-g<string>] [-h] [-n] [-N] [-s<char>] [-V]
----------------	---

Description

ctcheck lists the status of SafeNet ProtectServer devices (actually, of SafeNet Cryptoki providers) in machine-readable format. This could be used, for example, in automatic monitoring of the devices' health and activity level.

The devices can be local hardware or remote, depending on which Cryptoki provider is used. Normally, the Cryptoki provider is specified by the file pointed to by the symbolic link:

/opt/safenet/protecttoolkit5/ptk/libcryptoki.so

If local hardware is used, the device driver package must be installed and running (check it with **hsmstate** command). If a remote Cryptoki is used, its IP address must be given with the **CT_SERVER** environment parameter.

The exact information printed is determined by the command line options. The globals are always printed, unless the **-N** option is present. By default, the most interesting parameters are printed (use the **-h** option to see the default outputs). The globals and per-device details are controlled separately by simple lists of desired parameters. For example, to print just the device serial numbers, the battery status and the initialization status, you would use a string like this with the **-b** option:

ctcheck -bserialnumber~batterystatus~deviceinitialised

Output format is either in XML format or as a ~ (tilde)-separated list. The XML format should be self-documenting.

The tilde output format (see EXAMPLES) is as follows:

- Lines starting with **#** are comments and identify the fields in the following lines.
- The first non-comment line is the global information.
- Each subsequent non-comment line represents one device.
- Each line of information is a simple list of values each separated by the ~ (tilde) character (or as specified with the **-s** option)



Note: When operating in WLD/HA mode, this utility should only be used to view the configuration. Any changes to the configuration should be made in NORMAL mode. See ["Operation in WLD Mode" on page 56](#) and ["Operation in HA Mode" on page 57](#).

Options

The following options are supported:

Option	Description
-a	--all

Option	Description
	Print all device information (overrides -b options)
-b <string>	<p>--device-details=<string></p> <p><string> specifies what device information to output in a ~ (tilde)-delimited list of parameters. Enclose the string in "quotation marks" or 'apostrophes' to avoid shell interpretation of the separator characters.</p> <p>Parameters available:</p> <ul style="list-style-type: none"> • serialnumber - Serial number of device • model - Device model • devicerevision - Revision of device • firmwarerevision - Revision of firmware on device • ptkcrevision - Revision of SafeNet ProtectToolkit-C on device • deviceinitialised - 0 or 1. 0 may mean tampered. • slotcount - Number of slots on a device. • totalpublicmemory - Total secure memory - bytes or 'UNAVAILABLE'. • freepublicmemory - Available secure memory - bytes or 'UNAVAILABLE'. • freememory - Device's heap space (RAM) available - bytes or 'UNAVAILABLE'. • securitymode - 32-bit value or 'Default (No flags set)' • transportmode - 32-bit value or 'None' • batterystatus - LOW or GOOD • eventlogfull - 0 or 1. • fmsupport - 0 or 1 • batch - Device batch • dateofmanufacture - hh:mm:ss DD/MM/YYYY • clocklocal - hh:mm:ss DD/MM/YYYY (TimeZone) • pcbversion - Revision of PCB of device • fpgaversion - Revision of FPGA of device • externalpins - 32 bit value of external pin status • eventlogcount - Number of entries in log • fmlabel - Label of the FM inside the device • fmversion - Version of the FM inside the device • fmmanufacturer - Manufacturer of the FM inside the device • fmbuildtime - Build time of the FM inside the device • fmfingerprint - Fingerprint (hex string) identifying the FM image) of the FM • fmromsize - Amount of ROM the FM is occupying or 'UNAVAILABLE' • fmramsize - Amount of static RAM the FM is using or 'UNAVAILABLE' • fmstatus - 'Enabled', 'Disabled', 'No FM' or 'ERROR'
-d <device>	<p>--device=device</p> <p>Just print details for device number device (the first device is number 0)</p>
-f <x s>	--format= x s

Option	Description
	Output format: x for XML, s for separator (default)
-g<string>	--global-details=<string> <string> specifies what global information to output in a ~ (tilde)-delimited list of parameters. Enclose the string in "quotation marks" or 'apostrophes' to avoid expansion by the shell. Parameters available: <ul style="list-style-type: none"> devicecount - Number of active devices. applicationcount - Number of applications currently using Cryptoki or 'UNAVAILABLE' totalsessioncount - Number of sessions open on all devices.
-h	--help Display usage information.
-n	--number Just print the number of devices
-N	--noglobals Don't print the global information
-s<char>	--separator=<char> Separator for output (default is ~ (tilde))
-v	--version Print the program version

Diagnostics

The program returns 1 if errors are encountered, else 0.

Examples

The default case:

```
c:\>ctcheck
# global info: devicecount~applicationcount~totalsessioncount~
1~UNAVAILABLE~0~
# device info: serialnumber~model~devicerevision~firmwarerevision~ptkcrevision~d
eviceinitialised~slotcount~totalpublicmemory~freepublicmemory~freememory~securit
ymode~transportmode~batterystatus~eventlogfull~fmsupport~
518687~PSI-E2:PL220~6.00~5.00.06~5.3~TRUE~6~4091776~4054448~86986752~Default (No
flags set)~None~GOOD~FALSE~TRUE~
```

Default XML output:

```
c:\>ctcheck -fx
<?xml version="1.2" encoding="UTF-8"?>
<cryptoki>
  <devicecount>1</devicecount>
  <applicationcount>UNAVAILABLE</applicationcount>
  <totalsessioncount>0</totalsessioncount>
  <device>
    <serialnumber>518687</serialnumber>
```

```

    <model>PSI-E2:PL220</model>
    <devicerevision>6.00</devicerevision>
    <firmwarerevision>5.00.06</firmwarerevision>
    <ptkcrevision>5.3</ptkcrevision>
    <deviceinitialised>TRUE</deviceinitialised>
    <slotcount>6</slotcount>
    <totalpublicmemory>4091776</totalpublicmemory>
    <freepublicmemory>4054448</freepublicmemory>
    <freememory>86953984</freememory>
    <securitymode>Default (No flags set)</securitymode>
    <transportmode>None</transportmode>
    <batterystatus>GOOD</batterystatus>
    <eventlogfull>FALSE</eventlogfull>
    <fmsupport>TRUE</fmsupport>
  </device>
</cryptoki>

```

No globals, XML output, only list serial number and battery status:

```

c:\>ctcheck -Nfx -b"serialnumber~batterystatus"
<?xml version="1.2" encoding="UTF-8"?>
<cryptoki>
  <device>
    <serialnumber>518687</serialnumber>
    <batterystatus>GOOD</batterystatus>
  </device>
</cryptoki>

```

See Also

An **awk**(1) script called **ctalarm**(1m) is distributed with this program (not available for Windows) that post-processes the output of **ctcheck**(1m), decides if parameters are within site-specific limits and prints out an appropriate message. If parameters are not within limits, appropriate notices, warning or alarms can be raised. The script must be customized to the needs of the monitoring software being used and is provided as an example.

CTCONF

Configuration utility for the SafeNet ProtectToolkit-C environment.

Synopsis

ctconf	<pre>[-a<device>] [-b<name>] [-c<slots>] [-d<slot>] [-e] [-f<flags>] [-g<file>] [-h] [-i<file>] [-j<file>] [-k<file>] [-l] [-m<mode>] [-n<slot>] [-p] [-q] [-r<slot>] [-s] [-t] [-v] [-x] [--rtc-adj-access-control-rule=<secs>:<count>:<days>] [--rtc-adj-access-control=<0 1>]</pre>
---------------	--

Description

The **ctconf** utility is used to configure the operating parameters for SafeNet ProtectToolkit-C.

By default, **ctconf** will report configurable settings for the first device found. Some options are only applicable to either the hardware or software implementation of SafeNet ProtectToolkit-C.



Note: When operating in WLD/HA mode, this utility should only be used to view the configuration. Any changes to the configuration should be made in NORMAL mode. See ["Operation in WLD Mode" on page 56](#) and ["Operation in HA Mode" on page 57](#).

Options

The following options are supported:

Option	Description
-a<device>	--device-number=<device> Use the admin token on the specified device
-b<name>	--fm-cert=<name> FM validation certificate
-c<slots>	--create-slots=<slots> Create slots new User slots
-d<slot>	--delete-slot=<slot> Delete and remove User slot with ID slot (You cannot delete the admin slot).
-e	--event-log Prints the event log on stdout
-f<flags>	Configures security flags. Security flags are used to implement security policies. Multiple flags may be set simultaneously. For example the command: ctconf -ftu would set both the t and the u flags. When flags are set, any flags set previously are cleared. Setting ctconf -f0 clears all the flags and places the device in SafeNet Default Mode (no flags set). This security policy is described in the "Typical Security Policies"

Option	Description
	<p>section "SafeNet Default Mode" on page 70.</p> <p>Use other flags values to set flags as follows:</p> <ul style="list-style-type: none"> a FIPS Algorithms Only c No Public Crypto d DES Keys Even Parity Allowed e Entrust Ready F FIPS Mode (equivalent to -facIntu) i Increased Security Level I Mode Locked n No Clear PINs N Full Secure Messaging Encryption p Pure PKCS11 t Tamper Before Upgrade u Auth Protection U Full Secure Messaging Signing E User Specified EC Parameters allowed w Weak PKCS#11 Mechanisms <p>Each of these flags is fully described in "Security Flag Descriptions" on page 73.</p>
-g<file>	<p>--upgrade-fw=<file> Upgrade firmware with file</p>
-h	<p>--help Display usage information</p>
-i<file>	<p>--integrity-fw=<file> Verify the authenticity/integrity of a firmware file by specifying its filename.</p>
-j<file>	<p>--download-fm=<file> Download FM module file</p>
-k<file>	<p>--validate-fm=<file> Validate FM module file</p>
-l<fmid>	<p>--delete-fm --disable-fm --fmid=<fmid> Disable/delete an FM module. <fmid> specifies the FM ID in hex format.</p>
-m<n>	<p>--mode=<n> Set the transport mode for the HSM. The following transport modes can be set with <n>:</p> <ul style="list-style-type: none"> 0 No Transport Mode (Default) – to be applied when HSM is installed and configured. This mode will tamper the HSM if removed from the PCI bus.

Option	Description
	<p>1 Single Transport Mode – HSM will not be tampered after removal from the PCI bus. HSM will automatically change to No Transport Mode the next time the HSM is reset or power is removed and restored.</p> <p>2 Continuous Transport Mode – HSM will not be tampered by being removed from the PCI bus.</p> <p>See "Using Transport Mode to Avoid a Board Removal Tamper" on page 86.</p>
-n <slot>	<p>--init-token=<slot> Initialize the token in the specified slot</p>
-p	<p>--purge-log Purge event log. Note that a purge cannot be done until the event log is full.</p>
-q	<p>--query Query peripheral devices. Check all available serial ports, and attempt to activate drivers for the connected devices.</p>
-r <slot>	<p>--reset-token=<slot> Reset existing token in specified slot</p>
-s	<p>--fm-info Display FM module information</p>
-t	<p>--time-set Synchronizes the HSM's internal clock with the host system. This command is only valid when the RTC Status is either HSMADM_RTC_UNINITIALIZED or HSMADM_RTC_STAND_ALONE. Refer to the <i>SafeNet ProtectToolkit-C Programmers Guide</i> (in the Appendix titled <i>HSMAdmin.h Library Reference</i>).</p>
-v	<p>--verbose Display extended status information</p>
-x	<p>--tamper This will cause the Key Store memory on the HSM to be erased (as if tampered) and made ready for re-initialization. The -x option is only available on hardware-based SafeNet ProtectToolkit-C implementations.</p>
--rtc-adj-access-control-rule= <secs>:<count>:<days>	<p>This option sets the rule for RTC Adjustment Access Control. The RTC Adjustment Access Control Rule specifies the guard parameters that control RTC modification. If modification of the RTC is attempted outside of these guard parameters, it will fail.</p> <p>secstotal: amount of deviation (in seconds) within a guard duration. Range: 1-120</p> <p>counttotal number of adjustment that can be made within the guard duration. Range: 0-no maximum. 0 denotes that unlimited adjustments can be made.</p> <p>days: the guard duration in number of days. Range 1-12.</p>

Option	Description
	<p>The separator ':' is a compulsory argument. However, the values for <secs>, <count> and <days> can be NULL. A NULL equates to no modification.</p> <p>For example:</p> <pre>ctconf --rtc-adj-access-control-rule=12:0:1 ctconf --rtc-adj-access-control-rule=12:: ctconf --rtc-adj-access-control-rule=::4</pre> <p>Use ctconf -v to display the current settings for the RTC Adjustment Access Control Rule.</p>
<pre>--rtc-adj-access-control=0 1</pre>	<p>RTC Adjustment Access Control can be enabled once the RTC Adjustment Access Control Rule has been set.</p> <p>When RTC Adjustment Access Control is enabled, the functions provided by the HSMAdmin API (refer to the <i>SafeNet ProtectToolkit-C Programmer's Guide</i>) are governed by the RTC Adjustment Access Control Rule.</p> <p>By disabling RTC Adjustment Access Control, unlimited adjustments to the RTC may be performed.</p> <p>ctconf may be specified with both the --rtc-adj-access-control-rule and --rtc-adj-access-control command line parameters simultaneously.</p> <p>The RTC Adjustment Access Control Rule is given precedence over RTC Adjustment Access Control. Use ctconf -v to display the current settings for the RTC Adjustment Access Control Rule.</p>

CTFM

Functionality Module Management utility for the SafeNet ProtectToolkit-C environment.

Synopsis

ctfm	d	Delete FM [-a<device> -A]
	i	Import FM -l<certLabel> -f<fmFile> [-a<device> -A] [-c<certFile>]
	q	Query FM status [-a<device> -A]
	v	Verify an FM file -f<fmFile> -l<certLabel> [-a<device> -A] [-c<certFile>]

Description

The **ctfm** utility is restricted to use by the SafeNet ProtectToolkit-C administrator, to manage functionality modules on ProtectServer devices (HSMs).

With this tool it is possible to:

- Load a new FM (if an FM is already loaded, it is overwritten)
- Delete an FM so it becomes inactive
- Query the status of an FM (if any)
- Verify an FM file is correctly signed

In each case, the operation may apply to all HSMs or an individually-specified HSM.

By default, **ctfm** will report the FM state for the first device found.

The device Administrator PIN and Admin SO PIN must be initialized in order to run these commands. The **ctfm** utility will prompt the operator for new PINs if they are not initialized.

When the commands are executed, they may require the Admin PIN or Admin SO PIN. The utility will prompt the operator for the values (unless the values have been previously entered during execution of the same command).

Event log entries are created when FMs are loaded or disabled. To create event logs correctly, the HSM RTC should be initialized. If the **ctfm** utility detects that the RTC is not initialized, it will request approval to initialize the HSM RTC to match the system clock.

To load an FM, a trusted certificate must be present in the Admin Token of the HSM. Usually, a PEM-encoded certificate file is provided with the FM image file. If the utility detects that the certificate is not present, it will import the certificate from the file into the Admin Token and set it to *Trusted*.



Note: When operating in WLD/HA mode, this utility should only be used to view the configuration. Any changes to the configuration should be made in NORMAL mode. See ["Operation in WLD Mode" on page 56](#) and ["Operation in HA Mode" on page 57](#).

Commands

Command	Description
d	Delete FM This command is used to delete an FM so it becomes inactive on one or all HSMs.
i	Import FM This command is used to load a new FM onto one or all HSMs (if an FM is already loaded then it is overwritten). Existing FMs do not need to be disabled prior to executing this command. The command searches the device's Admin Token for a certificate label equal to the <certLabel> parameter. If the certificate object is present, the utility will ensure the certificate is set to <i>Trusted</i> . If the certificate object is not present, the utility will attempt to create a Trusted certificate from the contents of the <certFile>. If the <certFile> parameter is not provided, the utility will assume the filename is the <certLabel> with ".cert" appended. For example, if the certificate label is myfm then the utility will search for a file named myfm.cert . The device Administrator PIN (and possibly the Admin SO PIN) will be required.
q	Query FM Status This is the default command, used to query the status of an FM (if any) on one or all HSMs. Use this command to obtain the name, version information and disable status of an FM or to see if an FM is loaded at all. No PINs are required to perform this operation.
v	Verify an FM Signature This command is used to verify that an FM file has been signed correctly, without attempting to download the FM. The device Administrator PIN will be required. The behavior of the <certLabel> and <certFile> parameters is the same as for the Import FM command above.

Options

The following options are supported:

Option	Description
-a<device>	--device-number=<device> Use the Admin Token on the specified device The first device is numbered 0. If this option is absent then the first device is implied.
-A	--all-devices Apply command to all available devices.
-c<certFile>	--fm-cert-file=<certFile> FM validation certificate filename.
-f<fmFile>	--fm-file=<fmFile>

Option	Description
	Name of file holding a new FM.
-h, -?	--help Display usage information.
-l<certLabel>	--fm-cert-label=<certLabel> FM validation certificate object label.

CTIDENT

CTIDENT is a utility for establishing and maintaining trust between devices within the SafeNet ProtectToolkit-C environment.

Synopsis

ctident	gen	Generate new HSM ID key pair [-b] [-f] [-o<so_pin>] <targets>
	trust	Add trust for <peers> to <targets> [-b] [-f] [-o<so_pin>] <targets> [<peers>]
	remove	Remove trust for <peers> to <targets> [-b] [-o<so_pin>] <targets> [<peers>]
	list	List HSM ID keys [-b] [-t<types>] [-a] <targets>
	check	Check HSM ID keys [-b] <targets>

Description

The **ctident** utility establishes trust between devices. This includes operations performed by the Administrative Token SO to establish trust, as well as operations performed by any user to verify trust relationships.

A device trusts another peer when the device holds the peer's HSM Identity public-key in its Administrative Token.



Note: When operating in WLD/HA mode, this utility should only be used to view the configuration. Any changes to the configuration should be made in NORMAL mode. See ["Operation in WLD Mode" on page 56](#) and ["Operation in HA Mode" on page 57](#).

Commands

When specifying the command, the user need only supply the minimum number of characters to uniquely distinguish the command.

Command	Description
check	The check key command check is used to check HSM Identity keys for consistency on the devices specified by the <targets> parameter. Any anomalies will be reported. This command ensures that the peer keys match the device private key they represent, and ensures that all key objects have been created with appropriate security attributes.
gen	The generate key command gen is used to generate the HSM Identity key-pair on the devices specified by the <targets> parameter. If a device already has an identity key a key will not be generated and a warning will be issued, unless the -f parameter is used to force key re-generation. When a key is re-generated, the existing

Command	Description
	<p>key is destroyed BEFORE the new key has been generated to avoid any inconsistencies that could occur with multiple keys.</p> <p>To complete this command, ctident requires the SO PIN of the administrative token. The -o parameter can be used to supply a default SO PIN. Since multiple devices can be targeted with this command, differing PINs may be required for each device.</p> <p>When a default PIN is not provided or if the current PIN is incorrect, the PIN will be prompted for. The batch mode -b parameter can be used to disable PIN prompting.</p>
list	<p>The list key command list is used to list summary information for HSM Identity keys located on the devices specified by the <targets> parameter.</p> <p>The -t parameter restricts the types of keys listed. By default all HSM Identity key types are listed.</p> <p>The -a parameter lists all of the non-sensitive attributes for each key.</p>
remove	<p>The remove key command remove is used to remove HSM Identity keys from the devices specified by the <targets> parameter.</p> <p>The <peers> parameter specifies the peer device keys to remove. If the serial number format is used to identify peers, the peer device need not be available for the command to succeed since peer keys are identified by device serial number.</p> <p>If the <peers> parameter specifies the value local, the devices own local HSM Identity key-pair is removed. This is the only way to have ctident remove a devices own HSM Identity key-pair.</p> <p>To complete this command, ctident requires the SO PIN of the administrative token. The -o parameter can be used to supply a default SO PIN. Since multiple devices can be targeted with this command, differing PINs may be required for each device. When a default PIN is not provided or if the current PIN is incorrect, the PIN will be prompted for. The batch mode -b parameter can be used to disable PIN prompting.</p>
trust	<p>The trust key command 'trust' is used to add peer HSM Identity public-keys to the devices specified by the <targets> parameter.</p> <p>The <peers> parameter specifies one or more peer devices to trust.</p> <p>If a device already has a trusted identity key for a peer, the new key will not be trusted and a warning will be issued, unless the -f parameter is used to force the trust. When forcing trust, the existing peer key is destroyed BEFORE the new key is created to avoid any inconsistencies that could occur with multiple keys.</p> <p>Before trusting a key a number of checks are performed; the public key is checked to ensure it matches the device private key, and both the public and private key objects are checked to ensure they have been created with appropriate security attributes.</p> <p>To complete this command, ctident requires the SO PIN of the administrative token. The -o parameter can be used to supply a default SO PIN. Since multiple devices can be targeted with this command, differing PINs may be required for each device. When a default PIN is not provided or if the current PIN is incorrect, the PIN will be prompted for. The batch mode -b parameter can be used to disable PIN prompting.</p>

Options

Option	Description
<targets>	Specifies a comma-separated list of device numbers. The modifier, sn:<serial> allows device serial numbers to be specified as opposed to device positional numbers. The special value all denotes all devices.
<peers>	Specifies a comma-separated list of peer device numbers. The modifier, sn:<serial> allows device serial numbers to be specified as opposed to device positional numbers. The special value all denotes all devices other than the specific target device on which the command is currently being performed on. The special value local affects the devices own local HSM Identity key-pair and only has effect with the remove command.
-a	--attributes Output all non-sensitive attributes of a key.
-b	--batch Batch mode. Do not prompt for anything, including PINS. If the required information was not supplied on the command line ctident will report an error.
-f	--force Force the command, even if the key already exists.
-o<pin>	--so-pin=<pin> Specifies the security officer (SO) PIN. Use of this operation is a security risk due to the tools command line being visible in the systems process list.
-t<types>	--type=<types> Specifies a comma-separated list of key types. The available key types are: priv — local private keys pub — local public keys peer — peer public keys all — all key types

Exit Status

The **ctident** utility will return a zero(0) exit status when successful. A non-zero exit status is returned on an error. Warnings are *not* treated as errors.

CTKMU

Key Management Utility for the SafeNet ProtectToolkit-C environment.

Synopsis

ctkmu	c	Create key from entered components <code>-t<type> -n<name> -a<attribute> -k<num> [-s<slot>] [-z<size>] [-i<hex_string>] [-p]</code>
		Create key with components <code>-t<type> -n<name> -a<attribute> -k<num> -g [-s<slot>] [-z<size>] [-i<hex_string>]</code>
		Create key without components <code>-t<type> -n<name> -a<attribute> [-s<slot>] [-z<size>] [-i<hex_string>] [-C<curve_name>]</code>
	d	Delete object <code>-n<name> [-s<slot>]</code>
	e	Erase smart card <code>-c<slot></code>
	i	Import key(s) from single-custodian smart card <code>-w<name> -c<slot> [-s<slot>]</code>
		Import key(s) from multi-custodian smart cards <code>-c<slot> [-s<slot>]</code>
		Import key(s) from console <code>-a<attribute> -n<name> -t<type> -w<name> -y [-s<slot>] [-i<hex_string>] [-m] [-z<size>]</code>
		Import key(s) from file <code>-w<name> <filename> [-s<slot>] [-2]</code>
	idp	Import domain parameters <code>-n<name> -t<type> -a<attribute> <filename> [-s<slot>]</code>
	it	Import token <code><filename> [-s<slot>]</code>
	j	Import from PKCS #12 file <code>-n<name> -a<attribute> <filename> [-s<slot>] [-i<hex_string>]</code>
	l	List objects on token(s) <code>[-s<slot> [-v]] [-n<name>]</code>
	m	Modify attributes <code>-n<name> -a<attribute> [-s<slot>]</code>
	p	Initialize or change PINs <code>[-s<slot>] [-O]</code>

rt	Replicate token -d <slotlist> [-s <slot>]
s	Smart card status -c<slot>
t	Initialize/re-initialize token [-s<slot>] [-l<label>]
x	Export key(s) to single-custodian smart card -w<name> -c<slot> [-s<slot>] [-3] [-n<name>]
	Export key(s) to multi-custodian smart card -c<slot> [-s<slot>] [-3] [-n<name>] [-M]
	Export key(s) to file -w<name> <filename> [-s<slot>] [-3] [-n<name>]
	Export key(s) to console -n<name> -w<name> -y [-s<slot>] [-m]
	Export key(s) to PKCS #12 file [-s<slot>] --pkLabel --keyCertLabel [--certalgo] [--pkalgo]
xt	Export token -S<serial> <filename> [-s<slot>]

Description

The **ctkm** utility is used for SafeNet ProtectToolkit-C token management. This includes operations required by a token's SO, such as setting user PINs and re-initializing tokens, as well as those operations required by the normal User, such as object management.

A number of commands can be used with the **ctkm** utility to help with key creation, deletion, import, export, as well as PIN change, token initialization and replication.



Note: When operating in WLD/HA mode, this utility should only be utilized to view the configuration. Any changes to the configuration should be made when operating in NORMAL mode. Refer to ["Operation in WLD Mode" on page 56](#) and ["Operation in HA Mode" on page 57](#) for further details.

Commands

Command	Description
c	Create Key This command is used to generate new keys on the specified token. The -a parameter is used to specify the attributes, the -n parameter specifies the key's label and the -t parameter the new key type. "PKCS #11 Attributes" on page 168 contains further information on key attributes. Common

Command	Description
	uses for this command are generation of a random key, import of a split custodian key (using the -k flag), or creation of a split custodian key (using the -g and -k flags). When importing a split custodian key, optionally, a supported PIN pad device can be used (using the -p flag) to ensure that the key components are entered directly to the device.
d	Delete Key This command is used to delete a key on the specified token. This command will permanently destroy the key with the label specified with the -n parameter.
e	Erase Smart Card This command is used to erase a smart card in the specified slot and will leave the smart card in an un-initialized state.
i	Import Key This command is used to import keys previously exported with the export command (see below).
idp	Import Domain Parameters This command is used to store Domain Parameters objects onto a Token. The -s option indicates the slot e.g. -s1 for slot 1 – default is slot 0. The -n option indicates the label of the new object. The -t option specifies the key type, it may be ec or dsa or dh but only ec is supported. The -a option allows attributes to be specified. Only the 'P' private and 'M' Modifiable attributes are allowed. The default attribute if -a option is missing is CKA_PRIVATE=false and CKA_MODIFIABLE=false. The <filename> option specifies a test file that contains the information required to construct the domain parameters.
it	Import Token This command is used to import a token image into the specified token. The -s parameter identifies the token that will be replaced with the imported token image, by default slot 0 is used. The <filename> parameter specifies the token image file to import. To complete this operation, ctkm will prompt for the user PIN of the destination token. When importing into an un-initialized token, ctkm will prompt for the SO PIN of the destination token. If the device is running in FIPS mode, ctkm will prompt for the device administrator PIN of the destination token.
j	Import Private Key This command is used to import a Private Key and a Certificate from a PKCS #12 file format.
l	List Information This command is used to display information on the objects stored on the token in the specified slot. This command will list the actual keys, certificates and other objects, or, if the token is a smart card token previously used with the key export function information on that key backup set.
m	The Modify Attributes command ' m ' is used to toggle the specified attributes. That is, change from TRUE to FALSE and vice versa or add the attribute if it does not exist.

Command	Description
p	<p>The Pin command 'p' is used to initialize the User PIN or to change an existing PIN (either the User or SO PIN) the command will prompt. 'Cannot change the pin for the token in slot 1 as it is not initialized. You can use the command "ctkmu t -s 1" to initialize this token.'</p> <p>If the PIN is initialized the current PIN will be prompted for before the new PIN may be specified. To change the SO PIN, specify the -O option.</p>
rt	<p>The replicate token command 'rt' is used to replicate a source token to one or more destination tokens. The -s parameter identifies the source token to be replicated, by default slot 0 is used. The -d parameter specifies one or more destination tokens to replicate the source token to.</p> <p>If an error occurs replicating to a particular token, an error will be reported and that token will be skipped. This prevents offline or faulty devices from spoiling the replication process for other tokens.</p> <p>To complete this operation, ctkmu will prompt for the user PIN of the source token.</p> <p>When replicating to an un-initialized token, ctkmu will prompt for the SO PIN of the destination token. If the device is running in FIPS mode, ctkmu will prompt for the device administrator PIN of the destination token.</p>
s	<p>The Smart Card status command 's' is used to display information on the smart card token currently inserted in the specified slot. Details of the keys exported to the token will be displayed.</p>
t	<p>The Initialize/Reset Token command 't' allows for existing tokens to be initialized or re-initialized. If the specified token contains an initialized token the current SO PIN will be prompted for before a new Token label may be specified and the token re-initialized. If the token is un-initialized this command will only operate if the 'No clear PINs' flag is not specified for the HSM (otherwise only the Administrator may initialize tokens with the ctconf utility). In this case the new SO PIN and label may be specified. Once the token has been reset or initialized a new user PIN may also be set.</p>
x	<p>The Export Key command 'x' allows for keys to be exported to one or more smart cards or to a file or to the screen.</p> <p>Keys exported to the screen are wrapped with standard algorithm and are suitable for transport to foreign systems. Keys wrapped for smart card or file backup use proprietary algorithms and can only be restored to compliant SafeNet ProtectToolkit-based HSMs.</p> <p>The main difference between the standard and proprietary methods is that the proprietary method wraps all the attributes of the key so that when a key is restored it must contain the same attributes as the original.</p> <p>Keys wrapped for smart card backup may use one of two basic methods; keys may be exported as split custodian in which case they will be encrypted using a randomly generated key which is then split and distributed to a number of smart card tokens. Alternatively a key wrapping key may be specified which will then be used to encrypt the key specified for backup. This encrypted data can then be written to a smart card token or to a file.</p> <p>Please note that if the -j parameter is used to export a private key and certificate to a PKCS#12 file format the following considerations need to be made. Exportable private key types are: RSA, DSA, and ECDSA.</p> <ul style="list-style-type: none"> • If the private key being exported is marked CKA_EXPORTABLE=TRUE and CKA_EXTRACTABLE=FALSE, the toolkit will prompt for Security Officer (SO) to login to perform the export operation. • User performing the PKCS#12 private key export will be asked to provide two (2) passwords (one for Payload and one for HMAC). At this stage the user must take into account which 3rd

Command	Description
	<p>party tools will be used to extract the PKCS#12 file. For example, Microsoft Windows requires that the Payload and HMAC passwords be identical. OpenSSL, however, will extract Key and Certificate exported by ctkm using two different passwords. The user needs to decide which password policy best suits their needs.</p> <ul style="list-style-type: none"> The RC family of encryption algorithms (and others) are prohibited in FIPS mode. ctkm shall reject the command and display a warning message if they are used under this security policy.
xt	<p>The export token command 'xt' is used to export a token for later import to a specific device. The -s <slot> parameter identifies the source token to be exported, by default slot 0 is used. The -S parameter specifies the serial number of the intended device where token import will be later performed. The <filename> parameter specifies the output token image file. To complete this operation, ctkm will prompt for the user PIN of the source token.</p>

Options

Option	Description
-a <attributes>	<p>--attributes =<attributes></p> <p>Specifies attributes for an object / key. Valid attributes are:</p> <p>P CKA_PRIVATE=1 M CKA_MODIFIABLE=1 T CKA_SENSITIVE=1 W CKA_WRAP=1 w CKA_EXPORT=1 I CKA_IMPORT=1 U CKA_UNWRAP=1 X CKA_EXTRACTABLE=1 x CKA_EXPORTABLE=1 R CKA_DERIVE=1 E CKA_ENCRYPT=1 D CKA_DECRYPT=1 S CKA_SIGN=1 V CKA_VERIFY=1 L CKA_SIGN_LOCAL_CERT=1 C CKA_USAGE_COUNT=1 (can only be used with c command)</p>
-c <slot>	<p>--sc-slot-num =<slot></p> <p>Specifies the Smart Card slot to export to or import from.</p>
-C <curve_name>	<p>--curve-name =<label></p> <p>Specifies which curve to use. Valid values are:</p> <ul style="list-style-type: none"> P-192 (also known as prime192v1 and secp192r1) P-224 (also known as secp224r1)

Option	Description
	<ul style="list-style-type: none"> • P-256 (also known as prime256v1 and secp256r1) • P-384 (also known as secp384r1) • P-521 (also known as secp521r1) • c2nb191v1 • c2tnb191v1e • or any valid Domain Parameters object label <p>If -tec is specified, the -C parameter must be included in the command otherwise ctcert will exit with an error message.</p>
-d <slotlist>	<p>--dest =<slotlist></p> <p>Specifies a comma-separated list of tokens identified by slot number. The special value all denotes all initialized tokens with a token label identical to the source token label and where trust has been established between the devices.</p>
<filename>	Specifies a file to be created for export or used to import a key, certificate, token, or set of domain parameters.
-g	<p>--gen-comp</p> <p>Generate key components.</p>
-h, -?	<p>--help</p> <p>Display usage information.</p>
-j	<p>--pkcs12</p> <p>Export to PKCS#12 format.</p> <p>--pkLabel</p> <p>Private Key to be exported to PKCS#12 file.</p> <p>--keyCertLabel</p> <p>Certificate Label to be exported to PKCS#12 file.</p> <p>--pkalgo</p> <p>Private Key Encryption Algorithms. This parameter is optional. The default setting is DES3. Possible settings are: RC4_128, RC4_40, DES3, DES2, RC2_128, RC2_40. Note that if FIPS mode is ON, then none of the algorithms in the RC family are allowed.</p> <p>--certalgo</p> <p>Certificate Encryption Algorithm. This parameter is optional. In FIPS mode the default setting is DES3. If FIPS mode is OFF, the default setting is RC2_40. Possible settings are: RC4_128, RC4_40, DES3, DES2, RC2_128, RC2_40.</p>
-k <numb>	<p>--num-comp =<numb></p> <p>Number of key components required to be entered or number to be generated (when -g parameter is specified).</p>
-l <label>	<p>--label =<label></p> <p>Specify label.</p>
-m	--multi-part

Option	Description
	Do a multi-part key entry for console import/export.
-M	--NofM Causes the <i>N of M scheme</i> to be used for a multiple-custodians backup. This means that the key is split in such a way that the original key may be recovered with the co-operation of <i>any</i> of the custodians with a user specified, minimum number of custodians being required.
-n<name>	--name =<name> Name of the object to operate on.
-O	--SO-PIN Change the Security Officer PIN. Used with the change PIN command.
-P	--pinpad Use a supported PIN pad device for entering key components.
-s<slot>	--slot-num =<slot> Specifies the slot to operate on. Default is 0 (zero), however must be specified when using the I command and -v option for Slot 0.
-S<serial>	--serial =<serial> Specifies the device serial number.
-t<type>	--type=<type> The type of key to create. Options are: aes des des2 des3 rc2 rc4 cast idea seed rsa dsa ec .
-v	--verbose Displays the attributes that ctkm may change.
-w<name>	--wrap-key =<name> Name of the key used to wrap or unwrap.
-y	--console Import/Export using the console.
-z<size>	--size=<size> Size of the key to create/import (for AES, RC2, RC4, CAST, RSA, DSA and generic secret).
-2	--Cprov2 Import keys from a Cprov 2 formatted file. This is used when migrating keys from an older Cprov 2 key format to the current format (see "Key Migration from SafeNet ProtectToolkit-C V4.1" on page 172).
-3	--PTKC3 Generate export to smart card and file using the SafeNet ProtectToolkit-C version 3 format. Used when exporting keys to be sent to older style HSMs.

Exit Status

The **ctkmu** utility will return a zero(0) exit status when successful. A non-zero exit status is returned on an error. Warnings are not treated as errors.

CTLIMITS

ctlimits is a utility for establishing and managing usage limits on cryptographic keys within the SafeNet ProtectToolkit-C environment.

Synopsis

ctlimits	ct	Create ticket from offline specification <code>-k <keyspec> -S<serial_no> -i<key_id> -t<tok_label> -l<target_label> [-U<usertype>] [-m<message>] [-d<days>] [-L<limit>] [-s<date>] [-e<date>] [-c<cert_filename>] filename</code>
	pt	Present ticket to HSM <code>filename [-U<usertype>] [-O<objtype>] -k<keyspec> [-i<key_id>]</code>
	up	Apply limit attributes directly <code>-k<keyspec> [-U<usertype>] [-O<objtype>] [-i<key_id>] [-C<count>] [-L<limit>] [-s<date>] [-e<date>] [-c<cert_filename>]</code>
	vk	View key attributes <code>-k<keyspec> [-U<usertype>] [-O<objtype>] [-i<key_id>]</code>

Description

The **ctlimits** utility sets and/or modifies the usage limitation attributes of cryptographic objects within the SafeNet ProtectToolkit-C environment.

The utility will gracefully recognize older firmware and report meaningful error message.

Commands

Command	Description
ct	Create ticket from offline specification This command creates a SET ATTRIBUTES ticket in the file filename. This ticket may be presented to a SafeNet ProtectToolkit-C HSM using the ctlimits pt command. The ticket is signed with the authority of the user type specified by -U option (or the CKU_USER if no -U option is provided). <ul style="list-style-type: none"> The key specified by -k parameter is used to identify the signing key used to sign the ticket. The -k parameter may optionally provide the utility with a pin value. If none is supplied the utility will prompt the operator to enter one. If the -m option is specified then a message, which may be used to identify the ticket, is included into the file containing the ticket. To identify the target object completely all the -l, -t, -S and -i options must be specified At least one of the -c, -L, -s and -e options must be provided In order to indicate the change required. The valid time for the ticket is one day unless the -d option is used to specify a different duration.

Command	Description
pt	<p>Present ticket to HSM</p> <p>This command reads a SET ATTRIBUTES ticket from filename and attempts to find the key in the token indicated by the -l -t and optionally the -i options.</p> <p>If the key object is not found inside the token then the utility will attempt to login as the USER and will search again. In this case the USER pin is required. The -u option can be used to supply the USER pin or if this is not provided then the utility will prompt the operator to enter the USER pin.</p>
up	<p>Apply limit attributes directly</p> <p>This command sets or updates attributes on the target object directly without making an intermediate ticket file. The object must be modifiable.</p> <p>To identify the target object the -l and -t options must be provided. To further identify the target object the -i option may be specified.</p> <p>The target object will have its attributes updated according to the -C, -L, -s, -e and -c options. At least one of these options must be provided.</p> <p>After the command sets the new attributes it will lock the object by setting the CKA_MODIFIABLE to False (in a C_CopyObject operation).</p> <p>If the key object is not found inside the token then the utility will attempt to login as the USER and will search again. In this case the USER pin is required. The -k option can be used to supply the USER pin or if this is not provided then the utility will prompt the operator to enter the USER pin.</p>
vk	<p>View key attributes</p> <p>This command displays the current limits attributes of an object.</p> <p>To identify the target object the -k option must be provided. To further identify the target object the -i option may be specified.</p> <p>If the key object is not found inside the token then the utility will attempt to login as the USER and will search again. In this case the USER pin is required. The -k option can be used to supply the USER pin or if this is not provided then the utility will prompt the operator to enter the USER pin.</p>

Options

Option	Description
-U<user>	<p>--usertype=<user></p> <p>User type creating ticket – may be either SO or USER (default)</p>
-k<keyspec>	<p>--keyspec=<keyspec></p> <p>Specification of a key. The format used is TokenLabel(pin)/KeyLabel, where the pin is optional and TokenLabel may specify slot by number</p> <p>For example:</p> <p>-k MyToken(1234)/MyKey (Pin 1234) or</p> <p>-k MyToken/MyKey (no Pin - utility may prompt for pin)</p> <p>-k SLOTID=2/MyKey</p>
-O<objtype>	<p>--objtype=<objtype></p> <p>Object type of the key. May be secret_key, certificate, public_key, or private_key. The</p>

Option	Description
	default is private_key .
-m <message>	--message=<message> Optional message to add to ticket
-t <tok_label>	--token_label=<tok_label> Label of token containing the target object (may be numeric to refer to token by slot number)
-s <serial_no>	--tok_sno=<serial_no> Serial number of Token containing the target object.
-l <target_label>	--target_label=<target_label> Label of object that is the target of the operation
-i <key_id>	--target_key_id=<key_id> Key ID of object that is the target of the operation. key_id should be in HEX format
-C <count>	--usage_count=<count> Specify CKA_USAGE_COUNT value, 'count' is in decimal format.
-L <limit>	--usage_limit=<limit> Specify CKA_USAGE_LIMIT value, 'limit' is in decimal format.
-s <date>	--start_date=<date> Specify new CKA_START_DATE value for the target object. 'time' format is YYYYMMDD – time is GMT.
-e <date>	--end_date=<date> Specify new CKA_END_DATE value for the target object. 'time' format is YYYYMMDD – the time specified is GMT.
-c <cert_filename>	--cert=<cert_filename> Name of the file containing a public key certificate to be applied to CKA_ADMIN_CERT attribute
-d <days>	--duration=<days> Validity period of ticket in days

CTPERF

Performance reporting utility.

Synopsis

ctperf	<code>[-h] [-b<bytes>] [-c] [-C<curve_name>] [-e] [-i<count>] [-k] [-m<bits>] [-n<mechanism>] [-o<mechanism>] [-p] [-q] [-r] [-R] [-s<slot>] [t<seconds>] [-v] [-x] [-z<name>]</code>
---------------	---

Description

The **ctperf** utility reports on the performance of PKCS #11 cryptographic operations.



Note: This performance measurement is application-dependent, therefore the results are indicative only.

Options

The following options are supported:

Option	Description
-b<bytes>	--block-size=<bytes> Specify the block size to use for the symmetric cipher tests. For example, -b8 specifies 8 bytes, -b8k specifies 8 kilobytes. Default size is 4 kilobytes.
-c	--strict Strict PKCS #11.
-C<curve_name>	--curve-name=<label> Specifies which curve to use. Valid values are: <ul style="list-style-type: none"> P-192 (also known as prime 192v1 and secp192r1) P-224 (also known as secp224r1) P-256 (also known as prime 256v1 and secp256r1) P-384 (also known as secp384r1) P-521 (also known as secp521r1) c2nb191v1 c2tnb191v1e or any valid Domain Parameters object label If a curve name is not specified, the default P-192 is used.
-e	--EMC Runs tests suitable for EMC testing purposes.
-h, -?	--help Display usage information.
-i<count>	--iterations=<count>

Option	Description
	The number of iterations of the performance tests to run. Default is 1 , use -1 to specify an infinite count.
-k	--keygen Generation random keys (default uses fixed keys).
-m<bits>	--modulus=<bits> Modulus bit length.
-n<mechanism>	--exc-mechanism=<mechanism> Mechanisms to exclude from the test. This option may be repeated with additional mechanisms to specify more than one. See the -o option for a list of mechanisms. Default is no mechanisms.
-o<mechanism>	--inc-mechanism=<mechanism> Mechanisms to include in the test. This option may be repeated with additional mechanisms to specify more than one. Default is all mechanisms. (For details and a listing of SafeNet ProtectToolkit-C supported mechanisms please refer to the <i>SafeNet ProtectToolkit-C Programmer's Guide</i> .) The following mechanism tests are supported: <ul style="list-style-type: none"> -o sha1 SHA-1 mechanism -o sha224 SHA-224 mechanism -o sha256 SHA-256 mechanism -o sha384 SHA-384 mechanism -o sha512 SHA-512 mechanism -o md all Message Digest mechanisms -o md5 MD-5 mechanism -o rmd128 RMD-128 mechanism -o rmd160 RMD-160 mechanism -o aes all AES mechanisms -o aes_ecb AES ECB mechanism -o aes_cbc AES CBC mechanism -o aes_mac AES MAC mechanism -o des_all all DES mechanisms -o des_ecb64 DES ECB with fixed buffer size of 54 bytes -o des all single DES mechanisms -o des_ecb single DES-ECB mechanism -o des_cbc single DES-CBC mechanism -o des_mac single DES-MAC mechanism -o des3 all triple-DES mechanisms -o des3_ecb triple DES-ECB mechanism -o des3_ecb64 triple DES-ECB mechanism with fixed length 64 byte buffer

Option	Description
-o des3_cbc	triple DES-CBC mechanism
-o des3_mac	triple DES-MAC mechanism
-o idea	all IDEA mechanisms
-o idea_ecb	IDEA-ECB mechanism
-o idea_cbc	IDEA-CBC mechanism
-o idea_mac	IDEA-MAC mechanism
-o cast	all CAST mechanisms
-o cast_ecb	CAST ECB mechanism
-o cast_cbc	CAST CBC mechanism
-o cast_mac	CAST MAC mechanism
-o seed	all SEED mechanisms
-o seed_ecb	SEED ECB mechanism
-o seed_cbc	SEED CBC mechanism
-o seed_mac	SEED MAC mechanism
-o rc2	all RC2 mechanisms
-o rc2_ecb	RC2-ECB mechanism
-o rc2_cbc	RC2-CBC mechanism
-o rc4	RC4 mechanism
-o rsa_kg	RSA PKCS key generation mechanism
-o rsa_kg_x931	RSA C931 key generation mechanism
-o rsa	most RSA mechanisms
-o rsa_raw_enc	basic RSA public key transform
-o rsa_pkcs_enc	RSA public key enc with PKCS padding
-o rsa_pkcs_ver	RSA private key verify with PKCS padding
-o rsa_raw_dec	basic RSA private key transform
-o rsa_pkcs_dec	RSA private key decrypt with PKCS padding
-o rsa_9796_sign	RSA sign with ISO9796 padding
-o rsa_raw_dec_crt	RSA private key primitive with CRT key
-o rsa_9796_sign_crt	RSA ISO9796 sign, CRT key
-o rsa_ncrt	all RSA mechanisms using non CRT keys
-o dsa_kg	all DSA key generation mechanisms
-o dsa	all DSA mechanisms. That is: <ul style="list-style-type: none"> • dsa_verify • dsa_sign
-o sym	all symmetric algorithm mechanisms
-o asym	all asymmetric algorithm mechanisms
-o ec	all EC DSA operations. That is: <ul style="list-style-type: none"> • ecdsa_kgecdsa_sign • ecdsa_verify • ecdsa_sha1_sign

Option	Description
	<ul style="list-style-type: none"> • <code>ecdsa_sha1_verify</code> <p><code>-o cert_gen</code> invokes certificate-generation and signing tests</p> <p><code>-o dh_kg</code> Diffie-Hellmann key generation mechanism</p> <p><code>-o rng</code> the random number generation mechanism</p> <p><code>-o extra</code> the following:</p> <ul style="list-style-type: none"> • <code>des3_ses_kg</code>: DES3 Session Key Generation/Destruction • <code>des3_tok_kg</code>: DES3 Token Key Generation/Destruction • <code>des3_kw</code>: DES3 Key Wrap • <code>cert_gen</code>: Certificate Generation/Destruction • <code>ses</code>: Session Open/Close • <code>obj</code>: Object Creation/Search/Destroy • <code>login</code>: Login / Logout • <code>xor_dk</code>: XOR Derive Key <p><code>-o mc</code> reading of the monotonic counter object</p>
-p	--dsa-params Parameters generated for DSA.
-q	--quick Quick Keygen (key generation tests not performed).
-r	--random Execute a random selection of the performance tests.
-R	--Random Seed the random number generator. This option should be used with the -r option to generate a unique sequence of tests, otherwise the same pseudo random sequence will be repeated.
-s <slot>	--slot-num=<slot> Specify the slot number to perform test on.
-t <seconds>	--time-period=<seconds> Specify the measurement period. Default is 5 seconds.
-v	--verbose Verbose (provide more information).
-x	--csv Create a CSV (comma-separated variable) file.
-z <name>	--cryptoki-module=<name> Optionally, specify a different cryptoki module to use. May include full path.

CTSTAT

Show status for SafeNet ProtectToolkit-C tokens and objects.

Synopsis

ctstat	[-a] [-b] [-h] [-j] [-m] [-n<name>] [-s<slot>] [-t<name>] [-v]
---------------	---

Description

The **ctstat** utility is used to check the status of a token, determine what state the token is in and what, if any, objects it contains. With no arguments, **ctstat** will provide a summary report of all tokens found.



Note: When operating in WLD/HA mode, this utility should only be utilized to view the configuration. Any changes to the configuration should be made when operating in NORMAL mode. Refer to ["Operation in WLD Mode" on page 56](#) and ["Operation in HA Mode" on page 57](#) for further details.

Options

The following options are supported:

Option	Description
-a	--all Display the status for everything associated with the SafeNet ProtectToolkit-C token.
-b	--attributes Display the attributes associated with a token.
-h	--help Display usage information.
-j	--objects Display all objects associated with a token.
-m	--mechanism Display all mechanisms available on a token.
-n<name>	--object-name=<name> Display the attributes of the specified object.
-s<slot>	--slot-num=<slot> Specify the slot number to display statistics about.
-t<name>	--token=<name> Specify the token name to display statistics about.
-v	--verbose Verbose (provide more information).

Administration Utility (gCTAdmin) Reference

The Administration Utility (**gCTAdmin**) provides a graphical user interface to functions that allow management of the HSM hardware using a PKCS #11- sub-system. The functionality which is provided is identical to that of the command line utility **ctconf** (["Command Line Utilities Reference" on page 90](#)).



Note: The **gCTAdmin** application is a Java-based application. A working Java runtime that supports the Swing user interface must be installed. This application has been tested with JDK 1.2, JDK 1.3 and JDK 1.1 (with Swing installed). The screenshots throughout this manual may vary from platform to platform.



Note: When WLD mode is configured, this utility does not operate.

To start **gCTAdmin** using Microsoft Windows, locate the program folder titled **SafeNet ProtectToolkit C RT** or **SafeNet ProtectToolkit C SDK** in the Windows **Start** menu, and click on the appropriate shortcut. To start the admin utility in a UNIX environment, enter **gctadmin** at the command prompt.

To exit the utility, select **File>Exit** from the menu bar.

Select **Help** from the **Main Menu** for information about the current version of the software.

This chapter contains the following sections:

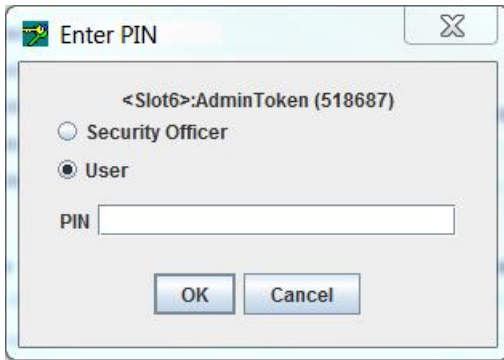
- ["Logging In and Out" below](#)
- ["Main gCTAdmin Interface" on the next page](#)
- ["Slot and Token Management" on page 132](#)
- ["HSM Management" on page 135](#)

Logging In and Out

After starting **GCTADMIN**, the utility will check if the HSM hardware has been initialized.

If the hardware has not been initialized, the utility will prompt the operator to initialize the Admin Token. For full details regarding initial configuration, please refer to ["Cryptoki Configuration" on page 36](#). Initialization is necessary for the Admin SO to create the Administrator user.

If the hardware has been initialized, the operator is prompted for entry of the Administrator PIN.



PIN entry is masked so only the '•' character will be displayed as characters are typed.

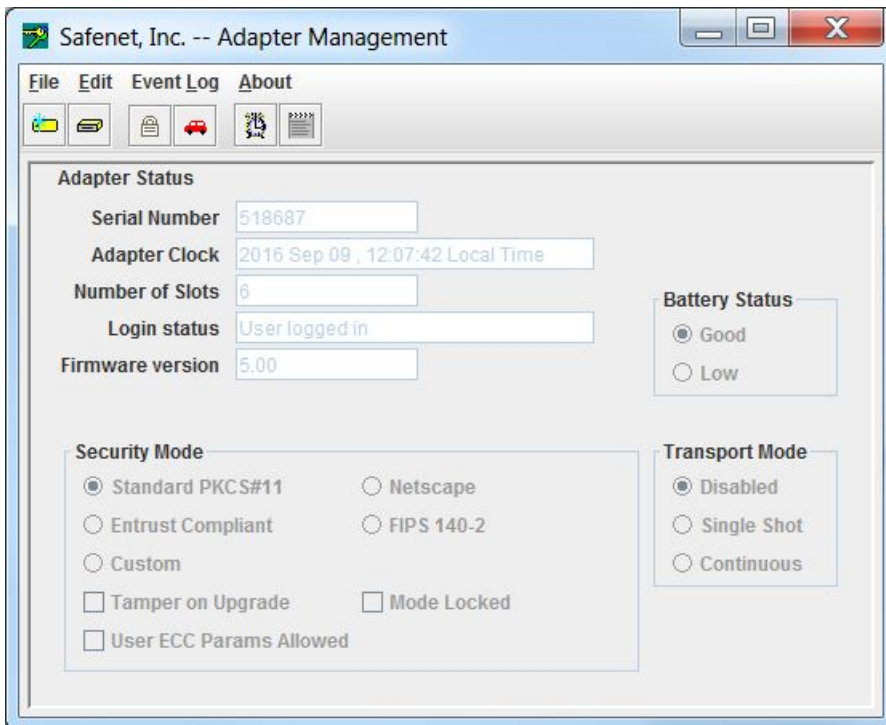
To log out from the main interface, select the **Logout** option from the **File** menu.

Main gCTAdmin Interface

Following a successful login, the main user interface is displayed ("[Main gCTAdmin interface](#)" below). The main interface shows the currently-selected HSM and a variety of its hardware settings.







In a host system containing multiple HSMs, other HSMs can be selected with **File>Select Adapter**. Choosing a different HSM will require a new login.

Figure 1: Main gCTAdmin interface



Toolbar Buttons

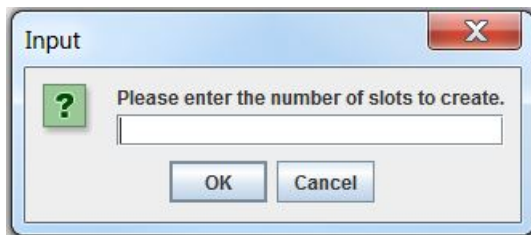
The buttons on the toolbar correspond to the following commands.

Button	Description	Button	Description
	Token Configuration		Transport Mode
	Create Slots		Set the Clock
	Security Mode		View the Event Log

Slot and Token Management

Creating Slots

To create slots on the HSM, select **File>Create Slot**, or click **Create Slots** on the toolbar. A dialog will prompt for the number of slots to be created.

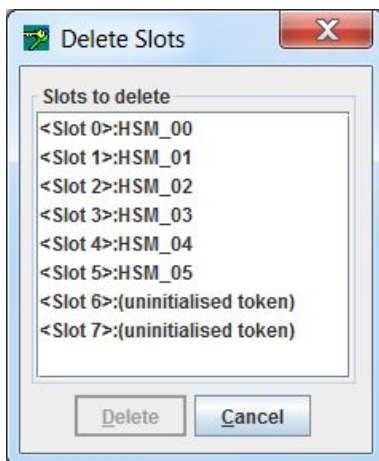


Note: It is not possible to add slots using **GCTADMIN** while other SafeNet ProtectToolkit-C applications are running.

Removing Slots

Before removing slots from SafeNet ProtectToolkit-C, ensure that the contained token and objects are not in use.

To remove a slot, select **File> Delete Slots**. A list of available slots is displayed. Select the slot to delete from the list and click the **Delete** button.



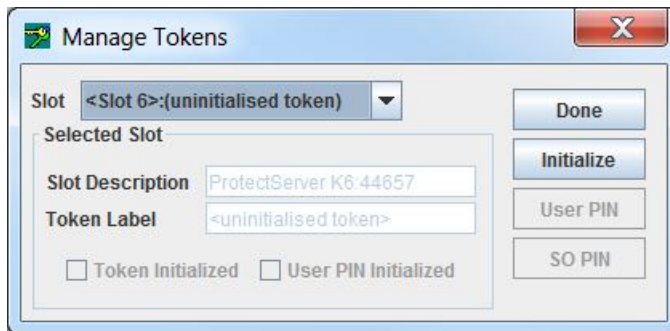


Note: The slot containing the Admin Token cannot be deleted.

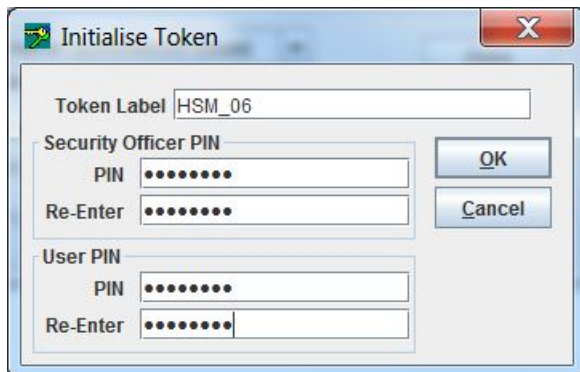
Initializing a Token

The initialization of a token is performed to set the user and token SO PIN.

1. To initialize a token, select **Edit> Tokens...** from the menu to open the **Manage Tokens** dialog.



2. Select an uninitialized token from the slot drop-down box.
3. Click **Initialize**. The **Initialize Token** dialog will prompt for the token label, SO PIN and User PIN. A token is considered initialized after entry of the SO PIN. The User PIN must be set at this time, but will not be required until an application requires storage on that slot.

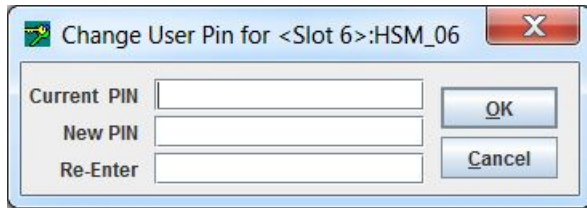


Note: PINs have to be entered twice to confirm correct entry.

4. Click **Done** to exit the *Manage Tokens* dialog.

Setting the Token User PIN

1. To set a token User PIN, select **Edit> Tokens...**
2. Select an initialized token from the slot drop-down box, then click **User PIN**. If the selected token does not have a current User PIN, the dialog will prompt for the SO PIN in order to authorize the creation of the new User PIN.
If the selected token already has a User PIN assigned, the dialog will prompt for the current and new User PIN to be entered.

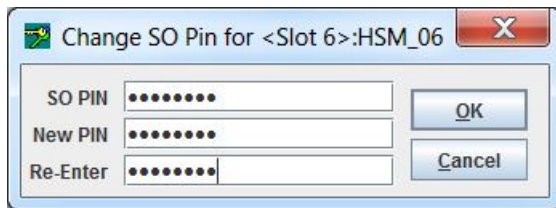


Note: PINs have to be entered twice to confirm correct entry.

- Click **Done** to exit the Manage Tokens dialog.

Setting the Token SO PIN

- To set a token SO PIN, select **Edit>Tokens...**
- Select an initialized token from the slot drop-down box, then click **SO PIN**. The dialog will prompt for the current and new SO PIN to be entered.



Note: Enter PINs twice to confirm correct entry.

- Click **Done** to exit the *Manage Tokens* dialog.

Resetting a Token

A token reset can only be done to initialized tokens. Admin tokens cannot be reset and any attempt to do so will display a warning.



Note: Resetting a token will erase all objects and user data on that token and set a new user PIN.

- Select an initialized token from the slot drop-down box, and then click **Reset** and enter the token SO PIN to open the **Initialize Token** dialog.
- Enter a token label, SO PIN and User PIN. A token is considered initialized after entry of the SO PIN. The User PIN does not have to be set until an application requires storage on that slot.



Note: PINs have to be entered twice to confirm correct entry.

- Click **Done** to exit the *Manage Tokens* dialog.

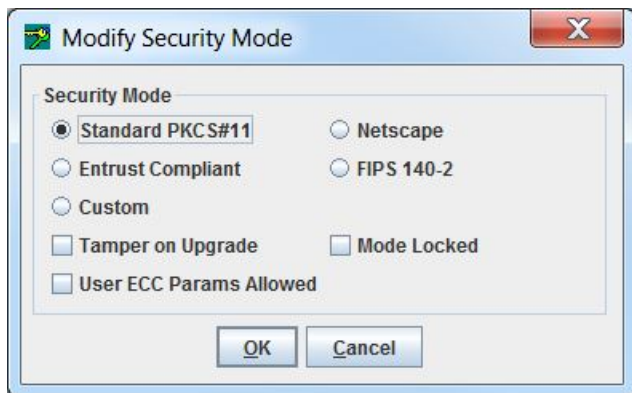
HSM Management

Setting the Security Policy

The most important aspect of SafeNet ProtectToolkit-C administration is choosing the settings, or *Security Policy*, which will determine how SafeNet ProtectToolkit-C can be used. The Administrator is strongly advised to read ["Security Policies and User Roles" on page 68](#), which explains how different settings affect the security and performance of the SafeNet ProtectToolkit-C environment.

To set the HSM security policy:

1. Select **Edit> Security Mode...**
2. Select the required settings from the **Modify Security Mode** dialog box.



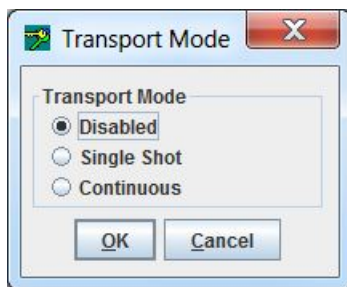
3. Click **OK** to store the selected security policy.

Setting the Transport Mode

The HSM transport mode is used to set the method in which the HSM responds when removed from the PCI bus.

To set the HSM transport mode:

1. Select **Edit> Transport Mode...** to open the **Transport Mode** dialog box.



2. Choose from the following selections:

Disabled	To be applied when HSM is installed and configured. This mode will tamper the HSM if removed from the PCI bus.
Single Shot	The HSM will not be tampered after removal from the PCI bus. HSM will automatically disable

	Transport Mode the next time the HSM is reset or power is removed and restored.
Continuous	The HSM will not be tampered by being removed from the PCI bus.



Note: The transport mode does not disable the tamper response mechanism entirely. Any attempt to physically attack the HSM will still result in a tamper event.

- Click **OK** to set the Transport Mode.

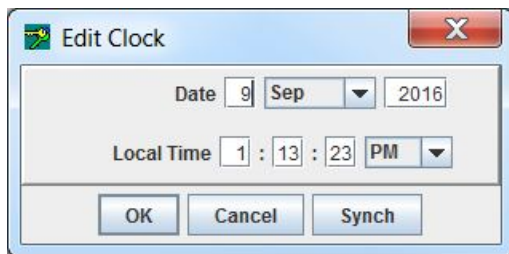
Clock Drift Correction

The HSM hardware's internal clock may occasionally need to be adjusted, due to clock drifts and other timing differences between the HSM and the host system. The clock can be adjusted manually or synchronized with the host system's clock (recommended).

To synchronize the HSM clock:

- Select **Edit> Clock**.

The current value of the HSM clock is displayed.



- Edit the date and time manually, or synchronize the HSM clock to the host clock (recommended) by clicking **Synch**.
- Click **OK** to close the dialog box.

Viewing and Purging the System Event Log

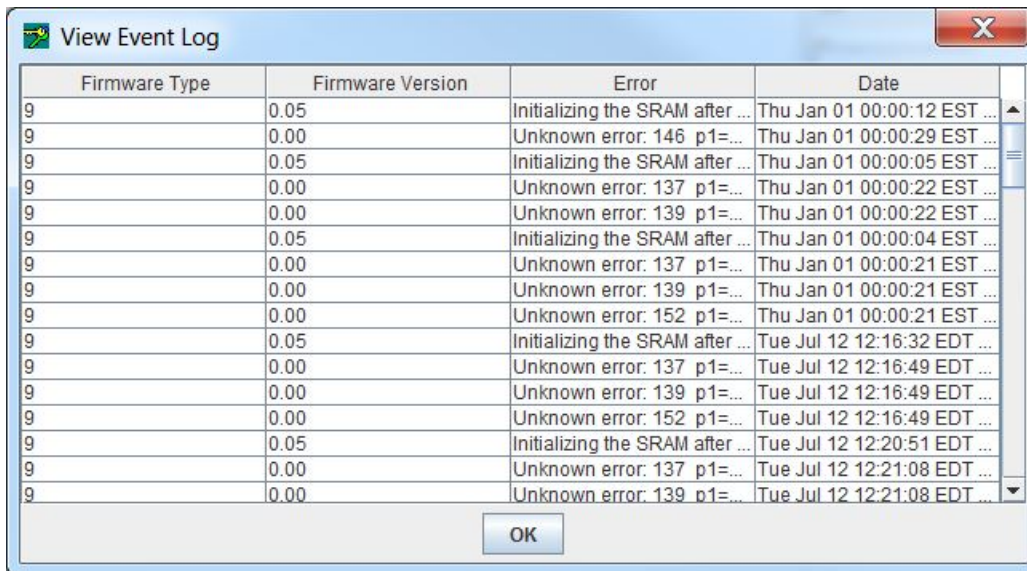
SafeNet ProtectToolkit-C maintains a system event log as a means of tracking serious hardware or operational faults, tamper events, and self-test error information. For full details on what the event log stores and how to interpret its data, please refer to ["Using the System Event Log" on page 87](#).

When the event log is full, the HSM will no longer store new event records and will need to be purged. The event log cannot be purged until it is full.

To view the event log:

Select **Event Log> Event Log View**.

A dialog is shown containing a list of events with columns for "Firmware Type", "Firmware Date", "Error", "Date".



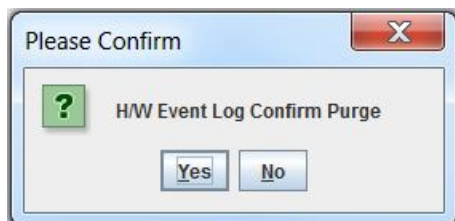
The 'View Event Log' dialog box displays a table of event logs. The table has four columns: Firmware Type, Firmware Version, Error, and Date. The data is as follows:

Firmware Type	Firmware Version	Error	Date
9	0.05	Initializing the SRAM after ...	Thu Jan 01 00:00:12 EST ...
9	0.00	Unknown error: 146 p1=...	Thu Jan 01 00:00:29 EST ...
9	0.05	Initializing the SRAM after ...	Thu Jan 01 00:00:05 EST ...
9	0.00	Unknown error: 137 p1=...	Thu Jan 01 00:00:22 EST ...
9	0.00	Unknown error: 139 p1=...	Thu Jan 01 00:00:22 EST ...
9	0.05	Initializing the SRAM after ...	Thu Jan 01 00:00:04 EST ...
9	0.00	Unknown error: 137 p1=...	Thu Jan 01 00:00:21 EST ...
9	0.00	Unknown error: 139 p1=...	Thu Jan 01 00:00:21 EST ...
9	0.00	Unknown error: 152 p1=...	Thu Jan 01 00:00:21 EST ...
9	0.05	Initializing the SRAM after ...	Tue Jul 12 12:16:32 EDT ...
9	0.00	Unknown error: 137 p1=...	Tue Jul 12 12:16:49 EDT ...
9	0.00	Unknown error: 139 p1=...	Tue Jul 12 12:16:49 EDT ...
9	0.00	Unknown error: 152 p1=...	Tue Jul 12 12:16:49 EDT ...
9	0.05	Initializing the SRAM after ...	Tue Jul 12 12:20:51 EDT ...
9	0.00	Unknown error: 137 p1=...	Tue Jul 12 12:21:08 EDT ...
9	0.00	Unknown error: 139 p1=...	Tue Jul 12 12:21:08 EDT ...

An 'OK' button is located at the bottom center of the dialog box.

To purge the event log:

1. Select **Event Log>Event Log Purge**. A confirmation dialog appears.



2. Click **Yes** to confirm you want to purge the event log.



Note: If the event log is not full, an error is displayed.

Updating HSM Firmware

The firmware that operates on the ProtectServer hardware can be upgraded to newer versions through a secure upgrade facility. This facility will only allow the HSM to be upgraded to firmware versions that have been digitally signed by SafeNet.



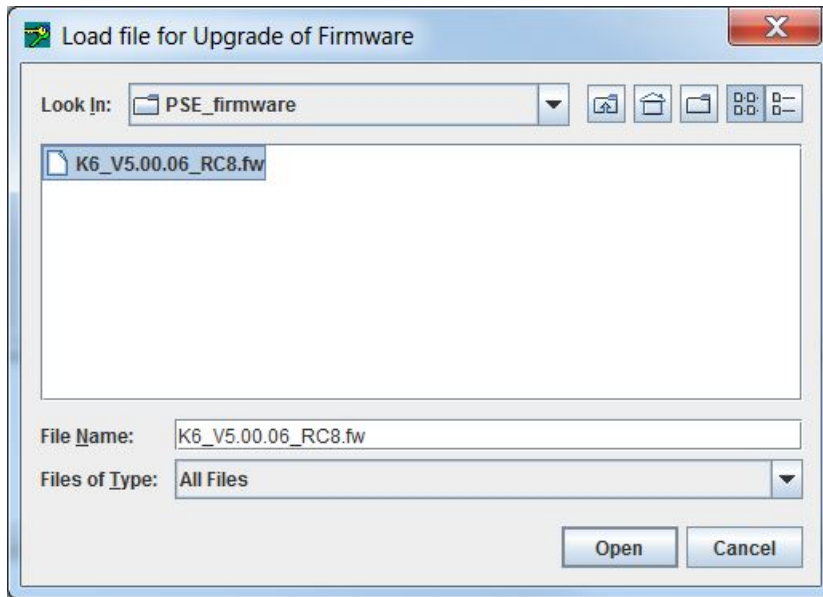
CAUTION: Depending on the active security policy, the HSM might execute a soft tamper before completing the upgrade process. This tamper will erase all key and configuration data on the HSM. See ["Security Policies and User Roles" on page 68](#)

Firmware upgrades are distributed in the form of a digitally-signed file. Before a firmware upgrade, ensure that:

- All important user data and keys have been backed up
- The current HSM configuration has been noted
- All applications using the HSM have been closed

To upgrade the HSM firmware:

1. Select **File> Upgrade Firmware**.
2. Select the firmware upgrade file and click **OK** to continue with the firmware upgrade.



Note: The upgrade process may take up to two minutes to complete. Following the upgrade, a dialog appears, stating the success or failure of the upgrade operation.

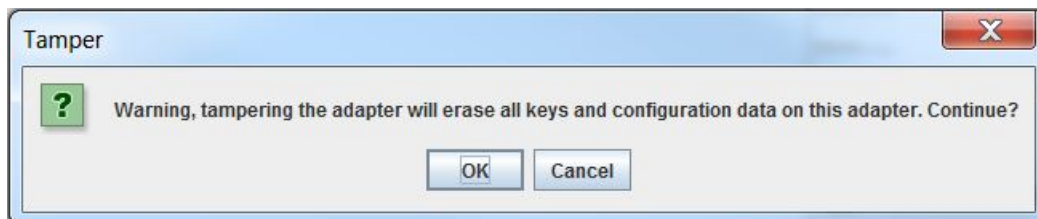
Tampering the HSM

It may be necessary to tamper the HSM at the end of its lifecycle, or after any other security-sensitive event requiring all stored data to be immediately destroyed.

A tamper formats the secure memory of the HSM, erasing all configuration and user data.

To tamper the HSM:

1. Select **File> Tamper Adapter**.
2. Click **OK** to confirm the action.



Key Management Utility (KMU) Reference

The Key Management Utility (KMU) provides a graphical user interface for key management functions, using a PKCS #11 sub-system. The utility provides the same functionality as the command line utility **ctkm** (["Command Line Utilities Reference" on page 90](#)).



Note: The KMU application is a Java-based application. A working Java runtime that supports the Swing user interface must be installed. This application has been tested with JDK 6, JDK 7, and JDK 8. The screenshots throughout this manual may vary from platform to platform.



Note: When operating in WLD/HA mode, this utility should only be used to view the configuration. Any changes to the configuration should be made in NORMAL mode. See ["Operation in WLD Mode" on page 56](#) and ["Operation in HA Mode" on page 57](#).

This chapter contains the following sections:

- ["Compatibility Issues" on the next page](#)
- ["Main KMU Interface" on the next page](#)
- ["Logging Into and Out From Tokens" on page 142](#)
- ["Creating Keys" on page 143](#)
 - ["Available Keys" on page 144](#)
 - ["Key Attribute Types" on page 144](#)
 - ["Creating a Random Secret Key" on page 145](#)
 - ["Creating a Random Key Pair" on page 146](#)
 - ["Creating Key Components" on page 147](#)
 - ["Entering a Key from Components" on page 149](#)
- ["Editing Key Attributes" on page 150](#)
- ["Deleting a Key" on page 151](#)
- ["Display Key Check Value" on page 151](#)
- ["Importing and Exporting Keys" on page 152](#)
- ["Key Backup Feature Tutorial" on page 158](#)
- ["Error Messages and Warnings" on page 163](#)

Compatibility Issues

Using KMU with SafeNet ProtectToolkit-J

SafeNet ProtectToolkit-J is SafeNet's Java Cryptography Architecture (JCA) and Java Cryptography Extension provider (JCE) software.

KMU may be used to set up tokens and keys for use with SafeNet ProtectToolkit-J V3 or later. The tokens and keys that are managed with KMU are fully compatible and may be utilized by SafeNet ProtectToolkit-J. The KMU may also be used to see and manipulate keys that have been created by SafeNet ProtectToolkit-J. For more information, consult the Key Management section in the SafeNet ProtectToolkit-J Reference Manual.

Please contact SafeNet for further details on its SafeNet ProtectToolkit-J products.

Using KMU with SafeNet ProtectToolkit-C V4.0, V3.x, and V2.x

This version of the KMU is not compatible for use with SafeNet ProtectToolkit-C version 4.0 or less.

The KMU can read backup files and smart cards created by SafeNet ProtectToolkit-C v2.x and v3.x but cannot create backup cards/files for these older versions.

The **ctkmu** command line utility is capable of creating backup cards/files for SafeNet ProtectToolkit-C v4.0 and V3.x HSMs. So, if you are exporting keys from a system running SafeNet ProtectToolkit-C 4.1 or above for import to an older system then use the **ctkmu** and the **-3** option.

Please contact SafeNet for a KMU that is compatible with older versions of this software.

Main KMU Interface

To start the KMU when using Microsoft Windows, locate the relevant program folder in the Windows Start menu and click on the appropriate shortcut. To start the KMU in a UNIX environment, enter **kmu** at the command prompt. To exit the KMU, select **Tokens> Exit** from the menu bar. Selecting **Help** from the main menu can retrieve information about the current KMU version.

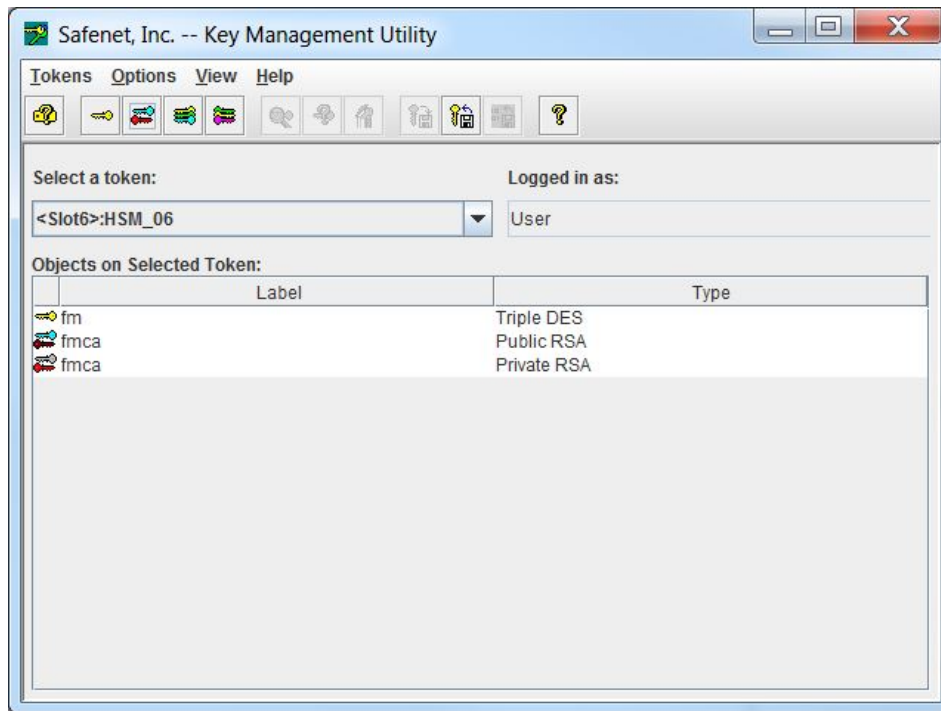
When the KMU is started, all toolbar functions are initially disabled. The user must first select a Token from the **Select a token** drop-down box, which will list all available tokens. Initialized tokens are displayed by their assigned label name. Uninitialized tokens are displayed as **<Slotn>:uninitialized token**.



Note: The KMU is unable to initialize tokens or change PINs. Use **gCTAdmin** or the command-line utility **ctconf** to perform these operations.

Once a token has been selected, the user is given the option to login. The PIN is authenticated, and a list of keys and other objects within the token are displayed in the **Objects on Selected Token** box. Appropriate buttons on the toolbar are enabled as shown in ["Key Management Utility Main Interface" on the next page](#).

Figure 1: Key Management Utility Main Interface



Token and Key Selection

Tokens are selected from the **Select a token** drop-down box. If an uninitialized token is selected, an error message is displayed. Use the admin utility **GCTADMIN** to initialize tokens.

The **Objects on Selected Token** box displays the objects currently stored on the selected token. This list displays the label and the type of each object. Select items from this list to perform the various functions.









Note: More than one key may be selected by drag-selecting to choose a range or SHIFT-LBUTTON to add/remove items to a selection. Operations that can accept more than one key will process all selected keys.

Toolbar Buttons

The buttons on the toolbar correspond to the following commands.

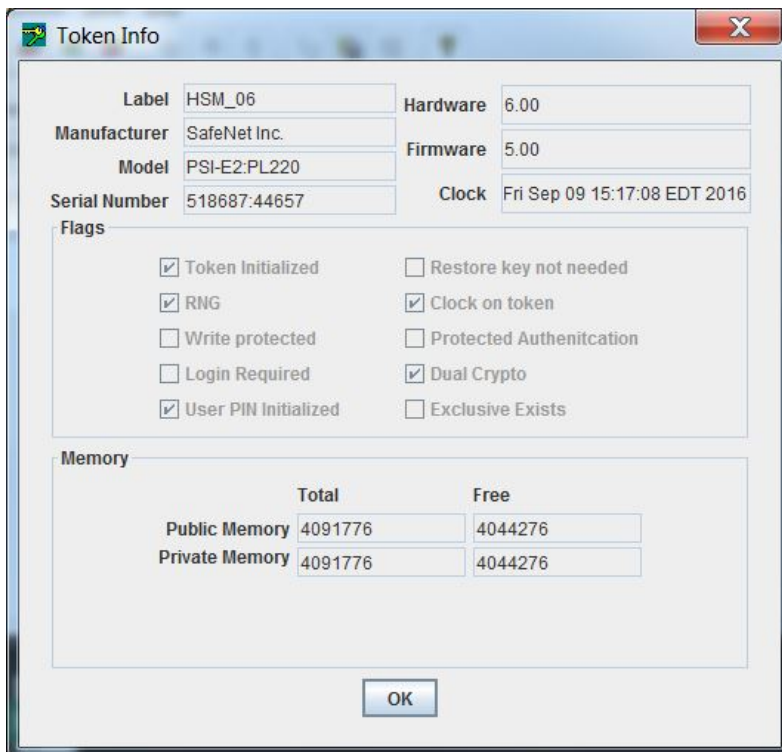
Button	Description	Button	Description
	Token Info		Edit Key Attributes
	Create Random Secret Key		Delete Key
	Create Key Pair		Import Key

Button	Description	Button	Description
	Create Key Components		Export Key
	Enter Key from Components		Import Domain Parameters
	Display Key Check Value		About KMU

The toolbar can be enabled or disabled from the **View** menu.

Retrieving Information about a Token

Click the **Token Info** button on the toolbar, or choose **Tokens > Token Info** from the menu bar. The **Token Info** dialog is displayed.



The **Token Info** dialog box displays the following information:

Label	HSM_06	Hardware	6.00
Manufacturer	SafeNet Inc.	Firmware	5.00
Model	PSI-E2:PL220	Clock	Fri Sep 09 15:17:08 EDT 2016
Serial Number	518687:44657		

Flags

<input checked="" type="checkbox"/> Token Initialized	<input type="checkbox"/> Restore key not needed
<input checked="" type="checkbox"/> RNG	<input checked="" type="checkbox"/> Clock on token
<input type="checkbox"/> Write protected	<input type="checkbox"/> Protected Authentication
<input type="checkbox"/> Login Required	<input checked="" type="checkbox"/> Dual Crypto
<input checked="" type="checkbox"/> User PIN Initialized	<input type="checkbox"/> Exclusive Exists

Memory

	Total	Free
Public Memory	4091776	4044276
Private Memory	4091776	4044276

OK

For more information on the items shown in this dialog, please refer to the PKCS #11 standard document.

Logging Into and Out From Tokens

To log in to a token:

1. Select an initialized token from the **Select a token** drop-down list.
2. Select a user type and enter the PIN corresponding to the selected token.



Note: Make sure that the CAPS lock is not on if the password contains lowercase characters.

PIN entry is masked so only the '•' character will be displayed as characters are typed. Some operations require the Security Officer (SO) to be logged in while other operations (private object operations) require the user to be logged in. It is also possible to open the token without logging in, but only public objects will be visible (also, depending on the security policy for the token, various operations like key generation might not be possible).

To log out from a token:

Select **Tokens > Logout From Token** from the menu bar.

Creating Keys

The KMU supports four key creation functions:

- ["Creating a Random Secret Key" on page 145](#)
- ["Creating a Random Key Pair" on page 146](#) (RSA public and private keys, for example)
- ["Creating Key Components" on page 147](#)
- ["Entering a Key from Components" on page 149](#)



Note: To refresh the key information displayed on the Main KMU Interface, select **Options> Refresh** from the menu bar. This displays a representation of what KMU has found on that token. If the token is modified by any other process or the KMU is out of sync with the token for any reason, choosing this menu option will refresh the list.

The KMU can also export and import keys for key backup and/or key escrow. This feature employs the PKCS #11 concept of key wrapping using high security key encryption keys (KEK) to wrap other KEKs and/or data keys. The KEK is a special key created with the *wrap* attribute, allowing it to be used for this purpose. KEKs are usually created as split custodian keys because of their enhanced security.




















Note: Only keys marked for export may be wrapped in this way, so it is possible to create keys that can never be extracted from the secure key storage.

Key Component creation is an important feature of SafeNet ProtectToolkit-C, since it allows key material to be split up and distributed among multiple trusted custodians. All custodians must combine their components to reconstruct the keys. Key custodians may use smart cards for key component and authentication PIN data storage, or use a disk file for key component storage.

Available Keys

The following key types are available when selecting a key operation:

Single Key Types		Key Pair Types	
	DES		RSA (Public)
	Double DES		RSA (Private)
	Triple DES		DSA (Public)
	AES (16, 24, or 36 bytes)		DSA (Private)
	IDEA		DH (Public)
	CAST128 (1 to 16 bytes)		DH (Private)
	RC2 (1 to 128 bytes)		EC (Public)
	RC4 (1 to 256 bytes)		EC (Private)
	SEED		

Key Attribute Types

You can specify what attributes a key will have when it is created. The following table describes the attributes which you can set when creating a key using the KMU.

Attribute	Description
Persistent	Stores the object on non-volatile memory. Persistent objects can be accessed after session termination.
Private	Defines whether the user PIN protects the object. A private object is only accessible to an application that has supplied the user PIN.
Sensitive	If a key is sensitive, the key's value cannot be revealed in plain text. Once a key becomes Sensitive it cannot be modified to be non-sensitive.
Modifiable	Indicates whether or not the object is modifiable, that is, if the object's attributes may be modified after creation.

Attribute	Description
Wrap	Indicates that the key may be used to wrap (that is, extract) other keys.
Unwrap	Indicates that the key may be used to unwrap keys.
Extractable	An extractable key can be wrapped (encrypted with another key) and extracted from the HSM.
Export	Indicates the key may be used to export other keys (similar to the wrap function).
Exportable	An exportable key may be wrapped (encrypted with another key), but only with keys marked with the Export attribute.
Derive	Indicates that the key can be used in key derivation functions.
Encrypt	Indicates that the key may be used for encryption.
Decrypt	Indicates that the key may be used for decryption.
Sign	Indicates that the key may be used for signing.
Verify	Indicates that the key may be used for verifying signatures or MAC values.

Creating a Random Secret Key

1. Select an initialized token from the **Select a Token** drop-down box and click on the **Secret Key** button in the toolbar. Alternatively, select **Options> Create> Secret Key** from the menu bar.

The **Generate Secret Key** dialog is displayed.



2. Choose the type of key you wish to generate from the **Mechanism** drop-down box. If you are generating an AES, CAST, RC2 or RC4 key, you must specify a Key Size.
3. Enter a label for the key into the Label input field.

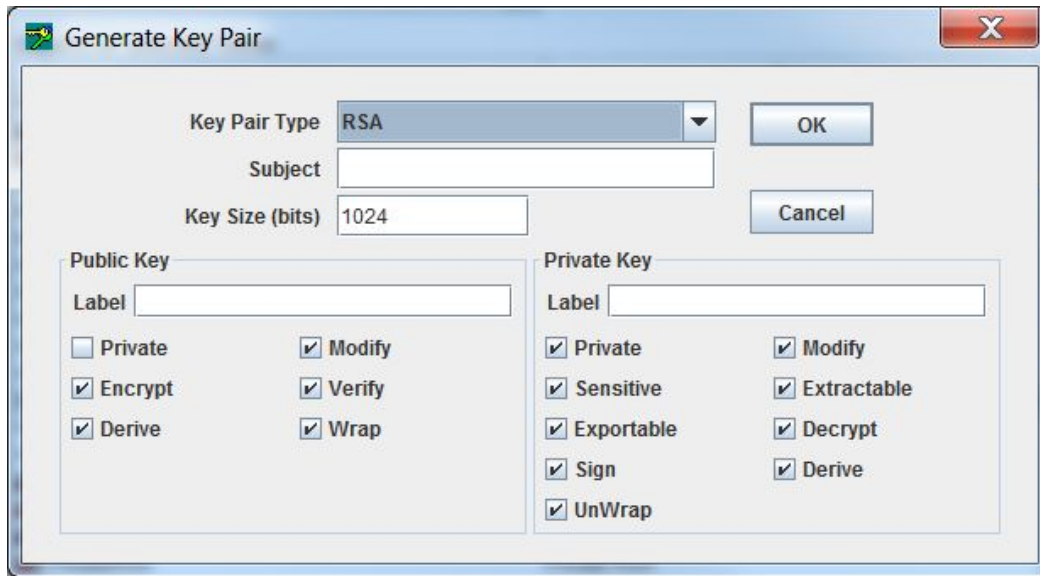
4. Select the desired key attributes by checking their boxes. See ["Key Attribute Types" on page 144](#) for descriptions of the individual attributes. There will be a default set of attributes checked for the key type.
5. Click **OK** to generate the secret key, or **Cancel** to reject your input and return to the previous menu.

The generated key will be displayed in the **Objects on Selected Token** box on the main KMU interface.

Creating a Random Key Pair

1. Select an initialized token from the **Select a Token** drop-down box and click on the **Key Pair** button in the toolbar. Alternatively, select **Options> Create> Key Pair** from the menu bar.

The **Generate Key Pair** dialog is displayed.



2. Select the type of key pair you wish to generate from the **Key Pair Type** drop-down box.

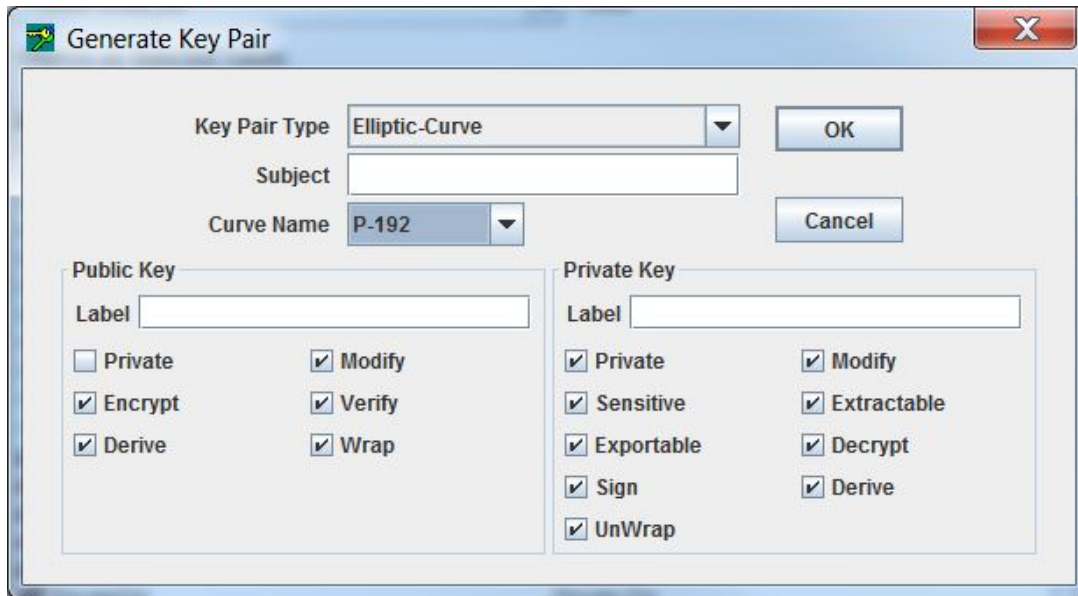
The **Subject** field can be left blank, in which case there will be no X.500 certificate information attached to the key pair. If you specify a **Subject**, it must be set according to X.500 distinguished name syntax. For example, **C=CA, O=safenet, CN=Alice**. The subject fields can be any of the following, and may be input in any order:

- C= Country code
- O= Organization
- CN= Common Name
- OU= Organizational Unit
- L= Locality name
- ST= State name

This information will be stored with the public and private key objects in the CKA_SUBJECT_STR attribute and also DER-encoded and stored in the CKA_SUBJECT attribute. This attribute will be propagated into any PKCS #10 and X.509 certificates derived from these keys.

3. Specify the **Key Size (bits)** or **Curve Name** (only enabled if **Key Pair Type** is **Elliptic Curve**). Available curve names are:
 - **P-192** (also known as **prime192v1** and **secp192r1**)
 - **P-224** (also known as **secp224r1**)

- **P-256** (also known as **prime256v1** and **secp256r1**)
- **P-384** (also known as **secp384r1**)
- **P-521** (also known as **secp521r1**)
- **c2tnb191v1**
- **c2tnb191v1e**



4. Label both the public key and the private key. Check or uncheck any available boxes to select the desired key attributes.



Note: The check boxes are enabled and disabled according to the selected Key Pair Type.

5. Press **OK** to generate the keys, or **Cancel** to discard your input and return to the previous menu.
Generated keys will be displayed under the **Objects on Selected Token** list on the main KMU user interface.

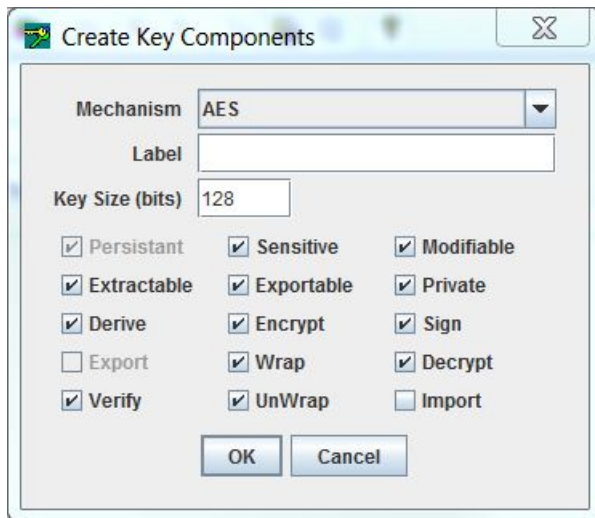
Creating Key Components

This function will create a random key as a number of components. These components may be recorded manually, either for backup purposes or so that they can be entered on another machine by using the **Enter Key** function.

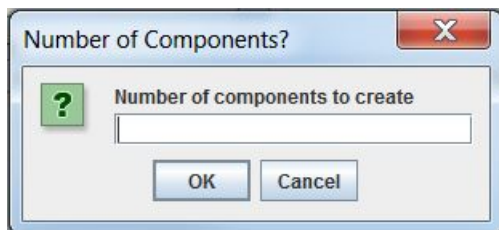
This is useful for the creation and distribution of Key Encryption Keys (KEKs) with multiple custodians. This function makes it possible to create a key whose value is unknown to any single party. Only by combining the components known by each custodian can the key be regenerated. Each component is randomly generated, and in itself does not expose any portion of the final key value.

To create key components:

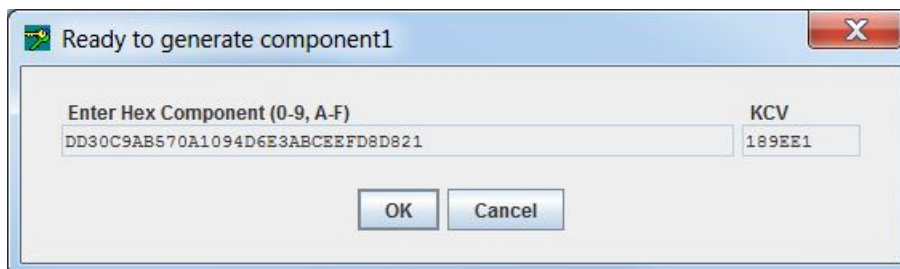
1. Select an initialized token from the **Select a Token** drop-down. Log in if necessary.
2. Choose **Options> Create> Generate Key Components** from the menu bar, to open the **Create Key Components** dialog box.



3. Select a key type from the **Mechanism** drop-down list.
4. Enter a label for the key into the **Label** field.
5. For key types AES, CAST, RC2 and RC4, specify the size of the key to be generated in the **Key Size (bits)** field.
6. Decide on the key attributes and click active checkboxes as required.
7. Click **OK** to continue, or **Cancel** to abort this operation and return to the previous menu.
8. When prompted by the KMU, enter in the **Number of Components** field the number of components that you wish the key to be split into. There is no limit on the number of components.



9. Click **OK** to start displaying the key components, or **Cancel** to abort this operation and return to the previous menu.
- A **Ready to generate component** dialog box will be displayed for each component determined in step 8.



10. Record the Component Value and Key Check Value (KCV), both given in hexadecimal, displayed in these dialogs. The KCV for the generated component is used to verify correct entry of the component during manual key component entry.

Entering a Key from Components

This function allows a key to be entered from one or more components.

To enter a key from components:



Note: The component entry can be masked by selecting **Options> Mask Component Entry** before beginning the operation.

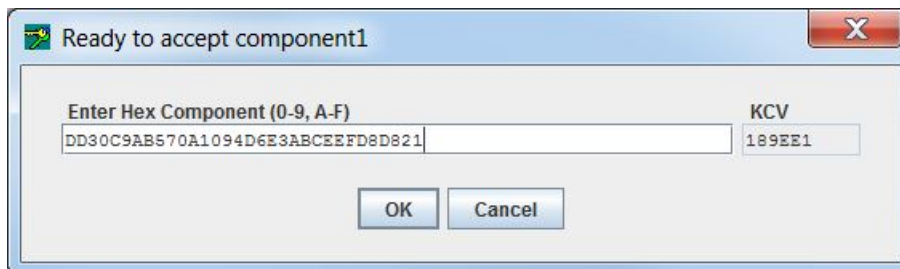
1. Select an initialized token from the **Select a Token** drop-down box and click **Enter Key From Components** on the toolbar. Alternatively, select **Options> Create> Enter Key From Components** from the menu bar.

The **Enter Key Components** dialog will open.



2. Select a key type from the **Mechanism** drop-down list.
3. Enter a label for the key into the *Label* field.
4. For key types AES, CAST, RC2 and RC4, specify the size of the key to be generated in the **Key Size (bits)** field.
5. Decide on the key attributes and click active checkboxes as required.
6. Click **OK** to continue, or **Cancel** to abort this operation and return to the previous menu.
7. When prompted by the KMU, enter the number of key components to combine in the **Number of Components** field. There is no limit on the number of components.
8. Click **OK** to continue and open the **Ready to accept component** dialog, or **Cancel** to abort this operation

A number of component dialogs will appear, corresponding with the number specified in the **Enter Key** dialog.





Note: The KCV appears automatically when the key component is entered, allowing the custodian to confirm correct entry. The KMU will check that the KCV matches that of the key components being input. If a mismatch is detected, an error is shown.

Key check value (KCV) of symmetric keys can be displayed by selecting a key and clicking **View** on the toolbar. Alternatively, select **Options> View** from the menu bar.

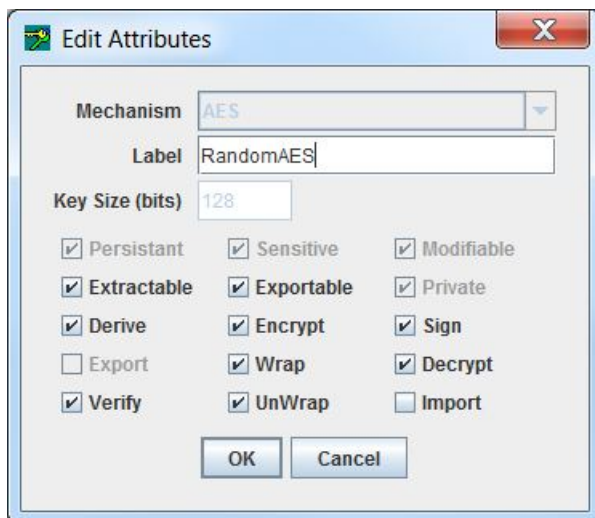
Refer to "[PKCS #11 Attributes](#)" on [page 168](#) for details on how the KCV is calculated.

Editing Key Attributes

The attributes available to edit depend on what attributes were set when the key was created. The **Edit Attributes** dialog box displays only the attributes that can be changed. Unavailable attributes are grayed out.

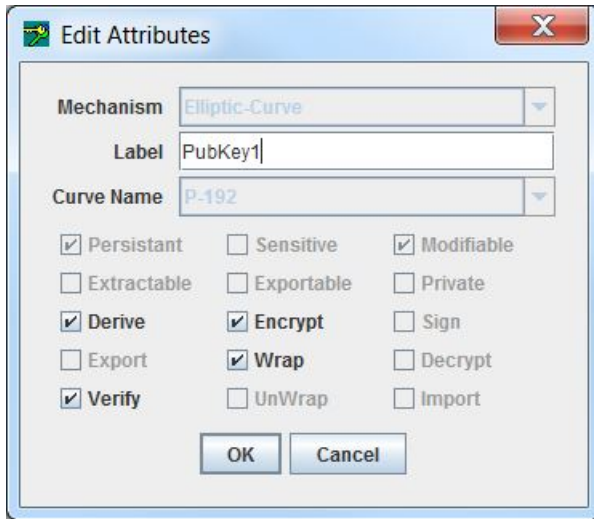
To edit key attributes:

1. Double-click on the key you want to edit.
The **Edit Attributes** dialog box is displayed.
2. Check the active boxes for the attributes you want to change.



Available curve names when **Elliptic Curve** is selected are:

- **P-192** (also known as **prime192v1** and **secp192r1**)
- **P-224** (also known as **secp224r1**)
- **P-256** (also known as **prime256v1** and **secp256r1**)
- **P-384** (also known as **secp384r1**)
- **P-521** (also known as **secp521r1**)
- **c2tnb191v1**
- **c2tnb191v1e**



Deleting a Key

To delete a key:

1. Select an initialized token from the **Token Selection** drop-down box. Enter the User PIN.
2. Select the key to be deleted from the **Objects on Selected Token** box, and click **Delete Key** on the toolbar.



Alternatively, select **Options> Delete** from the menu bar.

Display Key Check Value

You can check that a key matches an expected key value, without revealing anything about the actual key value, by viewing its Key Check Value (KCV).

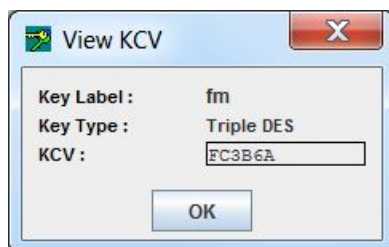
The KCV is a standard technique for obtaining an identification fingerprint from a key. The mechanism used, compatible with AS 2805, is simply the first three hex digits obtained by encrypting binary zeros with the key. Please refer to ["Creating Keys" on page 143](#) for details of KCV generation.

To display the KCV for a key:

Select a key from the **Objects on Selected Token** list and click the **View** button on the toolbar.



Alternatively, select **Options> View** from the menu bar.



Importing and Exporting Keys

The process of exporting and importing keys ensures that keys, certificate objects, and other PKCS#11 objects can be recovered after a failure or tamper event. Keys can be exported to files on the host system or to smart cards. When exporting to smart cards, you may export keys to a single card (single-custodian) or split the key over multiple cards (multiple-custodian). All PKCS#11 attributes, including security attributes, and the key/object's value are backed up.

It is not possible to back up the security officer and user PINs for a token. Before a restore/import operation, the destination token must be already initialized and the SO and user PINs set. A number of additional keys are generated, used, and then deleted during the backup process.

Exporting Keys

This function allows keys to be encrypted and written to smart cards, files, or the screen. The keys can then be transferred to other machines. See ["Secure Key Backup and Restoration" on page 81](#) for background information on backup and recovery methods, key splitting schemes and key attributes.

Preparation

Before attempting a key backup, please ensure that you have:

- a valid key that can be backed up
- a smart card reader connected (if backing up to smart cards)
- sufficient initialized and erased smart cards or disk space to back up the required data
- created a wrapping key (if wrapping keys to be backed up). See ["Creating Keys" on page 143](#) for instructions.

To export a key (or set of keys):

1. On the Key Management Utility main interface (see ["Key Management Utility Main Interface" on page 141](#)), select the token containing the key(s) to be exported from the **Select a token** box, and log on to the token.

The **Objects on Selected Token** list displays the available keys on the token.

2. Select one or more keys to export from the **Objects on Selected Token** list.
3. Right-click on the selected key(s) or select **Options> Export**.

Alternatively, click on the **Export Key** button on the toolbar.



The **Export Keys** dialog box displays. Details of selections appear in the **Selected Token** and **Selected Key(s)** fields.



Note: Wrapping keys must be created before the next step. See ["Creating Keys" on page 143](#) for instructions.

4. From the **Wrapping Key** drop-down list, select an appropriate wrapping key based on your choice of backup and recovery method. See the table below for further assistance.

To use the:	Select:
<i>Multiple custodians</i>	<Random key>

To use the:	Select:
method	
<i>Single custodian</i> method	The desired wrapping key. This key is used to encrypt the key (or set of keys) to be exported

5. In the **Options** area, make further selections as appropriate for the backup and recovery method and destination backup media to be used.

When using the *multiple custodians* backup and recovery method, only **Write to smart card(s)** and associated options may be selected.

Continue with the following steps for the destination backup media required.

To export the selected key(s) to smart cards:

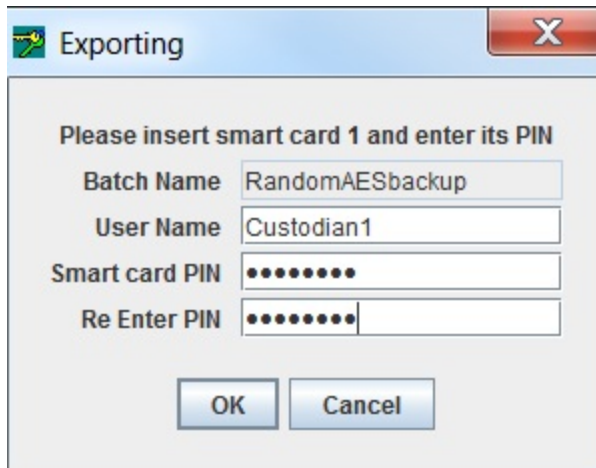
- In the **Options** area, select **Write to smart card(s)**.
- Enter an identifying name for the smart card set in the *Batch Name* field.
The batch name cannot be the same as the token label if the N of M key splitting scheme is to be used (see below).
- If the multiple custodians backup and recovery method is to be used (<Random key> selected from the **wrapping key** drop-down list) enter the number of custodians required.
- When using the multiple custodians backup and recovery method you may also elect to use the N of M key splitting scheme so that only N out of M custodians are needed to recover the key.

For example, if M = 3 and N = 2, only two out of the three custodians need to present their smart cards to recover the key. To use the N of M scheme select the **Use N of M** checkbox and enter the minimum number of custodians

required to recover the key (N) in the *No. of custodians for recovery* field. This field only displays after **Use N of M** has been selected. Note that N may not equal M.

- Click **OK** to begin the export operation or **Cancel** to abort it.

After clicking OK a dialog box displays and shows the *Batch Name*, a *User Name* entry field and a *Smart card PIN* entry field for a custodian (see "Importing and Exporting Keys" on page 152).



- Insert a smart card in the smart card reader.
- Any user name may be entered. The PIN entered can be that already established for the inserted smart card or a new one may be entered. The PIN must be entered again in the *Re-Enter PIN* field as an accuracy check. Click **OK**.
- If a new PIN was entered, a prompt for the old PIN displays. Enter the old PIN to complete the change.
If an incorrect smart card PIN is entered more times than the number specified for the card during its initialization, the smart card will become blocked. The card may then only be unlocked by entering the Security Officer PIN. Refer to the smart card initialization section for further details.
Data is now written to the smart card. If additional key shares are to be written to smart cards then a prompt for the next smart card displays.
- Remove the smart card from the smart card reader and repeat steps 5-9 until all the key shares required have been written to smart cards.
When the operation is complete, an *Export Successful* message box displays.
- Click **OK** to return to the main Key Management Utility interface.

To export the selected key(s) to a file:

Available for the wrapping key backup and recovery method only.

- In the Options area, select **Write to selected file**.
- Enter the path and filename of the file to be created in the *File to write* field. If a file with the same filename already exists at this location then it will be overwritten. Alternatively, browse to a location and enter a filename by clicking on the "..." button next to the *File to write* field.
- Click **OK** to begin the export operation or **Cancel** to abort it.

To export the selected key(s) to the console:

Available for the wrapping-key backup and recovery method only.

1. In the **Options** area, select **Write encrypted parts to the screen**.
2. Select **single** or **multi-part** export.
3. Click **OK** to begin the export operation or **Cancel** to abort it.

Importing Keys

Importing allows keys, stored on smart cards, in files or as encrypted parts that were exported to the screen, to be restored to a token. See the section "[Secure Key Backup and Restoration](#)" on page 81 for background information on backup and recovery methods, key splitting schemes and key attributes.

To import a key (or set of keys):

1. From the **Token Selection** drop-down box select the token that is to receive the imported keys and click the **Import Keys** button on the toolbar. Alternatively, select **Options>Import** from the menu bar.

The *Import Key(s)* dialog displays.

2. In the Options area, choose either **Read from smart card(s)**, **Read from selected file**, or **Import encrypted parts**, depending on the media that was used to store the key(s).

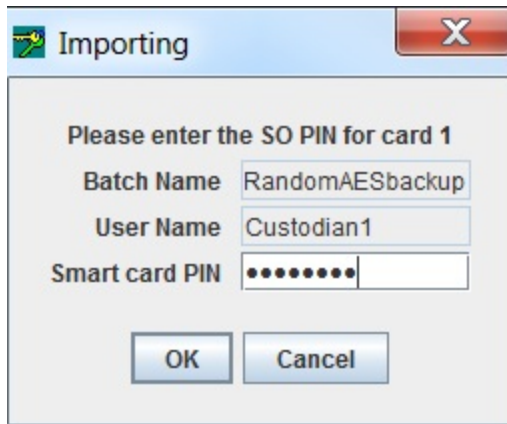
When choosing to read from smart card(s):

1. Select the backup and recovery method that was used to back up the key(s), either the *multiple custodians* or the *single custodian* method, by making the appropriate selection from the **Unwrap Key** drop-down list.

If the backup method was:	Select:
<i>Multiple custodians</i>	<Random key>
<i>Single custodian</i>	the particular wrapping key that was used to create the backup

2. In the **Options** area, select **Read from smart card(s)**.
3. Insert the smart card in the smart card reader.

4. Select the smart card from the **Selected Smartcard** drop-down list. Click **OK** to start the import operation, or **Cancel** to abort.
5. The following dialog box, displaying the current card number and batch name, prompts for the smart card PIN.



Enter the PIN for the smart card and click **OK**.

If an incorrect smart card PIN is entered, a prompt will display to enable re-entry. If an incorrect smart card PIN is entered more times than the number specified for the card during its initialization, the smart card will become blocked. The card may then only be unlocked by entering the Security Officer PIN. Refer to the smart card initialization section for further details.

If a smart card is from a different batch is inserted or if the card has already been read it will be rejected. A prompt will display to insert another card.

Data is now retrieved from the smart card. If additional key shares are required to recover the key(s) then a prompt for the next smart card displays.

6. Remove the smart card from the smart card reader and insert the next one. Repeat the previous step until all the key shares required have been retrieved from smart cards.

When the operation has completed, the message *Import Successful* message is displayed. The newly imported key(s) also display in the *Objects on Selected Token* table in the main Key Management Utility interface.
7. Click **OK** to return to the main Key Management Utility interface.

When choosing to read from a selected file:

1. From the **Unwrap Key** drop-down list, select the wrapping key that was used to create the backup.
If a wrong wrapping key is selected the error message, *Key used to import was not the same as the key used to export*, will display.
2. Select **Read** from selected file.
3. Enter the filename for the encrypted key file into the *File to Read* field. The “...” button can be used to find and select the file.
4. Click **OK** to import the selected key, or **Cancel** to abort this operation.

If the import key operation is a success, the message *Import command succeeded* is displayed. The newly imported key also displays in the *Objects on Selected Token* table in the main Key Management Utility interface.

When choosing to import encrypted parts:

1. From the **Unwrap Key** drop-down list, select the wrapping key that was used to create the backup.

If a wrong wrapping key is selected, the error message *Key used to import was not the same as the key used to export* will display.

2. Select Import encrypted parts.
3. Select either **Multi Part** or **Single Part** as applicable and click **OK** to continue.
4. Enter the encrypted key (or key parts) and click **OK** to import the key.

If the import key operation is a success, the message *Import command succeeded* is displayed. The newly imported key also displays in the *Objects on Selected Token* table in the main Key Management Utility interface.

Key Backup Feature Tutorial

This section illustrates the use of KMU for Key Backup, which can be used to ensure keys, certificate objects and other PKCS#11 objects can be recovered after a failure or tamper.

It contains the following subsections:

- ["Key Definitions" below](#)
- ["Creation of Encrypted Key Set to Backup \(Payload\)" on the next page](#)
- ["Back up to File" on the next page](#)
- ["Backup to Smart Card – Single Custodian Mode" on page 160](#)
- ["Backup to Smart Card – Multiple Custodian Mode" on page 161](#)

Two storage media options are available:

- smart card
- file (hard disk drive)

For smart card media, there are two modes available:

- single-custodian
- multiple-custodian

All the PKCS#11 attributes for any key/object, including the security attributes, are backed up along with the key/object's value.

When backing up to smart card, the utility will automatically prompt for additional smart cards if the size of the backup is larger than one smart card.

The security officer and user PINs for a token cannot be backed up. Before a restore operation, the destination token must be already initialized and the security officer and user PINs set.

There are a number of additional keys generated, used, and then deleted during the backup process.

Key Definitions

wK	Wrapping key. The top-level key for the backup process. This key must be valid for the operation $E2_x$. When performing a backup to file or single custodian to smart card, the custodian must provide this key. It is recommended that this be a triple length DES key. For the multiple Custodian backup, this key is created from the randomly generated split components for each custodian.
tK	A randomly generated transport key, which is a triple length DES key, using CKM_DES3_KEY_GEN. This is the key that the keys/objects to be backed up will be wrapped under. This key is used with W_x .
mK	A randomly generated MAC key, which is a triple length DES key, using CKM_DES3_KEY_GEN. This key is used with M_x .
E_x	Encryption using CKM_DES3_ECB_PAD with key 'x'.
$E2_x$	Encryption using CKM_(based on key type of 'x') with key 'x', e.g. CKM_DES3_ECB.
W_x	C_WrapKey() operation using CKM_WRAPKEY_DES3_CBC with key 'x'.

R_x	C_DeriveKey() operation using CKM_XOR_BASE_AND_DATA with key 'x' and provided data.
M_x	MAC generation, using CKM_DES3_MAC (4 byte MAC result) with key 'x'.

Creation of Encrypted Key Set to Backup (Payload)

The creation of the encoded payload to backup is common to all storage options. The payload can contain one or more keys/objects.

To create the encoded payload:

1. Generate tK.
2. For each key/object to be backed up:

$$w = W_{tK}(\text{Key/Object})$$

The format of the resulting Payload is as follows:

$$p = N |_1 w_1 [|_2 w_2 [|_3 w_3 [\dots |_N w_N]]]$$

where N = Number of keys/objects in the payload, $|_i$ = length of w_i , and w_i = The i'th wrapped key data, i.e. $W_{tK}(\text{Key/Object})$

3. Generate mK.
4. Calculate the MAC for the Payload, $m = M_{mK}(p)$.

Back up to File

This is the simplest form of backup. The only limitation is that the wrapping key must already exist. This key must be able to be recreated after a tamper/failure before a restore can be performed. It may be entered in components, have a known value, or be backed up using the multiple custodian backup mode (described below).

To back up to file:

1. Encode mK with tK, $emK = E_{tK}(mK)$
2. Encode tK with wK, $etK = E_{wK}(tK)$
3. Write the binary file containing the backed up Payload. The format of the file is:

Header	Contains the version of the Backup Feature
length p	Length of the encoded Payload
p	Encoded Payload
m	MAC of the Payload
length emK	Length of the Encoded MAC key
emK	Encoded MAC key
length etK	Length of the Encoded Transport key
etK	Encoded Transport key

4. Delete mK and tK.

Backup to Smart Card – Single Custodian Mode

This backup mode has more security than the backup to file mode because the payload is stored on a smart card instead of in a file. The payload data on the smart card is also protected by the custodian's PIN, i.e. the PIN must be presented and authenticated to the smart card before the data can be read.

The only limitation is that the wrapping key must already exist. This key must be able to be re-created after a tamper/failure before a restore can be performed. It may be entered in components, have a known value, or be backed up using the multiple custodian backup mode (described below).

If the payload cannot fit on one smart card, then the backup process will prompt the custodian to continue entering new smart cards, until the entire payload has been exported.

To back up to Smart Card:

1. Encode mK with tK, $emK = E_{tK}(mK)$
2. Encode tK with wK, $etK = E_{wK}(tK)$
3. Write the following data files to the smart card:

Header	<p>Not protected by custodian's PIN.</p> <p>Contains the following information about the payload:</p> <p>Contains the version of the backup feature</p> <p>Name of this backup payload</p> <p>MAC of the complete payload</p> <p>MAC of the payload component on this smart card, i.e. $M_{mK}(p')$</p> <p>Timestamp of payload creation</p> <p>Total number of custodians</p> <p>Number of the custodian who owns this smart card</p> <p>Number of the current card being written</p> <p>Flag to indicate if encoded transport key (etK) is on this smart card</p> <p>Flag to indicate if encoded MAC key (emK) is on this smart card</p> <p>Size of the complete payload</p> <p>Size of the payload component on this smart card</p> <p>Offset of this payload component in the complete payload</p> <p>Name of custodian who owns this smart card</p> <p>Payload</p> <p>Protected by the custodian's PIN.</p> <p>The component of the payload contained on this smart card. This may be the entire payload.</p>
etK	<p>Protected by the custodian's PIN.</p> <p>Encoded transport key</p> <p>This data file will only be located on the last smart card of the backup set.</p>
emK	<p>Protected by the custodian's PIN.</p> <p>Encoded MAC key</p> <p>This data file will only be located on the last smart card of the backup set.</p>

4. Delete mK and tK.

Backup to Smart Card – Multiple Custodian Mode

This backup mode has the most security. This is because the payload is stored on smart cards and the payload is split between a number of custodians. Also, the payload data on the smart card is protected by the custodian's PIN, i.e. the PIN must be presented and authenticated to the smart card before the data can be read.

The top level wrapping key (wK) is randomly generated, and each custodian has a component of this key. The entire set of smart cards is needed before the wrapping key can be successfully re-created.

If each custodian's payload component cannot fit on one smart card, then the backup process will prompt the custodian to continue entering new smart cards, until their payload component has been exported.

To back up to a Smart Card in Multiple Custodian Mode

1. Create an initial intermediate wrapping key, which is a triple length DES key, wK', with a value of zero.

Each custodian must then:

2. Generate random wrapping key component (24 bytes), wC
3. Derive new intermediate wrapping key $wK' = R_{wK'}(wC)$
4. Delete the previous intermediate wrapping key (wK'-1)
5. Write the following data files to the smart card:

Header	<p>Not protected by custodian's PIN.</p> <p>Contains the following information about the payload:</p> <p>Contains the version of the backup feature</p> <p>Name of this backup payload</p> <p>MAC of the complete payload</p> <p>MAC of the payload component on this smart card, i.e. $M_{mK}(p')$</p> <p>Timestamp of payload creation</p> <p>Total number of custodians</p> <p>Number of the custodian who owns this smart card</p> <p>Number of the current card being written</p> <p>Flag to indicate if encoded transport key (etK) is on this smart card</p> <p>Flag to indicate if encoded MAC key (emK) is on this smart card</p> <p>Size of the complete payload</p> <p>Size of the payload component on this smart card</p> <p>Offset of this payload component in the complete payload</p> <p>Name of custodian who owns this smart card</p>
wC	<p>Protected by the custodian's PIN.</p> <p>The wrapping key component for this custodian.</p>
Payload	<p>Protected by the custodian's PIN.</p> <p>The component of the payload contained on this smart card.</p>

The last custodian must then:

6. Encode mK with tK, $emK = E_{tK}(mK)$
7. Encode tK with the final wrapping key ($wK = wK'$), $etK = E_{wK}(tK)$

8. Write the following data files to the smart card:

etK	Protected by the custodian's PIN. Encoded transport key This data file will only be located on the last smart card of the last custodian of the backup set.
emK	Protected by the custodian's PIN. Encoded MAC key This data file will only be located on the last smart card of the last custodian of the backup set.

9. Delete mK, tK and wK.

Error Messages and Warnings

This section describes error messages and warnings specific to KMU.

KMU-specific warning messages

Warning Message	Action
Incompatible Cryptoki Version	An attempt was made to use a feature of another Cryptoki version
Token not initialized	Initialize tokens before attempting to access them
Re-initializing the token will erase all currently stored keys	Store all further needed objects on cards/files before initializing the token
No SO PIN entered	You must enter a SO PIN when initializing a token
No User PIN entered	You must enter a user PIN when initializing a token
No label entered	A key label should be entered when generating a key

KMU-specific error messages

Error Message	Action
Couldn't download SAM	The Software Application Module (SAM) that is used by the KMU when performing smart card operations was not set correctly. Check that the smart card reader is securely attached to the lower port of the SafeNet ProtectServer HSM. After having verified the connection, open a command prompt and change into the directory where the KMU is installed. Type: SAMDEVL and press Enter
Couldn't get smart card to respond	Check the connection between the smart card reader and the SafeNet ProtectServer HSM
Not enough room left on smart card for export	Re-initialize the card and erase its contents, and repeat the export operation
Smart card was not initialized	Re-initialize the smart card and retrieve information about the card
The key used to import was not the same as the key used to export	Retry the export/import operation using the same key for wrapping and unwrapping
Smart card was not initialized. Export Failed	Initialize the smart card before attempting to export keys
Smart card has already been processed	Retry the import operation using another card from the same batch
Smart card is from a different batch	Retry the import operation using another card from the given batch
The card inserted has already been written as	Insert a new card

Error Message	Action
part of this batch. Please insert another card	
There are no keys stored on the inserted card	Insert a card with keys stored on it
Please select a file to read	Specify the file containing the key before importing
Couldn't read the selected file	Check the file's read access rights
Unable to open selected file	Check the file's path and access rights
An error occurred when writing file	Retry the export to file operation making sure that the path is correct and adequate access to the file is provided
Chunk MAC's do not match. Import Failed	The MAC retrieved from the different data chunks written on smart cards doesn't match with the initial one
No wrapping key selected	Specify a wrapping key before attempting to write a key to the file
An unwrapping key is required for this batch	Retry the import operation after making sure that the correct unwrapping key was selected
An unwrapping key is not required for this batch	The keys were exported using a random wrapping key. Specify no unwrapping key for import operations
Could not generate KVC on a key with Encrypt (E) attribute set to false	Only KVC's of keys enabled for encryption can be calculated
No KVC entered	Re-enter the key with a KVC
KVC mismatch	The KVC of the key components and of the entered key do not match
Not a valid hex digit	An attempt was made to enter a wrong character in a key
No component entered	An attempt was made to store a key without having entered any key components
Wrong key size	When generating keys use the key ranges specified on page
User PIN's don't match	Make sure that you use the same user and user verification PINs.
SO PIN's don't match	Make sure that you use the same SO and SO verification PIN
Incorrect PIN length	Re-enter the PIN using a PIN length between 4 and 32
Incorrect PIN	An attempt was made to enter an incorrect character in the PIN string
User PIN is blocked	Unblock the PIN as SO, and then retry the operation

For other error messages generated by Cryptoki functions, see the PKCS#11 documentation.

APPENDIX A

Event Log Error Types

The following table lists the error entries that may be generated by the ProtectServer HSM firmware and written to the HSM's event log.

Event records are written sequentially and chronologically. If the date and time of a later entry in the log is stating an earlier time than an entry preceding it, it indicates that the real time clock or audit information has been altered.

Name	Description
POST_ERR_SRAM_WRITE	POST Error: Cannot write to SRAM
POST_ERR_SRAM_READ	POST Error: Cannot read from SRAM
POST_ERR_SDRAM_DATA_STUCK	POST Error: SDRAM, bit stuck
POST_ERR_SDRAM_DATA_SHORT	POST Error: SDRAM data bits short Param 1. Bit number Param 2. Value
POST_ERR_SDRAM_ADDR_STUCK	POST Error: SDRAM address bit stuck
POST_ERR_SDRAM_ADDR_SHORT	POST Error: SDRAM address bits short Param 1. Bit number
POST_ERR_SDRAM_BAD_BYTESEL	POST Error: SDRAM bad bytes select
POST_ERR_BAD_SECTOR0	POST Error: POST Sector checksum is not correct
POST_ERR_NOMEM	Cannot allocate memory
POST_ERR_OS_HASH	The OS hash value is incorrect
POST_ERR_KAT	Known answer test failed Param 1. Algorithm Identifier Param 2. Error Code
POST_ERR_RNG	RNG did not pass chi-squared test
POST_ERR_NO_THREAD	Unable to start POST Thread
POST_ERR_SMFS	Secure memory file system error Param 1. Error Number
POST_ERR_RTC	Unable to access RTC
POST_ERR_SER	Unable to access UART


Name	Description
EXCEPT_UNDEF	An undefined instruction has been executed Param 1. Address Param 2. Instruction
EXCEPT_SWI	A software interrupt generated Param 1. Address Param 2. Instruction
EXCEPT_PREFETCH	A Prefetch abort generated Param 1. Address
EXCEPT_DATA	A Data abort generated Param 1. Address
EXCEPT_IRQ	An unhandled IRQ received Param 1. Identifier
ERR_HOT_TAMPER	Hot tamper detected
LOG_FIRST_ENTRY	Initial event entry
LOG_INITIALIZING_SRAM	Initializing the SRAM after a tamper
LOG_EVENT_LOG_PURGED	Event log has been purged
ERROR_ASSERT	Runtime Assertion Param 1. File Param 2. Line
ERROR_INIT_RESOURCE	Out of resources in initialization Param 1. File Param 2. Line
ERROR_INIT_PLATFORM	Failed to detect hardware platform Param 1. File Param 2. Line
HEAP_INVALID_ADDRESS	Heap Invalid block address Param 1. Heap number Param 2. Address
HEAP_MEM_FREED_TWICE	Heap: Memory Freed twice Param 1. Address
PCCISES_TIMEOUT	PCCISES: Timeout error on device Param 1. Error
PCCISES_BAD_STAT	PCCISES: Bad device status

Name	Description
	Param 1. Status
PCCISES_BAD_DATA	PCCISES: Bad input data
PCCISES_RNG_STUCK	PCCISES: Continuous RNG test error Param 1. Value
PCCISES_LNAU_EXCEPTION	PCCISES: Large Number Arith Hardware exception (Unit,0)
PCCISES_FAILED_RESET	PCCISES: Failed to reset
PCCISES_RESOURCES	PCCISES: Insufficient resources to start driver
CPROV_OS_UPGRADED	OS Upgrade performed Param 1. Mod Param 2. Version
CPROV_OS_UPGRADE_FAILED	OS Upgrade failed
PROT_NO_SMPR	PROTECTION: HSM SMPR not found
PROT_CIPHER_ERROR	PROTECTION: Cipher operation failed
KEYGEN_ERR_PAIRWISE	Key generation: Pair-wise consistency failure
FM_OP_DOWNLOAD	FM Download Performed Param 1. Mod Param 2. Version
FM_OP_DISABLE	FM Disabled Param 1. Mod Param 2. Version
FM_MODULE_FAILED	FM failed to load Param 1. Mod Param 2. Version
PTKC_CFG_CHNG	SafeNet ProtectToolkit-C config change Param 1. New Val Param 2. Old Val

APPENDIX B

PKCS #11 Attributes

Objects, as described by PKCS #11, consist of a number of attributes that define both the *object* and its *access policy*. In general, the SafeNet ProtectToolkit-C system will define the object's attributes. Access policy should be provided by the user based on their particular requirements. The following attribute descriptions are intended to assist with these decisions.

Attribute	Description
CKA_LABEL	<p>This attribute specifies a textual label for an object. This label is used to assist in differentiating the various objects stored on a token.</p> <hr/> <p> Note: Although SafeNet ProtectToolkit-C does not require this attribute to be unique, various other tools may.</p> <hr/>
CKA_CLASS	<p>This attribute is assigned by the system when an object is created. There are a number of classes in common use:</p> <ul style="list-style-type: none">• CKO_PUBLIC_KEY• CKO_PRIVATE_KEY• CKO_SECRET_KEY• CKO_CERTIFICATE• CKO_CERTIFICATE_REQUEST• CKO_DATA
CKA_KEY_TYPE	<p>This attribute specifies the key type associated with the object. There are many key types supported by SafeNet ProtectToolkit-C. For example:</p> <ul style="list-style-type: none">• CKK_AES, CKK_DES, CKK_DES2, CKK_DES3, CKK_RSA, CKK_DSA• CKA_ENCRYPT• CKA_DECRYPT• CKA_SIGN• CKA_VERIFY• CKA_WRAP• CKA_UNWRAP <p>The previous attributes describe the cryptographic operations the key may be used for. Careful consideration should be given when assigning these attributes, to avoid key misuse.</p>
CKA_IMPORT	<p>This attribute is similar to the standard CKA_UNWRAP attribute. It is used to determine if a given key can be used to unwrap encrypted key material. The important difference between these attributes and their standard counterparts is that if CKA_IMPORT is set to True and CKA_UNWRAP attribute is set to False, then the only unwrap mechanism that</p>

Attribute	Description
	can be used is CKM_WRAPKEY_DES3_CBC. With this combination, the error code CKR_MECHANISM_INVALID will be returned for all other mechanisms.
CKA_EXPORT	This attribute is similar to the CKA_WRAP attribute, in that it specifies that the key may be used to encrypt a second key, so that it may be extracted from the HSM in an encrypted form. Unlike the CKA_WRAP attribute, however, only the <i>Security Officer</i> may specify this attribute.
CKA_SENSITIVE	This attribute specifies that the key object cannot be extracted from the token in the clear. Generally this attribute should be specified to ensure the key material is not exposed. When the <i>No Clear PINs</i> flag is set only sensitive keys may be created on the HSM.
CKA_EXTRACTABLE/ CKA_EXPORTABLE	These attributes are used to specify that the key may be extracted from the token in an encrypted (for example, wrapped) form. These attributes determine how the key may be backed up. Please consult the key backup section in "Unauthenticated Users" on page 79 for more information.

APPENDIX C

KMU Key Check Value (KCV) Calculation

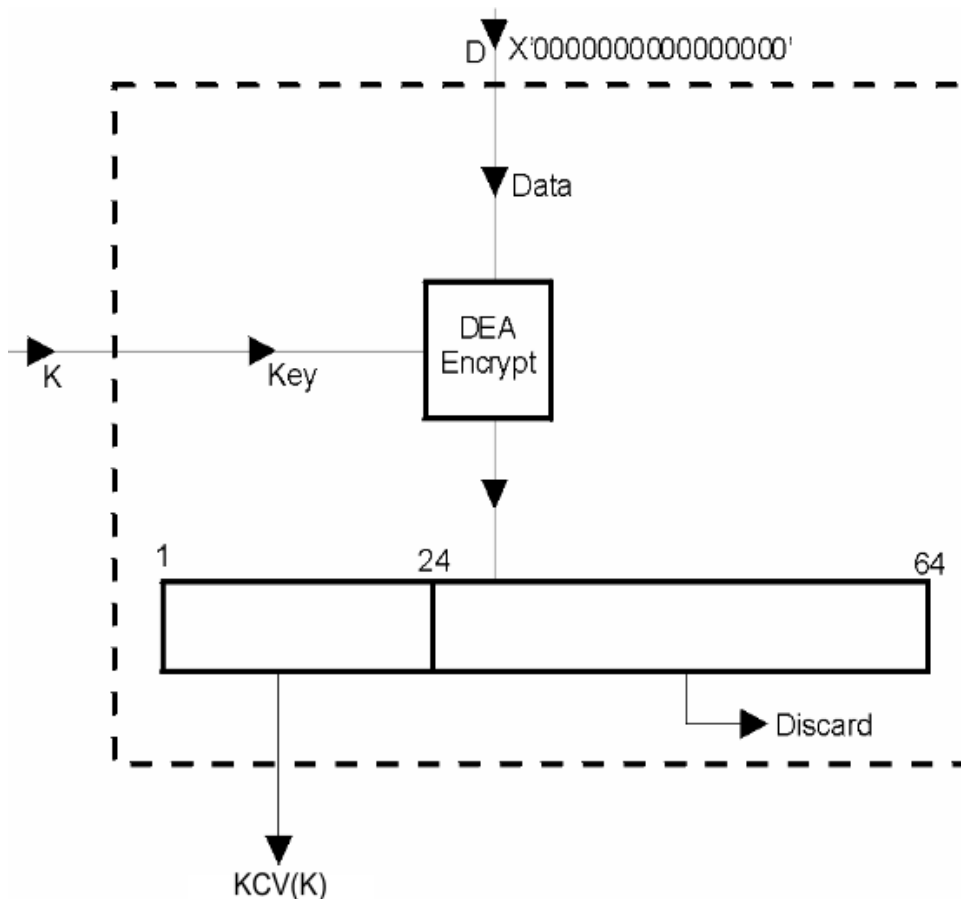
The Key Management Utility calculates and displays keys according to AS 2805.6.3.

Single-length Key KCV

The single-length key check value is a one-way cryptographic function of a key, used to verify that the key has been entered correctly.

The KCV is calculated by taking an input of constant D (64 Zero bits) and encrypting it with key K (64 bit). The 64 bit output is truncated to the most significant 24 bits which is reported as the keys KCV ("Single-length Key Check Value KCV(K)." below).

Figure 1: Single-length Key Check Value KCV(K).

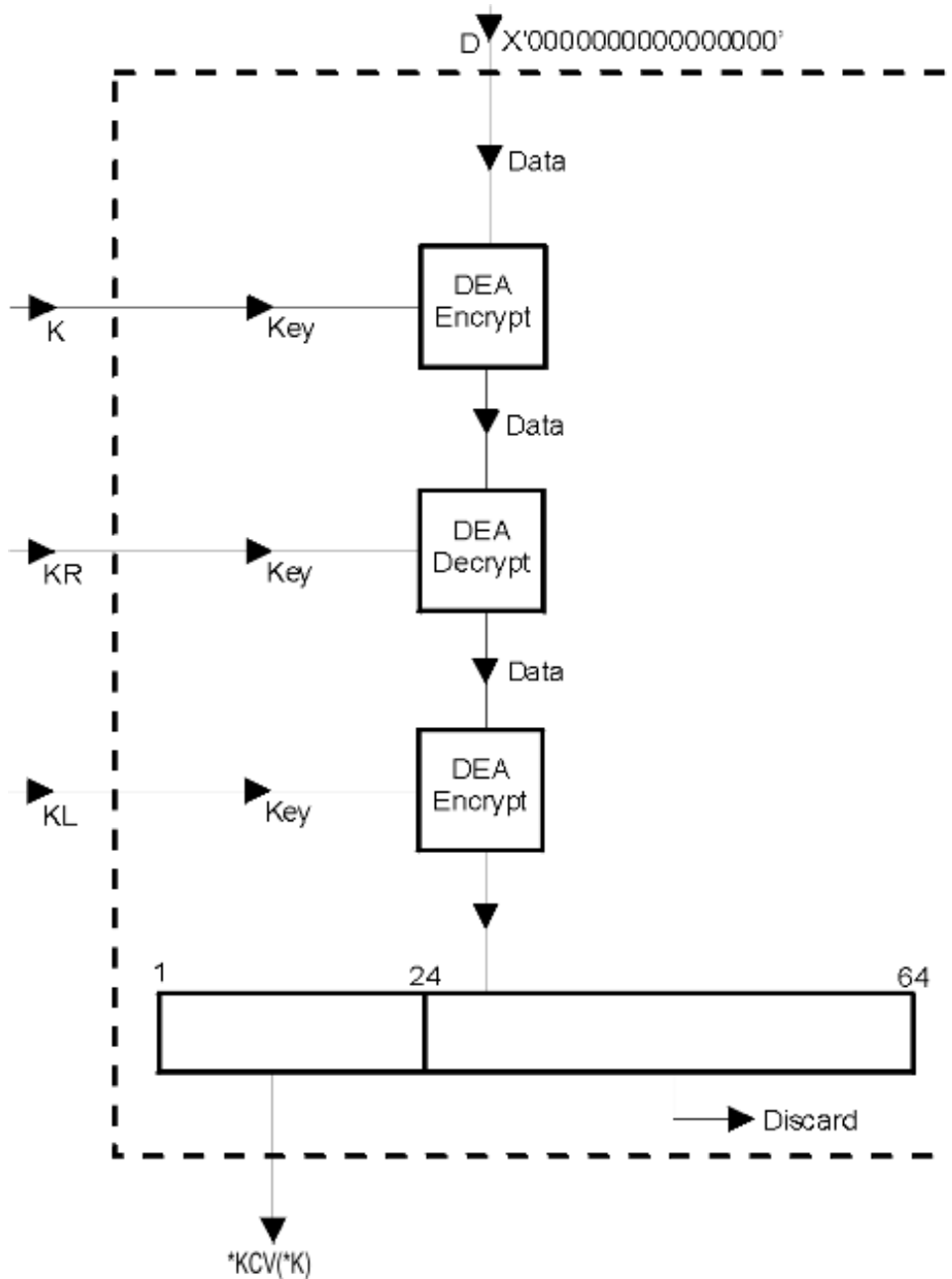


Double-length Key KCV

The double-length key check value is a one-way cryptographic function of a key, used to verify that the key has been correctly entered.

The KCV is calculated by taking an input of constant D (64 Zero bits) and key *K (128 bit string made up of two 64 bit values KL and KR). Data value D is encrypted with KL as the key. The result is decrypted with KR as the key. The result is then encrypted with KL as the key. The 64 bit output is truncated to the most significant 24 bits which is reported as the double-length keys *KCV ("Double-length Key Check Value *KCV(*K)" below).

Figure 2: Double-length Key Check Value *KCV(*K)



APPENDIX D

Key Migration from SafeNet ProtectToolkit-C V4.1

SafeNet ProtectToolkit-C version 4.1 has introduced new key wrapping mechanisms to support algorithms that meet the minimum key requirements for NIST 2010.

The old algorithms are still supported, but may be disabled if the HSM is operating in FIPS mode.

The smart card and file backups data contains version information that allows the **ctkm** or **KMU** to determine what mechanism should be applied to import that keys.

The **ctkm** and **KMU** will by default produce backup images using the new mechanisms. The **ctkm** utility has a **-3** option that will cause it to create backup images using the older mechanism so that they can be imported into an older HSM.

APPENDIX E

Sample EC Domain Parameter Files

This Appendix describes the domain parameters of two EC curves. It contains the following sections:

- ["C2tnB191v1" on the next page](#)
- ["brainpoolP160r1" on page 175](#)
- ["Hexadecimal to Decimal Conversion Table" on page 176](#)

C2tnB191v1

```
#
#This file describes the domain parameters of an EC curve
#
#File contains lines of text. All lines not of the form key=value are ignored.
#All values must be Hexidecimal numbers except m, k, k1, k2 and k3 which are decimal.
#Lines starting with ';' or '#' are comments.
#
#Keys recognised for fieldID values are -
#prime          - only if the Curve is based on a prime field
#m              - only if the curve is based on a 2^M field
#k              - only if the curve is 2^M field and is Trinomial basis
#k1, k2, k3     - these three only if 2^M field and Pentanomial basis
#
#You should have these combinations of fieldID values -
#prime          - if Curve is based on a prime field
#m              - if curve is based on 2^M and Basis is Gaussian normal basis
#m,k            - if curve is based on 2^M and Basis is Polynomial basis
#m,k1,k2,k3     - if curve is based on 2^M and Basis is Pentanomial basis
#
#These are the values common to prime fields and polynomial fields.
#curveA         - field element A
#curveB         - field element B
#curveSeed      - this one is optional
#baseX          - field element Xg of the point G
#baseY          - field element Yg of the point G
#bpOrder        - order n of the point G
#cofactor       - (optional) cofactor h
#
#
# Curve name C2tnB191v1

m          = 191
k          = 9
curveA     = 2866537B676752636A68F56554E12640276B649EF7526267
curveB     = 2E45EF571F00786F67B0081B9495A3D95462F5DE0AA185EC
baseX      = 36B3DAF8A23206F9C4F299D7B21A9C369137F2C84AE1AA0D
baseY      = 765BE73433B3F95E332932E70EA245CA2418EA0EF98018FB
bpOrder    = 400000000000000000000000000000004A20E90C39067C893BBB9A5
```

brainpoolP160r1

```
#
#This file describes the domain parameters of an EC curve
#
#File contains lines of text. All lines not of the form key=value are ignored.
#All values must be Hexidecimal numbers except m, k, k1, k2 and k3 which are decimal.
#Lines starting with ';' or '#' are comments.
#
#Keys recognised for fieldID values are -
#prime          - only if the Curve is based on a prime field
#m              - only if the curve is based on a 2^M field
#k              - only if the curve is 2^M field and is Trinomial basis
#k1, k2, k3     - these three only if 2^M field and Pentanomial basis
#
#You should have these combinations of fieldID values -
#prime          - if Curve is based on a prime field
#m              - if curve is based on 2^M and Basis is Gaussian normal basis
#m,k            - if curve is based on 2^M and Basis is Polynomial basis
#m,k1,k2,k3     - if curve is based on 2^M and Basis is Pentanomial basis
#
#These are the values common to prime fields and polynomial fields.
#curveA         - field element A
#curveB         - field element B
#curveSeed      - this one is optional
#baseX          - field element Xg of the point G
#baseY          - field element Yg of the point G
#bpOrder        - order n of the point G
#cofactor       - (optional) cofactor h
#
#
# Curve name brainpoolP160r1

prime          = E95E4A5F737059DC60DFC7AD95B3D8139515620F
curveA         = 340E7BE2A280EB74E2BE61BADA745D97E8F7C300
curveB         = 1E589A8595423412134FAA2DBDEC95C8D8675E58
baseX          = BED5AF16EA3F6A4F62938C4631EB5AF7BDBCDBC3
baseY          = 1667CB477A1A8EC338F94741669C976316DA6321
bpOrder        = E95E4A5F737059DC60DF5991D45029409E60FC09
```

Hexadecimal to Decimal Conversion Table

Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec
00	000	20	032	40	064	60	096	80	128	A0	160	C0	192	E0	224
01	001	21	033	41	065	61	097	81	129	A1	161	C1	193	E1	225
02	002	22	034	42	066	62	098	82	130	A2	162	C2	194	E2	226
03	003	23	035	43	067	63	099	83	131	A3	163	C3	195	E3	227
04	004	24	036	44	068	64	100	84	132	A4	164	C4	196	E4	228
05	005	25	037	45	069	65	101	85	133	A5	165	C5	197	E5	229
06	006	26	038	46	070	66	102	86	134	A6	166	C6	198	E6	230
07	007	27	039	47	071	67	103	87	135	A7	167	C7	199	E7	231
08	008	28	040	48	072	68	104	88	136	A8	168	C8	200	E8	232
09	009	29	041	49	073	69	105	89	137	A9	169	C9	201	E9	233
0A	010	2A	042	4A	074	6A	106	8A	138	AA	170	CA	202	EA	234
0B	011	2B	043	4B	075	6B	107	8B	139	AB	171	CB	203	EB	235
0C	012	2C	044	4C	076	6C	108	8C	140	AC	172	CC	204	EC	236
0D	013	2D	045	4D	077	6D	109	8D	141	AD	173	CD	205	ED	237
0E	014	2E	046	4E	078	6E	110	8E	142	AE	174	CE	206	EE	238
0F	015	2F	047	4F	079	6F	111	8F	143	AF	175	CF	207	EF	239
10	016	30	048	50	080	70	112	90	144	B0	176	D0	208	F0	240
11	017	31	049	51	081	71	113	91	145	B1	177	D1	209	F1	241
12	018	32	050	52	082	72	114	92	146	B2	178	D2	210	F2	242
13	019	33	051	53	083	73	115	93	147	B3	179	D3	211	F3	243
14	020	34	052	54	084	74	116	94	148	B4	180	D4	212	F4	244
15	021	35	053	55	085	75	117	95	149	B5	181	D5	213	F5	245
16	022	36	054	56	086	76	118	96	150	B6	182	D6	214	F6	246
17	023	37	055	57	087	77	119	97	151	B7	183	D7	215	F7	247
18	024	38	056	58	088	78	120	98	152	B8	184	D8	216	F8	248
19	025	39	057	59	089	79	121	99	153	B9	185	D9	217	F9	249

Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec
1A	026	3A	058	5A	090	7A	122	9A	154	BA	186	DA	218	FA	250
1B	027	3B	059	5B	091	7B	123	9B	155	BB	187	DB	219	FB	251
1C	028	3C	060	5C	092	7C	124	9C	156	BC	188	DC	220	FC	252
1D	029	3D	061	5D	093	7D	125	9D	157	BD	189	DD	221	FD	253
1E	030	3E	062	5E	094	7E	126	9E	158	BE	190	DE	222	FE	254
1F	031	3F	063	5F	095	7F	127	9F	159	BF	191	DF	223	FF	255

APPENDIX F

Glossary of terms

A

Adapter

The printed circuit board responsible for cryptographic processing in a HSM

AES

Advanced Encryption Standard

API

Application Programming Interface

ASO

Administration Security Officer

Asymmetric Cipher

An encryption algorithm that uses different keys for encryption and decryption. These ciphers are usually also known as public-key ciphers as one of the keys is generally public and the other is private. RSA and ElGamal are two asymmetric algorithms

B

Block Cipher

A cipher that processes input in a fixed block size greater than 8 bits. A common block size is 64 bits

Bus

One of the sets of conductors (wires, PCB tracks or connections) in an IC

C

CA

Certification Authority

CAST

Encryption algorithm developed by Carlisle Adams and Stafford Tavares

Certificate

A binding of an identity (individual, group, etc.) to a public key which is generally signed by another identity. A certificate chain is a list of certificates that indicates a chain of trust, i.e. the second certificate has signed the first, the third has signed the second and so on

CMOS

Complementary Metal-Oxide Semiconductor. A common data storage component

Cprov

ProtectToolkit C - SafeNet's PKCS #11 Cryptoki Provider

Cryptoki

Cryptographic Token Interface Standard. (aka PKCS#11)

CSA

Cryptographic Services Adapter

CSPs

Microsoft Cryptographic Service Providers

D

Decryption

The process of recovering the plaintext from the ciphertext

DES

Cryptographic algorithm named as the Data Encryption Standard

Digital Signature

A mechanism that allows a recipient or third party to verify the originator of a document and to ensure that the document has not be altered in transit

DLL

Dynamically Linked Library. A library which is linked to application programs when they are loaded or run rather than as the final phase of compilation

DSA

Digital Signature Algorithm

E

Encryption

The process of converting the plaintext data into the ciphertext so that the content of the data is no longer obvious. Some algorithms perform this function in such a way that there is no known mechanism, other than decryption with the appropriate key, to recover the plaintext. With other algorithms there are known flaws which reduce the difficulty in recovering the plaintext

F

FIPS

Federal Information Protection Standards

FM

Functionality Module. A segment of custom program code operating inside the CSA800 HSM to provide additional or changed functionality of the hardware

FMSW

Functionality Module Dispatch Switcher

H

HA

High Availability

HIFACE

Host Interface. It is used to communicate with the host system

HSM

Hardware Security Module

I

IDEA

International Data Encryption Algorithm

IIS

Microsoft Internet Information Services

IP

Internet Protocol

J

JCA

Java Cryptography Architecture

JCE

Java Cryptography Extension

K

Keyset

A keyset is the definition given to an allocated memory space on the HSM. It contains the key information for a specific user

KWRAP

Key Wrapping Key

M

MAC

Message authentication code. A mechanism that allows a recipient of a message to determine if a message has been tampered with. Broadly there are two types of MAC algorithms, one is based on symmetric encryption algorithms and the second is based on Message Digest algorithms. This second class of MAC algorithms are known as HMAC algorithms. A DES based MAC is defined in FIPS PUB 113, see <http://www.itl.nist.gov/div897/pubs/fip113.htm>. For information on HMAC algorithms see RFC-2104 at <http://www.ietf.org/rfc/rfc2104.txt>

Message Digest

A condensed representation of a data stream. A message digest will convert an arbitrary data stream into a fixed size output. This output will always be the same for the same input stream however the input cannot be reconstructed from the digest

MSCAPI

Microsoft Cryptographic API

MSDN

Microsoft Developer Network

P

Padding

A mechanism for extending the input data so that it is of the required size for a block cipher. The PKCS documents contain details on the most common padding mechanisms of PKCS#1 and PKCS#5

PCI

Peripheral Component Interconnect

PEM

Privacy Enhanced Mail

PIN

Personal Identification Number

PKCS

Public Key Cryptographic Standard. A set of standards developed by RSA Laboratories for Public Key Cryptographic processing

PKCS #11

Cryptographic Token Interface Standard developed by RSA Laboratories

PKI

Public Key Infrastructure

ProtectServer

SafeNet HSM

ProtectToolkit C

SafeNet's implementation of PKCS#11. Protecttoolkit C represents a suite of products including various PKCS#11 runtimes including software only, hardware adapter, and host security module based variants. A Remote client and server are also available

ProtectToolkit J

SafeNet's implementation of JCE. Runs on top of ProtectToolkit C

R

RC2/RC4

Ciphers designed by RSA Data Security, Inc.

RFC

Request for Comments, proposed specifications for various protocols and algorithms archived by the Internet Engineering Task Force (IETF), see <http://www.ietf.org>

RNG

Random Number Generator

RSA

Cryptographic algorithm by Ron Rivest, Adi Shamir and Leonard Adelman

RTC

Real Time Clock

S**SDK**

Software Development Kits Other documentation may refer to the SafeNet Cprov and Protect Toolkit J SDKs. These SDKs have been renamed ProtectToolkit C and ProtectToolkit J respectively. ⓘ The names Cprov and ProtectToolkit C refer to the same device in the context of this or previous manuals. ⓘ The names Protect Toolkit J and ProtectToolkit J refer to the same device in the context of this or previous manuals.

Slot

PKCS#11 slot which is capable of holding a token

SlotPKCS#11

Slot which is capable of holding a token

SO

Security Officer

Symmetric Cipher

An encryption algorithm that uses the same key for encryption and decryption. DES, RC4 and IDEA are all symmetric algorithms

T**TC**

Trusted Channel

TCP/IP

Transmission Control Protocol / Internet Protocol

Token

PKCS#11 token that provides cryptographic services and access controlled secure key storage

TokenPKCS#11

Token that provides cryptographic services and access controlled secure key storage

U**URI**

Universal Resource Identifier

V**VA**

Validation Authority

X**X.509**

Digital Certificate Standard

X.509 Certificate

Section 3.3.3 of X.509v3 defines a certificate as: "user certificate; public key certificate; certificate: The public keys of a user, together with some other information, rendered unforgeable by encipherment with the private key of the certification authority which issued it"