

ProtectToolkit 5.9.1 PTK-C ADMINISTRATION GUIDE



Document Information

| | |
|--------------|---------------------------|
| Last Updated | 2024-04-18 12:26:35-04:00 |
|--------------|---------------------------|

Trademarks, Copyrights, and Third-Party Software

Copyright 2009-2024 Thales Group. All rights reserved. Thales and the Thales logo are trademarks and service marks of Thales Group 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 Thales Group 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 Thales Group'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, Thales Group 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, Thales reserves the right to make any change or improvement in the specifications data, information, and the like described herein, at any time.

Thales Group 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 Thales Group 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.

Thales Group 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 Thales Group be held liable for any third party actions and in particular in case of any successful attack against systems or equipment incorporating Thales products. Thales Group 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 Thales Group.

CONTENTS

| | |
|---|-----------|
| Preface: About the ProtectToolkit-C Administration Guide | 9 |
| Document Conventions | 9 |
| Support Contacts | 11 |
| Chapter 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 |
| Chapter 2: Cryptoki Configuration | 14 |
| The ProtectToolkit-C Model | 14 |
| Slots and Tokens | 15 |
| User Slots | 15 |
| Smart Card Slots | 15 |
| The Admin Slot | 16 |
| PKCS #11 Objects | 16 |
| Administration Objects | 16 |
| User Roles | 16 |
| PINs and Passwords | 17 |
| Initial Configuration | 17 |
| Synchronizing the HSM Time With the System Clock | 18 |
| Setting the Admin Token PINs | 18 |
| Selecting and Setting a Security Policy | 19 |
| Setting up Slots | 19 |
| Multiple Adapter HSMs | 20 |
| Token Initialization | 20 |
| Trust Management | 21 |
| Secure Messaging | 23 |
| Messaging Mode Configuration | 24 |
| Configuring Session Protection | 25 |
| Establishing Trust Relationships | 26 |
| Token Replication | 28 |
| Replicate Master Tokens to a Single Slot or List of Slots | 28 |
| Work Load Distribution Model (WLD) and High Availability (HA) | 30 |
| WLD | 30 |
| HA | 31 |
| ProtectToolkit-C Configuration | 31 |
| WLD Slots | 31 |

| | |
|--|-----------|
| Distribution Scheme | 31 |
| Token Replication | 32 |
| WLD System Setup | 32 |
| Configuring WLD Slots | 36 |
| Operation in WLD Mode | 37 |
| Operation in HA Mode | 37 |
| HA Mode Logging | 38 |
| External Key Storage | 39 |
| External Key Storage Model | 40 |
| External Key Storage Configuration | 42 |
| Creating Externally Stored Objects | 45 |
| Real-Time Clock | 46 |
| Setting the Rule for RTC Adjustment Access Control | 46 |
| Enabling/Disabling RTC Adjustment Access Control | 47 |
| Chapter 2: ProtectToolkit-C Configuration Items | 48 |
| General Configuration Items | 48 |
| Work Load Distribution and High Availability Configuration Items | 48 |
| Logger Configuration Items | 49 |
| External Key Storage Configuration Items | 51 |
| Secure Messaging Configuration Items | 51 |
| Software-Only Mode Configuration Items | 52 |
| Chapter 3: Security Policies and User Roles | 53 |
| PKCS #11 Compliance and Security | 54 |
| Typical Security Policies | 54 |
| PKCS #11 Compatibility Mode | 54 |
| Default Mode | 54 |
| FIPS Mode | 55 |
| Entrust Compliant Modes | 56 |
| Netscape Compliant Mode | 56 |
| Restricted Mode | 56 |
| Security Flags | 56 |
| Configuring Security Flags | 57 |
| Security Flag Descriptions | 58 |
| Security Policy Options | 62 |
| User Roles | 64 |
| Administration Security Officer (ASO) | 64 |
| Administrator | 64 |
| Security Officer (SO) | 65 |
| Token Owner (User) | 65 |
| Unauthenticated Users | 65 |
| Chapter 4: Audit Logging | 67 |
| Audit Logging Overview | 67 |
| Logged Events | 67 |
| The Auditor Role | 67 |

| | |
|--|-----------|
| Audit Key | 68 |
| Log Verification | 69 |
| Log Capacity and Rotation | 69 |
| Configuring and Using Audit Logging | 69 |
| Initialize the Audit User and Create the Audit Key | 69 |
| Enable Audit Logging | 70 |
| Configure Audit Logging | 70 |
| Verify the Logs | 71 |
| Disable Audit Logging | 72 |
| Audit Log Events and Structure | 73 |
| Logged Events | 73 |
| Entry Structure | 74 |
| SNMP Monitoring | 76 |
| Querying the ProtectServer Network HSM via SNMP | 76 |
| HSM Information | 77 |
| Appliance Information | 78 |
| Chapter 6: Operational Tasks | 80 |
| Changing a User or Security Officer PIN | 80 |
| Secure Key Backup and Restoration | 81 |
| Determining Backup Requirements | 81 |
| Available Backup and Recovery Methods | 81 |
| A Typical Key Backup and Recovery Scheme | 82 |
| Key Backup Procedure | 84 |
| Key Restore Procedure | 85 |
| Adding and Removing Slots | 86 |
| Adding Slots | 86 |
| Removing Slots | 86 |
| Re-initializing a Token | 86 |
| Multifactor Authentication (One-Time Password) | 87 |
| Activating Your SafeNet 110 OTP Token | 88 |
| Initializing Multifactor Authentication | 88 |
| Logging In Using Multifactor Authentication | 89 |
| Re-initializing Multifactor Authentication For the User Role | 90 |
| Removing Multifactor Authentication From a Role | 91 |
| Connecting and Removing Smart Card Readers | 91 |
| Using Transport Mode to Avoid a Board Removal Tamper | 92 |
| Adjusting the HSM Clock | 92 |
| Changing Secure Messaging Mode | 93 |
| Managing Session Key Rollover | 93 |
| Using the System Event Log | 93 |
| Viewing and Interpreting the Event Log | 93 |
| Purging the Event Log | 93 |
| Updating the HSM Firmware | 93 |
| Update Prerequisites | 94 |
| Updating the Firmware | 94 |
| Installing a Functionality Module | 94 |

| | |
|---|------------|
| Loading an FM Causes Halt and Reset | 95 |
| Key Entry via PIN Pad | 95 |
| Using a PIN pad for Key Entry | 95 |
| Hexadecimal to Decimal Conversion Table | 97 |
| Tampering or Decommissioning the HSM | 97 |
| Hardware Tamper Procedures | 98 |
| Software Tamper Procedure | 99 |
| RMA and Shipping Back to Thales | 99 |
| Chapter 7: Command Line Utilities Reference | 101 |
| ctcert | 102 |
| ctcheck | 113 |
| ctconf | 117 |
| ctfm | 122 |
| ctident | 125 |
| ctkmu | 129 |
| ctlimits | 139 |
| ctmultitoken | 142 |
| Syntax | 142 |
| Operating Modes | 144 |
| Named and User-Defined Curves | 150 |
| ctotp | 151 |
| ctperf | 154 |
| ctstat | 160 |
| auditverify | 161 |
| Chapter 8: Administration Utility (gCTAdmin) Reference | 163 |
| Logging In and Out | 163 |
| Main gCTAdmin Interface | 164 |
| Toolbar Buttons | 164 |
| Slot and Token Management | 165 |
| Creating Slots | 165 |
| Removing Slots | 165 |
| Initializing a Token | 166 |
| Setting the Token User PIN | 167 |
| Setting the Token SO PIN | 167 |
| Resetting a Token | 168 |
| HSM Management | 168 |
| Setting the Security Policy | 168 |
| Setting the Transport Mode | 169 |
| Clock Drift Correction | 170 |
| Viewing and Purging the System Event Log | 170 |
| Updating HSM Firmware | 171 |
| Tampering the HSM | 172 |
| Chapter 9: Key Management Utility (KMU) Reference | 173 |
| Compatibility Issues | 174 |

| | |
|--|-----|
| Main KMU Interface | 174 |
| Token and Key Selection | 175 |
| Toolbar Buttons | 175 |
| Retrieving Information about a Token | 176 |
| Logging Into and Out From Tokens | 176 |
| Creating Keys | 177 |
| Available Keys | 178 |
| Key Attribute Types | 178 |
| Creating a Random Secret Key | 179 |
| Creating a Random Key Pair | 180 |
| Creating Key Components | 182 |
| Entering a Key from Components | 184 |
| Editing Key Attributes | 185 |
| Deleting a Key | 186 |
| Display Key Check Value | 186 |
| Importing and Exporting Keys | 186 |
| Exporting Keys | 187 |
| Importing Keys | 190 |
| Key Backup Feature Tutorial | 193 |
| Key Definitions | 194 |
| Creation of Encrypted Key Set to Backup (Payload) | 194 |
| Backup to File | 194 |
| Backup to Smart Card - Single Custodian Mode | 195 |
| Backup to Smart Card - Multiple Custodian Mode | 196 |
| | |
| Chapter 10: PKCS #11 Attributes | 198 |
| | |
| Chapter 11: KMU Key Check Value (KCV) Calculation | 200 |
| Single-length Key KCV | 200 |
| Double-length Key KCV | 200 |
| | |
| Appendix A: Key Migration from ProtectToolkit-C V4.1 | 202 |
| | |
| Appendix B: Sample EC Domain Parameter Files | 203 |
| C2tnB191v1 | 203 |
| brainpoolP160r1 | 204 |
| | |
| Glossary | 205 |

PREFACE: About the ProtectToolkit-C Administration Guide

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

- > ["Introduction" on page 12](#)
- > ["Cryptoki Configuration" on page 14](#)
- > ["Security Policies and User Roles" on page 53](#)
- > ["Audit Logging" on page 67](#)
- > ["SNMP Monitoring" on page 76](#)
- > ["Operational Tasks" on page 80](#)
- > ["Key Entry via PIN Pad" on page 95](#)
- > ["Command Line Utilities Reference" on page 101](#)
- > ["Key Management Utility \(KMU\) Reference" on page 173](#)
- > ["Administration Utility \(gCTAdmin\) Reference" on page 163](#)
- > ["PKCS #11 Attributes" on page 198](#)
- > ["KMU Key Check Value \(KCV\) Calculation" on page 200](#)
- > ["Key Migration from ProtectToolkit-C V4.1" on page 202](#)
- > ["Sample EC Domain Parameter Files" on page 203](#)

This preface also includes the following information about this document:

- > ["Document Conventions" below](#)
- > ["Support Contacts" on page 11](#)

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

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 cross-references to other documents in this documentation set. |
| <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 refer to the documentation before contacting support. If you cannot resolve the issue, contact your supplier or [Thales Customer Support](#).

Thales Customer 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 Thales and your organization. Please consult this support plan for further information about your entitlements, including the hours when telephone support is available to you.

Customer Support Portal

The Customer Support Portal, at <https://supportportal.thalesgroup.com>, is where you can find solutions for most common problems. The Customer Support Portal is a comprehensive, fully searchable database of support resources, including software and firmware downloads, release notes listing known problems and workarounds, a knowledge base, FAQs, product documentation, technical notes, and more. You can also use the portal to create and manage support cases.

NOTE You require an account to access the Customer Support Portal. To create a new account, go to the portal and click on the **REGISTER** link.

Telephone

The support portal also lists telephone numbers for voice contact ([Contact Us](#)).

CHAPTER 1: Introduction

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 Thales Technical Support Customer Portal. See ["Support Contacts" on page 11](#) for more information.

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

- > ProtectServer Network HSM
- > 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 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 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 ProtectToolkit-C Administrator, responsible for installation, configuration, security policy and number of applications (or users) of ProtectToolkit-C. This configuration of ProtectToolkit-C will determine the functionality and services available to the 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 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 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.

- > *ProtectServer HSM and ProtectToolkit Installation Guide*
- > *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 Thales 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 **ctkmu** refers to the **ctkmu** utility operating in NORMAL mode, while, **ctkmu (WLD)** refers to the **ctkmu** utility operating in WLD mode. Refer to section ["Operation in WLD Mode" on page 37](#) for details.

CHAPTER 2: Cryptoki Configuration

A number of steps must be taken in order for applications to operate correctly with ProtectToolkit-C. The 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 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 ProtectToolkit-C. The chapter then covers the steps required to configure a system utilizing 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.

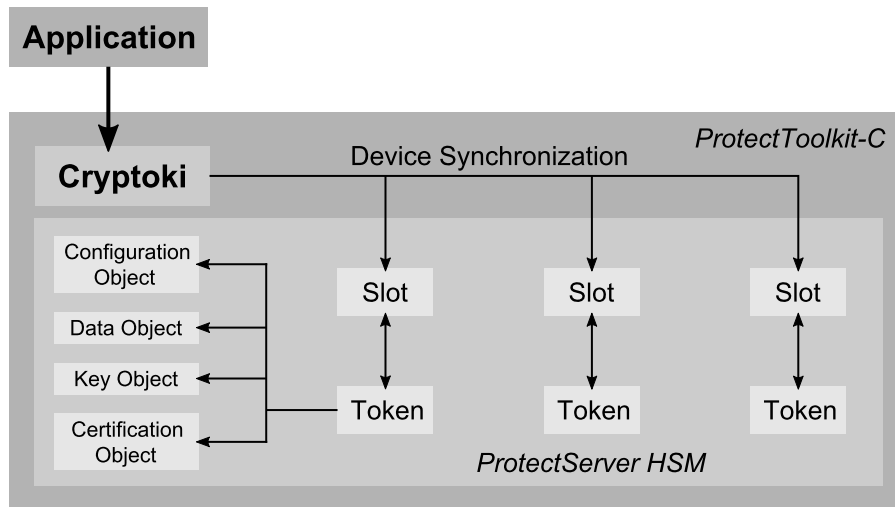
This chapter contains the following sections:

- > ["The ProtectToolkit-C Model" below](#)
- > ["Initial Configuration" on page 17](#)
 - ["Trust Management" on page 21](#)
 - ["Establishing Trust Relationships" on page 26](#)
 - ["Token Replication" on page 28](#)
- > ["Work Load Distribution Model \(WLD\) and High Availability \(HA\)" on page 30](#)
- > ["External Key Storage" on page 39](#)
- > ["Real-Time Clock" on page 46](#)
- > ["ProtectToolkit-C Configuration Items" on page 48](#)

The ProtectToolkit-C Model

The model for ProtectToolkit-C is based on standard PKCS #11 processing. 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: 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.

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 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. For more information about administration objects, see "[Administration Objects](#)" below.

PKCS #11 Objects

As shown in "[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 198.

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

For 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 53.

PINs and Passwords

PINs and passwords are used to authenticate users and to provide access to secured computer systems. In Cryptoki and ProtectToolkit-C, they are defined as variable-length strings of characters selected from the ANSI C character set. User PINs are case-sensitive, and must be 4-32 characters in length. If you are using "[Multifactor Authentication \(One-Time Password\)](#)" on page 87, the PIN length becomes 10-38 characters (userpin + OTP).

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 ProtectToolkit-C implementations utilizing the 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:

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

- > you can access the ProtectToolkit-C utilities used to carry out configuration tasks, as described in [Configuration Items](#) in the *ProtectServer HSM and ProtectToolkit Installation Guide*.

Synchronizing the HSM Time With the System Clock

When setting up a ProtectServer HSM for the first time, use the **ctconf** utility to synchronize the HSM with the system clock, to ensure that logs are produced with the correct time stamps (see "[ctconf](#)" on page 117).

To synchronize the HSM time with the system clock

Use the following command:

```
ctconf -t
```

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.

To set the Admin token PINs

1. From a command prompt, type **ctconf** and press **ENTER**.

You are prompted to set the Administration Security Officer (ASO) PIN. User PINs are case-sensitive, and must be 4-32 characters in length.

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. User PINs are case-sensitive, and must be 4-32 characters in length.

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 94](#).

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 ProtectToolkit-C is allowed to function. Setting the security policy is the most important part of ProtectToolkit-C initial configuration.

A number of security settings offered by 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 53](#) 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, 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 15](#).

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 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 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 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 53](#).

To initialize a token

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 \(WLD\) and High Availability \(HA\)" on page 30](#) for details.

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.

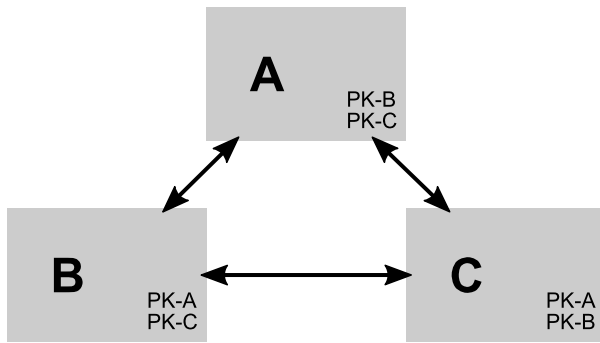
Public-key cryptography establishes trust between HSMs. Private keys are used for signing extracted information and unwrapping tokens. Public keys are used for wrapping tokens and verifying signed information. An RSA key-pair must be generated on the administrative token of each device. This key-pair 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 2: Simple trust relationships



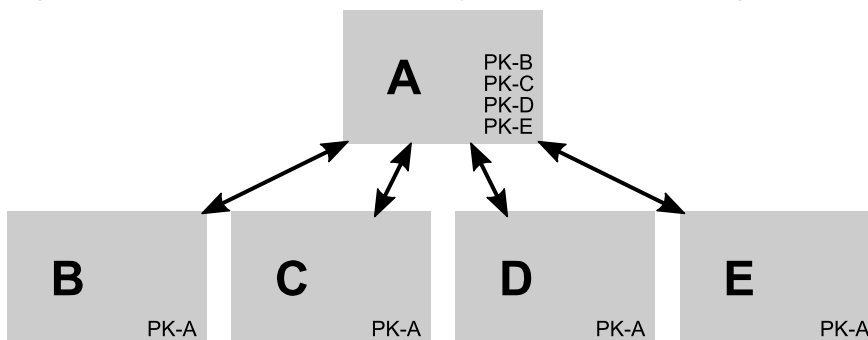
["Relationships of mutual trust" on the next page](#) shows a system where every HSM shares a relationship of mutual trust with every other HSM. In this scenario, token replication can be performed from any HSM to any other HSM on the system.

Figure 3: 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.

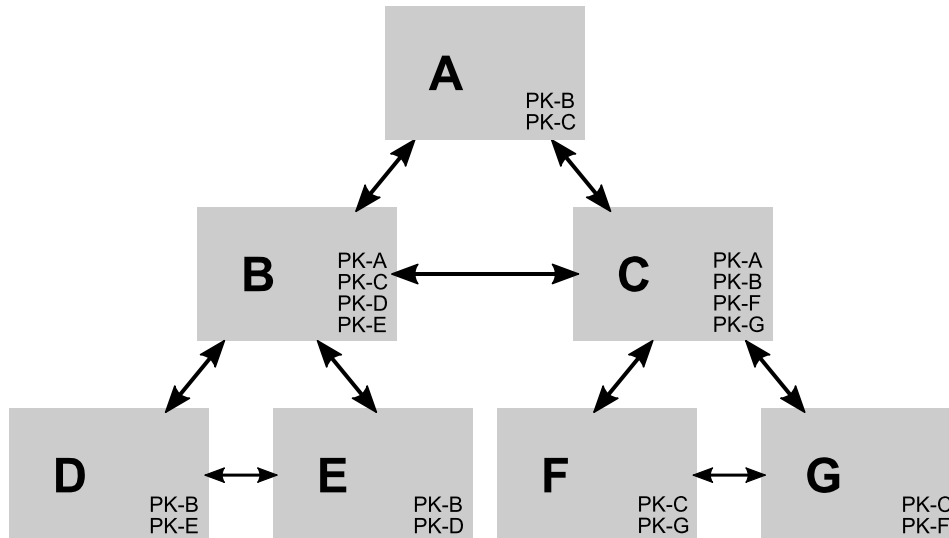
"Trust relationships in a typical WLD/HA configuration" below illustrates a system in a typical WLD configuration. In this system, HSM A has been selected to hold the master tokens.

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 4: 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 5: Complex trust topology



The **ctident** utility provides the mechanism for establishing, maintaining and removing trust relationships on HSMs. It can also be used to rollover the HSM identity keys used in trust management. 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" on page 26](#) shows how to establish trust among HSMs.

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 an Anonymous Diffie-Hellman key.

To configure and enable SMS

1. Configure secure messaging mode.

You may need to change the session key rollover default configuration. See the section ["Configuring Session Protection" on page 25](#) 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 page 25](#).

Messaging Mode Configuration

Anonymous Diffie-Hellman (ADH) mode selection

With the No Clear PINs flag set (see ["No Clear PINs" on page 60](#)), the ProtectServer Client software uses the default, NIST-approved ADH2 mode for secure messaging using SHA-512-based MAC. The default mode can be overridden and set to the legacy ADH mode (which uses SHA-1).

To change the SMS mode to ADH, set the configuration item ET_PTKC_SMS_MODE to ADH. For more information about this configuration item, refer to ["Secure Messaging Configuration Items" on page 51](#).

```
$ ET_PTKC_SMS_MODE=ADH
```

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 more information about this configuration item, refer to ["Secure Messaging Configuration Items" on page 51](#).

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](#) in the *ProtectServer HSM and ProtectToolkit Installation Guide* 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 more information about this configuration item, refer to ["Secure Messaging Configuration Items" on page 51](#).

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](#) in the *ProtectServer HSM and ProtectToolkit Installation Guide* 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 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 53](#).

| 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 ProtectToolkit-C **ctconf** utility command, **ctconf -f<flags>**. Refer to ["Security Policies and User Roles" on page 53](#) 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.

To Set SMPR Security Flags

1. Obtain the serial number of the HSM.

This can be done by executing the command `ctconf -a<device>` from a command line, where <device> is the number of the HSM in the list of HSMs.

2. Create the following configuration item:

```
ET_PTKC_<serial>_SMPR
```

...where <serial> is the serial number of the HSM found in step 1.

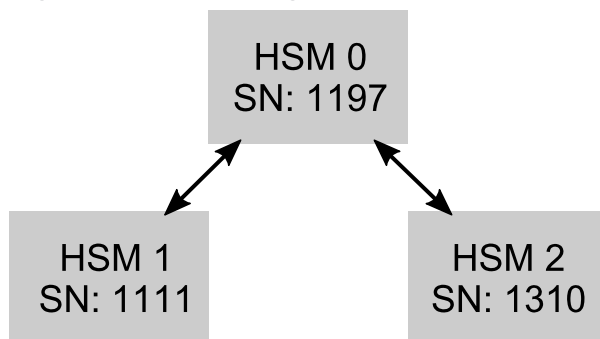
This change can be made at the temporary, user or system levels on both UNIX and Windows platforms. Refer to [Configuration Items](#) in the *ProtectServer HSM and ProtectToolkit Installation Guide* for further details on how to go about this if required.

3. Set one or more flags by assigning a value to the configuration item. For more information about the valid values for this configuration item, refer to ["Secure Messaging Configuration Items" on page 51](#). For example, if both Auth Protection and Auth Replies are required, assign the value **SR**.

Establishing Trust Relationships

The following example describes how to set-up the trust relationships illustrated in ["Establishing trust relationships example configuration" below](#). 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 6: Establishing trust relationships example configuration



To configure a trust relationship between HSMs

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

Use the `ctkmu` 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:\>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)
<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 ProtectToolkit-C Model](#)" on page 14 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.

CAUTION! Back up any objects on the target tokens before attempting to replicate a master token, because all objects on the target tokens are erased before the objects from the master token are loaded.

When a system is operating in WLD mode, the token label identifies its associated virtual WLD slot. Refer to "[Work Load Distribution Model \(WLD\) and High Availability \(HA\)](#)" on page 30 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 following command:

```
ctkmu rt -d <slotlist> [-s <slot>]
```

Refer to "[ctkmu](#)" on page 129 for more information about the preceding command.

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 58 for more information.

The **ctkmu 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)

Replicate Master Tokens to a Single Slot or List of Slots

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

To replicate Master Tokens to a single slot or list of slots

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

Use the **ctkmu** utility. Refer to "[ctkmu](#)" on page 129 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:\>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)
<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 **ctkmu** utility with the **rt** command with two parameters: the slot exporting the token and the list of slots receiving the token (see "[ctkmu](#)" on page 129). 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 58 for more information.

Examples:

Replicate token from slot 0 to slot 5

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

Replicate token from slot 0 to slot 9

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

Replicate token from slot 1 to slot 4 and slot 10

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

To replicate a Master Token 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 20 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 (WLD) and High Availability (HA)

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.

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 WLD. Within 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 (see ["Establishing Trust Relationships" on page 26](#)). The **ctkmu** utility replicates a token once trust has been established.

HA

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

The HA feature keeps track of the commands sent to a session. In case of session failure, 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.

ProtectToolkit-C Configuration

To enable WLD/HA, ProtectToolkit-C must be configured to operate in WLD or HA mode. Refer to ["Operation in WLD Mode" on page 37](#) and ["Operation in HA Mode" on page 37](#) for details on each.

When applications use the ProtectToolkit-C interface in WLD/HA mode, the system of physical HSMs appears as a single virtual HSM. 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 36](#)). *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).
- > In WLD/HA mode, token and session objects on a WLD slot are only visible to the session that generated the objects.

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

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 (see ["Establishing Trust Relationships" on page 26](#)). The **ctkmu** utility is used to replicate a token once trust has been established. See ["Trust Management" on page 21](#) and ["Token Replication" on page 28](#) 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.

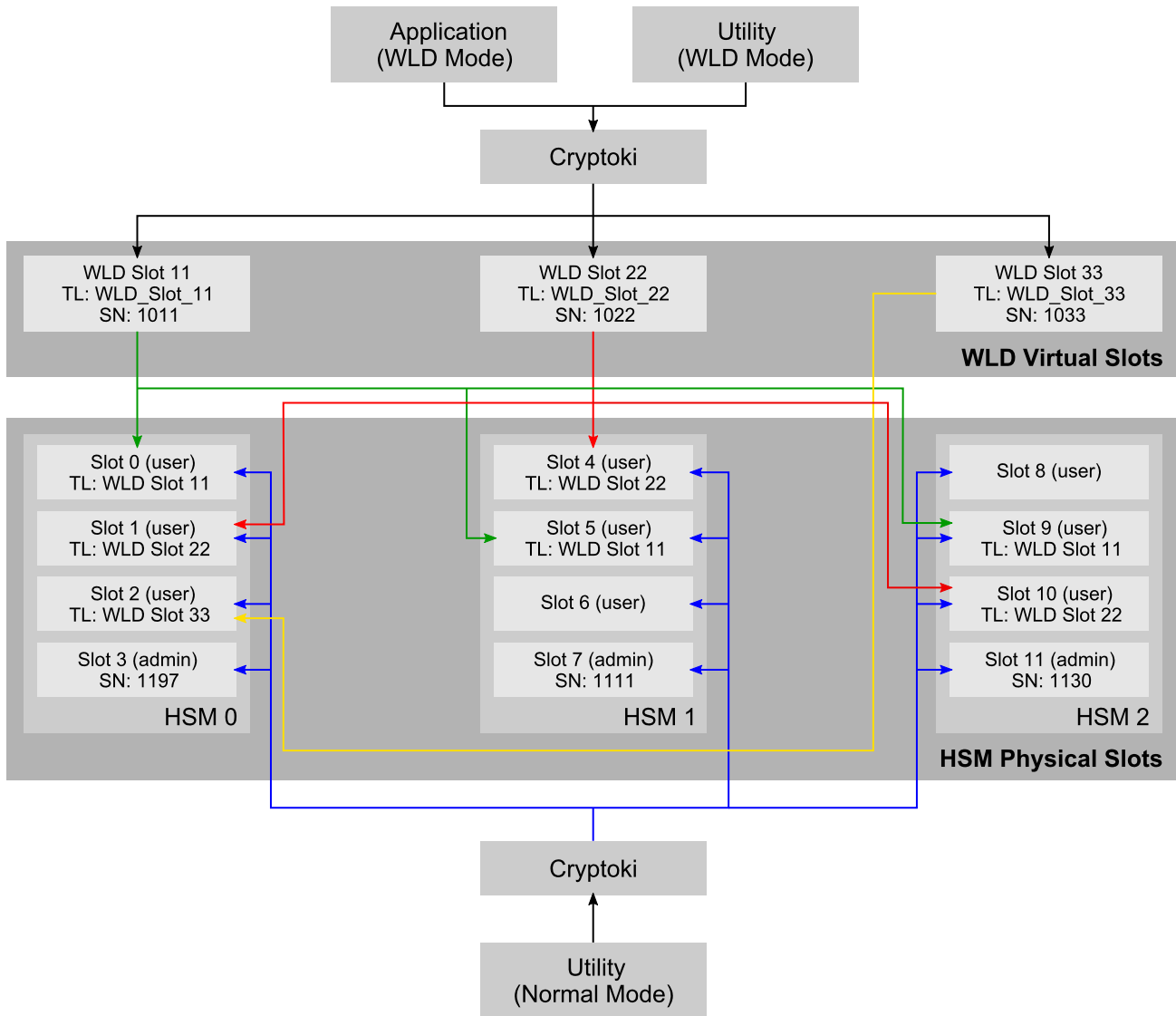
WLD System Setup

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 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 7: 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 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.

To configure the system for WLD

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\(s\)](#) in the "Configuration Items" section of the *ProtectServer HSM and ProtectToolkit Installation Guide* 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 37](#) 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 17](#) for further details.

4. Create User Slots.

Create User slots for each HSM, as described below. Refer to ["Initial Configuration" on page 17](#) 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 28](#) 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 21](#) 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 28](#) 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:

- 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
Copyright (c) Safenet, Inc.

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 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. For more information about configuring this environment variable, refer to ["Work Load Distribution and High Availability Configuration Items" on page 48](#).

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.

To configure WLD slots at the system level**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).

1. Open the file: **/etc/default/et_ptkc**

2. 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, 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**.

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 ProtectToolkit-C in WLD mode by editing the value of the ET_PTKC_GENERAL_LIBRARY_MODE environment variable. For more information about editing the value of this environment variable, refer to ["ProtectToolkit-C Configuration Items" on page 48](#).

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 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](#) in the *ProtectServer HSM and ProtectToolkit Installation Guide*):

```
ET_PTKC_WLD_SLOT_0=TokName
ET_PTKC_GENERAL_LIBRARY_MODE=WLD
```

Operation in HA Mode

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

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_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).

You must configure the Cryptoki library to operate ProtectToolkit-C in HA mode by editing the values of the ET_PTKC_GENERAL_LIBRARY_MODE, ET_PTKC_HA_RECOVER_DELAY, and ET_PTKC_HA_RECOVER_WAIT environment variables. For more information about editing the values of these environment variables, refer to "[ProtectToolkit-C Configuration Items](#)" on page 48.

To configure a basic HA system

To configure a basic HA system across two 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](#) in the *ProtectServer HSM and ProtectToolkit Installation Guide*):

```
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. The ET_PTKC_HA_LOG_FILE and ET_PTKC_HA_LOG_NAME configuration items can be used to configure HA mode logging. For more information about editing these configuration items, refer to "[Work Load Distribution and High Availability Configuration Items](#)" on page 48.

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 C_Initialize/C_Finalize 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 ET_HSM_NETCLIENT_SERVERLIST or in the order the 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 ET_PTKC_HA_RECOVER_DELAY and ET_PTKC_HA_RECOVER_WAIT are enabled. For more information about these configuration items, refer to "Work Load Distribution and High Availability Configuration Items" on page 48 . 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

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 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.

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

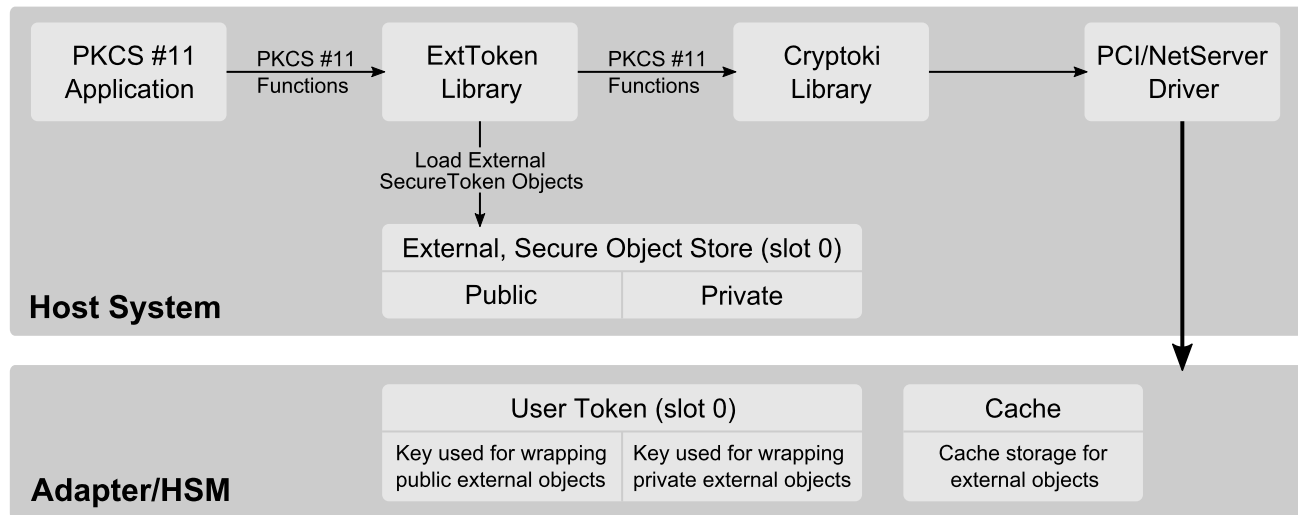
This section contains the following subsections:

- > ["External Key Storage Model" below](#)
- > ["External Key Storage Configuration" on page 42](#)
- > ["Creating Externally Stored Objects" on page 45](#)

External Key Storage Model

"External Key Storage" below shows how secure external key storage is achieved on the host system and HSM.

Figure 8: 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. ProtectToolkit-C's **ctperf** utility can be used to gauge performance on individual systems (see "[ctperf](#)" on page 154). 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`. For more information about this configuration item, refer to "[External Key Storage Configuration Items](#)" on page 51. 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 **ctkmu**. 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. For more information about this configuration item, refer to "[External Key Storage Configuration Items](#)" on page 51

The SafeNet implementation of JCA/JCE (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 keys to 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 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 |

| Path | Purpose | ID |
|--|---|----|
| 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 |

Configuring External Key Storage

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 40](#), 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:

- > ["To configure External Key Storage for application development" below](#)
- > ["To configure External Key Storage for Runtime operation" on the next page](#)

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](#) in the "Configuration Items" section of the *ProtectServer HSM and ProtectToolkit Installation Guide* for details.

To configure External Key Storage 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. For more information about configuring this environment variable, refer to ["External Key Storage Configuration Items" on page 51](#).
4. Configure the ET_PTKC_EXTTOKEN_PATH environment variable. For more information about configuring this environment variable, refer to ["External Key Storage Configuration Items" on page 51](#).

5. Configure the `ET_PTKC_EXTTOKEN_MAXLOADED` environment variable. For more information about configuring this environment variable, refer to ["External Key Storage Configuration Items" on page 51](#).

To check 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\Protect Toolkit 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 by checking if the folder has been created and contains an `.ord` file and an `.ort` file.

The `ET_PTKC_EXTTOKEN_PATH` determines the location of the storage files. For more information about this environment variable, refer to ["External Key Storage Configuration Items" on page 51](#).

To configure External Key Storage 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. For more information about configuring this environment variable, refer to ["External Key Storage Configuration Items" on page 51](#).
4. Configure the **ET_PTKC_EXTTOKEN_PATH** environment variable. For more information about configuring this environment variable, refer to ["External Key Storage Configuration Items" on page 51](#).
5. Configure the **ET_PTKC_EXTTOKEN_MAXLOADED** environment variable. For more information about configuring this environment variable, refer to ["External Key Storage Configuration Items" on page 51](#).

To check 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). For more information about configuring this environment variable, refer to ["External Key Storage Configuration Items" on page 51](#).
3. Enter the command **ctkmu I -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 by checking if the folder has been created and contains an **.ord** file and an **.ort** file.

The **ET_PTKC_EXTTOKEN_PATH** determines the location of the storage files. For more information about this environment variable, refer to ["External Key Storage Configuration Items" on page 51](#).

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.

To backup and restore keys by storing them externally

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**. For more information about this environment variable, refer to ["External Key Storage Configuration Items" on page 51](#).
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**.

- 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.
- 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
```

- 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 (refer to ["External Key Storage Configuration Items" on page 51](#)). 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
```

- 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 117](#) 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 |

To set guard parameters:

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
```

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 *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.

CHAPTER 2: ProtectToolkit-C Configuration Items

This chapter lists the available ProtectToolkit-C configuration items and, where applicable, their default values and valid range of values.

NOTE Thales recommends leaving configuration items at their default value or setting them to a valid value specified in the following table. If the value of a configuration item must be changed and no valid values are given, contact [Thales Customer Support](#) for assistance.

For more information about using configuration items see [Configuration Items](#).

General Configuration Items

The configuration items in the table below are used to configure ProtectToolkit-C more generally.

| Configuration Item | Meaning |
|------------------------------|--|
| ET_PTKC_GENERAL_LIBRARY_MODE | <p>The Cryptoki library operating mode.</p> <p>Valid values:</p> <ul style="list-style-type: none"> > NORMAL - Standard PKCS #11 mode. > WLD - Work Load Distribution mode. > HA - High Availability mode. <p>Default=NORMAL</p> |

Work Load Distribution and High Availability Configuration Items

The configuration items in the table below are used to configure ProtectToolkit-C for Work Load Distribution (WLD) and High Availability (HA). For more information about these modes, refer to "[Work Load Distribution Model \(WLD\) and High Availability \(HA\)](#)" on page 30.

| Configuration Item | Meaning |
|---------------------|--|
| ET_PTKC_HA_LOG_FILE | <p>The name of the ProtectToolkit-C file where the Cryptoki library generates log messages while operating in HA mode.</p> <ul style="list-style-type: none"> > Windows default= c:\ptk_halog.log > Linux default=/ptk_halog.log |

| Configuration Item | Meaning |
|--------------------------|---|
| ET_PTKC_HA_LOG_NAME | The name of the application. Default= ptk_cryptoki |
| ET_PTKC_HA_RECOVER_DELAY | The number of 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. Default= 0 |
| ET_PTKC_HA_RECOVER_WAIT | Whether the system will poll and attempt recovery if an HSM has failed. This configuration item is valid only if HA mode is enabled. Valid values: > YES > NO |
| ET_PTKC_WLD_SLOT_n | The configuration parameters of a WLD slot in a WLD system. In the name of this configuration item, <i>n</i> is an integer (in the range 0 to 99) that defines the slot number. Slot numbers allocated within an application must be unique. The value of this configuration item is specified in the following format: <WLDTokenLabel>[,<WLDTokenSerial#>][,<WLDSlotDescription>]] > <WLDTokenLabel> This variable 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#> This variable 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> This variable is optional. You can assign any PKCS #11 slot description you wish for this WLD Slot. The default value is "WLD Slot:n", where <i>n</i> is the same as the value of <i>n</i> in the configuration variable name. |

Logger Configuration Items

The configuration items in the table below are used to configure the logger library. For more information about the logger library, refer to [PKCS#11 Logger Library](#).

NOTE Values for the logger configuration items are located in the **HKEY_LOCAL_MACHINE\SOFTWARE\SafeNet\PTKC\LOGGER** key on Windows and stored in **/etc/default/et_ptkc** file on Linux.

| Configuration Item | Meaning |
|------------------------|--|
| ET_PTKC_LOGGER_FILE | <p>The name of the ProtectToolkit-C file where the logger library writes log information.</p> <ul style="list-style-type: none"> > Windows default=\ctlog.log > Linux default=~/ctlog.log |
| ET_PTKC_LOGGER_LOGMEM | <p>Whether all numeric data, buffer addresses, and the contents of buffer addresses at the input and output of functions (excluding PIN values) are included in log messages.</p> <p>Valid values:</p> <ul style="list-style-type: none"> > TRUE > FALSE - the contents of buffer addresses at the input and output of functions are omitted, while numeric data and buffer addresses are retained. <p>Default=TRUE</p> |
| ET_PTKC_LOGGER_LOGPID | <p>Whether the calling process ID (PID) is included in log messages.</p> <p>Valid values:</p> <ul style="list-style-type: none"> > TRUE > FALSE <p>Default=TRUE</p> |
| ET_PTKC_LOGGER_LOGPIN | <p>Whether the PIN values passed to C_Login, that are used to log into tokens, are included in log messages.</p> <p>Valid values:</p> <ul style="list-style-type: none"> > TRUE > FALSE <p>Default=FALSE</p> |
| ET_PTKC_LOGGER_LOGTID | <p>Whether the thread ID (TID) is included in log messages.</p> <p>Valid values:</p> <ul style="list-style-type: none"> > TRUE > FALSE <p>Default=TRUE</p> |
| ET_PTKC_LOGGER_LOGTIME | <p>Whether the date and time of each message is included in the log.</p> <p>Valid values:</p> <ul style="list-style-type: none"> > TRUE > FALSE <p>Default=TRUE</p> |

| Configuration Item | Meaning |
|--------------------------|---|
| ET_PTKC_LOGGER_PKCS11LIB | <p>Whether the logger is configured for HSM or Software-Only operating mode on Windows.</p> <p>> Valid values:</p> <ul style="list-style-type: none"> • C:\Program Files\SafeNet\Protect Toolkit 5\Protect Toolkit C SDK\bin\hsm\cryptoki.dll (for HSM mode) • C:\Program Files\SafeNet\Protect Toolkit 5\Protect Toolkit C SDK\bin\sw\cryptoki.dll (for Software Only mode) |

External Key Storage Configuration Items

The configuration items in the table below are used to configure external key storage. For more information about configuring external key storage, refer to ["External Key Storage" on page 39](#).

| Configuration Item | Meaning |
|----------------------------|--|
| ET_PTKC_EXTTOKEN_MAXLOADED | <p>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.</p> <p>Default=100</p> |
| ET_PTKC_EXTTOKEN_PATH | <p>The fully qualified directory path that determines where ExtToken library stores its data files. These data files will contain the encrypted key material.</p> <p>Default=C:\ETExtToken</p> |
| ET_PTKC_EXTTOKEN_PKCS11LIB | <p>The fully qualified file path to Cryptoki library to be used after configuring external key storage for application development or runtime operation. Located in the HLKM(or HKCU)\SOFTWARE\SafeNet\PTKC\EXTTOKEN registry key on Windows.</p> <p>Valid values:</p> <ul style="list-style-type: none"> > C:\Program Files\SafeNet\Protect Toolkit 5\Protect Toolkit C RT\hsm (When configuring external key storage for runtime operation) > C:\Program Files\SafeNet\Protect Toolkit 5\ProtectToolkit C SDK\bin\hsm (When configuring external key storage for application development in HSM operating modes) > C:\Program Files\SafeNet\Protect Toolkit 5\ProtectToolkit C SDK\bin\sw (When configuring external key storage for application development in software-only operating mode) |

Secure Messaging Configuration Items

The configuration items in the table below are used to configure the Secure Messaging System (SMS). For more information about the SMS, refer to ["Secure Messaging" on page 23](#).

| Configuration Item | Meaning |
|-----------------------|--|
| ET_PTKC_<serial>_SMPR | <p>The Secure Messaging Policy Register (SMPR) security mode flag(s) to enable. In the name of this configuration item, <serial> is the serial number of the HSM.</p> <p>Valid values:</p> <ul style="list-style-type: none"> > E - Only messages sent to the HSM that contain sensitive data are encrypted (No clear PINs). > S - Only messages sent to the HSM are signed (Auth Protection). > R - Only messages received from the HSM are signed (Auth Replies). |
| ET_PTKC_SMS_BLOCKS | <p>The number of blocks that must be encrypted or decrypted by the application before session key rollover is triggered.</p> <p>Default=4294967296</p> |
| ET_PTKC_SMS_HOURS | <p>The number of hours that must elapse before session key rollover is triggered.</p> <p>Default=24</p> |
| ET_PTKC_SMS_MODE | <p>Whether ProtectToolkit uses the legacy Anonymous Diffie-Hellman (ADH) mode for secure messaging when the No Clear PINs flag is set.</p> <p>Valid values:</p> <ul style="list-style-type: none"> > ADH <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>NOTE PTK firmware versions 5.01.00 and newer support ADH2 only. ADH is included for use with legacy firmware older than 5.01.00. Setting the SMS mode to ADH with newer firmware will return an error message.</p> </div> |

Software-Only Mode Configuration Items

The configuration items in the table below are used to configure ProtectToolkit-C for software-only mode. For more information about configuring software-only mode, refer to [Software-Only Mode Configuration](#).

| Configuration Item | Meaning |
|---------------------|---|
| ET_PTKC_SW_DATAPATH | <p>The directory within the local file system where keys and configuration information are stored.</p> <ul style="list-style-type: none"> > Windows default=C:\cryptoki > Linux default=\$HOME/.cryptoki/cryptoki |

CHAPTER 3: Security Policies and User Roles

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

A security policy is a set of security settings that control how 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 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 64](#).

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 56](#) for specific procedures.

PKCS #11 Compliance and Security

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 ProtectToolkit-C.

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 56](#).

For some policies, security flags may be available that alter security behavior without invalidating the policy. See ["Security Policy Options" on page 62](#).

For the complete **ctconf** command reference, see ["ctconf" on page 117](#).

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
```

Default Mode

By default (after initial HSM installation or following a tamper event), Default Mode is applied to 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 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

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 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

NOTE This flag requires a valid ProtectServer Identity Key/Certificate on the HSM. See for details and procedures.

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 or 3072 bits (verify - 1024 or 1536 bits)
- > **DSA**: must be 2048, 3072, or 4096 bits (verify - 1024 or 1536 bits)
- > **DH**: must be 2048 bits at minimum
- > **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

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 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 "[Default Mode](#)" on page 54, for more information about this security policy.

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

| To set flag: | Use flags value: |
|--|------------------|
| "Auth Protection" on the next page | u |
| "DES Keys Even Parity Allowed" on the next page | d |
| "Enable PCI Audit Logs" on the next page | b |
| "Entrust Ready" on the next page | e |
| "FIPS Algorithms Only" on page 59 | a |
| "FIPS Mode" on page 59 | F |
| "Full Secure Messaging Encryption" on page 59 | N |
| "Full Secure Messaging Signing" on page 60 | U |
| "Increased Security Level" on page 60 | i |
| "Mode Locked" on page 60 | l |
| "No Clear PINs" on page 60 | n |
| "No Public Crypto" on page 61 | c |
| "Pure PKCS11 (PKCS#11 Compatibility Mode)" on page 61 | p |
| "Tamper Before Upgrade" on page 61 | t |
| "User Specified ECC DomainParameters Allowed" on page 61 | E |

| To set flag: | Use flags value: |
|--|------------------|
| "Weak PKCS#11 Mechanisms" on page 62 | w |

Each of these flags is fully described below.

For the complete **ctconf** command reference, see ["ctconf" on page 117](#).

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 62](#).

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.

NOTE This flag requires a valid ProtectServer Identity Key/Certificate on the HSM. See for details and procedures.

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.

Enable PCI Audit Logs

The *Enable PCI Audit Logs* flag permits the collection of logs accessible to the **audit** user.

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 55 for more information.

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 55.

NOTE This flag requires a valid ProtectServer Identity Key/Certificate on the HSM. See for details and procedures.

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.

NOTE This flag requires a valid ProtectServer Identity Key/Certificate on the HSM. See for details and procedures.

The *No Clear PINs* flag must be set to enable *Full Secure Messaging Encryption* to encrypt data.

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.

NOTE This flag requires a valid ProtectServer Identity Key/Certificate on the HSM. See for details and procedures.

The *No Clear PINs* flag must be set to enable *Full Secure Messaging Encryption* to encrypt data.

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.

NOTE This flag requires a valid ProtectServer Identity Key/Certificate on the HSM. See for details and procedures.

The *No Clear PINs* flag must be set to enable *Full Secure Messaging Encryption* and *Full Secure Messaging Signing*.

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.

This flag requires a valid ProtectServer Identity Key/Certificate on the HSM. See for details and procedures.

Pure PKCS11 (PKCS#11 Compatibility Mode)

CAUTION! 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

CAUTION! 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

If you are using firmware 5.06.04 or higher, setting this security flag will allow you to change the value of the `CKA_EXPORTABLE` attribute of an object from `FALSE` to `TRUE`.

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 | |

| Security Policies | Impact of Security Flags on Policies | | | | | | | | | | | | |
|---------------------------------------|--------------------------------------|---|---|---|---|---|---|---|---|---|---|---|---|
| | a | c | d | e | i | l | n | N | p | t | u | U | E |
| Entrust Compliant Mode 2 ² | x | ✓ | x | x | | | x | x | | | x | x | |
| 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 ProtectToolkit-C configuration process, different *user roles* are assigned to those responsible for application administration and use.

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

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

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

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 the 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 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 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.

CHAPTER 4: Audit Logging

This chapter described how to use audit logging to provide security audits of HSM activity. It contains the following sections:

- > ["Audit Logging Overview" below](#)
- > ["Configuring and Using Audit Logging" on page 69](#)
- > ["Audit Log Events and Structure" on page 73](#)

Audit Logging Overview

Each event that occurs on the HSM can be recorded in the HSM event log, allowing you to audit your HSM usage. The HSM event log is viewable and configurable by the **audit** user only.

NOTE Audit logging is available on ProtectServer External 2 and ProtectServer External 2 Plus only; it is not supported on ProtectServer PCIe 2.

Logged Events

The types of events that can be logged include:

- > Administrative events
- > Object Management events
- > Object Use events

Events are logged whether they fail or succeed. For a complete list of logged events, see ["Audit Log Events and Structure" on page 73](#).

The Auditor Role

The audit logging function is controlled by two roles that must be used together:

- > The **audit** appliance account (use SSH or PuTTY to log in as **audit**, instead of **admin**, or **pseoperator**)
- > The Auditor HSM account (must be initialized, setting the Auditor PIN)

On ProtectServer, audit logging is managed by an **audit** user (an appliance system role), in combination with the HSM audit role, through a subset of PSESH commands. The **audit** user can perform only the audit-logging and self-related tasks. Other HSM appliance users have no access to the audit logging commands.

Upon first login, the **audit** user is asked to change their password. That user must initialize the HSM Auditor role before configuring audit logging.

To simplify configuration,

- > The log path is kept internal.
- > The log rotation is initially set to "never".

Audit User on the Appliance

The appliance **audit** user is a standard user account on ProtectServer, with the default password "password".

The **audit** user has a limited set of operations available, as reflected in the reduced command set available when logged in to the shell (PSESH).

```
login as: audit
Using keyboard-interactive authentication.
Password:
Last login: Thu Jul 13 10:21:02 2017 from 10.124.0.32
```

```
PSe 1.11-01 Command Line Shell - Copyright (c) 2001-2017 SafeNet, Inc. All rights reserved.
```

```
[PSe-II] psesh:>help
```

The following top-level commands are available:

| Name | (short) | Description |
|--------|---------|--------------------------|
| audit | a | > Manage Audit Log Files |
| help | h | Get Help |
| exit | e | Exit PSE-II Shell |
| syslog | sy | > Syslog |
| user | u | Set User Password |

Auditor Role on the HSM

The Auditor role allows complete separation of Audit responsibilities from the Admin Security Officer and User roles. The Admin SO and User are unable to work with the log files, and the Auditor is unable to perform administrative tasks on the HSM.

Use the PSESH command **audit audit init** to initialize the Auditor role and set the Auditor PIN. See [audit audit](#) in the "PSESH Commands" section of the *PSESH Command Reference Guide* for command syntax.

Audit Key

Log records are HMACed using an Audit Key, which is later used to verify the logs. The HSM generates the Audit Key from a unique set of parameters entered by the Auditor. If the key is lost or destroyed, these parameters can be re-entered to regenerate the same key. With the same parameters, the key can also be regenerated on another HSM. This allows one HSM's logs to be verified by another HSM.

Audit Key generation requires a minimum of three unique parameters, each at least 8 characters long. For additional security, a key can be generated using input from multiple people, so that one person alone can never regenerate the key.

The Audit Key is stored in the Administrative token, and has the following fixed attributes:

- > Always sensitive
- > Encryption, signing, wrapping, unwrapping are disabled
- > Available only to the Auditor role in the HSM (CKA_AUDIT_KEY)

Use the PSESH command **audit audit secret** to generate the Audit Key. See [audit audit](#) in the "PSESH Commands" section of the *PSESH Command Reference Guide* for command syntax.

Log Verification

The Auditor must export the logs to a client machine using **scp/pscp**, and then use the **auditverify** utility to verify and view the extracted logs. The **auditverify** utility requires the Auditor to sign in using the Auditor PIN. See ["Verify the Logs" on page 71](#) for the complete procedure.

See ["Audit Log Events and Structure" on page 73](#) for a guide on reading the audit logs.

Log Capacity and Rotation

When the HSM logs an event, the log is stored on the HSM, which has a limited capacity. The Auditor must set a schedule for log rotation (hourly, daily, or weekly). Logs will be then periodically packaged and stored on the appliance, which has a much greater storage capacity.

CAUTION! The default log rotation setting is "never". Failing to set a log rotation schedule may allow the HSM storage to fill up, interfering with cryptographic processes.

Short-term log storage within the HSM is important only in the rare situations where the HSM remains functioning but cannot reach the appliance file system.

Configuring and Using Audit Logging

This section describes how to enable audit logging, configure the log rotation, and how to copy and verify the audit logs. It contains the following sections:

- > ["Initialize the Audit User and Create the Audit Key" below](#)
- > ["Enable Audit Logging" on the next page](#)
- > ["Configure Audit Logging" on the next page](#)
- > ["Verify the Logs" on page 71](#)
- > ["Disable Audit Logging" on page 72](#)

Initialize the Audit User and Create the Audit Key

The Admin SO and the Auditor must both be present to initialize the Audit role and create the Auditor PIN. This procedure assumes that you have already initialized the Admin token on the ProtectServer HSM.

To initialize the Audit user and create the Audit Key

1. Using an SSH connection (or a local serial connection), login to PSESH on the ProtectServer appliance as **audit** (not as **admin**), using the initial password "password". The first time you login as **audit**, you will be prompted to create a new, more secure password.
2. Initialize the Auditor role with the following command. The Admin SO must enter the SO PIN before the Auditor can set the new Auditor PIN.

```
psesh:> audit audit init
```

```
psesh:>audit audit init
```

```
Please Enter the SO PIN:
```

```
Please Enter the new Auditor's PIN:
```

Please re-enter the new Auditor's PIN:

Command Result : 0 (Success)

- The Auditor can now generate the Audit Key. You will be prompted for the Auditor PIN, and to enter a minimum of 3 unique parameters, each at least 8 bytes in length (see ["Audit Key" on page 68](#) for more information).

```
psesh:> audit audit secret
```

```
psesh:>audit audit secret
```

```
Please Enter the Auditor's PIN:
Please enter number of params (minimum 3): 3
Please enter parameter #0:12345678
Please enter parameter #1:87654321
Please enter parameter #2:18273645
Audit Key created successfully
```

Command Result : 0 (Success)

Enable Audit Logging

The Admin SO must enable audit logging on the HSM.

To enable Audit logging

- On a client machine, set the *Enable PCI Audit Logs* flag using **ctconf**. You will be prompted for the Admin SO PIN:

```
ctconf -fb
```

```
>ctconf -fb
ProtectToolkit C Configuration Utility
Copyright (c) Safenet, Inc.
```

```
Please enter Administrator's pin (Device 0, S/N: 518687):
```

```
Set new security mode:
Security Mode      : PCI Audit Logging Enabled
```

See ["Enable PCI Audit Logs" on page 58](#) for more information.

- You must reset the HSM to load the new Audit Key:

```
hsmreset
```

CAUTION! Whenever the Audit Key is regenerated, you must reset the HSM in order to load the new key. If you do not load the new key, the HSM will still generate logs, but you will be unable to verify them.

Configure Audit Logging

Configure audit logging using the PSESH commands available to the **audit** user. See [audit](#) in the "PSESH Commands" section of the *PSESH Command Reference Guide* for full syntax. The following procedure must be performed by the Auditor.

To configure Audit logging

1. Using an SSH connection (or a local serial connection), login to PSESH on the ProtectServer appliance as **audit**.

2. Enable the audittrace service:

```
psesh:> audit service enable
psesh:>audit service enable

Audit Log is enabled
Audit Log is started

Command Result : 0 (Success)
```

3. Configure the rotation schedule. By default, logs do not rotate. You can choose an hourly, daily, or weekly rotation schedule.

```
psesh:> audit log rotation {-hourly | -daily | -weekly}
psesh:>audit log rotation -daily

Setting Daily rotation.

Command Result : 0 (Success)
```

Verify the Logs

The Auditor must package the logs and transfer them to a client machine in order to verify them.

To verify the logs

1. Using an SSH connection (or a local serial connection), login to PSESH on the ProtectServer appliance as **audit**.

2. Package the logs for export:

```
psesh:> syslog tarlogs

Generating package list...
Generating tarlogs...
The tar file containing logs is now available via scp as filename 'pselogs.tgz'.

Command Result : 0 (Success)
```

3. Use **scp/pscp** to transfer the package from the appliance. On a client machine, enter one of the following commands:

- Windows: **pscp audit@<appliance_IP>:pselogs.tgz <filename>**
- Linux: **scp audit@<appliance_IP>:pselogs.tgz <filename>**

...where <filename> is the new package filename. Use "." to keep **pselogs.tgz**, but ensure that there is no other file with that name in the destination directory; it will be overwritten.

4. Extract the log files into a directory.
5. Use the auditverify tool to verify the file **applog** in the extracted directory:

```
auditverify -l applog
```

```

Please Enter the Auditor's PIN:
Starting to verify
2017-07-12 14:12:29,success,0,Audit Log initial message
,000000000000000000000000000000000000000000000000000000000000000000000000,692f41f2ec2bbb42411c7b2c5e
3230b39dab28bd5178ef1b3e71b34331500765
2017-07-12 14:53:44,success,0,CS_Initialize:
,692f41f2ec2bbb42411c7b2c5e3230b39dab28bd5178ef1b3e71b34331500765,6afe98063371c25d675616827e
c51d5d23f879312d935c230ebe566db3e064a0
2017-07-12 14:53:44,success,1,CS_OpenSession:
,6afe98063371c25d675616827ec51d5d23f879312d935c230ebe566db3e064a0,868b4457c44c525febad5c87d9
d27ee745829aa38f9ac6bf2405a788f8c3ea89
2017-07-12 14:53:44,success,1,CS_OpenSession:
,868b4457c44c525febad5c87d9d27ee745829aa38f9ac6bf2405a788f8c3ea89,8e65ee17ce0d0b835fd746558d
5c114a45baf6e4e7f579b1f7b22f204db51538
2017-07-12 14:53:44,success,1,CS_FindObjects:
,8e65ee17ce0d0b835fd746558d5c114a45baf6e4e7f579b1f7b22f204db51538,7ff4201694d9b5a68b6f3e205c
75380e10975cddd9fff45641cd82fdb7d7eee17
2017-07-12 14:53:44,success,1,CS_GetAttributeValue:
,7ff4201694d9b5a68b6f3e205c75380e10975cddd9fff45641cd82fdb7d7eee17,c2fd9b7bd90e370a8684259f12
0beda70f3ce2a7aa217e753f02864618066fc8
2017-07-12 14:53:44,success,1,CS_CloseSession:
,c2fd9b7bd90e370a8684259f120beda70f3ce2a7aa217e753f02864618066fc8,a3ef1d28edcf2b1eb4efa2f7d0
75241e2bf1253f85b7dc36895b2ce07cd4732b
...<snip>...
2017-07-11 19:12:40,success,0,CS_Login:
,afc0b246dda667297c4a546c5c7db3b241381ed103589acf920f4c681dbedf14,527710e30d5ff9f13f2922a0a4
ffaaeb7d25724587f92224e27d9e6f7abf4618
2017-07-11 19:12:40,success,0,CS_GenerateKeyPair:
,527710e30d5ff9f13f2922a0a4ffaaeb7d25724587f92224e27d9e6f7abf4618,12ef60bbd62da32a7daf16b276
9a557a342ee0ad02f790386340af942d684ace
2017-07-11 19:12:40,success,0,CS_CloseSession:
,12ef60bbd62da32a7daf16b2769a557a342ee0ad02f790386340af942d684ace,7ba56613669ef06ac298014ac8
b51bcff09fe00a0561a16de53ff7ba567d91eb
2017-07-11 19:12:40,success,0,CS_Finalize:
,7ba56613669ef06ac298014ac8b51bcff09fe00a0561a16de53ff7ba567d91eb,29bdaa88157935cb3d7962f7cb
af0c8311a1da7440e34b1a8aee9fcdda6bd360
File is verified successfully

```

Disable Audit Logging

The Admin SO or the Auditor can stop audit logging. Audit logging will also be stopped by any event that resets the security flags on the HSM, such as a tamper event or factory reset.

To stop audit logging as Admin SO

1. Login to a client machine.
2. Use **ctconf** to remove all security flags, including the *Enable PCI Audit Logs* flag. Enter the Administrator's PIN when prompted:

ctconf -f0

```
Please enter Administrator's pin (Device 0, S/N: 518687):
```

```
Set new security mode:
```

```
Security Mode      : Default (No flags set)
```

To stop audit logging as Auditor

1. Using an SSH connection (or a local serial connection), login to PSESH on the ProtectServer appliance as **audit**.

2. Disable the audittrace service:

audit service disable

```
psesh:>audit service disable
```

```
Audit Log Service is disabled
```

```
Stopping audittrace:
```

```
[ OK ]
```

```
Audit Log Service is stopped
```

```
Command Result : 0 (Success)
```

NOTE Disabling the audittrace service will only prevent audit logs from being recorded. The HSM will continue to generate logs as long as the *Enable PCI Audit Logs* flag is set, potentially impacting HSM performance. For more information about the *Enable PCI Audit Logs* security flag, see "[Enable PCI Audit Logs](#)" on page 58.

Audit Log Events and Structure

This section provides a summary of the events collected in audit logs, and a brief description of the information included in each log entry.

Logged Events

The events logged fall under three categories, as shown in the following table:


```
,8e65ee17ce0d0b835fd746558d5c114a45baf6e4e7f579b1f7b22f204db51538,7ff4201694d9b5a68b6f3e205c75380e10975cddd9ff45641cd82fdb7d7eee17  
2017-07-12 14:53:44,success,1,CS_GetAttributeValue:  
,7ff4201694d9b5a68b6f3e205c75380e10975cddd9ff45641cd82fdb7d7eee17,c2fd9b7bd90e370a8684259f120beda70f3ce2a7aa217e753f02864618066fc8  
2017-07-12 14:53:44,success,1,CS_CloseSession:  
,c2fd9b7bd90e370a8684259f120beda70f3ce2a7aa217e753f02864618066fc8,a3ef1d28edcf2b1eb4efa2f7d075241e2bf1253f85b7dc36895b2ce07cd4732b
```

Message Chaining

Each entry is signed by the Audit Key. To ensure that the audit log data is not tampered with, each entry includes both its own signature and the signature of the previous entry. Note that the first entry includes a string of zeroes for the previous signature.

SNMP Monitoring

This section describes Simple Network Management Protocol (SNMP v2c) support for remote monitoring certain conditions of ProtectServer Network HSMs. Thales provides the following Management Information Base files (MIBs) with the ProtectToolkit client software:

> **SAFENET-PTK-GLOBAL-MIB.mib**

The global MIB, describing the tree from the Thales Enterprise OID, to the PTK sub-tree.

> **SAFENET-PTK-APPLIANCE-MIB.mib**

Defines SNMP access to information about the ProtectServer appliance.

> **SAFENET-PTK-HSM-MIB.mib**

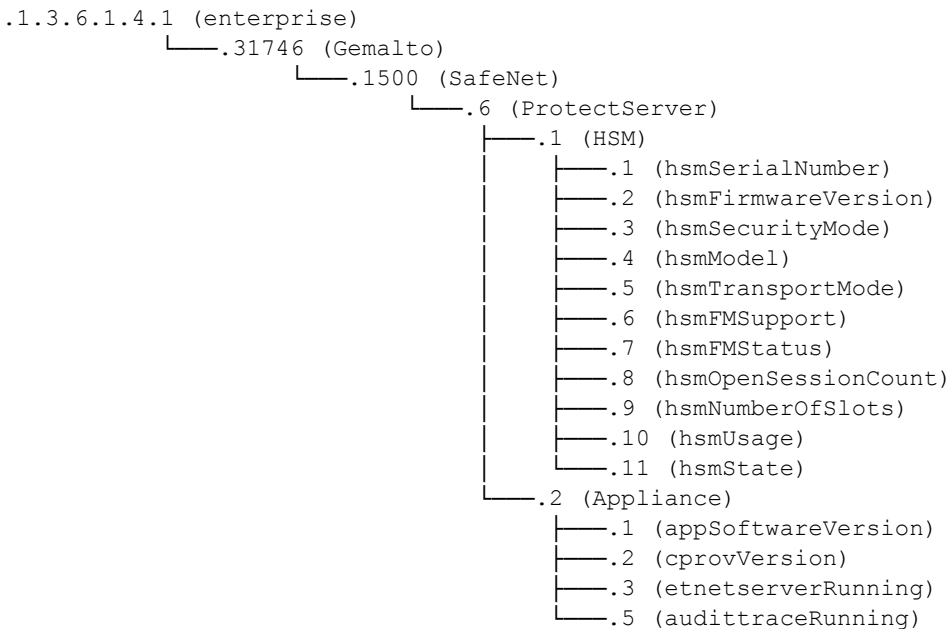
Defines SNMP access to information about the ProtectServer K6 HSM.

These MIBs are included in the client installer package directory **SNMP-MIB**. They must be loaded in your preferred SNMP client.

On Linux, if you are using **snmp-utils**, you can either edit the conguration file **snmpd.conf** in your home directory, or add the MIBs with the command line using **snmpcmd -m <colon-separated_list_of_MIBs>**.

Querying the ProtectServer Network HSM via SNMP

You can query the ProtectServer Network HSM for information by specifying the following Object Identifiers (OIDs):



For example, querying the OID **.1.3.6.1.4.1.31746.1500.6.1.2** will return the current HSM firmware version:

```
$ snmpget -c community -v2c 172.20.11.186 .1.3.6.1.4.1.31746.1500.6.1.2
```

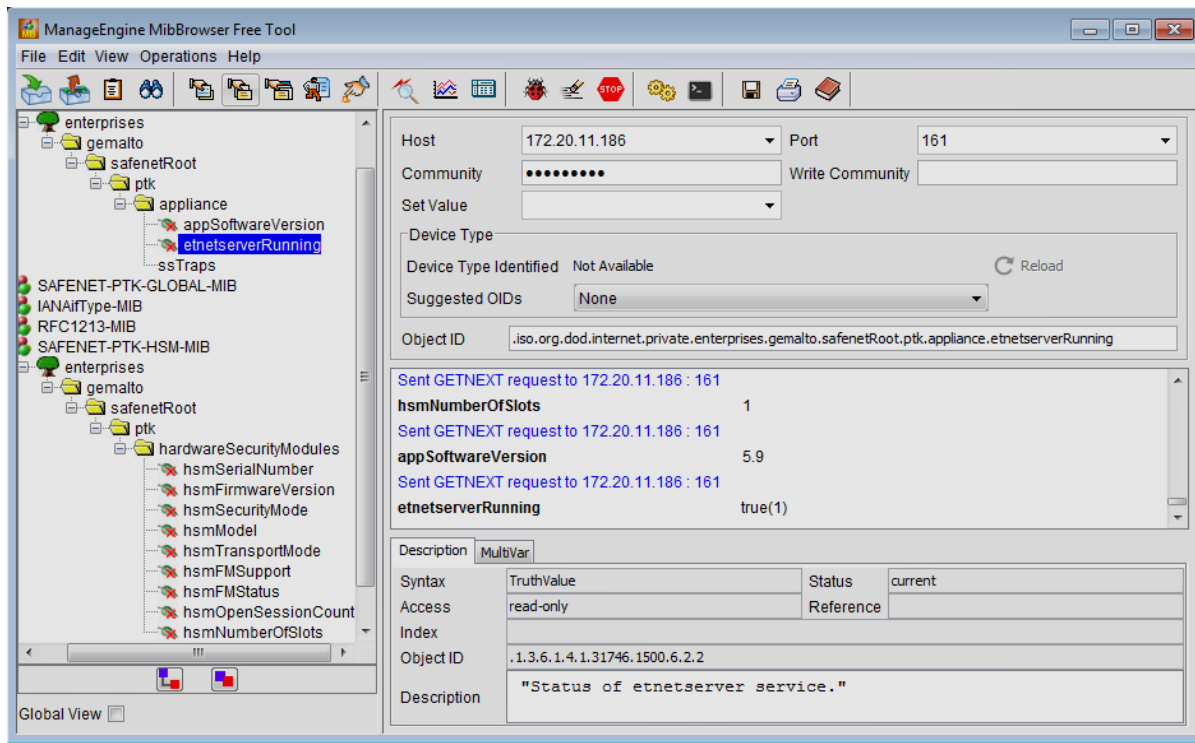
```
.1.3.6.1.4.1.31746.1500.6.1.2 = STRING : 5.06.00
```

The MIBs allow you to simplify queries to use the strings listed above, instead of specifying the entire OID:

```
$ snmpget -c community -v2c 172.20.11.186 etnetserverRunning
```

```
SAFENET -PTK - APPLIANCE - MIB :: etnetserverRunning = INTEGER : true (1)
```

The following example uses a Windows SNMP client:



NOTE SNMP information is placed in an internal cache on the ProtectServer appliance, so information reported by querying these OIDs could be up to 60 seconds old.

HSM Information

The following table describes the HSM information that is retrievable via SNMP.

| Name | OID | Description |
|--------------------|-------------------------------|---|
| hsmSerialNumber | .1.3.6.1.4.1.31746.1500.6.1.1 | Serial number of the HSM adapter. |
| hsmFirmwareVersion | .1.3.6.1.4.1.31746.1500.6.1.2 | Current HSM firmware version. |
| hsmSecurityMode | .1.3.6.1.4.1.31746.1500.6.1.3 | Security flags currently set on the HSM (see "Security Flags" on page 56). |
| hsmModel | .1.3.6.1.4.1.31746.1500.6.1.4 | Model identifier for the HSM. |

| Name | OID | Description |
|---------------------|--------------------------------|--|
| hsmTransportMode | .1.3.6.1.4.1.31746.1500.6.1.5 | Transport mode currently set on the HSM (see "Using Transport Mode to Avoid a Board Removal Tamper" on page 92). |
| hsmFMSupport | .1.3.6.1.4.1.31746.1500.6.1.6 | Indicates whether FMs are supported on the HSM. |
| hsmFMStatus | .1.3.6.1.4.1.31746.1500.6.1.7 | Current status of FM(s) loaded on the HSM. |
| hsmOpenSessionCount | .1.3.6.1.4.1.31746.1500.6.1.8 | Current number of open sessions on the HSM. |
| hsmNumberOfSlots | .1.3.6.1.4.1.31746.1500.6.1.9 | Current number of slots/tokens on the HSM. |
| hsmUsage | .1.3.6.1.4.1.31746.1500.6.1.10 | Current percentage of HSM CPU capacity in use (see hsmstate in the "ProtectToolkit Software Installation" section of the <i>ProtectServer HSM and ProtectToolkit Installation Guide</i>). |
| hsmState | .1.3.6.1.4.1.31746.1500.6.1.11 | Current state of the HSM (see hsmstate in the "ProtectToolkit Software Installation" section of the <i>ProtectServer HSM and ProtectToolkit Installation Guide</i>). |

Appliance Information

The following table describes the HSM appliance information that is retrievable via SNMP.

| Name | OID | Description |
|--------------------|-------------------------------|---|
| appSoftwareVersion | .1.3.6.1.4.1.31746.1500.6.2.1 | Current appliance software version. |
| cprovVersion | .1.3.6.1.4.1.31746.1500.6.2.2 | Current version of the ProtectToolkit-C PKCS#11 Cryptoki provider. |
| etnetserverRunning | .1.3.6.1.4.1.31746.1500.6.2.3 | Indicates whether the etnetserver service is currently running on the appliance. |

| Name | OID | Description |
|-------------------|-------------------------------|--|
| audittraceRunning | .1.3.6.1.4.1.31746.1500.6.2.5 | Indicates whether the audittrace service is currently running on the appliance. |

CHAPTER 6: Operational Tasks

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

Before attempting any of the procedures in this chapter, ensure that 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 101. Many of these functions can also be achieved with the GUI-based tools, which are described in "[Key Management Utility \(KMU\) Reference](#)" on page 173 and "[Administration Utility \(gCTAdmin\) Reference](#)" on page 163.

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 86
- > "[Re-initializing a Token](#)" on page 86
- > "[Multifactor Authentication \(One-Time Password\)](#)" on page 87
- > "[Connecting and Removing Smart Card Readers](#)" on page 91
- > "[Using Transport Mode to Avoid a Board Removal Tamper](#)" on page 92
- > "[Adjusting the HSM Clock](#)" on page 92
- > "[Changing Secure Messaging Mode](#)" on page 93
- > "[Managing Session Key Rollover](#)" on page 93
- > "[Using the System Event Log](#)" on page 93
- > "[Updating the HSM Firmware](#)" on page 93
- > "[Tampering or Decommissioning the HSM](#)" on page 97
- > "[RMA and Shipping Back to Thales](#)" on page 99
- > "[Installing a Functionality Module](#)" on page 94
- > "[Key Entry via PIN Pad](#)" on page 95

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

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 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.

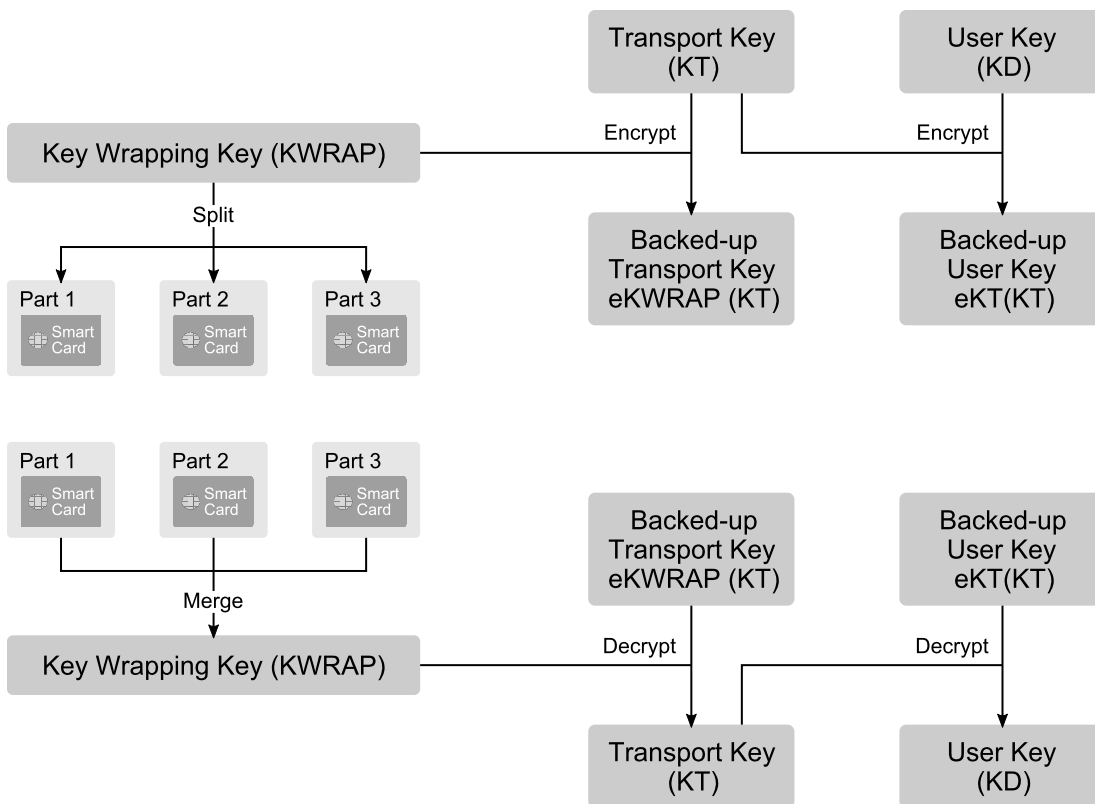
*As defined in A. Shamir - *How to Share a Secret*, Communications of the ACM, Vol 22, no. 11, November 1979, pp 612-613.

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 9: 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, 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 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 **kmu** can be used to create keys and change their attributes. See the "[Command Line Utilities Reference](#)" on page 101 and "[Key Management Utility \(KMU\) Reference](#)" on page 173 respectively. More details about key attributes can be found in "[PKCS #11 Attributes](#)" on page 198.

Key Backup Procedure

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

- > a valid key or keyset 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.
- > If you are exporting keys using the standard *multiple custodians* method, you must set the **Weak Mechanisms** flag to ON (see "[Security Flags](#)" on page 56). This flag is not necessary when using the *N of M* method.

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 101 and "[Key Management Utility \(KMU\) Reference](#)" on page 173 for the complete **ctkmu** and **kmu** references respectively.

The following examples use the **ctkmu** command line utility. See "[Importing and Exporting Keys](#)" on page 186 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 or keyset 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.

Example 3: Keyset Recovery

This example will restore a keyset from a disk file named **keyset.bak** for administration with the Keyset Management Utility. It will be decrypted (unwrapped) with the wrapping key **MyWRAP1**. This operation will prompt for the keyset password and the device Administrator password.

```
ctkmu i -s3 -wMyWRAP1 keyset.bak
```

NOTE When restoring the MACHINE_Keyset or the SYSTEM_Keyset, enter the default value **password** as the user password. The device administrator password used to create the backup will also be prompted for.

Adding and Removing Slots

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

NOTE It is not possible to add slots using **ctconf** while other 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 20](#).

Removing Slots

Before removing slots from 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 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. User PINs are case-sensitive, and must be 4-32 characters in length.

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.

Multifactor Authentication (One-Time Password)

ProtectToolkit supports multifactor authentication using the SafeNet 110 OTP Token. This authentication scheme adds another layer of security by requiring both the memorized token PIN and a 6-digit number randomly generated by the SafeNet 110 OTP Token. When you press the button, a 6-digit number is generated. This number is valid for only 30 seconds (approximately the time that it is displayed on the token's screen). This time limit ensures that any person logging in to the HSM must have the physical device in hand.

User PINs are case-sensitive, and must be 4-32 characters in length. If you are using multifactor authentication, the PIN length becomes 10-38 characters (userpin + OTP).

NOTE This feature is not compatible with High Availability (HA) or Work Load Distribution (WLD) configurations.

You can activate multifactor authentication for:

- > the ["Administration Security Officer \(ASO\)" on page 64](#) and/or ["Administrator" on page 64](#) roles on the Admin slot
- > the ["Security Officer \(SO\)" on page 65](#) and/or ["Token Owner \(User\)" on page 65](#) roles on individual token slots

Each person who holds one or more of these roles requires their own SafeNet 110 OTP Token to use multifactor authentication. The physical tokens allow you to customize your authentication scheme to suit your security needs. Contact your Thales Customer Support representative to purchase SafeNet 110 OTP Tokens.

Figure 10: The SafeNet 110 OTP Token (PN: 955-000237-001)



This section contains instructions for the following tasks:

- > ["Activating Your SafeNet 110 OTP Token" on the next page](#)
- > ["Initializing Multifactor Authentication" on the next page](#)

- > ["Logging In Using Multifactor Authentication" on the next page](#)
- > ["Re-initializing Multifactor Authentication For the User Role" on page 90](#)
- > ["Removing Multifactor Authentication From a Role" on page 91](#)

Activating Your SafeNet 110 OTP Token

When you order SafeNet 110 OTP Tokens, Thales sends you a series of secure emails containing the information you need to activate them. Follow the instructions in the emails to unzip the following encrypted files:

- > **TokenSeed.xml**
- > **PSKCPassword.txt**

Initializing Multifactor Authentication

This procedure allows you to enable multifactor authentication for a role on a ProtectServer HSM token slot.

NOTE If you wish to perform ["Token Replication" on page 28](#) between HSMs using multifactor authentication, you must use the same OTP Token to initialize multifactor on both HSMs.

Prerequisites

- > The HSM token must be initialized and a PIN set for the specified role (Administration SO, Administrator, Security Officer or User)
- > SafeNet 110 OTP Token
- > **TokenSeed.xml** and **PSKCPassword.txt** files provided by Thales via secure email. The SafeNet 110 OTP Token serial number must match one listed in the **TokenSeed.xml** file.

NOTE If you are initializing multifactor authentication on a Linux client, run **dos2unix** on each file before continuing.

```
>dos2unix <filename>
```

To initialize multifactor authentication

1. Since the random numbers generated by the SafeNet OTP token are time-sensitive, sync the HSM time with the clock on the client machine (["ctconf" on page 117](#)).

ctconf -t

```
>ctconf -t
ProtectToolkit C Configuration Utility 5.7.0
Copyright (c) Safenet, Inc. 2009-2018
```

```
Please enter Administrator's pin (Device 0, S/N: 518687):
The clock is set to: 12/10/2018 16:18:28 (-5:00+DST)
```

2. Use the **ctotp** utility to initialize multifactor authentication for the desired role. You must specify the slot, the SafeNet 110 OTP Token serial number, and filepaths to the **TokenSeed.xml** and **PSKCPassword.txt** files. Include the **-O** option to specify the Security Officer or Administration Security Officer role. When prompted, enter the role's standard token PIN (["ctotp" on page 151](#)).


```
ctotp init -s<slotnum> -t<serialnum> -x<path_to_TokenSeed.xml> -p<path_to_PSKCPassword.txt> [-O]
```

```
>ctotp init -s0 -tGALT10282872 -xTokenSeed.xml -pPSKCPasswd.txt -O
```

```
Please Enter the Security Officer Token PIN:
```

```
=====
```

```
OTP Initialization Successful.
```

```
=====
```

3. Use the **ctotp** utility to log in to the role. The first login synchronizes the SafeNet 110 OTP Token with the HSM's clock, so that the 30-second window will be accurate for future logins ("**ctotp**" on page 151).

```
ctotp login -s<slotnum> [-O]
```

- a. Press the button on the SafeNet 110 OTP Token. A six-digit one-time password is displayed on the screen.



- b. At the PIN prompt, enter the token PIN for the specified role, together with the one-time password from the SafeNet 110 OTP Token. For example, if your token PIN is **tokenPIN**, you would enter:

```
tokenPIN123456
```

```
>ctotp login -s0 -O
```

```
Please Enter the Security Officer Token PIN:
```

```
=====
```

```
OTP Login Successful.
```

```
=====
```

NOTE Once multifactor authentication is initialized, the `pin` and `pinLen` parameters passed to `C_Login()` must contain the token PIN and the current 6-digit one-time password. See "[Logging In Using Multifactor Authentication](#)" below.

Logging In Using Multifactor Authentication

This procedure describes how to log in to a ProtectServer HSM slot using multifactor authentication.

Prerequisites

- > Multifactor authentication must be initialized for the role
- > Ensure that you have your token PIN and the correct SafeNet 110 OTP Token ready

To log in using multifactor authentication

1. Use **CTbrowse** or one of the PTK command-line utilities to initiate login or perform an action that requires login. You will be prompted for the PIN associated with the role.
2. Press the button on the SafeNet 110 OTP Token. A six-digit one-time password is displayed on the screen.

123456

3. At the PIN prompt, enter the token PIN for the specified role, together with the one-time password from the SafeNet 110 OTP Token. For example, if your token PIN is **userPIN**, you would enter:

userPIN123456

The password generated by the SafeNet 110 OTP Token changes every 30 seconds, so you must complete the login procedure within this time.

Re-initializing Multifactor Authentication For the User Role

The Security Officer can re-initialize multifactor authentication for the User if required. This capability is useful if a SafeNet 110 OTP Token associated with the User role is lost or damaged, and the User needs to initialize another one. There is no mechanism to re-initialize multifactor authentication for the Security Officer role.

This procedure is performed by the Security Officer with input from the User.

Prerequisites

- > The Security Officer must be present and prepared to log in with their token PIN (and SafeNet 110 OTP Token, if applicable)
- > The User must be present and prepared to enter their standard token PIN
- > A new or unused SafeNet 110 OTP Token
- > **TokenSeed.xml** and **PSKCPassord.txt** files provided by Thales via secure email. The SafeNet 110 OTP Token serial number must match one listed in the **TokenSeed.xml** file.

To re-initialize multifactor authentication for the User role

1. Use the **ctotp** utility to re-initialize multifactor authentication for the User. You must specify the slot, the new SafeNet 110 OTP Token's serial number, and filepaths to the **TokenSeed.xml** and **PSKCPassord.txt** files ("**ctotp**" on page 151).

```
ctotp reinit -s<slotnum> -t<serialnum> -x<path_to_TokenSeed.xml> -p<path_to_PSCKPassord.txt>
```

- a. You are prompted for the Security Officer PIN. If you have multifactor authentication enabled for the SO role, enter the standard SO PIN followed by the 6-digit one-time password from the SO's SafeNet 110 OTP Token.

- b. You are prompted for the token User PIN. Only the standard PIN is required.

```
>ctotp reinit -s0 -tGALT10282854 -xTokenSeed.xml -pPSKCPassord.txt
```

Please Enter the Security Officer Token PIN:

Please Enter the Token PIN:

```

=====
OTP Re-Initialization Successful.
=====

```

Removing Multifactor Authentication From a Role

If you no longer wish to use multifactor authentication, you can use this procedure to remove the requirement from your own role.

Prerequisites

- > Standard PIN for the role
- > SafeNet 110 OTP Token associated with the role

To remove multifactor authentication from a role

1. Use the **ctotp** utility to remove the multifactor authentication requirement from the desired role by specifying the slot for that role. If you are removing the multifactor requirement for the Security Officer or Administration Security Officer, include the **-O** option ("[ctotp](#)" on page 151).

ctotp del -s<slotnum> [-O]

You are prompted for the token PIN. Enter the token PIN for the specified role, together with the one-time password from the SafeNet 110 OTP Token.

```
>ctotp del -s0 -O
```

```
Please Enter the Security Officer Token PIN:
```

```

=====
OTP Deletion Successful.
=====

```

In the future, logging in with this role requires only the standard token 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 adapter 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. When applied to ProtectServer External 2 or ProtectServer External 2 Plus, Transport Mode prevents a tamper event from occurring when the tamper key is turned to the tamper position, or the tamper button is pressed.

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:

| Value | Mode Name | Mode Description |
|-------|-----------------------------|--|
| 0 | No Transport Mode (Default) | Default mode that is applied when HSM is installed and configured. This mode will tamper the HSM if it is removed from the PCI bus, the tamper key is turned to the tamper position, or the tamper button is pressed. |
| 1 | Single Transport Mode | HSM will not be tampered after removal from the PCI bus, or when the tamper key is turned to the tamper position, or the tamper button is pressed. 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 removal from the PCI bus, or when the tamper key is turned to the tamper position, or the tamper button is pressed. |

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.

- From a command prompt, type:

```
ctconf -t
```

Changing Secure Messaging Mode

See ["Secure Messaging" on page 23](#).

Managing Session Key Rollover

See ["Messaging Mode Configuration" on page 24](#).

Using the System Event Log

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.

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 the HSM Firmware

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

Thales.

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 53](#) for more information on security policies.

Firmware upgrades are distributed in the form of a digitally-signed file. Refer to the CRN for a list of supported firmware versions.

Update Prerequisites

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

Updating the Firmware

The HSM firmware is upgraded using the **ctconf** utility (see ["ctconf" on page 117](#)).

To update the ProtectServer HSM firmware

1. Enter the following at a command prompt, where *<filename>* refers to the name of the firmware upgrade file:

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

The user is prompted for the Administrator password.

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

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

Installing a Functionality Module

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 Thales.

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 Thales 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
```

Loading an FM Causes Halt and Reset

When you load an FM, the HSM is automatically halted and reset. The halt/reset is reported as an error in the event logs and in **/var/log/messages**. This error can be safely ignored.

Key Entry via PIN Pad

ProtectServer HSMs support key component entry via a compatible Verifone PIN pad. You must order the PIN pad directly from Thales; only Thales-distributed PIN pads are configured to work with ProtectServer.

Using a PIN pad for Key Entry

The ProtectServer HSM administrator can use these directions to enter key components via a compatible PIN pad. You require:

- > compatible PIN pad with USB connector
- > physical access to the ProtectServer HSM
- > a client or host machine with **ctkmu** installed
- > key components ready for entry in 3-digit decimal format (see ["Hexadecimal to Decimal Conversion Table" on page 97](#))

To use a PIN pad for key entry

1. Connect the PIN pad to the USB port on the HSM card. It must be connected directly to the HSM and not one of the other USB ports on the appliance/host.



Micro-D subminiature (MSDM) connector USB Port

The PIN pad powers up and performs its startup processes.

2. On the client machine, use **ctkmu** to initiate the key entry procedure. You must include the **-p** option to use the PIN pad. See ["ctkmu" on page 129](#) for full command syntax.

```
>ctkmu c -s<slot> -t<key_type> -a<attributes> -n<name> -k<number_of_components> -p
```

3. The PIN pad prompts the user to enter the first byte of the first key component. Key components must be entered on the PIN pad in decimal. Refer to ["Hexadecimal to Decimal Conversion Table" on page 97](#). Depending on the PIN pad model you received from Thales, the PIN pad responds in one of the following ways:

- The byte expected by the PIN pad is displayed. When you see this message, you have 20 seconds to enter the 3-digit byte before the operation times out.

```
ENTER 1 / 8
```

```
* * *
```

- The byte expected by the PIN pad is displayed for 2 seconds, followed by the ENTER prompt. When you see this prompt, you have 20 seconds to enter the 3-digit byte before the operation times out.

```
Component 1
  1/8
```

```
ENTER
```

```
* * *
```

Continue following the prompts on the PIN pad until all bytes have been entered.

- When the entire component has been entered, the PIN pad displays the key component value (KCV) and prompts you to confirm it is correct by pressing the appropriate button.

```
KVC:      | Continue
BAC10C    | Cancel
```

- The PIN pad prompts the user to enter the first byte of the second key component. Continue following the prompts until all key components are entered.
- When all key components have been entered, **ctkm** displays the KCV for the complete key and prompts you to confirm it.

```
Key 'des_1' KCV : 8CA64D
Is this correct? [Y/n]: y
```

```
Key "des_1" was created
```


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 |
| 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 |

Tampering or Decommissioning the HSM

A tamper event formats the secure memory of the HSM, erasing all cryptographic material, configuration, and user data. This is triggered automatically when someone attempts to tamper with the HSM in any of the following ways:

- > Removing a ProtectServer PCIe 2 from its PCIe bus.
- > Opening the ProtectServer PCIe 2 host chassis, if a chassis intrusion switch is connected (for more information about connecting a chassis intrusion switch, see [The Tamper-Input Header](#) in the "ProtectServer PCIe2 Hardware Installation" section of the *ProtectServer and ProtectToolkit Installation Guide*).

- > Opening the ProtectServer External 2 or ProtectServer External 2 Plus appliance chassis.

This function protects your important keys in the case of physical attack on the HSM. It is also an important part of any decommissioning procedure, when the HSM has reached the end of its lifecycle, or after a security-sensitive event which requires all stored data to be immediately destroyed.

NOTE FMs that have been loaded onto the HSM are not deleted from the HSM after a tamper event. To delete FMs from the HSM, use the **ctfm** utility before tampering the HSM. For more information about deleting FMs using the **ctfm** utility, see ["ctfm" on page 122](#).

CAUTION! If FMs are present on the HSM that modify login behaviour, the user will be permanently locked out of the HSM after a tamper event. To avoid an RMA, you must delete these FMs by using the **ctfm** utility before tampering the HSM. For more information about deleting FMs using the **ctfm** utility, see ["ctfm" on page 122](#).

To deliberately tamper the HSM, you can use a hardware or software procedure depending on your reasons for tampering and your access to the physical HSM.

Hardware Tamper Procedures

The hardware tamper procedure is different for each variant of the ProtectServer 2 HSM hardware.

ProtectServer PCIe 2 HSM

There are two methods of performing a hardware tamper of the ProtectServer PCIe 2:

- > Remove the adapter from the PCIe bus.
- > Open the host chassis if a chassis intrusion switch is connected. For more information about connecting a chassis intrusion switch, see [The Tamper-Input Header](#) in the "ProtectServer PCIe2 Hardware Installation" section of the *ProtectServer and ProtectToolkit Installation Guide*.

If you wish to remove the ProtectServer PCIe 2 from the host PCIe bus without triggering a tamper event, see ["Using Transport Mode to Avoid a Board Removal Tamper" on page 92](#).

ProtectServer External 2 HSM

The ProtectServer External 2 appliance has a keyed tamper lock on the rear panel (see [ProtectServer External 2 rear panel](#) in the "ProtectServer External 2 Installation and Configuration" section of the *ProtectServer HSM and ProtectToolkit Installation Guide*).

To hardware tamper the ProtectServer External 2 HSM

1. Insert the tamper key into the tamper lock and turn it to the vertical (Tamper) position.
All tokens, key material, and user configuration on the HSM are destroyed.
2. If you wish to re-initialize the HSM for continued use, turn the tamper key back to the horizontal (Active) position.
3. If you are decommissioning the appliance, log in to PSESH as **admin** and perform a factory reset of the appliance configuration:

```
psesh:>sysconf appliance factory
```

ProtectServer External 2 Plus HSM

The ProtectServer External 2 Plus has a tamper button on the rear panel (see [Rear panel view](#) in the "ProtectServer External 2 Plus Product Overview" section of the *ProtectServer HSM and ProtectToolkit Installation Guide*).

To hardware tamper the ProtectServer External 2 Plus HSM

1. Press the tamper button on the back of the ProtectServer External 2 Plus appliance. Pressing the Tamper button flags the HSM to be placed in a tamper state. At this point, all keys and tokens still exist on the HSM and running applications will work normally.
2. Log in to PSESH as **admin** or **pseoperator** and reboot the appliance.

```
psesh:> sysconf appliance reboot
```

After the reboot, the HSM is tampered and erased.

3. If you are decommissioning the appliance, log in to PSESH as **admin** and perform a factory reset of the appliance configuration:

```
psesh:>sysconf appliance factory
```

Software Tamper Procedure

You can also tamper the HSM using the **ctconf** utility. However, the following constraints apply:

- > Only the administrator may tamper the HSM, due to the highly destructive nature of this action.
- > All sessions must be closed before performing a software tamper and no user should be accessing the HSM during the software tamper procedure.

The tamper procedure is the same regardless of the HSM variant. If you are performing a tamper as part of decommissioning a ProtectServer External 2 appliance, you must also factory reset the appliance configuration.

To tamper the HSM using software

1. Use the **ctconf** utility to trigger the tamper event:

```
ctconf -x
```

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

2. If you are decommissioning a ProtectServer External 2 or ProtectServer External 2 Plus appliance, log in to PSESH as **admin** and perform a factory reset of the appliance configuration:

```
psesh:>sysconf appliance factory
```

RMA and Shipping Back to Thales

Although rare, you might need to ship a ProtectServer HSM back to Thales.

Contact your Thales representative to obtain the Return Material Authorization (RMA) and instructions for packing and shipping. Refer to ["Support Contacts" on page 11](#).

The RMA process is completely secure and Thales cannot access any sensitive key material contained in the HSM. To take maximum precaution with the contents of your HSM before it leaves your possession, back up all key materials and then decommission the HSM, forcibly clearing all of its contents, before returning it to Thales. For more information about decommissioning the HSM, see ["Tampering or Decommissioning the HSM" on page 97](#).

CHAPTER 7: Command Line Utilities Reference

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

- > "ctcert" on the next page
- > "ctcheck" on page 113
- > "ctconf" on page 117
- > "ctfm" on page 122
- > "ctident" on page 125
- > "ctlimits" on page 139
- > "ctmultitoken" on page 142
- > "ctotp" on page 151
- > "ctkmu" on page 129
- > "ctperf" on page 154
- > "ctstat" on page 160
- > "auditverify" on page 161

ctcert

Certificate Management Utility for the ProtectToolkit-C environment.

The **ctcert** utility provides basic support for the creation of X.509v3 certificates using 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 37](#) and ["Operation in HA Mode" on page 37](#) in the "Cryptoki Configuration" section of the *ProtectToolkit-C Administration Guide* for more information about these operating modes.

Syntax

Generate certificate with existing keys and self-sign

```
ctcert c -l<label> [-s<slot>] [-b<YYYYMMDDhhmmss[Z]>] [-d<duration><h|d|m|y>] [-e<YYYYMMDDhhmmss[Z]>] [-x<name>] [-P] [-S<mechanism>]
```

Generate certificate with new keys and self-sign

```
ctcert c -l<label> [-s<slot>] [-b<YYYYMMDDhhmmss[Z]>] [-d<duration><h|d|m|y>] [-e<YYYYMMDDhhmmss[Z]>] [-x<name>] [-P] [-S<mechanism>] [-t<type>] [-C<curve_name>] [-z<bits>]
```

Generate certificate with existing keys and sign with existing key

```
ctcert c -l<label> -c<label> -k [-s<slot>] [-b<YYYYMMDDhhmmss[Z]>] [-d<duration><h|d|m|y>] [-e<YYYYMMDDhhmmss[Z]>] [-x<name>] [-P] [-S<mechanism>] [-i<slot>]
```

Generate certificate with new keys and sign with existing key

```
ctcert c -l<label> -c<label> -k [-s<slot>] [-b<YYYYMMDDhhmmss[Z]>] [-d<duration><h|d|m|y>] [-e<YYYYMMDDhhmmss[Z]>] [-x<name>] [-P] [-S<mechanism>] [-i<slot>] [-t<type>] [-C<curve_name>] [-z<bits>]
```

Import certificate

```
ctcert i -f<file> -l<label> [-s<slot>]
```

List certificate

```
ctcert l [-s<slot>]
```

Generate certificate request with new keys

```
ctcert r -I<label> -k [-s<slot>] [-S<mechanism>] [-t<type>] [-C<curve_name>] [-z<bits>]
```

Generate certificate request with existing keys

```
ctcert r -I<label> [-s<slot>] [-S<mechanism>]
```

Set trusted certificate

```
ctcert t -I<label> [-s<slot>]
```

Export certificate or certificate request to file

```
ctcert x -I<label> -f<name> [-s<slot>]
```

Export certificate or certificate request to screen

```
ctcert x -I<label> [-s<slot>]
```

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 -I<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 -I<label> option specifies the label for the newly-imported certificate or certificate request object.</p> |

| Command | Description |
|----------|---|
| 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> |
| 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 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 syntax.

| 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> |

| Option | Description |
|--|---|
| <p>-C<curve_name></p> | <p>--curve-name=<label> Specifies which curve to use. Valid values:</p> <ul style="list-style-type: none"> > brainpoolP160r1 > brainpoolP160t1 > brainpoolP192r1 > brainpoolP192t1 > brainpoolP224r1 > brainpoolP224t1 > brainpoolP256r1 > brainpoolP256t1 > brainpoolP320r1 > brainpoolP320t1 > brainpoolP384r1 > brainpoolP384t1 > brainpoolP512r1 > brainpoolP512t1 > c2tnb191v1 > c2tnb191v1e > curve25519 > ed25519 > P-192 (also known as prime192v1 and secp192r1) > P-224 (also known as secp224r1) > P-224K1 (also known as secp224k1) > P-256 (also known as prime256v1 and secp256r1) > P-384 (also known as secp384r1) > P-521 (also known as secp521r1) > secp256k1 > or any valid ECC Domain Parameter object label <p>If -tec is specified, the -C parameter must be included in the command, or ctcert will exit with an error message. All hashing mechanisms are supported.</p> |
| <p>-d<integer[h d m y]></p> | <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> |

| Option | Description |
|-------------------------------|---|
| -e <YYYYMMDDhhmmss[Z]> | <p>--cert-end=<YYYYMMDDhhmmss[Z]></p> <p>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.</p> <p>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></p> <p>Specifies a text file containing a PEM encoding of a certificate or certificate request object.</p> <p>This option is only valid with the Import Command (i), Export Command (x) or Generate Certificate Request Command (r).</p> |
| -h, -? | <p>--help</p> <p>Display usage information.</p> |
| -i <slot> | <p>--ca-slot=<slot></p> <p>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.</p> <p>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.</p> <p>This option is only valid with the -c option.</p> |
| -k | <p>--key-gen</p> <p>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> <div data-bbox="571 1283 1433 1373" style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>NOTE -k is a mandatory parameter when generating a certificate with new keys.</p> </div> |
| -l <label> | <p>--label=<label></p> <p>This option specifies a label for a new or existing certificate request or public key object. Refer to the description of each command for relevant details.</p> |
| -P | <p>--P</p> <p>This option will sign the newly created certificate using an RSA-PSS mechanism. This option is only valid if the key used to sign the certificate is RSA.</p> |

| Option | Description | | | | | | | | | | | | | | | | | | | | |
|-------------------------|--|-----------|-----|-------------------------|-------------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-------------------|--|-------------------|--|-------------------|--|-------------------|--|
| -s <slot> | <p>--slot=<slot></p> <p>Specifies the slot:</p> <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> | <p>--sig-hash-alg=<sign_alg></p> <p>Specifies the RSA or DSA hashing algorithm for certificate request or certificate generation. Valid mechanisms are:</p> <table border="1" data-bbox="533 701 1473 1178"> <thead> <tr> <th data-bbox="533 701 1002 772">RSA/ECDSA</th> <th data-bbox="1002 701 1473 772">DSA</th> </tr> </thead> <tbody> <tr> <td data-bbox="533 772 1002 825">> SHA1 (default)</td> <td data-bbox="1002 772 1473 825">> SHA1 (default)</td> </tr> <tr> <td data-bbox="533 825 1002 877">> SHA224</td> <td data-bbox="1002 825 1473 877">> SHA224</td> </tr> <tr> <td data-bbox="533 877 1002 930">> SHA256</td> <td data-bbox="1002 877 1473 930">> SHA256</td> </tr> <tr> <td data-bbox="533 930 1002 982">> SHA384</td> <td data-bbox="1002 930 1473 982">> SHA384</td> </tr> <tr> <td data-bbox="533 982 1002 1035">> SHA512</td> <td data-bbox="1002 982 1473 1035">> SHA512</td> </tr> <tr> <td data-bbox="533 1035 1002 1087">> SHA3-224</td> <td data-bbox="1002 1035 1473 1087"></td> </tr> <tr> <td data-bbox="533 1087 1002 1140">> SHA3-256</td> <td data-bbox="1002 1087 1473 1140"></td> </tr> <tr> <td data-bbox="533 1140 1002 1192">> SHA3-384</td> <td data-bbox="1002 1140 1473 1192"></td> </tr> <tr> <td data-bbox="533 1192 1002 1245">> SHA3-512</td> <td data-bbox="1002 1192 1473 1245"></td> </tr> </tbody> </table> <p>NOTE ECDSA is used for a certificate and EC is used for a key pair.</p> | RSA/ECDSA | DSA | > SHA1 (default) | > SHA1 (default) | > SHA224 | > SHA224 | > SHA256 | > SHA256 | > SHA384 | > SHA384 | > SHA512 | > SHA512 | > SHA3-224 | | > SHA3-256 | | > SHA3-384 | | > SHA3-512 | |
| RSA/ECDSA | DSA | | | | | | | | | | | | | | | | | | | | |
| > SHA1 (default) | > SHA1 (default) | | | | | | | | | | | | | | | | | | | | |
| > SHA224 | > SHA224 | | | | | | | | | | | | | | | | | | | | |
| > SHA256 | > SHA256 | | | | | | | | | | | | | | | | | | | | |
| > SHA384 | > SHA384 | | | | | | | | | | | | | | | | | | | | |
| > SHA512 | > SHA512 | | | | | | | | | | | | | | | | | | | | |
| > SHA3-224 | | | | | | | | | | | | | | | | | | | | | |
| > SHA3-256 | | | | | | | | | | | | | | | | | | | | | |
| > SHA3-384 | | | | | | | | | | | | | | | | | | | | | |
| > SHA3-512 | | | | | | | | | | | | | | | | | | | | | |
| -t <type> | <p>--type=<type></p> <p>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.</p> <p>If -tec is specified, the -C parameter must be included in the command, or ctcert will exit with an error message. All hashing mechanisms are supported.</p> | | | | | | | | | | | | | | | | | | | | |
| -x <name> | <p>--attribute-file=<name></p> <p>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" on the next page.</p> <p>This option is only valid with the Generate Certificate command (c).</p> | | | | | | | | | | | | | | | | | | | | |
| -z <bits> | <p>--size=<bits></p> <p>Use with the -k option to specify the new key size in bits. The default key size is 1024 bits.</p> | | | | | | | | | | | | | | | | | | | | |

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> |

| Tag | Description |
|-------------------------|---|
| 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> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>NOTE The maximum value of a certificate serial number is 99999999.</p> </div> |
| 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> , C= <string> }</pre> <p>The meaning of each name component in each name/value pair is as follows:</p> <ul style="list-style-type: none"> CN - Common Name OU - Organizational Unit Name O - Organization Name C - Country Name <p>The following example illustrates a well-formed issuer-distinguished name:</p> <pre>issuer { CN=any string , OU=Testing , O=SafeNet , C=CA }</pre> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <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> </div> <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> |

| Tag | Description |
|----------------------------|--|
| 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 ,] [unotice="<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 unotice. The cps qualifier string is the URI of the Certification Practice Statement that relates to this policy, and the unotice qualifier is a string that is included in the certificate as a user notice that relates to the certificate policy. Both the cps and unotice 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> <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> |

| Tag | Description |
|-----------------|---|
| 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 -lecdsaCert1 -k</pre> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>NOTE Use ECDSA for a certificate and EC for a key pair.</p> </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 | Use existing keys with label “Test” and sign with a key with that has the label “CA Key”. <code>ctcert c -c"CA Key" -lTest</code> |
| Example 6 | Use existing certificate request with a label “Test Cert” and sign with a key that has the label “CA Key”. <code>ctcert c -c"CA Key" -l"Test Cert"</code> |
| Example 7 | To create a new certificate request with the label “User” and generate new keys (RSA is default) <code>ctcert r -k -lUser</code> |
| Example 8 | To export a previously generated certificate request as a PEM object and store this in the file name mycert.txt . <code>ctcert x -lUser -fmycert.txt</code> |
| Example 9 | To use RSA-PSS to generate an RSA certificate with new keys using the SHA512 algorithm. <code>ctcert c -k -s0 -ltest1234 -t rsa -z2048 -P -S sha512</code> |

ctcheck

SafeNet Cryptoki provider status enquiry utility.

ctcheck lists the status of 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 37](#) and ["Operation in HA Mode" on page 37](#) in the "Cryptoki Configuration" section of the *ProtectToolkit-C Administration Guide* for more information about these operating modes.

Syntax

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

| Option | Description |
|-----------|--|
| -a | --all Print all device information (overrides -b options) |

| Option | Description |
|--------------------|--|
| -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 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> |

| Option | Description |
|--------------------|--|
| -f <x s> | --format= x s 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 ~) |
| -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 ProtectToolkit-C environment.

The **ctconf** utility is used to configure the operating parameters for 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 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 37](#) and ["Operation in HA Mode" on page 37](#) in the "Cryptoki Configuration" section of the *ProtectToolkit-C Administration Guide* for more information about these operating modes.

Syntax

```
ctconf [-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>]
```

| Option | Description |
|------------|---|
| -a<device> | --device-number=<device> Use the admin token on the specified device |
| -b<name> | --fm-cert=<name> Specifies the certificate used to validate an FM specified with -k<FM_file> . |
| -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 |

| Option | Description |
|-------------|---|
| -f<flags> | <p>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.</p> <p>When flags are set, any flags set previously are cleared.</p> <p>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" section "Default Mode" on page 54.</p> <p>Use other flags values to set flags as follows:</p> <ul style="list-style-type: none"> a FIPS Algorithms Only b Enable PCI Audit Logs c No Public Crypto d DES Keys Even Parity Allowed e Entrust Ready F FIPS Mode (equivalent to -facIntu) i Increased Security Level 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 EC Parameters allowed w Weak PKCS#11 Mechanisms <p>Each of these flags is fully described in "Security Flag Descriptions" on page 58.</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<FM_file> | <p>--validate-fm=<file> Validate FM module file. You must also specify the certificate used to validate the FM (-b<name>).</p> |

| Option | Description |
|------------------|---|
| -I <fmID> | --delete-fm --disable-fm --fmid =<fmid> Disable/delete an FM module, specifying the FM ID in hex format. |
| -m <n> | --mode =<n> Set the transport mode for the HSM. The following transport modes can be set with <n>: 0 No Transport Mode (Default) - to be applied when HSM is installed and configured. This mode will tamper the HSM if it is removed from the PCI bus, the tamper key is turned to the tamper position, or the tamper button is pressed. 1 Single Transport Mode - HSM will not be tampered after removal from the PCI bus, or when the tamper key is turned to the tamper position, or the tamper button is pressed. 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 removal from the PCI bus, or when the tamper key is turned to the tamper position, or the tamper button is pressed. See "Using Transport Mode to Avoid a Board Removal Tamper" on page 92 . |
| -n <slot> | --init-token =<slot> Initialize the token in the specified slot |
| -p | --purge-log Purge event log. Note that a purge cannot be done until the event log is full. |
| -q | --query Query peripheral devices. Check all available serial ports, and attempt to activate drivers for the connected devices. |
| -r <slot> | --reset-token =<slot> Reset existing token in specified slot |
| -s | --fm-info Display FM module information |

| Option | Description |
|--|--|
| -t | <p>--time-set</p> <p>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.</p> <p>For more information about RTC status values, see HSMADM_SetRtcStatus in the "hsmadmin.h Library Reference" section of the <i>ProtectToolkit-C Programming Guide</i>.</p> |
| -v | <p>--verbose</p> <p>Display extended status information</p> |
| -x | <p>--tamper</p> <p>This will cause the Key Store memory on the HSM to be erased (as if tampered) and made ready for re-initialization.</p> <p>The -x option is only available on hardware-based ProtectToolkit-C implementations.</p> |
| <p>--rtc-adj-access-control-rule= <secs>:<count>:<days></p> | <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> <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> |

| Option | Description |
|-------------------------------------|---|
| --rtc-adj-access-control=0 1 | <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 (see HSMAdmin.H Library Reference in the <i>ProtectToolkit-C Programming Guide</i>) are governed by the RTC Adjustment Access Control Rule. 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 ProtectToolkit-C environment.

The **ctfm** utility is restricted to use by the ProtectToolkit-C administrator, to manage functionality modules on ProtectServer devices (HSMs).

With this tool it is possible to:

- > Load a new FM (if a patched 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.

Starting with ProtectToolkit version 5.4, you can upload multiple custom FMs to an HSM and use them simultaneously. Only one PKCS#11 patched FM can be loaded and used at a time. If a patched FM is already loaded, it is overwritten by the new FM. Refer to [Custom Functions](#) and [PKCS#11 Patched Functions](#) in the *FM SDK Programming Guide* for descriptions of these FM types.

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 37](#) and ["Operation in HA Mode" on page 37](#) in the "Cryptoki Configuration" section of the *ProtectToolkit-C Administration Guide* for more information about these operating modes.

Syntax

Delete FM

```
ctfm d -i<fmID> [-a<device>|-A]
```

Import FM

```
ctfm i -I<certLabel> -f<fmFile> [-a<device>|-A] [-c<certFile>]
```

Query FM status**ctfm q** [-a<device>|-A]**Verify an FM file****ctfm v-f**<fmFile> [-i<certLabel>] [-a<device>|-A] [-c<certFile>]**Commands**

| Command | Description |
|----------|---|
| d | <p>Delete FM</p> <p>This command is used to delete an FM so it becomes inactive on one or all HSMs. You must specify the FM's hex ID with the -i<fmID> option.</p> |
| i | <p>Import FM</p> <p>This command is used to load a new FM onto one or all HSMs. Starting with ProtectToolkit version 5.4, you can upload multiple custom FMs to an HSM and use them simultaneously. Only one PKCS#11 patched FM can be loaded and used at a time. If a patched FM is already loaded, it is overwritten by the new FM. Refer to Custom Functions and PKCS#11 Patched Functions in the <i>FM SDK Programming Guide</i> for descriptions of these FM types. Existing FMs do not need to be disabled prior to executing this command.</p> <p>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>.</p> <p>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.</p> <p>The device Administrator PIN (and possibly the Admin SO PIN) will be required.</p> |
| q | <p>Query FM Status</p> <p>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.</p> <p>No PINs are required to perform this operation.</p> |
| v | <p>Verify an FM Signature</p> <p>This command is used to verify that an FM file has been signed correctly, without attempting to download the FM.</p> <p>The device Administrator PIN will be required.</p> <p>The behavior of the <certLabel> and <certFile> parameters is the same as for the Import FM command above.</p> |

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> Name of file holding a new FM. |
| -h, -? | --help Display usage information. |
| -i <fmID> | --fmid= <fmID> Specifies the FM ID in hex format. |
| -l <certLabel> | --fm-cert-label= <certLabel> FM validation certificate object label. |

ctident

Utility for establishing and maintaining trust between devices within the ProtectToolkit-C environment.

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 37](#) and ["Operation in HA Mode" on page 37](#) in the "Cryptoki Configuration" section of the *ProtectToolkit-C Administration Guide* for more information about these operating modes.

Syntax

Generate new HSM ID key pair

```
ctident gen [-b] [-f] [-o<so_pin>] <targets>
```

Add trust for <peers> to <targets>

```
ctident trust [-b] [-f] [-o<so_pin>] <targets> <peers>
```

Remove trust for <peers> from <targets>

```
ctident remove [-b] [-o<so_pin>] <targets> <peers>
```

List HSM ID keys

```
ctident list [-b] [-t<types>] [-a] <targets>
```

Check HSM ID keys

```
ctident check [-b] <targets>
```

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. |

| Command | Description |
|---------------|--|
| gen | <p>The generate key command gen is used to generate the HSM Identity key-pair on the devices specified by the <targets> parameter.</p> <p>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 regeneration. When a key is regenerated, the existing 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 keys are listed.</p> <p>The -a parameter lists all of the non-sensitive attributes for each key/cert.</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.</p> <p>The batch mode -b parameter can be used to disable PIN prompting.</p> |

| Command | Description |
|--------------|--|
| 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. |

| Option | Description |
|-------------------|--|
| -t <types> | --type =<types> Specifies a comma-separated list of key types. The available key types are: pri - local private keys pub - local public keys peer - peer public keys all - all key types |
| -u <pin> | Specifies the Administrator PIN. Use of this operation is a security risk due to the tools command line being visible in the systems process list. |

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 ProtectToolkit-C environment, used for 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 **ctkmu** 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 37](#) and ["Operation in HA Mode" on page 37](#) for further details.

Syntax

Create key from entered components

```
ctkmu c -t<type> -n<name> -a<attribute> -k<num> [-s<slot>] [-z<size>] [-i<hex_string>] [-p]
```

Create key with components

```
ctkmu c -t<type> -n<name> -a<attribute> -k<num> -g [-s<slot>] [-z<size>] [-i<hex_string>]
```

Create key without components

```
ctkmu c -t<type> -n<name> -a<attribute> [-s<slot>] [-z<size>] [-i<hex_string>] [-C<curve_name>]
```

Delete object

```
ctkmu d -n<name> [-s<slot>]
```

Erase smart card

```
ctkmu e -c<slot>
```

Import key(s) from single-custodian smart card

```
ctkmu i -w<name> -c<slot> [-s<slot>]
```

Import key(s) from multi-custodian smart cards

```
ctkmu i -c<slot> [-s<slot>]
```

Import key(s) from console

```
ctkmu i -a<attribute> -n<name> -t<type> -w<name> -y [-s<slot>] [-i<hex_string>] [-m] [-z<size>]
```

Import key(s) from file

```
ctkmu i -w<name> <filename> [-s<slot>] [-2]
```

Import domain parameters

```
ctkmu idp -n<name> -t<type> -a<attribute> <filename> [-s<slot>]
```

Import token

```
ctkmu it <filename> [-s<slot>]
```

Import from PKCS #12 file

```
ctkmu j -n<name> -a<attribute> <filename> [-s<slot>] [-i<hex_string>]
```

List objects on token(s)

```
ctkmu l -s<slot> [-v] [-n<name>]
```

Modify attributes

```
ctkmu m -n<name> -a<attribute> [-s<slot>]
```

Initialize or change PINs

```
ctkmu p [-s<slot>] [-O]
```

Replicate token

```
ctkmu rt -d <slotlist> [-s <slot>]
```

Smart card status

```
ctkmu s -c<slot>
```

Initialize/re-initialize token

```
ctkmu t [-s<slot>] [-l<label>]
```

Export key(s) to single-custodian smart card

```
ctkmu x -w<name> -c<slot> [-s<slot>] [-3] [-4] [-n<name>]
```

Export key(s) to multi-custodian smart card

```
ctkmu x -c<slot> [-s<slot>] [-3] [-4] [-n<name>] [-M]
```

Export key(s) to file

```
ctkmu x -w<name> <filename> [-s<slot>] [-3] [-4] [-n<name>]
```

Export key(s) to console

```
ctkmu x -n<name> -w<name> -y [-s<slot>] [-m]
```

Export key(s) to PKCS #12 file

```
ctkmu x [-s<slot>] -j --pkLabel --keyCertLabel [--certalgo] [--pkalgo] <filename>
```

Export token

```
ctkmu xt -S<serial> <filename> [-s<slot>]
```

Commands

| Command | Description |
|------------|--|
| c | <p>Create Key</p> <p>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 198 contains further information on key attributes. Common 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.</p> |
| d | <p>Delete Key</p> <p>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.</p> |
| e | <p>Erase Smart Card</p> <p>This command is used to erase a smart card in the specified slot and will leave the smart card in an uninitialized state.</p> |
| i | <p>Import Key</p> <p>This command is used to import keys previously exported with the export command (see below).</p> |
| idp | <p>Import Domain Parameters</p> <p>This command is used to store Domain Parameters objects onto a Token.</p> <p>The -s option indicates the slot e.g. -s1 for slot 1 - default is slot 0.</p> <p>The -n option indicates the label of the new object.</p> <p>The -t option specifies the key type, it may be ec or dsa or dh but only ec is supported.</p> <p>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.</p> <p>The <filename> option specifies a test file that contains the information required to construct the domain parameters.</p> |
| it | <p>Import Token</p> <p>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.</p> <p>To complete this operation, ctkmu will prompt for the user PIN of the destination token.</p> <p>When importing into 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> |
| j | <p>Import Private Key</p> <p>This command is used to import a Private Key and a Certificate from a PKCS #12 file format.</p> |

| Command | Description |
|-----------|--|
| l | <p>List Information</p> <p>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.</p> |
| m | <p>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.</p> |
| 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 uninitialized 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 uninitialized 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> |

| Command | Description |
|------------------|--|
| <p>x</p> | <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 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 <code>CKA_EXPORTABLE=TRUE</code> and <code>CKA_EXTRACTABLE=FALSE</code>, 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 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 ctkmu using two different passwords. The user needs to decide which password policy best suits their needs. > The RC family of encryption algorithms (and others) are prohibited in FIPS mode. ctkmu shall reject the command and display a warning message if they are used under this security policy. <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>NOTE When logging in to a smart card, the card is locked after 7 consecutive incorrect PIN attempts. You must re-initialize the card to set a new PIN.</p> </div> |
| <p>xt</p> | <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, ctkmu will prompt for the user PIN of the source token.</p> |

Options

| Option | Description |
|----------------|--|
| -a<attributes> | <p>--attributes =<attributes> Specifies attributes for an object / key. Valid attributes are:</p> <ul style="list-style-type: none"> 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) <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>NOTE You can only set CKA_EXPORTABLE to TRUE if you are using firmware 5.06.04 or newer and have set the <i>Weak PKCS#11 Mechanisms</i> security flag. For more information about CKA_EXPORTABLE and setting the <i>Weak PKCS#11 Mechanisms</i> security flag, see CKA_EXPORT, CKA_EXPORTABLE and "Weak PKCS#11 Mechanisms" on page 62, respectively.</p> </div> |
| -c<slot> | <p>--sc-slot-num =<slot> Specifies the Smart Card slot to export to or import from.</p> |

| Option | Description |
|------------------------|---|
| -C <curve_name> | <p>--curve-name =<label></p> <p>Specifies which curve to use. Valid values are:</p> <ul style="list-style-type: none"> > brainpoolP160r1 > brainpoolP160t1 > brainpoolP192r1 > brainpoolP192t1 > brainpoolP224r1 > brainpoolP224t1 > brainpoolP256r1 > brainpoolP256t1 > brainpoolP320r1 > brainpoolP320t1 > brainpoolP384r1 > brainpoolP384t1 > brainpoolP512r1 > brainpoolP512t1 > c2tnb191v1 > c2tnb191v1e > curve25519 > ed25519 > P-192 (also known as prime192v1 and secp192r1) > P-224 (also known as secp224r1) > P-224K1 (also known as secp224k1) > P-256 (also known as prime256v1 and secp256r1) > P-384 (also known as secp384r1) > P-521 (also known as secp521r1) > secp256k1 > or any valid ECC Domain Parameter object label <p>If -tec is specified, the -C parameter must be included in the command otherwise ctkmu 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> | <p>Specifies a file to be created for export or used to import a key, certificate, token, or set of domain parameters.</p> |

| Option | Description |
|------------------------|--|
| -g | --gen-comp Generate key components. |
| -h, -? | --help Display usage information. |
| -j | --pkcs12 Export to PKCS#12 format. -pkLabel Private Key to be exported to PKCS#12 file. -keyCertLabel Certificate Label to be exported to PKCS#12 file. -pkalgo 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. -certalgo 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. |
| -k<numb> | --num-comp =<numb> Number of key components required to be entered or number to be generated (when -g parameter is specified). |
| -l<label> | --label =<label> Specify label. |
| -m | --multi-part 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. |

| Option | Description |
|---------------------------|---|
| -O | --SO-PIN Change the Security Officer PIN. Used with the change PIN command. |
| -o <userpin> | Specifies the Security Officer PIN in the command line along with the command, instead of prompting for it afterwards. Useful for automation, but not recommended for use in production environments since the password is exposed in plaintext. Can be specified with all commands that require SO login. |
| -p | --pinpad Use a supported PIN pad device for entering key components. See "Key Entry via PIN Pad" on page 95 for complete PIN pad instructions. |
| -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 |
| -u <userpin> | Specifies the slot user PIN in the command line along with the command, instead of prompting for it afterwards. Useful for automation, but not recommended for use in production environments since the password is exposed in plaintext. Can be specified with all commands that require token login. |
| -v | --verbose Displays the attributes that ctkm may change. |
| -w<name> | --wrap-key =<name> Name of the key used to wrap or unwrap. NOTE If you are specifying a DES3 key as the wrapping key for an export key operation (with the x command), you must also include the -3 option. If -3 is not included, the operation fails and an <code>Export operation failed 0x63 - key type inconsistent error</code> is returned. |
| -y | --console Import/Export using the console. |

| Option | Description |
|------------------|--|
| -z <size> | <p>--size=<size> Size of the key to create/import (for AES, RC2, RC4, CAST, RSA, DSA and generic secret).</p> <div data-bbox="576 384 1433 642" style="border: 1px solid #ccc; padding: 5px;"> <p>NOTE If the FIPS Mode security policy is enabled, the cryptographic operations of RSA, DSA, DH, and EC algorithms are restricted to key sizes within a specified range. For more information about the size limitations of keys that are created or imported in FIPS Mode, see "FIPS Mode Operational Restrictions" on page 55 in the "Security Policies and User Roles" section of the <i>ProtectToolkit-C Administration Guide</i>.</p> </div> |
| -2 | <p>--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 ProtectToolkit-C V4.1" on page 202).</p> |
| -3 | <p>--PTKC3 Generate export to smart card and file using the ProtectToolkit-C version 3 format (CKM_WRAPKEY_DES3_CBC), instead of the default wrapping mechanism (CKM_WRAPKEY_AES_KWP). Used when exporting keys to be sent to older style HSMs.</p> <div data-bbox="576 1041 1433 1129" style="border: 1px solid #ccc; padding: 5px;"> <p>NOTE Wrapping and unwrapping operations using this option will fail in FIPS mode.</p> </div> |
| -4 | <p>--PTKC4 Generate export to smart card and file using the ProtectToolkit-C version 4 format (CKM_WRAPKEY_AES_CBC), instead of the default wrapping mechanism (CKM_WRAPKEY_AES_KWP). Used when exporting keys to be sent to older style HSMs.</p> <div data-bbox="576 1352 1433 1402" style="border: 1px solid #ccc; padding: 5px;"> <p>NOTE Wrapping operations using this option will fail in FIPS mode.</p> </div> |

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 ProtectToolkit-C environment.

The utility will recognize older firmware and report meaningful error messages.

Syntax

Create ticket from offline specification

```
ctlimits ct -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
```

Present ticket to HSM

```
ctlimits pt <filename> [-U<usertype>] [-O<objtype>] -k<keyspec> [-i<key_id>]
```

Apply limit attributes directly

```
ctlimits up -k<keyspec> [-U<usertype>] [-O<objtype>] [-i<key_id>] [-C<count>] [-L<limit>] [-s<date>] [-e<date>] [-c<cert_filename>]
```

View key attributes

```
ctlimits vk -k<keyspec> [-U<usertype>] [-O<objtype>] [-i<key_id>]
```

Commands

| Command | Description |
|-----------|---|
| ct | <p>Create ticket from offline specification</p> <p>This command creates a SET ATTRIBUTES ticket in the file filename.</p> <p>This ticket may be presented to a ProtectToolkit-C HSM using the ctlimitspt 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).</p> <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> User type creating ticket - may be either SO or USER (default)</p> |
| -k<keyspec> | <p>--keyspec=<keyspec> Specification of a key. The format used is TokenLabel(pin)/KeyLabel, where the pin is optional and TokenLabel may specify slot by number For example: -k MyToken(1234)/MyKey (Pin 1234) or -k MyToken/MyKey (no Pin - utility may prompt for pin) -k SLOTID=2/MyKey</p> |

| Option | Description |
|---------------------------|--|
| -O <objtype> | --objtype =<objtype> Object type of the key. May be secret_key , certificate , public_key , or private_key . The 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 |

ctmultitoken

The **ctmultitoken** utility is a simple demonstration tool that allows you to perform basic cryptographic functions on a ProtectServer HSM. It allows you to specify an operation, and one or more tokens on which to perform that operation. The **ctmultitoken** utility runs the operations and returns a summary of the results.

Syntax

ctmultitoken -mode <mode> {-slots <slot_list> | -nslots <slot_threads>} [options...]

| Argument(s) | Shortcut | Description |
|---------------------------------|-------------|--|
| -alarm <secs> | -al | Sound periodic alarm (every <secs> seconds) if error occurs. |
| -blob <blob_count> | -b | Number of data blobs to be signed during each multisign operation. |
| -curve <curve_num> | -crv | ID number of ECC curve. If user-defined (99), then must specify -parmfile . |
| -eciesdata <filename> | -ecd | Specifies the file to receive the plaintext data used. |
| -eciesenc <filename> | -ece | Specifies the file to receive the encrypted data. |
| -ecieskey <filename> | -eck | Specifies the file to receive the DER-encoded private key. |
| -enddate <YYYYMMDD> | -end | Validity end date for key, in YYYYMMDD format. |
| -force | -f | Avoid prompts for responses. |
| -gcmaad <bytes> | -gad | Specify the length of the AAD data used for GCM/GMAC. The AAD data can not be larger than 1024 bytes. |
| -gcmiv <bits> | -giv | Specify the length of the IV (in bits) to be used for GCM/GMAC. Valid values: 0,96,128 |
| -help | -h | Display help information and operating modes only. |
| -kdfchoice <kdf_index> | -kdf | Select key derivation function - specify choice list index. |
| -kdfscnt <counter_index> | -kds | Select key derivation session counter type - specify choice list index. |
| -key <key_size> | -k | Size of key: asymmetric in bits (default = 1024 for RSA, 2048 for DSA). Symmetric in bytes (i.e. 16, 24, 32 for AES/ARIA). |
| -keychoice <key_index> | -kc | Select key type to derive/generate - specify choice list index. |

| Argument(s) | Shortcut | Description |
|-------------------------------|--------------|---|
| -keyderiv <keysize> | -kde | Size of key to derive with (ex. 1024 for X9.42 Diffie Hellman). |
| -kwicv | -kiv | Use external ICV for the key wrap mechanism. |
| -logfile <filename> | -l | File for results logging. |
| -mode <mode> | -m | Operating mode. See "Operating Modes" on the next page . |
| -multipartsig | -msig | Use multipart signatures. |
| -nodec | -nod | Decryption operation will not be performed. Only symmetric and asymmetric encryption will be performed and measured. |
| -nodestroy | -n | Leaves created objects on the HSM after test completes. |
| -noenc | -noe | Perform only one encryption operation. Only symmetric and asymmetric decryption will be performed and measured. |
| -nosign | -nos | Perform only one sign operation. Only verify will be performed and measured. |
| -nounwrap | -nou | Unwrapping operation will not be performed. Only wrapping will be performed and measured. |
| -noverify | -nov | Verify operation will not be performed. Only sign will be performed and measured. |
| -noverifyr | -nvr | Do not verify decryption results. |
| -nowrap | -now | Perform only one wrapping operation. Only unwrapping will be performed and measured. |
| -nslots <slot_threads> | -ns | Create multiple threads on the same slot(s). Specify <slot>x<number of threads>, with multiple slots separated by commas. The example below creates 5 threads on slot 1 and 20 threads on slot 2: Example: -nslots 1x5,2x20 You must specify either this option or -slots . See "-slots <slots>" on the next page . |
| -packet <packet_size> | -p | Size of packet used in operation. |
| -parmfile <param_file> | -prm | File for EC curve parameters or OAEP source data (0 = none for OAEP). |

| Argument(s) | Shortcut | Description |
|------------------------------|-------------|---|
| -password <password> | -pwd | Specify password to use for token. |
| -pbkd2prf | | Specify the type of PRF to use for PBKD2-based key derivation. |
| -prftype <type> | -prf | Specify the type of PRF to use for PRF-based key derivation. |
| -sharefile <filename> | -shf | Shared data file used for operation. |
| -silent | -sil | Disables system "beep" that is generated when a error occurs. |
| -slots <slots> | -s | <p>List of slots to use (slot numbers separated by commas). List the same slot multiple times to create multiple threads on that slot. The example below creates 2 threads on slot 1 and 3 threads on slot 2:</p> <p>Example: -slots 1,1,2,2,2</p> <p>To create many threads on the same slot, use -nslots instead. See "-nslots <slot_threads>" on the previous page.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>NOTE Multithreaded requests cannot be sent if ctmultitoken is being used in Software-only mode.</p> </div> |
| -startdate <YYYYMMDD> | -sta | Validity start date for key in format <YYYYMMDD>. |
| -subprime <size> | -sub | Size of the subprime in bits. |
| -symm <mechanism> | -sym | Select symmetric key mechanism for symderive/pbegen or key choice for symgen (can also use -keychoice). |
| -template | -tp | Attaches a generic unwrap template or derive template for the wrapunwrap or symderive mode respectively. |
| -timed <secs> | -t | Fixed amount of time to run (seconds). |
| -usage <uses> | -u | Number of times a key is allowed to be used. |
| -verbose | -v | Show all thread performances. Default is only first and last threads. |

Operating Modes

The following table lists the available operating modes for the **ctmultitoken** utility. The operating mode is specified using the **-mode** parameter.

| Mode | Description |
|--------------------------|-------------------------|
| aescmac | AES CMAC sign |
| aesenc | AES ECB encrypt |
| aesenccbc | AES CBC encrypt |
| aesengcm | AES GCM encrypt |
| aesenkw | AES KW encrypt |
| aesencofb | AES OFB encrypt |
| aesmac | AES MAC sign |
| aeswrapkw | AES KW wrap |
| aeswrapkwp | AES KWP wrap |
| ariaenc | ARIA ECB encrypt |
| ariaenccbc | ARIA CBC encrypt |
| ariamac | ARIA MAC sign |
| bip32childderive | BIP32 Child Key derive |
| bip32masterderive | BIP32 Master Key derive |
| des3encofb | DES3 OFB encrypt |
| descmac | DES3 CMAC sign |
| desenc | DES3 ECB encrypt |
| desenccbc | DES3 CBC encrypt |
| desmac | DES3 MAC sign |
| desx919mac | DES3 X919 MAC sign |
| dsakeygen | DSA Key Generation |
| dsasigver | DSA bare sign |
| ecdhderive | ECDH derive key |

| Mode | Description |
|-----------------------------------|--|
| ecdsagbcsha256sigver | SHA256 ECDSA-GBCS sign |
| ecdsakeygen | ECDSA Key Generation |
| ecdsasha1sigver | SHA1 ECDSA sign |
| ecdsasha224sigver | SHA224 ECDSA sign |
| ecdsasha256sigver | SHA256 ECDSA sign |
| ecdsasha384sigver | SHA384 ECDSA sign |
| ecdsasha512sigver | SHA512 ECDSA sign |
| ecdsasha3-224-sigver | SHA3-224 ECDSA sign |
| ecdsasha3-256sigver | SHA3-256 ECDSA sign |
| ecdsasha3-384sigver | SHA3-384 ECDSA sign |
| ecdsasha3-512sigver | SHA3-512 ECDSA sign |
| ecdsasigver | ECDSA sign |
| ecedwardskeygen | EC Edwards Key Generation |
| eciesshimxorhmacsha1 | ECIES XOR with HMAC SHA1 decrypt |
| eciesshimxorhmacsha1shared | ECIES XOR with HMAC SHA1 and shared data decrypt |
| eciesxorhmacsha1 | ECIES XOR enc/dec with HMAC SHA1 |
| eciesxorhmacsha1shared | ECIES XOR enc/dec with HMAC SHA1 and shared data |
| eddsakeygen | EdDSA Key Generation |
| eddsasha1sigver | SHA1 EDDSA sign |
| eddsasha224sigver | SHA224 EDDSA sign |
| eddsasha256sigver | SHA256 EDDSA sign |
| eddsasha384sigver | SHA384 EDDSA sign |
| eddsasha512sigver | SHA512 EDDSA sign |

| Mode | Description |
|----------------------------|--|
| eddsasha3-224sigver | SHA3-224 EdDSA sign |
| eddsasha3-256sigver | SHA3-256 EdDSA sign |
| eddsasha3-384sigver | SHA3-384 EdDSA sign |
| eddsasha3-512sigver | SHA3-512 EdDSA sign |
| eddsasigver | EDDSA sign |
| keccak-1600 | KECCAK-1600 |
| md5 | MD5 Hashing |
| milenage | MILENAGE sign |
| pbegen | PBE key generation |
| randgen | Random number generation |
| rc4enc | RC4 encrypt |
| rsa1863primekeygen | RSA FIPS 186-3 using Primes key generation |
| rsaenc | RSA encrypt |
| rsakeygen | RSA key generation |
| rsaoaepenc | RSA OAEP encrypt |
| rsasigver | RSA sign |
| rsax931keygen | RSA X9.31 key generation |
| seedenc | SEED ECB encrypt |
| seedencbc | SEED CBC encrypt |
| seedmac | SEED MAC sign |
| sha1 | SHA-1 Hashing |
| sha1dsasigver | SHA1 DSA sign |
| sha1hmac | SHA1 HMAC sign |

| Mode | Description |
|---------------------------|-------------------------|
| sha1rsapsssigver | SHA1 RSA PSS sign |
| sha1rsasigver | SHA1 with RSA sign |
| sha224 | SHA-224 Hashing |
| sha224dsasigver | SHA224 DSA sign |
| sha224hmac | SHA224 HMAC sign |
| sha224rsaoapenc | SHA224 RSA OAEP encrypt |
| sha224rsapsssigver | SHA224 RSA PSS sign |
| sha224rsasigver | SHA224 with RSA sign |
| sha256 | SHA-256 Hashing |
| sha256dsasigver | SHA256 DSA sign |
| sha256hmac | SHA256 HMAC sign |
| sha256rsaoapenc | SHA256 RSA OAEP encrypt |
| sha256rsapsssigver | SHA256 RSA PSS sign |
| sha256rsasigver | SHA256 with RSA sign |
| sha384 | SHA-384 Hashing |
| sha384dsasigver | SHA384 DSA sign |
| sha384hmac | SHA384 HMAC sign |
| sha384rsaoapenc | SHA384 RSA OAEP encrypt |
| sha384rsapsssigver | SHA384 RSA PSS sign |
| sha384rsasigver | SHA384 with RSA sign |
| sha512 | SHA-512 Hashing |
| sha512dsasigver | SHA512 DSA sign |
| sha512hmac | SHA512 HMAC sign |

| Mode | Description |
|---------------------------|--------------------------------------|
| sha512rsaoepenc | SHA512 RSA OAEP encrypt |
| sha512rsapsssigver | SHA512 RSA PSS sign |
| sha512rsasigver | SHA512 with RSA sign |
| sha3-224 | SHA3-224 Hashing |
| sha3-256 | SHA3-256 Hashing |
| sha3-384 | SHA3-384 Hashing |
| sha3-512 | SHA3-512 Hashing |
| symderive | Symmetric key derivation |
| symgen | Symmetric key generation |
| tuak | TUAK sign |
| wrapunwrap | Wrap/unwrap operations |
| x942dhderive | X9.42 DH Derive |
| x942dhkeygen | X9.42 DH Key Pair Generation |
| x942dhparamsgen | X9.42 DH Domain Parameter Generation |

Notes

1. If you are performing RSA operations, you have the option of specifying a key size (512, 1024, 2048, 4096, 8192). If no key size is specified, the default key size of 1024 will be used. For example:

```
ctmultitoken -mode rsasigver -key 512 -slots 1
```

2. If you are performing wrapunwrap operation, it will perform the following operations:
 - Generate RSA key pair and a symmetric DES key.
 - Wrap DES key with RSA public key.
 - Unwrap wrapped key above with RSA private key.
 - Verify the unwrapped key.
3. A thread will be spawned to perform tests on each slot specified. A slot can be specified multiple times, in which case multiple threads will be created for the slot.
4. Options for the following modes can be used with the default 1024 bit key size only:
 - sha256rsasign - SHA256 with RSA
 - sha384rsasign - SHA384 with RSA

- sha512rsassign - SHA512 with RSA

If you specify a keysize on the command line (any of 1024, 2048 or 4096), the result is the 1024 bit benchmark speed, and a file called "1024" or "2048" or "4096" is created - that is the keysize parameter is parsed as a filename to which results are saved.

5. To run **ctmultitoken** in **symdrive** mode, you must first turn on the Weak PKCS#11 Mechanisms flag (see ["Weak PKCS#11 Mechanisms" on page 62](#)).

Named and User-Defined Curves

ProtectServer HSMs employ named and user-defined curves. **ctmultitoken** supports this option, as illustrated in the following example:

```
./ctmultitoken -mode ecdsasigver -s 1,1,1,1,1,1,1,1
```

Prime field curves:

```
[0]secp224k1 [1]secp224r1 (P-224) [2]secp256k1 [3]secp384r1 (P-384)
[4]secp521r1 (P-521)
```

X9.62 prime curves

```
[5]X9_62_prime192v1 (P-192) [6]X9_62_prime256v1 (P-256)
```

X9.62 two field curves:

```
[7]X9_62_c2tnb191v1e [8]X9_62_c2tnb191v1
```

Brainpool Curves:

```
[9]brainpoolP160r1 [10]brainpoolP160t1 [11]brainpoolP192r1
[12]brainpoolP192t1 [13]brainpoolP224r1 [14]brainpoolP224t1
[15]brainpoolP256r1 [16]brainpoolP256t1 [17]brainpoolP320r1
[18]brainpoolP320t1 [19]brainpoolP384r1 [20]brainpoolP384t1
[21]brainpoolP512r1 [22]brainpoolP512t1
```

Montgomery curves:

```
[23]curve25519
```

Please pick a curve (0-23):

Here, you would provide the filepath to the file specifying the Elliptical Curve parameters. The format and content of the parameter file follow industry standards. See ["Sample EC Domain Parameter Files" on page 203](#) for some examples.

ctotp

Utility to initialize (enable), reinitialize, or disable the One-Time Password (OTP) feature for a specified slot and role.

One-Time Password introduces multifactor authentication to the SafeNet ProtectToolkit-C environment. The OTP is a 6-digit number displayed on the SafeNet 110 OTP Token. This 6-digit number is automatically changed every 30 seconds on the token screen. When OTP is enabled for a slot, the User or Security Officer must enter the token PIN, followed by the 6-digit OTP, to log in to the slot. With OTP disabled, only the role's token PIN is required.

See "[Multifactor Authentication \(One-Time Password\)](#)" on page 87 for detailed procedures.

Syntax

Initialize/enable OTP on the specified slot

```
ctotp init -s<slot_num> -t<token_SN> -x<xml_file> -p<passcode_file> [-O]
```

Log in to the specified slot using OTP

```
ctotp login -s <slot_num> [-O]
```

Re-initialize OTP on the specified slot

```
ctotp reinit -s<slot_num> -t<token_SN> -x<xml_file> -p<passcode_file>
```

Disable OTP on the specified slot

```
ctotp del -s<slot_num> [-O]
```

NOTE Since the SafeNet 110 OTP token is time-based, ensure that the HSM time is in sync with the client by running **ctconf -t** on the client machine before you initialize OTP.

Commands

| Command | Description |
|--------------|---|
| del | Disable OTP for the specified slot (-s). To disable OTP for the Security Officer role, include the -O option. |
| init | Initialize/enable OTP for the specified slot (-s). You must specify the SafeNet 110 OTP Token serial number (-t), and filepaths to TokenSeed.xml (-x) and PSCKPassword.txt (-p). To initialize OTP for the Security Officer role, include the -O option. |
| login | Log in to the HSM token. To log in as the Security Officer, include the -O option. |

| Command | Description |
|---------------|--|
| reinit | Re-initialize OTP for the User on the specified slot (-s) using a different SafeNet 110 OTP Token. The Security Officer must log in to use this command. You must specify the SafeNet 110 OTP Token serial number (-t), and filepaths to TokenSeed.xml (-x) and PSCKPassword.txt (-p). You may re-initialize OTP for the User or Administrator roles only. |

Options

| Option | Description |
|---------------------------|--|
| -s <slotnum> | --slot-num =<slotnum> Specifies the slot on which to initialize, re-initialize, or disable OTP. |
| -t <token_SN> | --token-name =<label> Specifies the desired SafeNet 110 OTP Token serial number (located on the back of the device). This serial number must match a number in the provided TokenSeed.xml file. |
| -x <xml_file> | Specifies the full or relative filepath to the TokenSeed.xml file. |
| -p <password_file> | Specifies the full or relative filepath to the PSCKPassword.txt file. |
| -O | Specifies that the command applies to the Security Officer role (or the Administration Security Officer role on the Admin token). |
| -h, -? | --help Display help information. |

Examples

Initialize/enable OTP on the specified slot

```
ctotp.exe init -s0 -tGALT10282853 -xC:\otp\seed.xml -pC:\otp\passcode.txt -O
```

```
Please Enter the Security Officer Token PIN:
```

```
=====
```

```
OTP Initialization Successful.
```

```
=====
```

Log in to the specified slot using OTP

```
>ctotp login -s0
```

```
Please Enter the Token PIN:
```



```
=====
OTP Login Successful.
=====
```

Re-initialize OTP on the specified slot

```
ctotp reinit -s0 -tGALT10282857 -xc:/otp/seed.xml -pc:/otp/passcode.txt
```

```
Please Enter the Security Officer Token PIN:
```

```
Please Enter the Token PIN:
```

```
=====
OTP Re-Initialization Successful.
=====
```

Disable OTP on the specified slot

```
ctotp delete -s0
```

```
Please Enter the Token PIN:
```

```
=====
OTP Deletion Successful.
=====
```

Exit Status

The **ctotp** 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.

ctperf

Reports on the performance of PKCS #11 cryptographic operations.

NOTE This performance measurement is application-dependent, therefore the results are indicative only.

Syntax

```
ctperf [-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>]
```

Options

| 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. |

| Option | Description |
|------------------------|--|
| -C <curve_name> | <p>--curve-name=<label></p> <p>Specifies which curve to use. Valid values are:</p> <ul style="list-style-type: none"> > brainpoolP160r1 > brainpoolP160t1 > brainpoolP192r1 > brainpoolP192t1 > brainpoolP224r1 > brainpoolP224t1 > brainpoolP256r1 > brainpoolP256t1 > brainpoolP320r1 > brainpoolP320t1 > brainpoolP384r1 > brainpoolP384t1 > brainpoolP512r1 > brainpoolP512t1 > c2tnb191v1 > c2tnb191v1e > curve25519 > ed25519 > P-192 (also known as prime192v1 and secp192r1) > P-224 (also known as secp224r1) > P-224K1 (also known as secp224k1) > P-256 (also known as prime256v1 and secp256r1) > P-384 (also known as secp384r1) > P-521 (also known as secp521r1) > secp256k1 > or any valid Domain Parameters object label <p>If a curve name is not specified, the default P-192 is used.</p> |
| -e | <p>--EMC</p> <p>Runs tests suitable for EMC testing purposes.</p> |
| -h,-? | <p>--help</p> <p>Display usage information.</p> |
| -i <count> | <p>--iterations=<count></p> <p>The number of iterations of the performance tests to run. Default is 1, use -1 to specify an infinite count.</p> |

| Option | Description |
|----------------------------|--|
| -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. |

| Option | Description |
|---------------|--|
| -o<mechanism> | <p>--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 ProtectToolkit-C supported mechanisms please refer to the <i>ProtectToolkit-C Programmer's Guide</i>.)</p> <p>The following mechanism tests are supported:</p> <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 -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 |

| Option | Description |
|--------|--|
| | <ul style="list-style-type: none"> -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 > ecdsa_sha1_verify -o cert gen invokes certificate-generation and signing tests -o dh_kg Diffie-Hellmann key generation mechanism -o rng the random number generation mechanism |

| Option | Description |
|---------------------|---|
| | <p>-o extra the following:</p> <ul style="list-style-type: none"> > des3_ses_kg: DES3 Session Key Generation/Destruction > des3_tok_kg: DES3 Token Key Generation/Destruction > des3_kw: DES3 Key Wrap > cert_gen: Certificate Generation/Destruction > ses: Session Open/Close > obj: Object Creation/Search/Destroy > login: Login / Logout > xor_dk: XOR Derive Key <p>-o mc reading of the monotonic counter object</p> |
| -p | <p>--dsa-params Parameters generated for DSA.</p> |
| -q | <p>--quick Quick Keygen (key generation tests not performed).</p> |
| -r | <p>--random Execute a random selection of the performance tests.</p> |
| -R | <p>--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.</p> |
| -s <slot> | <p>--slot-num=<slot> Specify the slot number to perform test on.</p> |
| -t <seconds> | <p>--time-period=<seconds> Specify the measurement period. Default is 5 seconds.</p> |
| -v | <p>--verbose Verbose (provide more information).</p> |
| -x | <p>--csv Create a CSV (comma-separated variable) file.</p> |
| -z <name> | <p>--cryptoki-module=<name> Optionally, specify a different cryptoki module to use. May include full path.</p> |

ctstat

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 37](#) and ["Operation in HA Mode" on page 37](#) for further details.

On some UNIX platforms, **ctstat** is a built-in command. To use PTK's **ctstat**, you must first set the environment path by running **setvars.sh**. If you decide to set the environment path manually, ensure that the PTK path is set at the beginning of the path variable, before the system path.

Syntax

```
ctstat [-a] [-b] [-h] [-j] [-m] [-n<name>] [-s<slot>] [-t<name>] [-v]
```

| Option | Description |
|-----------------------|---|
| -a | --all Display the status for everything associated with the 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). |

auditverify

Verification utility for ProtectServer audit log records.

The **auditverify** utility allows the Auditor to read and verify audit logs that have been encrypted using the Audit Key on the HSM appliance. The Auditor's PIN must be entered at the prompt.

NOTE This utility is not applicable to ProtectServer PCIe 2 HSMs.

Syntax

auditverify **-l**<logfilename> [**-d**<devicenum>]

| Argument(s) | Description |
|-------------------------|--|
| -d <devicenum> | --device=<device> If the client machine is connected to multiple ProtectServer HSMs, this argument specifies which Audit Key to use when verifying the log file. |
| -l <logfilename> | --logfile=<logfilename> Specify the filename of the audit log to be verified. |

Example

```
>auditverify -l applog
Please Enter the Auditor's PIN:
Starting to verify
2017-07-12 14:12:29,success,0,Audit Log initial message
,00000000000000000000000000000000000000000000000000000000000000000000,692f41f2ec2bbb42411c7b2c5e323
0b39dab28bd5178ef1b3e71b34331500765
2017-07-12 14:53:44,success,0,CS_Initialize:
,692f41f2ec2bbb42411c7b2c5e3230b39dab28bd5178ef1b3e71b34331500765,6afe98063371c25d675616827ec51
d5d23f879312d935c230ebe566db3e064a0
2017-07-12 14:53:44,success,1,CS_OpenSession:
,6afe98063371c25d675616827ec51d5d23f879312d935c230ebe566db3e064a0,868b4457c44c525febad5c87d9d27
ee745829aa38f9ac6bf2405a788f8c3ea89
2017-07-12 14:53:44,success,1,CS_OpenSession:
,868b4457c44c525febad5c87d9d27ee745829aa38f9ac6bf2405a788f8c3ea89,8e65ee17ce0d0b835fd746558d5c1
14a45baf6e4e7f579b1f7b22f204db51538
2017-07-12 14:53:44,success,1,CS_FindObjects:
,8e65ee17ce0d0b835fd746558d5c114a45baf6e4e7f579b1f7b22f204db51538,7ff4201694d9b5a68b6f3e205c753
80e10975cddd9fff45641cd82fdb7d7eee17
2017-07-12 14:53:44,success,1,CS_GetAttributeValue:
,7ff4201694d9b5a68b6f3e205c75380e10975cddd9fff45641cd82fdb7d7eee17,c2fd9b7bd90e370a8684259f120be
da70f3ce2a7aa217e753f02864618066fc8
2017-07-12 14:53:44,success,1,CS_CloseSession:
,c2fd9b7bd90e370a8684259f120beda70f3ce2a7aa217e753f02864618066fc8,a3ef1d28edcf2b1eb4efa2f7d0752
41e2bf1253f85b7dc36895b2ce07cd4732b
...<snip>...
2017-07-11 19:12:40,success,0,CS_Login:
,afc0b246dda667297c4a546c5c7db3b241381ed103589acf920f4c681dbedf14,527710e30d5ff9f13f2922a0a4ffa
aeb7d25724587f92224e27d9e6f7abf4618
2017-07-11 19:12:40,success,0,CS_GenerateKeyPair:
```

```
,527710e30d5ff9f13f2922a0a4ffaaeb7d25724587f92224e27d9e6f7abf4618,12ef60bbd62da32a7daf16b2769a5  
57a342ee0ad02f790386340af942d684ace  
2017-07-11 19:12:40,success,0,CS_CloseSession:  
,12ef60bbd62da32a7daf16b2769a557a342ee0ad02f790386340af942d684ace,7ba56613669ef06ac298014ac8b51  
bcff09fe00a0561a16de53ff7ba567d91eb  
2017-07-11 19:12:40,success,0,CS_Finalize:  
,7ba56613669ef06ac298014ac8b51bcff09fe00a0561a16de53ff7ba567d91eb,29bdaa88157935cb3d7962f7cbaf0  
c8311alda7440e34b1a8aee9fcdda6bd360  
File is verified successfully
```

CHAPTER 8: 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**. See ["ctconf" on page 117](#) in the "Command Line Utilities Reference" section of the *ProtectToolkit-C Administration Guide* for more information about this utility.

NOTE The **gCTAdmin** application is a Java-based application. A working Java runtime that supports the Swing user interface must be installed. The screenshots throughout this manual may vary from platform to platform.

When WLD mode is configured, this utility does not operate.

To start **gCTAdmin** using Microsoft Windows, locate the program folder titled **ProtectToolkit C RT** or **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:

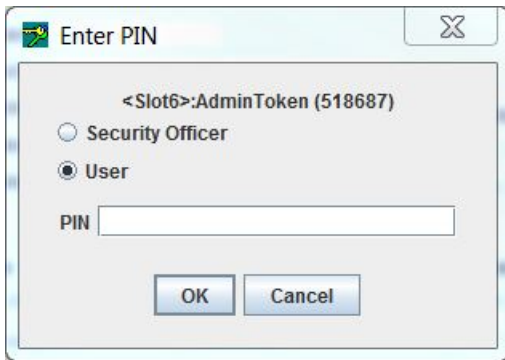
- > ["Logging In and Out" below](#)
- > ["Main gCTAdmin Interface" on the next page](#)
- > ["Slot and Token Management" on page 165](#)
- > ["HSM Management" on page 168](#)

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 14](#) in the *ProtectToolkit-C Administration Guide*. 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.

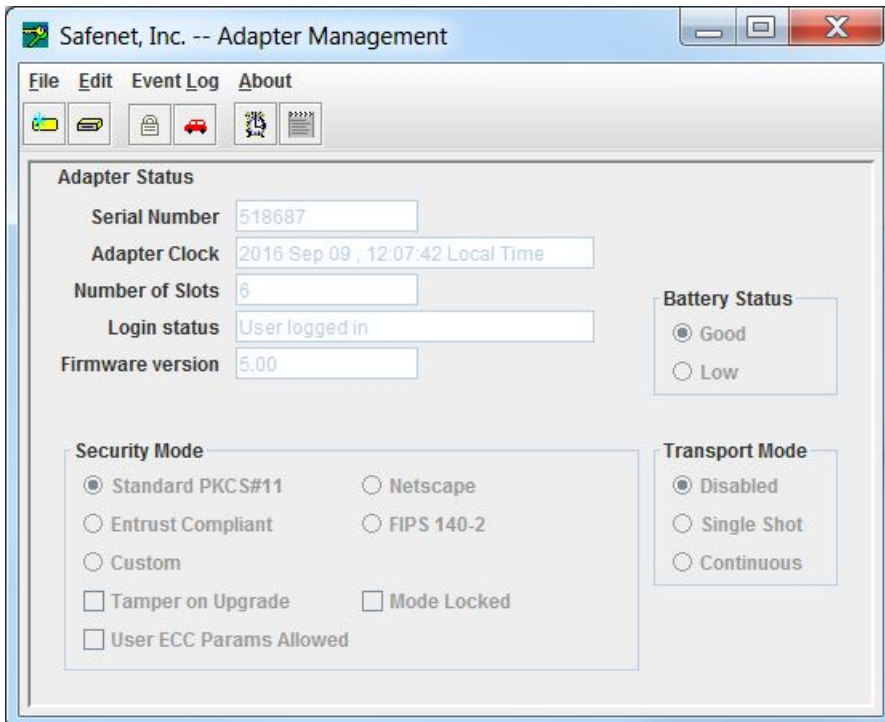
To log out from the main interface, select the **Logout** option from the **File** menu.

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 11: Main gCTAdmin interface



Toolbar Buttons

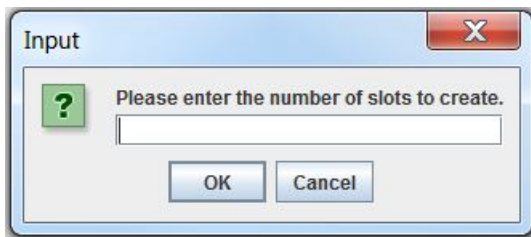
The buttons on the toolbar correspond to the following commands.

| | |
|---|---------------------|
|  | Token Configuration |
|  | Create Slots |
|  | Security Mode |
|  | Transport Mode |
|  | Set the Clock |
|  | 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.



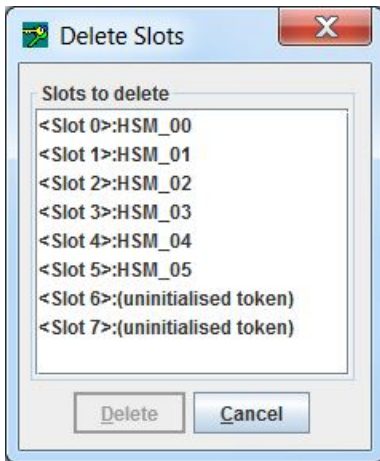
NOTE It is not possible to add slots using **GCTADMIN** while other ProtectToolkit-C applications are running.

Removing Slots

Before removing slots from 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.



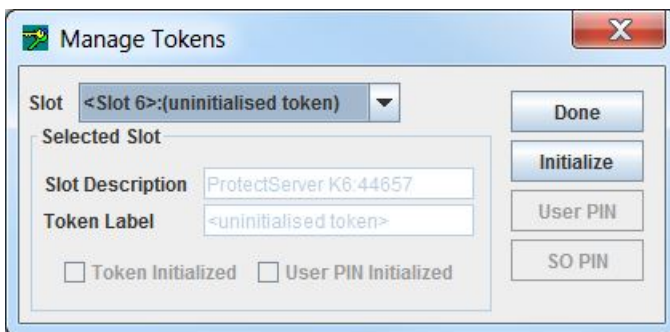
NOTE The slot containing the Admin Token cannot be deleted.

Initializing a Token

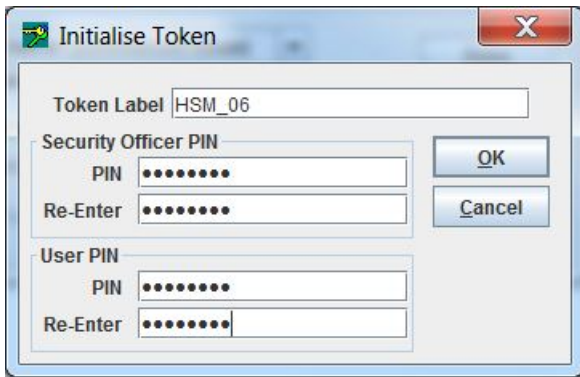
The initialization of a token is performed to set the user and token SO PIN.

To initialize a token

1. 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. User PINs are case-sensitive, and must be 4-32 characters in length.



NOTE PINs have to be entered twice to confirm correct entry.

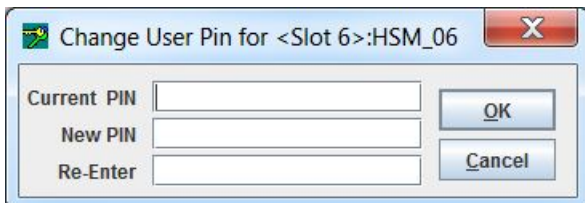
4. Click **Done** to exit the *Manage Tokens* dialog.

Setting the Token User PIN

To set a token user PIN

1. 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. User PINs are case-sensitive, and must be 4-32 characters in length.

If the selected token already has a User PIN assigned, the dialog will prompt for the current and new User PIN to be entered.

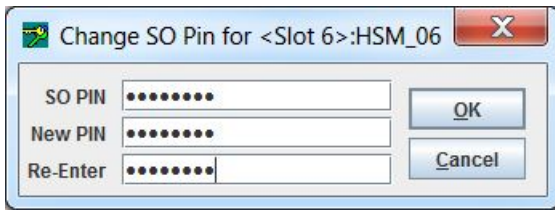


NOTE PINs have to be entered twice to confirm correct entry.

3. Click **Done** to exit the *Manage Tokens* dialog.

Setting the Token SO PIN

1. To set a token SO PIN, select **Edit>Tokens....**
2. 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. User PINs are case-sensitive, and must be 4-32 characters in length.



NOTE Enter PINs twice to confirm correct entry.

3. 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.

To reset a token

1. 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.
2. 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. User PINs are case-sensitive, and must be 4-32 characters in length.

NOTE PINs have to be entered twice to confirm correct entry.

3. Click **Done** to exit the *Manage Tokens* dialog.

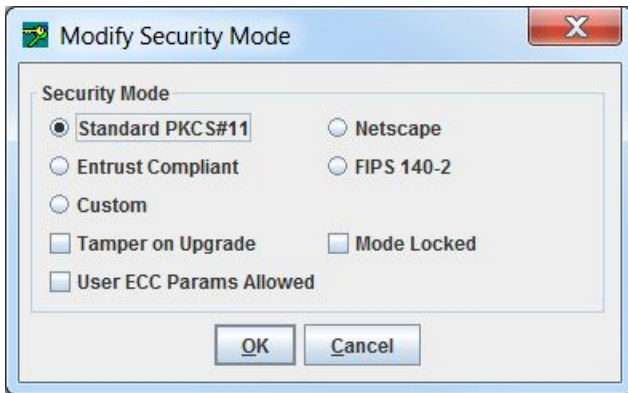
HSM Management

Setting the Security Policy

The most important aspect of ProtectToolkit-C administration is choosing the settings, or *Security Policy*, which will determine how ProtectToolkit-C can be used. The Administrator is strongly advised to read "[Security Policies and User Roles](#)" on page 53 in the *ProtectToolkit-C Administration Guide*, which explains how different settings affect the security and performance of the 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.



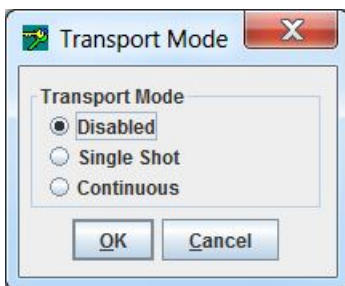
3. Click **OK** to store the selected security policy.

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.

3. Click **OK** to set the Transport Mode.

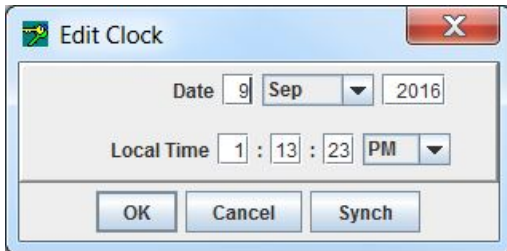
Clock Drift Correction

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

1. Select **Edit> Clock**.

The current value of the HSM clock is displayed.



2. Edit the date and time manually, or synchronize the HSM clock to the host clock (recommended) by clicking **Synch**.

3. Click **OK** to close the dialog box.

Viewing and Purging the System Event Log

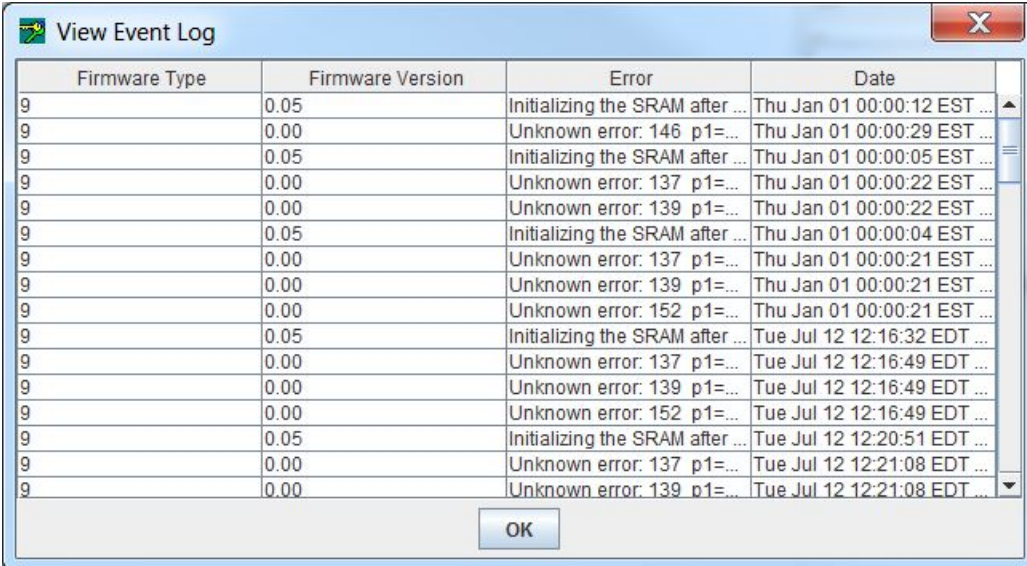
ProtectToolkit-C maintains a system event log as a means of tracking serious hardware or operational faults, tamper events, and self-test error information. For full details on what the event log stores and how to interpret its data, please refer to ["Using the System Event Log" on page 93](#) in the "Operational Tasks" section of the *ProtectToolkit-C Administration Guide*.

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**.

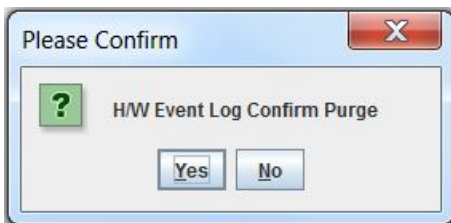
A dialog is shown containing a list of events with columns for "Firmware Type", "Firmware Date", "Error", "Date".



| 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 ... |

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.

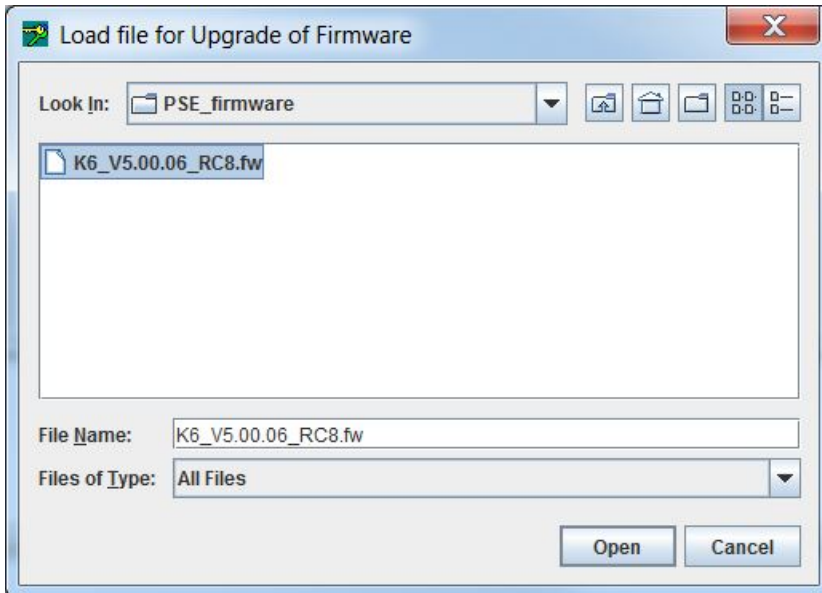
CAUTION! Depending on the active security policy, the HSM might execute a soft tamper before completing the upgrade process. This tamper will erase all key and configuration data on the HSM. See "[Security Policies and User Roles](#)" on page 53 in the *ProtectToolkit-C Administration Guide*.

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.

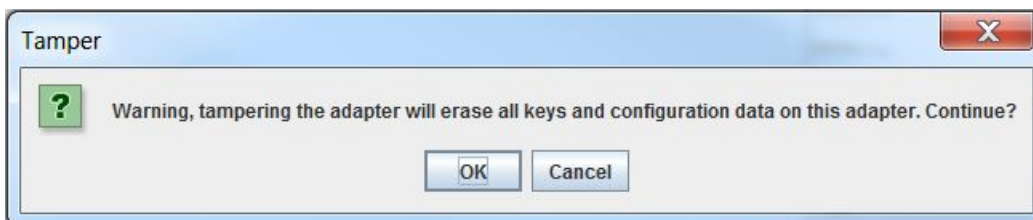
Tampering the HSM

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.



CHAPTER 9: 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 `ctkmu`. See ["ctkmu" on page 129](#) in the "Command Line Utilities Reference" section of the *ProtectToolkit-C Administration Guide* for more information about this utility.

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.

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 37](#) and ["Operation in HA Mode" on page 37](#) in the "Cryptoki Configuration" section of the *ProtectToolkit-C Administration Guide* for more information about these operating modes.

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 176](#)
- > ["Creating Keys" on page 177](#)
 - ["Available Keys" on page 178](#)
 - ["Key Attribute Types" on page 178](#)
 - ["Creating a Random Secret Key" on page 179](#)
 - ["Creating a Random Key Pair" on page 180](#)
 - ["Creating Key Components" on page 182](#)
 - ["Entering a Key from Components" on page 184](#)
- > ["Editing Key Attributes" on page 185](#)
- > ["Deleting a Key" on page 186](#)
- > ["Display Key Check Value" on page 186](#)
- > ["Importing and Exporting Keys" on page 186](#)
- > ["Key Backup Feature Tutorial" on page 193](#)

Compatibility Issues

Using KMU with ProtectToolkit-J

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 ProtectToolkit-J V3 or later. The tokens and keys that are managed with KMU are fully compatible and may be utilized by ProtectToolkit-J. The KMU may also be used to see and manipulate keys that have been created by ProtectToolkit-J. For more information, see [Key Management](#) in the *ProtectToolkit-J Reference Guide*.

Please contact Thales for further details on its SafeNet ProtectToolkit-J products.

Using KMU with ProtectToolkit-C V4.0, V3.x, and V2.x

This version of the KMU is not compatible for use with ProtectToolkit-C version 4.0 or less.

The KMU can read backup files and smart cards created by 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 ProtectToolkit-C v4.0 and V3.x HSMs. So, if you are exporting keys from a system running ProtectToolkit-C 4.1 or above for import to an older system then use the **ctkmu** and the **-3** option.

Please contact Thales for a KMU that is compatible with older versions of this software.

Main KMU Interface

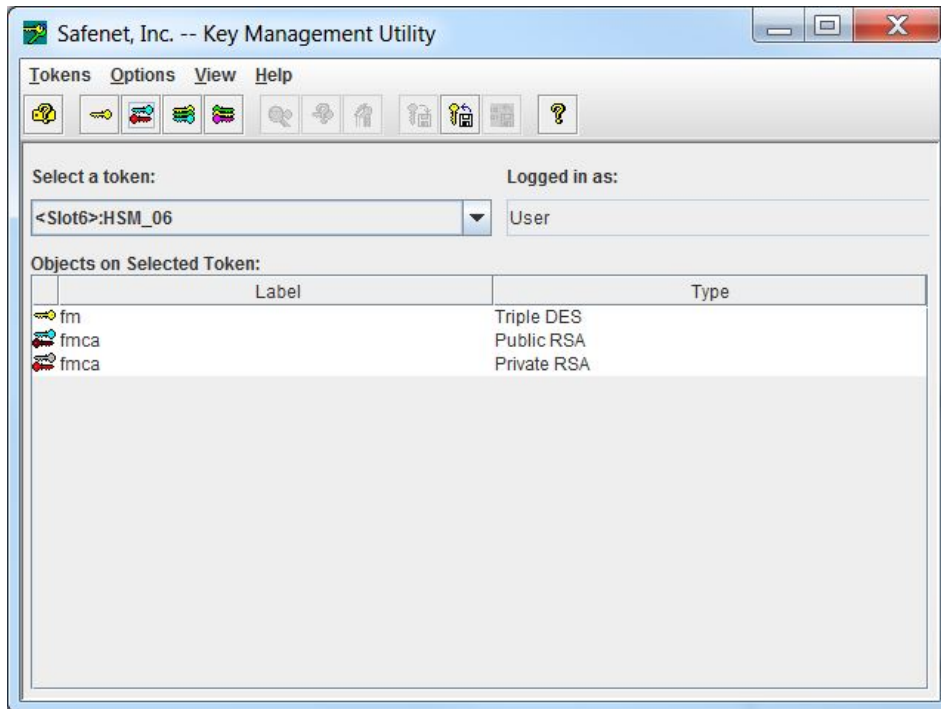
To start the KMU when using Microsoft Windows, locate the relevant program folder in the Windows Start menu and click on the appropriate shortcut. To start the KMU in a UNIX environment, enter **kmu** at the command prompt. To exit the KMU, select **Tokens> Exit** from the menu bar. Selecting **Help** 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 12: 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.

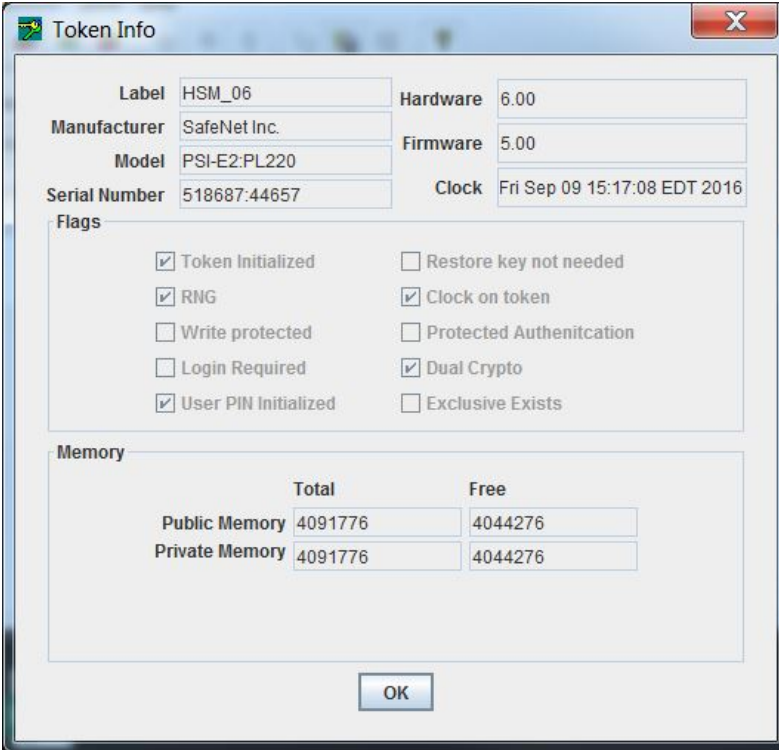
| | | | |
|---|--------------------------|---|---------------------|
|  | Token Info |  | Edit Key Attributes |
|  | Create Random Secret Key |  | Delete Key |
|  | Create Key Pair |  | Import Key |
|  | Create Key Components |  | Export Key |

| | | | |
|---|---------------------------|---|--------------------------|
|  | 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.



| Label | | Hardware | |
|--|--|------------------------------|--|
| HSM_06 | | 6.00 | |
| Manufacturer | | Firmware | |
| SafeNet Inc. | | 5.00 | |
| Model | | Clock | |
| PSI-E2:PL220 | | 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 | |

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 179](#)
- > ["Creating a Random Key Pair" on page 180](#) (RSA public and private keys, for example)
- > ["Creating Key Components" on page 182](#)
- > ["Entering a Key from Components" on page 184](#)

NOTE To refresh the key information displayed on the Main KMU Interface, select **Options> Refresh** from the menu bar. The display 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 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. |

| Attribute | Description |
|--------------------|--|
| 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. |
| 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.



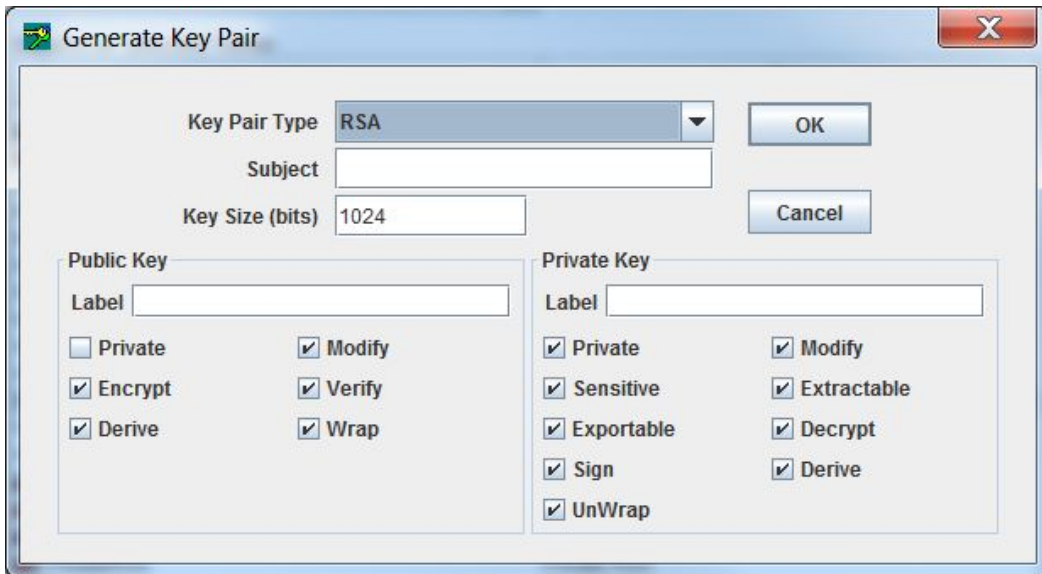
2. Choose the type of key you wish to generate from the **Mechanism** drop-down box. If you are generating an AES, CAST, RC2 or RC4 key, you must specify a Key Size.
3. Enter a label for the key into the Label input field.
4. Select the desired key attributes by checking their boxes. See "[Key Attribute Types](#)" on page 178 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.



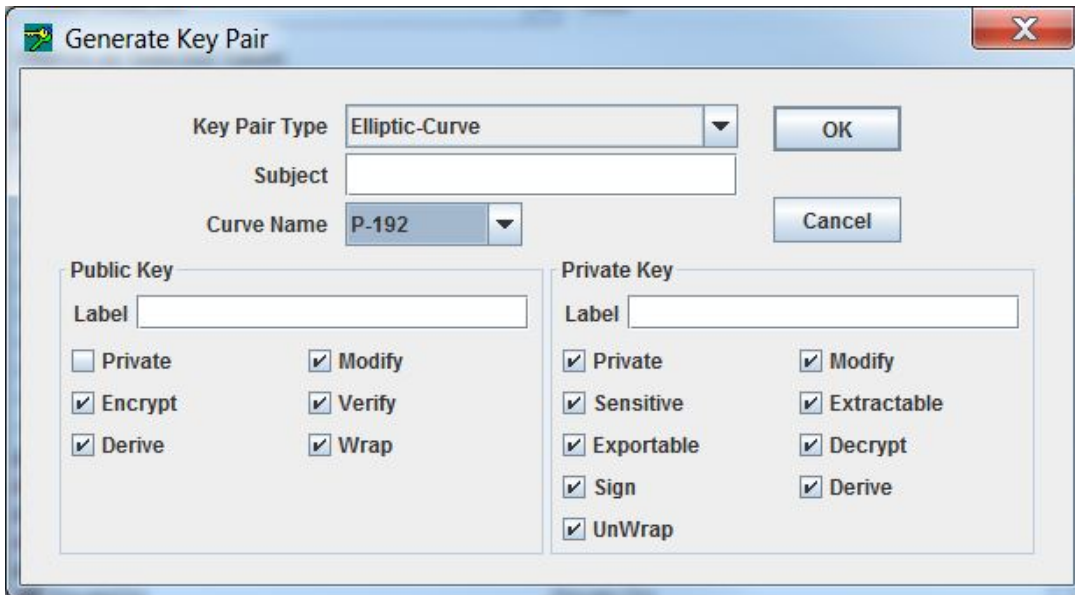
2. Select the type of key pair you wish to generate from the **Key Pair Type** drop-down box.

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**).



NOTE If the FIPS Mode security policy is enabled, the cryptographic operations of RSA, DSA, DH, and EC algorithms are restricted to key sizes within a specified range. For more information about the size limitations of keys that are created or imported in FIPS Mode, see "[FIPS Mode Operational Restrictions](#)" on page 55 in the "Security Policies and User Roles" section of the *ProtectToolkit-C Administration Guide*.

- 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.

- 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.

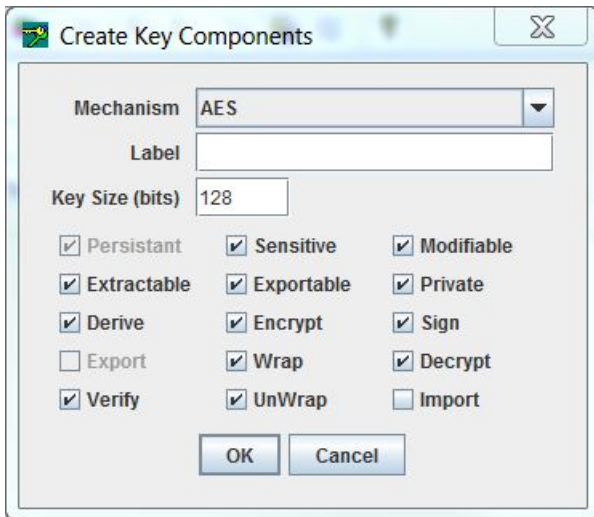
Creating Key Components

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.

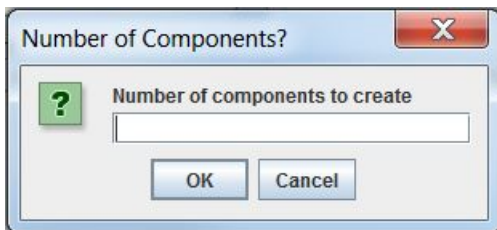
This is useful for the creation and distribution of Key Encryption Keys (KEKs) with multiple custodians. This function makes it possible to create a key whose value is unknown to any single party. Only by combining the components known by each custodian can the key be regenerated. Each component is randomly generated, and in itself does not expose any portion of the final key value.

To create key components

- Select an initialized token from the **Select a Token** drop-down. Log in if necessary.
- Choose **Options > Create > Generate Key Components** from the menu bar, to open the **Create Key Components** dialog box.

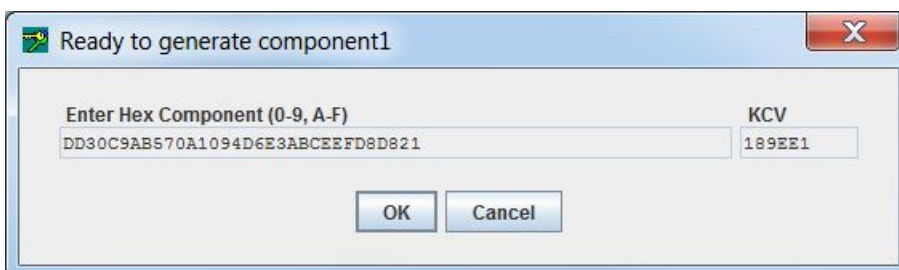


3. Select a key type from the **Mechanism** drop-down list.
4. Enter a label for the key into the **Label** field.
5. For key types AES, CAST, RC2 and RC4, specify the size of the key to be generated in the **Key Size (bits)** field.
6. Decide on the key attributes and click active checkboxes as required.
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

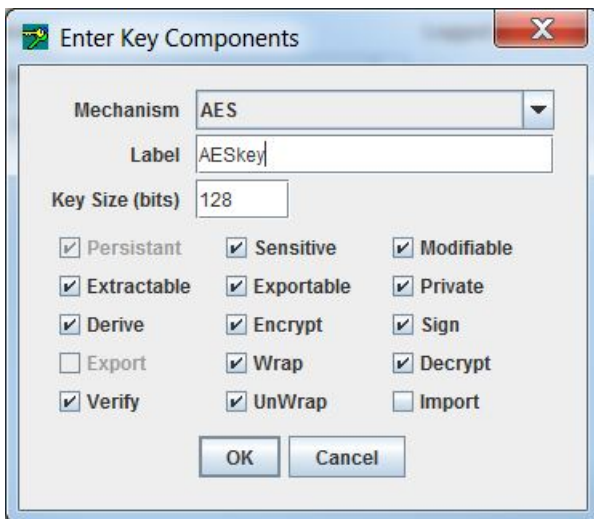
This function allows a key to be entered from one or more components.

To enter a key from components

NOTE The component entry can be masked by selecting **Options> Mask Component Entry** before beginning the operation.

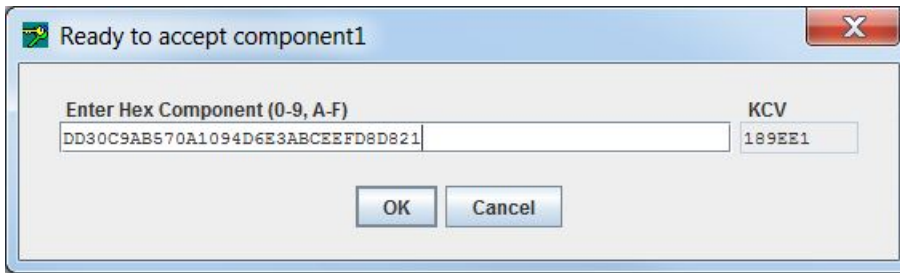
1. Select an initialized token from the **Select a Token** drop-down box and click **Enter Key From Components** on the toolbar. Alternatively, select **Options> Create> Enter Key From Components** from the menu bar.

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 n** 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.

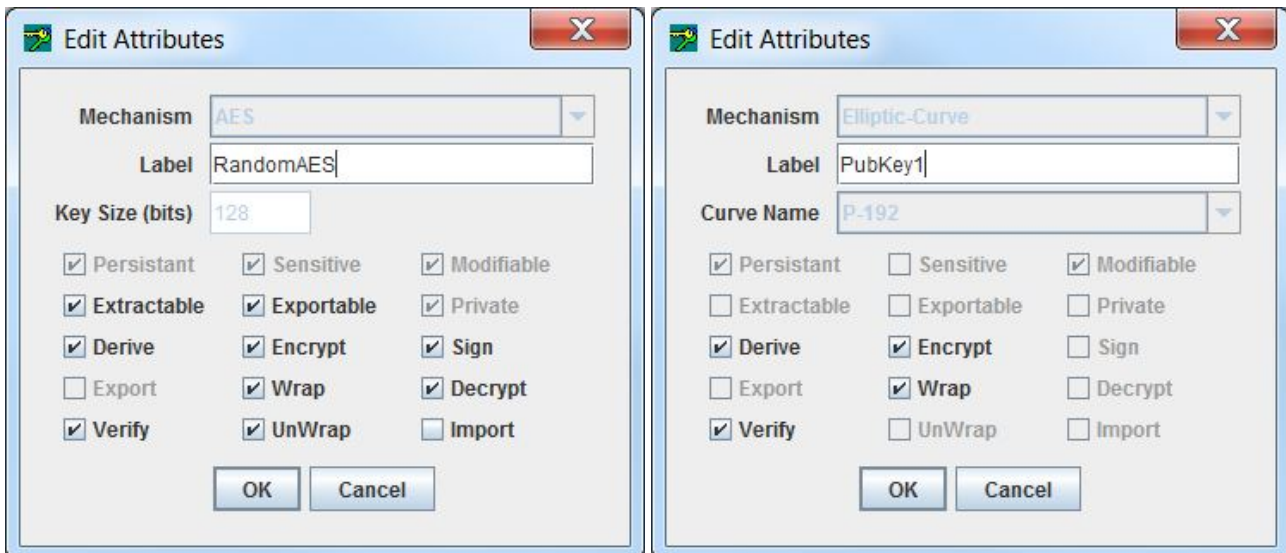
Refer to "[PKCS #11 Attributes](#)" on page 198 in the *ProtectToolkit-C Administration Guide* for details on how the KCV is calculated.

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.



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).

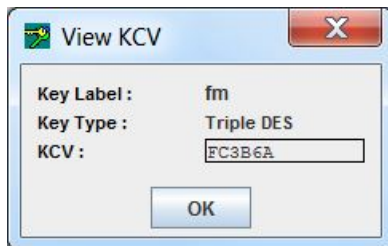
The KCV is a standard technique for obtaining an identification fingerprint from a key. The mechanism used, compatible with AS 2805, is simply the first three hex digits obtained by encrypting binary zeros with the key. Please refer to ["Creating Keys" on page 177](#) for details of KCV generation.

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](#) in the "Operational Tasks" section of the *ProtectToolkit-C Administration Guide* 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 177](#) 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 175](#)), 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 177](#) 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> method | <Random key> |
| <i>Single custodian</i> method | The desired wrapping key. This key is used to encrypt the key (or set of keys) to be exported |

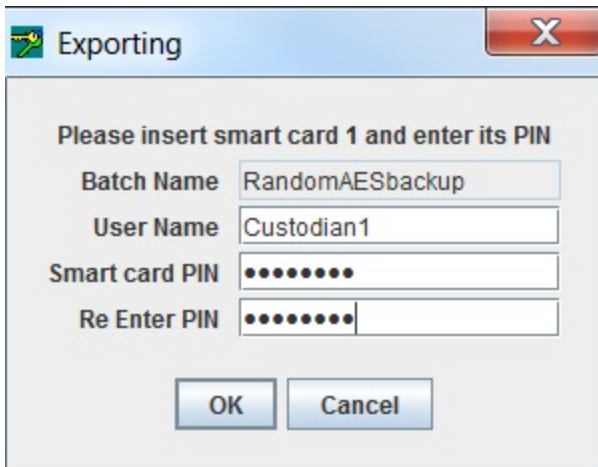
5. In the **Options** area, make further selections as appropriate for the backup and recovery method and destination backup media to be used.

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

1. In the **Options** area, select **Write to smart card(s)**.
2. 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).
3. 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.
4. 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.
5. 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 186](#)).



6. Insert a smart card in the smart card reader.
7. 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**.
8. 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, a prompt will display to enable re-entry. When logging in to a smart card, the card is locked after 7 consecutive incorrect PIN attempts. You must re-initialize the card to set a new PIN.
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.
9. 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.
10. 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.

1. In the Options area, select **Write to selected file**.
2. 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.
3. 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 ["Secure Key Backup and Restoration" on page 81](#) in the "Operational Tasks" section of the *ProtectToolkit-C Administration Guide* for background information on backup and recovery methods, key splitting schemes and key attributes.

NOTE If the FIPS Mode security policy is enabled, the cryptographic operations of RSA, DSA, DH, and EC algorithms are restricted to key sizes within a specified range. For more information about the size limitations of keys that are created or imported in FIPS Mode, see ["FIPS Mode Operational Restrictions" on page 55](#) in the "Security Policies and User Roles" section of the *ProtectToolkit-C Administration Guide*.

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.

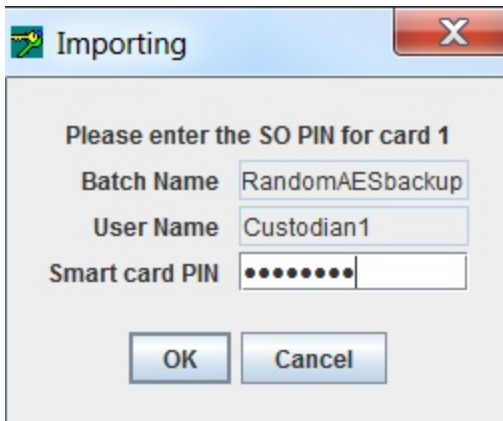
2. In the Options area, choose either **Read from smart card(s)**, **Read from selected file**, or **Import encrypted parts**, depending on the media that was used to store the key(s).

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. When logging in to a smart card, the card is locked after 7 consecutive incorrect PIN attempts. You must re-initialize the card to set a new PIN.

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" on the next page](#)
- > ["Creation of Encrypted Key Set to Backup \(Payload\)" on the next page](#)
- > ["Backup to File" on the next page](#)
- > ["Backup to Smart Card - Single Custodian Mode" on page 195](#)
- > ["Backup to Smart Card - Multiple Custodian Mode" on page 196](#)

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.

NOTE When logging in to a smart card, the card is locked after 7 consecutive incorrect PIN attempts. You must re-initialize the card to set a new PIN.

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.

NOTE The KMU application does not support using DES3 keys to make backups. You must use the **ctkmu** command-line application. Include the **-3** option to specify DES3. For example:

```
ctkmu x -s0 -w des3key -3 backup.bin
```

See ["ctkmu" on page 129](#) in the "Command Line Utilities Reference" section of the *ProtectToolkit-C Administration Guide* for complete command syntax.

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:

$$p = Nl_1w_1[l_2w_2[l_3w_3[\dots l_Nw_N]]]$$
 where N = Number of keys/objects in the payload, l_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, $m = M_{mK}(p)$.

Backup 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 backup 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 | <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 last custodian 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 last custodian of the backup set.</p> |

9. Delete mK, tK and wK.

CHAPTER 10: 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 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> <p>NOTE Although ProtectToolkit-C does not require this attribute to be unique, various other tools may.</p> |
| 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 ProtectToolkit-C. For example:</p> <ul style="list-style-type: none">> CKK_AES, CKK_DES, CKK_DES2, CKK_DES3, CKK_RSA, CKK_DSA, CKK_BIP32> 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> |

| Attribute | Description |
|------------------------------------|---|
| CKA_IMPORT | 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 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. For more information about setting these attributes to back up keys, see "Secure Key Backup and Restoration" on page 81 in the "Operational Tasks" section of the <i>ProtectToolkit-C Administration Guide</i> . |

CHAPTER 11: KMU Key Check Value (KCV) Calculation

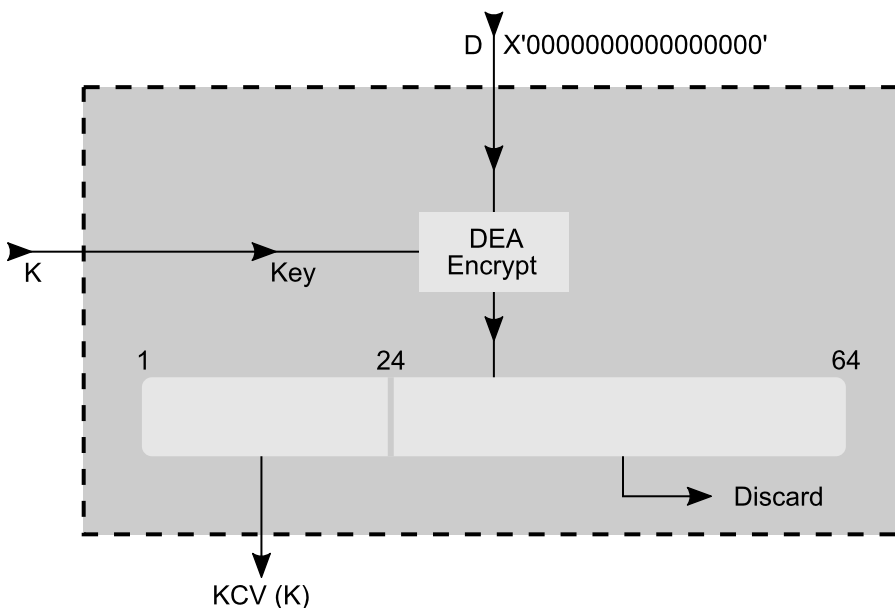
The Key Management Utility calculates and displays keys according to AS 2805.6.3.

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 13: Single-length Key Check Value KCV(K).

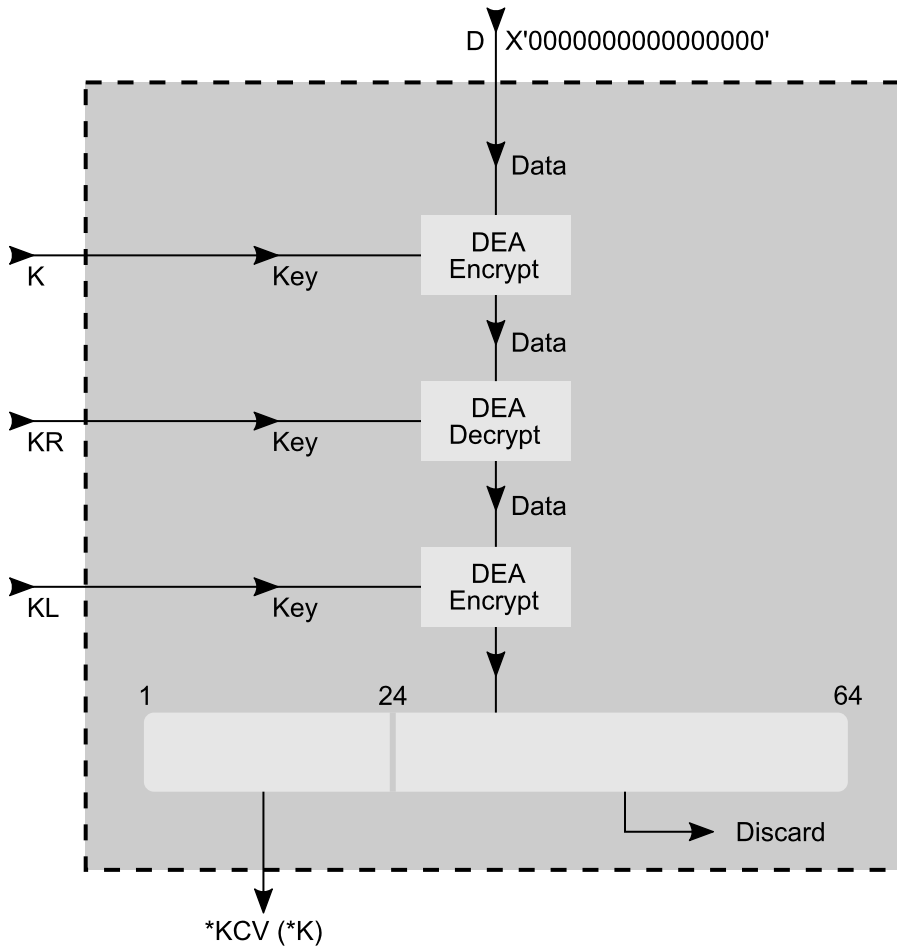


Double-length Key KCV

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\(*K\)](#)" on the next page).

Figure 14: Double-length Key Check Value *KCV(*K)



APPENDIX A: Key Migration from ProtectToolkit-C V4.1

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 **ctkmu** or **KMU** to determine what mechanism should be applied to import that keys.

The **ctkmu** and **KMU** will by default produce backup images using the new mechanisms. The **ctkmu** 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 B: Sample EC Domain Parameter Files

This Appendix describes the domain parameters of two EC curves.

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

```

Glossary

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"