ProtectToolkit-C

Administration Guide

SafeNet
THE DATA PROTECTION COMPANY

Part Number 007-008393-006
Version 5.1

Trademarks

Disclaimer

SafeNet makes no representations or warranties with respect to the contents of this document and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Furthermore, SafeNet reserves the right to revise this publication and to make changes from time to time in the content hereof without the obligation upon SafeNet to notify any person or organization of any such revisions or changes.

We have attempted to make these documents complete, accurate, and useful, but we cannot guarantee them to be perfect. When we discover errors or omissions, or they are brought to our attention, we endeavor to correct them in succeeding releases of the product.

SafeNet invites constructive comments on the contents of this document. Send your comments, together with your personal and/or company details to the address below:

SafeNet, Inc.
4690 Millennium Drive
Belcamp, Maryland USA 21017

Technical Support

If you encounter a problem while installing, registering or operating this product, please make sure that you have read the documentation. If you cannot resolve the issue, please contact your supplier or SafeNet support. SafeNet 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 SafeNet and your organization. Please consult this support plan for further information about your entitlements, including the hours when telephone support is available to you.

Contact method	Contact	
Address	SafeNet, Inc. 4690 Millennium Drive Belcamp, Maryland 21017 USA	
Phone	Global	+1 410-931-7520
	Australia	1800.020.183
	China	(86) 10 8851 9191
	France	0825 341000
	Germany	01803 7246269
	India	000.800.100.4290
	Netherlands	0800.022.2996
	New Zealand	0800.440.359
	Portugal	800.1302.029
	Singapore	800.863.499

	Spain	900.938.717
	Sweden	020.791.028
	Switzerland	0800.564.849
	United Kingdom	0800.056.3158
	United States	(800) 545-6608
Web	www.safenet-inc.com	
Support and Downloads	www.safenet-inc.com/support Provides access to the SafeNet Knowledge Base and quick downloads for various products.	
Technical Support Customer Portal	https://serviceportal.safenet-inc.com Existing customers with a Technical Support Customer Portal account can log in to manage incidents, get the latest software upgrades, and access the SafeNet Knowledge Base.	

Revision History

Revision	Date	Reason
A	14 August 2015	Release 5.1
B	17 August 2015	Updated support contacts table.

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1
INTRODUCTION

ProtectToolkit C is a cryptographic service provider that implements the PKCS #11 application programming interface (API) as specified by RSA labs, and 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 and verification as well as permanent key storage. ProtectToolkit C is compliant with PKCS #11 V 2.10.

To provide the highest level of security, ProtectToolkit C interfaces directly to an intelligent cryptographic Hardware Security Module (HSM). The HSM includes high-speed DES and RSA hardware acceleration as well as generic security processing. Secure key storage is also included in form of persistent, tamper resistant CMOS storage. In addition, multiple adapters may be used in a single host computer in order to improve throughput or to provide redundancy.

The environment under which an application can make requests for Cryptoki processing is herein referred to as the ProtectToolkit C Runtime.

Who Should Read This Manual?

This manual is primarily intended for the ProtectToolkit C *Administrator*. This person is responsible for performing administrative tasks such as specifying the configuration, security policy and number of applications (or *users*) of ProtectToolkit C. This configuration of ProtectToolkit C will determine the type of functionality and/or services that will be available to the ProtectToolkit C applications. The Administrator is strongly encouraged to thoroughly read this manual before attempting any operations.

The normal ProtectToolkit C *User* may gain additional insight into the structure and features of the ProtectToolkit C product by reading its contents, and hence this document may serve as a valuable reference for any user.

This manual also provides the details for configuration of some standard PKCS #11 applications, which are compatible with the ProtectToolkit C product.

Chapter Overview

Chapter 2 describes how the ProtectToolkit C system is initially set up. The System Administrator or person responsible for the installation process should read this chapter, in addition to the ProtectToolkit C Installation Guide.

Chapter 3 describes the ProtectToolkit C application model and then discusses the initial configuration of the system. This chapter is primarily intended for the *Administrator*; however an understanding of the concepts may be useful to the normal *user*.

Chapter 4 discusses the various operational aspects of the system. This chapter is applicable to both the *Administrator* and normal *user*.

Chapter 5 discusses different aspects that administrators must consider when selecting and setting a security policy for the ProtectToolkit C environment. Changes to security policy changes affect users, and in some cases, also depend upon the roles to which they have been assigned.

Chapter 6 describes some of the most common operational procedures a User, Administrator or Security Officer may perform during normal ProtectToolkit C operation.

Chapter 7 contains reference sections for the command line utilities provided with ProtectToolkit C. This chapter is applicable to both the *Administrator* and normal *user*.

Chapter 8 contains a reference to the graphical user interface (GUI) utilities provided with ProtectToolkit C. This chapter is applicable to both the *Administrator* and normal *user*.

Appendix A contains a reference to the Event Log types.

Appendix B contains a reference to PKCS #11 attributes.

Appendix C contains a reference to the KMU key management utility.

Appendix D contains a reference to Key Migration from ProtectToolkit C V4.1.

Appendix E contains a reference to External Key Storage.

Appendix F contains samples of EC Domain Parameter files.

Further Documentation

SafeNet Manuals

In addition to this administration manual the following manuals may contain relevant information. They are referenced in this manual when applicable.

Software

- ProtectToolkit C Installation Guide
- ProtectToolkit C Programming Guide
- ProtectSerrver HSM Access Provider Installation Guide

Hardware

- ProtectServer Internal Express 2 (PSI-E2) Installation Guide
- ProtectServer External 2 (PSE2) Installation Guide

SafeNet Application Integration Guides

A number of integration guides are available that outline the use of SafeNet products with third party applications. To obtain further information, contact your local SafeNet representative.

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 for further details.

CHAPTER 2
CONFIGURATION ITEMS

Overview

Configuration items are created and maintained on the host operating system (platform) where ProtectToolkit C is installed to store ProtectToolkit C configuration information.

This chapter covers configuration items in detail as it is important that their characteristics and use be well understood if ProtectToolkit C is to be setup and configured successfully.

In some cases ProtectToolkit C will automatically create configuration items. In most cases though, if a change controlled by a configuration item is to be made to ProtectToolkit C configuration, then that configuration item must be manually created and set to the value required using the information contained in this section.

Configuration items may exist at any one of four configuration levels. When a configuration item is queried, four locations corresponding to these levels are searched in order of precedence. This is explained in more detail below.

The four levels, in order of precedence, are:

- temporary configuration
- user configuration
- system configuration and
- default configuration

Default configuration items cannot be changed, however changes to configuration items can be made at the system, user or temporary levels and these changes will override the corresponding values at the default configuration level.

Any entries made at the temporary configuration level override any corresponding entries at the user or system levels and any entries at the user configuration level will override corresponding entries at the system level.

The exact nature and location of these configuration areas is platform specific. On Windows systems, user and system configuration information is stored in the Registry. On UNIX-based systems, configuration files are used. Temporary configuration items are implemented using environment variables on both Windows and UNIX-based platforms.

Regardless of the platform used a common convention has been followed to name configuration items. Understanding this naming convention will assist you to locate and change the appropriate configuration items when required.

Configuration items are hierarchical in structure, with the root node always being "ET". Child nodes of the root represent the class of the item, and are typically product abbreviations, such as "PTKC" (ProtectToolkit C) or "HSM" (Hardware Security Module). Nodes under class represent the component, such as "LOGGER" or "SMS". Finally, nodes under component represent the configuration item, such as "FILE" or "MODE". Putting it all together, configuration items are of the form:

ET_<class>_<component>_<item>

Platform-specific Details

Windows

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

User Configuration is the registry tree starting from *HKEY_CURRENT_USER\SOFTWARE\SafeNet*.

System Configuration is the registry tree starting from *HKEY_LOCAL_MACHINE\SOFTWARE\SafeNet*.

The User and System Configuration registry trees have a corresponding key for the class and component nodes. Entries contained in the component node key are strings whose names are of the form: *ET_<class>_<component>_<item>*.

Example

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

ET_PTKC_LOGGER_ FILE

This is located in the key:

HKEY_LOCAL_MACHINE\SOFTWARE\SafeNet\PTKC\LOGGER

UNIX

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

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

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

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

Example

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

ET_PTKC_LOGGER_FILE=/ctlog.log

CHAPTER 3
OPERATING MODE SETUP

Operating Mode Setup Overview

ProtectToolkit C can be used in any one of three operating modes. These are:

- **PCI Mode** in conjunction with a compatible SafeNet Hardware Security Module (HSM) such as the *ProtectServer PSI-E2* installed locally
- **Network Mode** over a TCP/IP network, in conjunction with a compatible SafeNet HSM such as the *PSE2*
- **Software-Only Mode**, on a local machine without access to a hardware security module, for development and testing purposes

The setup steps for each of these modes are summarized below with references to further information in other manuals or this chapter as applicable.

Once operating mode setup is complete proceed with Cryptoki configuration. This is covered in the next chapter.

PCI and Network Operating Modes

1 Install the hardware.

See the installation manuals provided with the hardware for further information.

2 Install and configure access provider software.

Access provider software must be installed and configured to support the operating mode required. For full details see the *ProtectServer HSM: Access Provider Installation Guide.*

It is not necessary to install access provider software when ProtectToolkit C is used in software-only mode for development and testing purposes.

3 Install ProtectToolkit C.

Install ProtectToolkit C on your computer system. Please refer to the *ProtectToolkit C Installation Guide* for further details.

4 Configure the secure messaging system (SMS).

Refer to the *Secure Messaging Overview, Messaging Mode Configuration,* and *Configuring Session Protection* sections in this chapter for further information.

5 Establish network communication (Network Operating Mode only).

In order to establish network communication the client must be configured to use one or more servers that are available on the same network. Refer to the *Specifying the Network Server(s)* section in this chapter for further information.

Software-only Mode

1 Install the ProtectToolkit C software development kit (SDK).

Install the ProtectToolkit C Software Development Kit (SDK) on your computer system. Refer to the *ProtectToolkit C Installation Guide* for further details.

2 Make configuration changes, if required.

Further changes may be made to customize the installation and optimize its performance. Refer to the *Software Only Mode Configuration* section in this chapter for further information if required.

Secure Messaging Overview

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

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

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

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

To configure and enable an SMS complete the following steps.

1 Configure secure messaging mode.

You may need to change the session key rollover default configuration. See the section *Configuring Session Key Rollover* below 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* below.

Messaging Mode Configuration

Configuring Session Key Rollover

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

Two mechanisms can be used to trigger session key rollover.

- The first mechanism triggers session key rollover once a preset number of blocks have been encrypted or decrypted by the application.
- 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 as follows.

Preset Number of Blocks Trigger

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

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

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

This change can be made at the temporary, user or system levels on both UNIX and Windows platforms. Refer to the Configuration Items chapter for further details on how to go about this if required.

Preset Number of Hours Trigger

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

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

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

This change can be made at the temporary, user or system levels on both UNIX and Windows platforms. Refer to the Configuration Items chapter 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 “administrators 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.

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 for full details on security policies, setting flags and the use of this command.

SMPR Security Flags

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

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

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

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

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. Replace `<device>` with the number of the HSM required.

2. Create the following configuration item:

   `ET_PTKC_<serial>_SMPR`

   Replace `<serial>` with the serial number of the HSM found in step 1.

   This change can be made at the temporary, user or system levels on both UNIX and Windows platforms. Refer to the Configuration Items chapter 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 using one or more of the *Configuration Item Value* letters given in the table above. For example, if both *Auth Protection* and *Auth Replies* are required assign the value `SR`.

Specifying the Network Server(s)

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

- Server Name = 127.0.0.1
- Server Port = 12396

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

The full syntax for the ET_HSM_NETCLIENT_SERVERLIST configuration item is:

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

UNIX Example

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

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

Windows Example

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

- Locate the registry key:

  HKEY_LOCAL_MACHINE\SOFTWARE\SafeNet\HSM\NETCLIENT
- Assign the value *ptkc.mydomain.com* to the entry:

  ET_HSM_NETCLIENT_SERVERLIST

Software-only Mode Configuration

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

Storage Location Assignment

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

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

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

This change can be made at the temporary, user or system levels on both UNIX and Windows platforms. Refer to the Configuration Items chapter for further details on how to go about this if required.

Fixing Command Line Utility Low Performance

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

```
ET_PTKC_SW_DETECTPERIPHERALS
```

This change can be made at the temporary, user or system levels on both UNIX and Windows platforms. Refer to the Configuration Items chapter for further details on how to go about this if required.

Enabling Smart Card Access under UNIX

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

CHAPTER 4
CRYPTOKI CONFIGURATION

Introduction

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.

The ProtectToolkit C Model

The model for ProtectToolkit C is based on standard PKCS #11 processing as illustrated in the following diagram for ProtectToolkit C running in hardware mode. It demonstrates how an application communicates its requests to a token via the PKCS #11 interface.

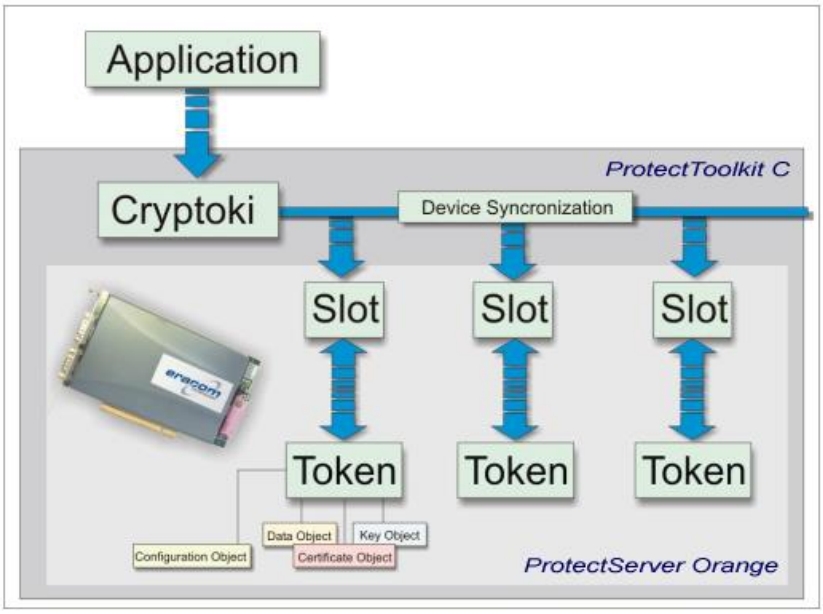


Figure 1 - ProtectToolkit C Model

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 end-user 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. You can add more slots, as required, for the local configuration. There are practical limits as to how many slots you can create. 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 external serial ports on the HSM. The smart card tokens can be used for storage of data objects. Their primary purpose is for key backup and key restoration. In order to protect objects stored on the token from un-authenticated 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. Initially, smart card tokens are blank and require initialization before they can be used. The storage format and layout of files on the tokens is proprietary and can store a maximum of 5 objects (up to the storage capacity of the actual token). Objects may be deleted; however, the storage allocated to the object is not reclaimed until the token is re-initialized by the Security Officer or Administrator.

The Admin Slot

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

The Admin slot holds the *Admin token* and it is on this token that the administration objects reside (See the discussion on administration objects on the next page).

PKCS #11 Objects

As shown in Figure 1, each token may contain a number of *objects*. The PKCS #11 standard allows for different types of objects which are classified as follows:

- Data objects, which are defined by an application
- Certificate objects, which represent digital certificates such as X.509, for example
- 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 please consult PKCS #11 Attributes.

There is a practical limit to the number of objects that can be stored in a token. It is recommended that the number of tokens stored in any single token be less than 1000, and that the number of tokens 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. The administration objects 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) and
- 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.

It should be noted that the services available to the various roles are highly dependent upon 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.

PINs and Passwords

In general both PINs and passwords are used to authenticate users and to provide access to secured computer systems. Most commonly PINs are defined as 4 digit numbers in the range 0000-9999 while passwords may be alphanumeric and of varying length.

NOTE: These common definitions have not been adopted in the *PKCS #11 V2.10 Cryptographic Token Interface (Cryptoki)* standard, and as ProtectToolkit C implements this standard, they do not apply when using this product.

PINs, as defined in the standard (and as implemented for ProtectToolkit C), are variable length strings of characters selected from the ANSI C character set. In ProtectToolkit C PINs:

- are case sensitive
- must be between 1 and 32 characters in length

The term *password* is not defined as something distinct from a PIN in Cryptoki environments. You will find the term used from time to time in Cryptoki related documentation instead of "PIN", in line with common usage.

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 PSI-E2 HSM.

After the third failed PIN presentation, the device imposes a delay (incrementing in multiples of 5 seconds) until the next presented PIN is checked.

For example, after the third failed attempt, the device imposes a delay of 1*5 seconds, after the fourth the delay is 2*5=10 seconds, after the fifth, the delay is 3*5=15 seconds, and so on.

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

Preparation

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 discussed previously in this manual

Setting the Admin Token PINs

Following an initial installation or after a 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.

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

  A prompt displays for the Administration Security Officer (ASO) PIN.

- Enter the ASO PIN and press *Enter*. Then, when prompted, enter this 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 above.

  A prompt displays for the Administrator PIN.

- Enter the Administrator PIN and press *Enter*. Then enter this PIN again when prompted for confirmation.

- Onboard each HSM is a Real Time Clock (RTC). If the RTC is out of synchronization with the host system clock, a prompt is displayed to allow synchronization of the clock. 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 display. For example:

```
Current Adapter Configuration for Device 0:

Model              : 8000:PL450
Batch              : 1809
Manufacturing date: 11/11/2004 10:59:59
Serial Number      : 3262
Adapter Clock      : 20/12/2004 09:22:33 (+10:00)
Board Revision     : G
Firmware Version   : 1.40.00
Cprov Version      : 3.22
Hardware Status    : BATTERY OK  PCB v0  FPGA v0  EXT PINS 0
Free Memory        : 11288416
SM Size Free/Total: 1022528/1046528
Security Mode      : Default (No flags set)
Transport Mode     : None
FM Support         : Enabled
FM Status          : "HKIMM-MA V2.16" is active.
Open Session Count: 0
Number of Slots    : 1
RTC Access Control: Enabled (SEC=10 COUNT=1500 DAYS=1)
```

  Following this, the following message displays:

```
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')
```

An *FM* is a *functionality module*. For more information see Installing a Functionality Module in the Operational Tasks chapter.

  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 from a security perspective. Implementing the security policy is, without doubt, the most important aspect of ProtectToolkit C initial configuration.

A number of security settings offered as a part of 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 for full details and implementation instructions.

Setting up Slots

The Administrator will have to decide on the number of slots required for their 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 separated from each other.

For further information on the type of slots and tokens please refer to the Slots and Tokens section in the Cryptoki Configuration chapter.

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

Example:

```
ctconf –c2
```

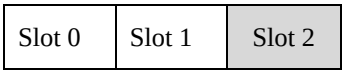
Since only the Administrator is authorized to create new slots, the Administrator PIN will be prompted for.

The previous 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.

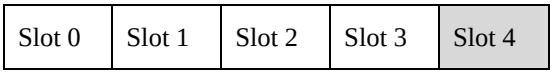
Example:

If current configuration were as follows:



where slot 0 and 1 are user slots and 2 being the Admin slot.

If two slots were added, the configuration would look as follows:



where slot 0, 1, 2 and 3 are user slots and 4 now being the Admin slot.

Multiple Adapter HSMs

When multiple adapter HSMs (such as the PSI-E2) are installed in a single machine there will be multiple Admin slots, one per HSM. In this situation 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.

Token Initialization

Following the creation of a slot within ProtectToolkit C, the next task is to *initialize the token* within that slot 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 question of who is responsible for token initialization is dependent on the Security Policy that has been set for the adapter. In the case where 'clear PINs' are allowed, any user deciding to take 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 please see

Security Policies and User Roles.

To initialize a token on a particular slot, as the Administrator, the **ctconf** utility is used. Once a token is initialized it may only be re-initialized, 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 a prompt for the token SO PIN to be entered.

NOTE: A token initialization will destroy all objects on that token. This is an important consideration when re-initializing a token that has already been used.

Following initialization of a token, the token SO should change the PIN set by the Administrator. This can be achieved with the **ctkmu** utility. A description of how to change the SO PIN is given in Chapter 6.

The next task for the token SO is to initialize the token user PIN. To do this the utility **ctkmu** is provided.

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. It is advisable, however, for the User to now change his PIN from the one the SO assigned to him. The User can achieve this by repeating the above command.

Trust Management

Trust management comes into play where a need exists to transfer secure data or keys from one HSM to another HSM through the process of token replication. Environments where Work Load Distribution (WLD) along with High levels of availability (HA) are used, are an example of such a system. Refer to section entitled Work Load Distribution Model (WLD) and High Availability (HA) or further details.

Currently, trust management is supported on the Protect Server, and Protect Server – External HSMs.

When a WLD system is configured, it is necessary to replicate tokens across those HSM User slots that are associated with a common WLD virtual slot. For the HSM that imports the token, it is essential that the token is deemed trustworthy before it is utilized; in other words, the token must not have been altered during transmission and the token was imported from a trustworthy source. For the HSM that exports the token, it is essential that the HSM that imports the token is also deemed trustworthy.

Public key cryptography is used to establish 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. A 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. A HSM trusts another HSM (the peer HSM) when the HSM holds the HSM Identity Public-Key of the peer in its administrative token. This is referred to as the **peer** HSM Identity Public-Key.

Figure 2 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 preformed between HSM B and HSM C (with either device originating the tokens) as token replication requires a relationship of mutual trust between HSMs.

Figure 2 - Simple trust relationships

Figure 3 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.

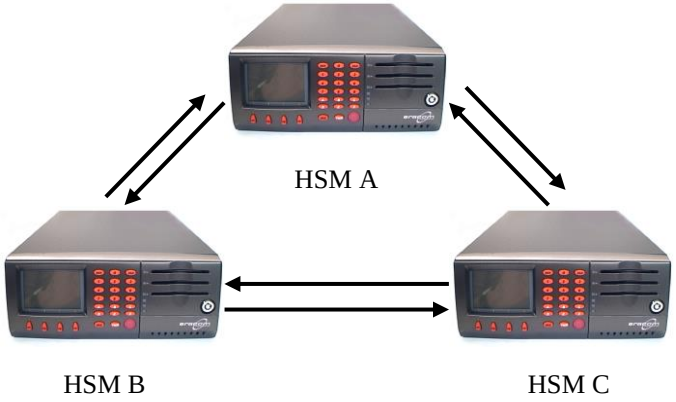


Figure 3 - Relationships of mutual trust

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

Figure 4 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.

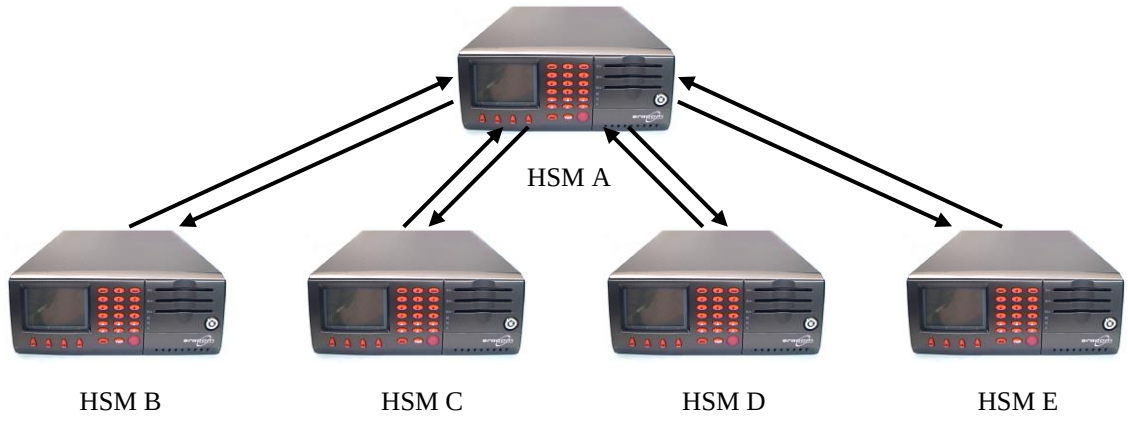


Figure 4 - Trust relationships in a typical WLD/HA configuration

Complex trust topologies can be configured depending upon system and administrative requirements. ***Figure 5*** illustrates an example of a complex trust topology.

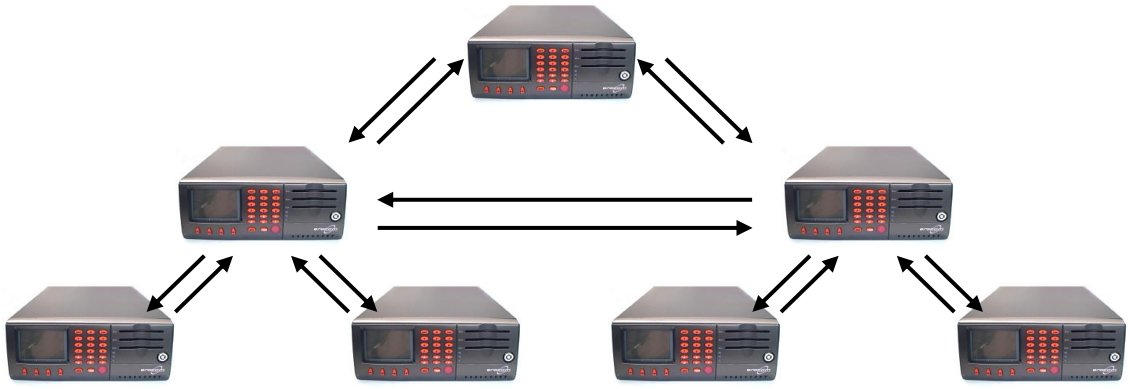


Figure 5 - Complex trust topology

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

The following example shows how to establish trust among HSMs. The *ctident* utility may be used to display trust relationships, check trust relationships and remove trust relationships. It may also be used to rollover the HSM identity keys that are used in trust management.

Establishing Trust Relationships

The following example describes how to set-up the trust relationships illustrated in ***Figure 6*** . 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

Configuration

1. **Generate a list of all the slots on the system to establish the serial number of the admin token**. The *ctkmu* utility may be utilized for this. The slots for each device on the system are assigned in the following order: User slots, Smart card slots, Administration slot. The serial number of the admin token is listed in brackets after the words AdminToken. In the example below, the serial number of the Administration token for the first device is 1197. For example:

```
C:\>ctkmu l
ProtectToolkit C Key Management Utility $Revision: 3.10.2.1 $
Copyright (c) SafeNet, Inc. 2006

Cryptoki Version  = 2.10
Manufacturer      = SafeNet, Inc.
WLD_Slot_11                     (Slot 0)
WLD_Slot_22                     (Slot 1)
WLD_Slot_33                     (Slot 2)
AdminToken (1197)               (Slot 3)
<uninitialised token>           (Slot 4)
<uninitialised token>           (Slot 5)
<uninitialised token>           (Slot 6)
AdminToken (1111)               (Slot 7)
<uninitialised token>           (Slot 8)
<uninitialised token>           (Slot 9)
<uninitialised token>           (Slot 10)
AdminToken (1310)               (Slot 11)
```

2. **Generate the HSM Identity Key-Pair on each device.** It is necessary to generate a HSM Identity Key-Pair on each device participating in a trust relationship. To achieve this, the *ctident* utility with the *gen* command and appropriate command line parameters is used. The Administration Token SO pin is required to be entered for each device when using this command. In a system where all the HSMs are to participate in a trust relationship, the *ctident gen* command can be utilized with the *all* parameter. Alternatively, only those devices on the system that are participating in token replication can be specified by their serial number as illustrated below.

   Example:

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

   The *ctident gen* command also permits devices to be specified by their device positional number. The device positional numbers are dynamically assigned at the time that 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. **Create trust relationship from the master device to destination devices.** This involves sharing the HSM Identity public-key of HSM 0 to HSM 1 and HSM 2. The *ctident trust* command is used to achieve this. 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 is required to be entered for each device when using this command.

   Example:

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

4. **Create trust relationship from destination devices to master device**. This involves sharing the HSM Identity public-keys of HSM 1 and HSM 2 with HSM 0. The *ctident trust* command is used to achieve this. In the example below, the first command line illustrates how to share the HSM Identity public-key of HSM 1 with HSM 0. The second command line illustrates how to share the HSM Identity public-key of HSM 2 with HSM 0. The Administration Token SO pin is required to be entered for each device when using this command.

   Example:

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

Token Replication

Token replication allows a user to replicate their tokens across one or more HSMs. Token replication is required especially when configuring a system to operate in WLD mode. Token replication is not a suitable mechanism to use in place of token export.

Currently, token replication is supported on the Protect Server, and Protect Server – External HSMs.

Token replication can only be performed on User Tokens (Smart card and Administration Slots are not supported). Refer to the section entitled The ProtectToolkit C Model for a description of slot types. Token replication can occur from any User slot to any other User slot on the same HSM or a different HSM. During token replication, all the objects contained within the master token as well as the master token label are replicated. The later is an important factor when a system is operating in WLD mode as the token label identifies which virtual WLD slot that the token is associated with. Refer to the section entitled The Work Load Distribution Model (WLD) and High Availability (HA) for further details.

Once a token has been replicated, any objects that are created or modified on that token will **not** be automatically replicated to those tokens replicated from the same token. If a token is modified, and a requirement exists for consistency among tokens, then 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.

The *ctkmu* utility with the *rt* command is used to replicate tokens. Refer to Chapter 7 for further details on the *ctkmu* utility. The SO pin of token in the master slot and the SO pins of the tokens in the slots that the token is imported to must be the same. The User pin of the token in the master slot and the User pins of the tokens in the slots that the token is imported to must be the same. When replicating to an un-initialized token, the SO pin of the token is required to be entered. If the *No Clear PINs* flag is set, the User pin of the Administration token on the device importing the token is also required. Refer to the Security Flag Descriptions section in Chapter 5 for further details on the *No Clear PINs* flag.

The *ctkmu rt* command utilizes slot positional numbers to identify the master slot and the destination slots. The slot positional numbers are dynamically assigned at the time that 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. It is important that the system is stable when using this command.

The following examples illustrates how to replicate a token from 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) and from 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).

Alternative 1 – Master Tokens Replicated to a Single Slot or List of Slots

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

1. **Generate a list of all the slots on the system to establish the slot positional number**. The *ctkmu* utility may be utilized for this. Refer to Chapter 7 for further details. For each device, slots 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 following token labels; WLD_Slot_11 (Slot 0), WLD_Slot_22 (Slot 1), 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 un-initialized 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 $Revision: 3.10.2.1 $
Copyright (c) SafeNet, Inc. 2006

Cryptoki Version  = 2.10
Manufacturer      = SafeNet, Inc.
WLD_Slot_11                     (Slot 0)
WLD_Slot_22                     (Slot 1)
WLD_Slot_33                     (Slot 2)
AdminToken (1197)               (Slot 3)
<uninitialised token>           (Slot 4)
<uninitialised token>           (Slot 5)
<uninitialised token>           (Slot 6)
AdminToken (1111)               (Slot 7)
<uninitialised token>           (Slot 8)
<uninitialised token>           (Slot 9)
<uninitialised token>           (Slot 10)
AdminToken (1310)               (Slot 11)
```

- **Replicate the token.** The *ctkmu* utility with the *rt* command is used to replicate tokens. The command can take two parameters: the slot that the token is exported from and the list of slots that the token is imported to. Refer to Chapter 7 for further details. The SO pin of token in the master slot and the SO pins of the tokens in the slots that the token is imported to must be the same. The User pin of the token in the master slot and the User pins of the tokens in the slots that the token is imported to must be the same. When replicating to an un-initialized token, the SO pin of the token is required to be entered. If the *No Clear PINs* flag is set, the User pin of the Administration token on the device importing the token is also required. Refer to the Security Flag Descriptions section in Chapter 5 for further details on the *No Clear PINs* flag.

  The example shows how to replicate the master tokens from HSM 0 to HSM 1 and from HSM 0 to HSM 2 as follows:

  - Replicate token from slot 0 to slot 5
  - Replicate token from slot 0 to slot 9
  - Replicate token from slot 1 to slot 4 and slot 10

  Example:

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

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

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

Alternative 2 – Token Replicated to Many Tokens

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

1. **Generate a list of all the slots on the system to establish the slot positional numbers.** To utilize this method, the token label of the slot that is to import the token must be the same as the token label of the master 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 the section entitled Token Initialization for further details.

   Example:

```
C:\>ctkmu l
ProtectToolkit C Key Management Utility $Revision: 3.10.2.1 $
Copyright (c) SafeNet, Inc. 2006
Cryptoki Version  = 2.10
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)
<uninitialised token>                    (Slot 6)
AdminToken (1111)                        (Slot 7)
<uninitialised token>                    (Slot 8)
WLD_Slot_11                              (Slot 9)
WLD_Slot_22                              (Slot 10)
AdminToken (1310)                        (Slot 11)
```

2. **Replicate the tokens.** The *ctkmu* utility with the *rt* command is used 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 as the master token.

   Example:

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

Work Load Distribution Model (WLD) and High Availability (HA)

High levels of scalability, availability, reliability and increased throughput can be easily achieved by ensuring there is no restriction on the number of HSMs that can work in union. In addition, the application can be relieved from its own load sharing processing to focus on its primary tasks by enabling the built-in configurable WLD mode. A high availability/load balancing setup provides a reliable solution and boosts overall performance.

Work Load Distribution (WLD)

Load Distribution is a design approach where work is balanced across a system by transferring units of work among processing modules during execution. The demand placed on any particular processing module is reduced by distributing the work to multiple processing modules within the system. In a well balanced system, this 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 implemented is termed Work Load Distribution and works as follows. Within ProtectToolkit C, a distribution engine portions the requests for work, which it then distributes to an appropriate HSM to perform. The distribution engine implements a distribution scheme to determine which HSM is selected. The tokens that are utilized within the scheme must be replicated across the HSMs, as appropriate to the system design. A good system design should address throughput requirements, resource portioning and fault tolerance/disaster recovery. The ctident utility provides the mechanism to establish trust between HSMs that share tokens. The ctkmu utility provides the mechanism to replicate a token once trust has been established.

High Availability (HA)

Enterprises have requirements to maintain their services and keep them up and running with a high degree of reliability to provide the highest level of security. By providing redundancy and availability in services, High Availability (HA) constitutes a critical feature.

Within ProtectToolkit C, the HA feature is implemented in the Cryptoki library), and is a capability to keep track of the commands sent to a session so that it can re-establish a new session by simply replaying these commands in case of session failure. This approach offers the best solution 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, i.e. you cannot have HA without WLD.

HSMs

A WLD/HA system may consist of any number of PSI-E2 or any number of PSE2 HSMs. The use of PSI-E2 and PSE2 HSMs concurrently is not supported.

ProtectToolkit C

For Work Load Distribution/High Availability to be enabled ProtectToolkit C must be configured to operate in WLD or HA mode. Refer to section Operation in WLD Mode and Operation in HA Mode for further details.

When applications are using the ProtectToolkit C interface in WLD/HA mode, the system of physical HSMs appears as a single virtual HSM. ProtectToolkit C achieves this through the use of virtual WLD slots. When an application wishes to make use of a WLD slot, it does so via the standard PKCS #11 function calls. The distribution engine distributes the session over those physical HSM slots that are associated with the WLD slot.

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. Refer to section Configuring WLD Slots for further details. **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 load share.

NOTE: The HA system cannot support more than 16 slots and hence it is required that an administrator limits the WLD slot numbers to be less than 16 (from 00 to 15 inclusive).

Distribution Scheme

The distribution of application requests is performed on a PKCS #11 session basis. When an application opens a session to a WLD slot, the distribution engine uses a distribution scheme to select the initial physical HSM slot to be used for servicing the open session request. Once the session has been opened, all other requests performed on that session are routed to the initially selected 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 distribution 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, the randomized nature of the distribution scheme will ensure an even distribution of sessions across the available physical HSM slots.

Token Replication

ProtectToolkit C provides tooling to support replication of token information to other SafeNet HSMs in a protected form. The *ctident* utility provides the mechanism to establish trust between HSMs that share tokens. The *ctkmu* utility provides the mechanism to replicate a token once trust has been established. Refer to sections Trust Management and Token Replication for further details.

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

CAUTION: Ensure that the tokens in WLD are always 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 Example

This section illustrates how to setup a system for Work Load Distribution. The system in the example contains 3 remote HSMs with ProtectToolkit C running on a Windows platform. The example illustrates how to configure 3 virtual WLD slots.

Figure 7, details the resulting configuration that is established in the example. 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. As illustrated in the figure, 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 slots made accessible to them by the Cryptoki Library are only the WLD virtual slots. An application or utility configured to operate in WLD mode cannot access the HSM slots directly.

The associations between the virtual WLD slots and the physical HSM slots in this configuration are shown by the arrows. 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. The system configuration is as follows.

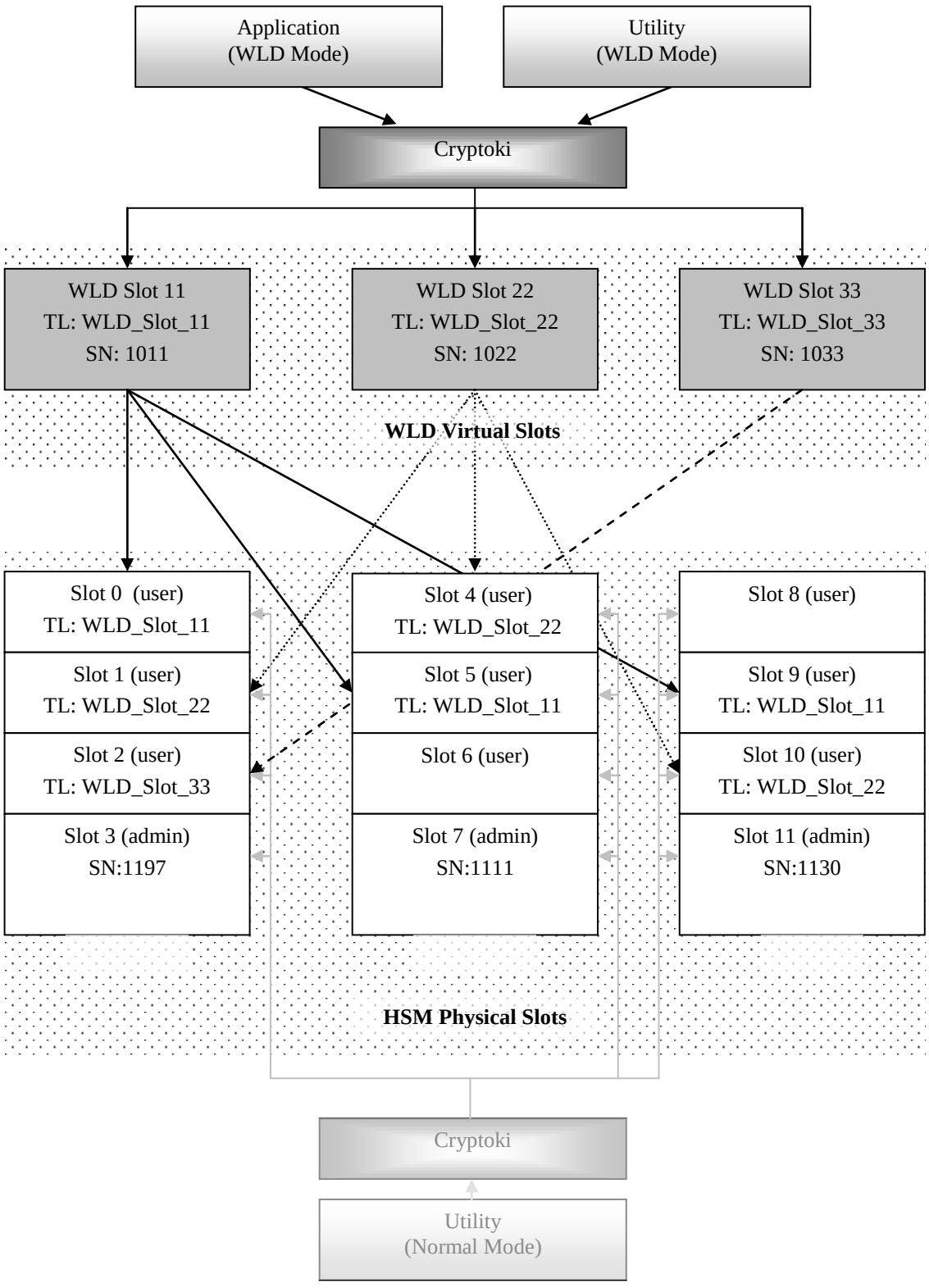


Figure 7 - Example of WLD configuration

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 ***Figure 7***, each WLD slot shares the same token label as the HSM slots that are associated with it. In the figure, the characters TL are an abbreviation for the token label. For example WLD Slot 22 shares the token label WLD_Slot_22 with the HSM User slots that are associated with it. That is, Slot 1, Slot 4 and Slot 10.

In ***Figure 7***, the characters SN are an abbreviation for the serial number of the token. It is necessary to be aware of the Admin token serial numbers when establishing the configuration for WLD operation. Each WLD slot must be configured with a serial number that is allocated by the user.

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

Configuration

1. **Establish Network Communication.** Set the environment variable *ET_HSM_NETCLIENT_SERVERLIST* with a list of the IP addresses of the HSMs in the order HSM0, HSM1, HSM2. Refer to section titled Specifying the Network Server(s) for further details.

2. Set the *Library Mode* to NORMAL. The HSM slots must be accessible to set up the system. For the HSM slots to be accessible, the utilities which access them must operate in NORMAL mode. This is achieved by setting the Cryptoki Library mode to NORMAL. Refer to the *Operation in WLD Mode* section for further details.

3. Initialize Admin Tokens and Security Policy. If a HSM has not been initialized, the Admin token and Security Policy for each HSM must be configured. Refer to the *Setting the Admin Token PINs* and *Selecting and Setting a Security Policy* sections for further details.

4. Create User Slots. Create User slots for each HSM, as described below. Refer to the *Setting up Slots* section 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 section entitled Configuring WLD Slots 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 the section entitled Token Replication for further details.

7. **Establish Trust.** For token replication to be performed from the HSM holding the master tokens to another HSM, both HSMs must have a trust relationship with the other HSM. Refer to section Trust Management 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 section Token Replication 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 the section entitled Configuring WLD Slots 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

   - 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 is set, 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. Refer to section Operation in WLD Mode for further details.

11. **Check the WLD Slot Configuration.** Run the *ctkmu* (*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:

```
C:\>ctkmu l
ProtectToolkit C Key Management Utility $Revision: 3.10.2.7 $
Copyright (c) SafeNet, Inc. 2006

Cryptoki Version  = 2.10
Manufacturer      = SafeNet, Inc.
WLD_Slot_11                     (Slot 11)
WLD_Slot_22                     (Slot 22)
WLD_Slot_33                     (Slot 33)
```

Configuring WLD Slots

If ProtectToolkit C is to operate in WLD Mode, virtual WLD slots must be configured.

The environment variables of the format ET_PTKC_WLD_SLOT_n specify the configuration parameters for the WLD slots. It is mandatory that an ET_PTKC_WLD_SLOT_n environment variable is configured for every WLD slot.

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

The format of these variables is as follows:

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

Where:

<WLDTokenLabel>	is mandatory and is the PKCS #11 Token Label for this WLD Token and is also used to identify the HSM Tokens to be used for WLD. The <WLDTokenLabel> should be unique within the complete list of WLD Slot Configurations.
• <WLDTokenSerial#>	is optional, and is the PKCS #11 Token Serial Number for this WLD Token, with the default being the WLD Slot Number, i.e. n from the configuration variable name.
• <WLDSlotDescription>	is optional, and is the PKCS #11 Slot Description for this WLD Slot, with the default being "WLD Slot:#", where # is the WLD Slot number, i.e. n from the configuration variable name.

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

UNIX

Under UNIX variants, the variable name and value are stored in the file "et_ptkc" in the directory /etc/default and/or $HOME/.safenet, depending upon choosing the system or user location respectively.

Example:

To configure WLD slots at the system level:

- Open the file: /etc/default/et_ptkc
- Make the following entries:

  ET_PTKC_WLD_SLOT_0=WLD Token 0,1002,PIN generation slot

  ET_PTKC_WLD_SLOT_5=WLD Token 5

  ET_PTKC_WLD_SLOT_6= WLD Token 6,,Password generation slot

NOTE: For WLD Slot 5, ProtectToolkit C will default the PKCS #11 Token Serial Number to 5, and the PKCS #11 Slot Description "WLD Slot:5". For WLD Slot 6, the PKCS #11 Token Serial Number will default to 6.

Windows

Under Win32, the variable name and value are stored in the HKLM and/or HKCU registry, in the key SOFTWARE\SafeNet\PTKC\WLD, depending upon choosing the system or user location respectively.

Example:

To configure WLD slots at the system level:

- Locate the registry key:

  HKEY_LOCAL_MACHINE\SOFTWARE\SafeNet\PTKC\WLD

1. Assign the *ET_PTKC_WLD_SLOT_n* variables the values shown in the UNIX example above.

Operation in WLD Mode

If ProtectToolkit C is to operate in WLD Mode, the Cryptoki Library mode must be configured.

The environment variable ET_PTKC_GENERAL_LIBRARY_MODE specifies the mode that the Cryptoki Library operates in. This variable controls whether the WLD model or the normal PKCS #11 model is applied to slot and token usage. Refer to the section entitled The Work Load Distribution Model (WLD) for further details.

This variable can have values of NORMAL or WLD or HA. If this variable is not defined, or contains an invalid value, then ProtectToolkit C will operate in the normal PKCS #11 mode.

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

While establishing the configuration it may be advisable to configure WLD mode via the temporary configuration parameter then when configuration is stable set the environment variable at the user or system configuration level.

It is possible to have a number of applications running in WLD mode and a number of applications 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 it is advisable that WLD mode is set at the user or system level and that NORMAL mode is set in environment variable operating in the context of the application which uses it. Refer to Overview for further details.

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

A simple example an how to configure a basic WLD system.

Assume the label of the tokens participating in the WLD is “TokName”and there is two PSE2’s with the address 192.168.1.100 and 192.168.1.101

Set these configuration items (for details see Chapter 2 Configuration Items)

ET_HSM_NETWORK_SERVERLIST=192.168.1.100 192.168.1.101
ET_PTKC_WLD_SLOT_0=TokName
ET_PTKC_GENERAL_LIBRARY_MODE=WLD

Operation in HA Mode

If ProtectToolkit C is to operate in HA Mode, the Cryptoki Library is provided a capability to keep track of the commands sent to a session so that it can re-establish a new session, in case of session failure, by simply replaying these commands.

ProtectToolkit C HA mode provides the following functionality:

- Detect that a session has terminated because of HSM failure and automatically establish a new session on a remaining HSM.
- After an HSM failure is detected, the feature will periodically attempt to automatically bring the HSM back on line.
- Restart an object search and step the search up to the point of failure.
- Restart a Encrypt, Decrypt, Sign, Verify, SignRecover, VerifyRecover and Digest operation and replay the Update operations (up to a certain limit of data length).
- Create a log entry to notify significant events
- Recover session objects that were created by:
    - C_CopyObject
    - C_DeriveKey
    - C_UnwrapKey
    - C_GenerateKey *
    - C_GenerateKeyPair *

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

The environment variable ET_PTKC_GENERAL_LIBRARY_MODE specifies the mode that the Cryptoki Library operates in. This variable controls whether the WLD mode and HA features are enabled. Refer to the section entitled Work Load Distribution Model (WLD) and High Availability (HA) for further details.

This variable can have values of NORMAL, WLD, or HA. If this variable is not defined, or if it contains an invalid value, then ProtectToolkit C will operate in the normal PKCS #11 mode.

The environment variable ET_PTKC_HA_RECOVER_DELAY represents the number of minutes to wait after detecting a failed HSM before attempting to reconnect to the failed HSM. The recovery is not attempted if the value is zero.

The environment variable ET_PTKC_HA_RECOVER_WAIT enables the HA feature to poll and attempt recovery if an HSM has failed. This variable can have values of YES or NO and is valid only if the HA feature is enabled using the variable ET_PTKC_GENERAL_LIBRARY_MODE=HA.

The library will issue a log entry in the following circumstances

When a HSM failure is detected and the library recovers that session the following message is generated:-

HSM Failure detected: hsmIdx=1, hsmSlotId=0

If the application performs operations such that the library is not capable of recovering the session in event of a HSM failure then following warning is generated (one the first occurrence only):

Session potentially not recoverable: <reason description>

If ET_PTKC_HA_RECOVER_DELAY and ET_PTKC_HA_RECOVER_WAIT are setup so that the library will try to reconnect a lost HSM then the following log entry is generated at each attempt:

```
Found HSM Dead ","HSM  Failed"
```

Example

The following is a simple example illustrating how to configure a basic WLD system.

Assume the label of the tokens participating in the WLD is “TokName”and there is two PSE2’s with the address 192.168.1.100 and 192.168.1.101

Set these configuration items (for details on how see Chapter 2 Configuration Items)

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

HA Mode Logging

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

Configuration Name	**Possible Values**
ET_PTKC_HA_LOG_FILE	Log filename and location – default “/ptk_cryptoki” For example, ET_PTKC_HA_LOG_FILE=C:\temp\ha_log.log (Windows) or ET_PTKC_HA_LOG_FILE=/tmp/hsm_log.log (Unix)
ET_PTKC_HA_ LOG_NAME	Application name – default “ptk_cryptoki”

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 a 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 PSI-E2 HSMs are detected. This is the same order as 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. It indicates that the library has seen a HSM has been failed and is holding off all application threads while it attempts to recover the lost HSM.

External Key Storage

Introduction

The secure memory available on ProtectServer HSMs is limited to 4MB.

The maximum number of keys (by type and size) that can be stored is limited by available memory only.

Applications in which secure memory requirements exceed those stated above can utilize the External Token Support Library (ExtToken) to overcome this limitation. ExtToken facilitates secure, external-to-the-HSM storage for token objects. ExtToken manages external token support transparently to host applications. Host applications can utilize 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 the ProtectToolkit C product (PTK-C) and is a part of the standard installation of the PTKcprt package incorporated in the PTK-C product release. The ExtToken library is supported on Windows only.

ExtToken supports the secure external storage of token objects for the purpose of RSA signing, checking certificates, DES key exchange and DES encryption of transaction messages, to name a few. To reduce the processing overhead introduced in the secure and external storage of token objects, the HSM utilizes internal cache memory to store the most recently utilized token objects. The number of token objects stored in cache is configurable by the user.

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

Implementation

The following figure illustrates how external key storage is achieved on the host system and HSM.

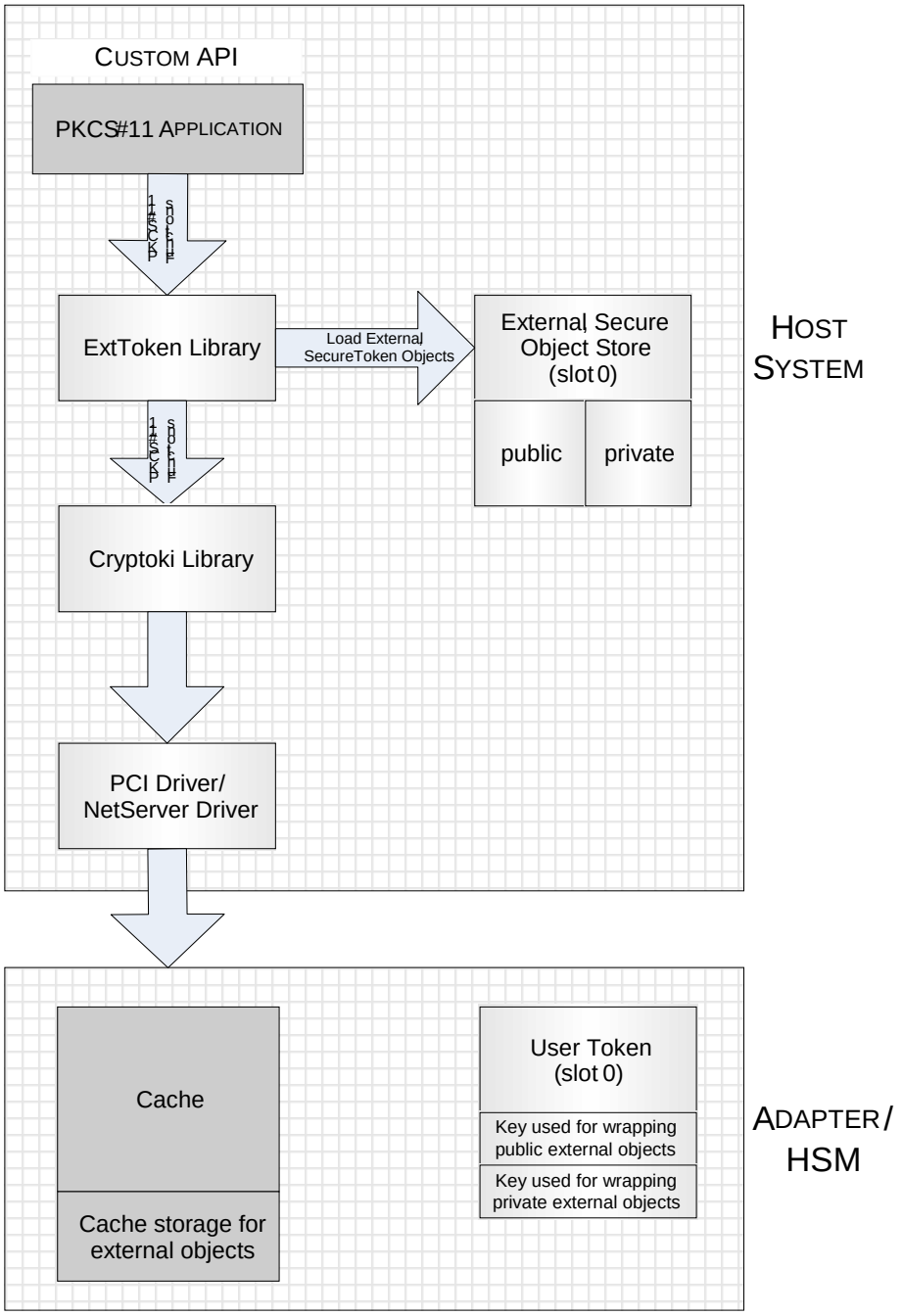


Figure 8 - External Key Storage

PKCS#11 applications interface to the ExtToken library via standard PKCS#11 function calls. The ExtToken library makes use of another PKCS#11 provider, the Cryptoki library. The Cryptoki library is responsible for enforcing security policies and storing all data not related to the token objects of an external token. All cryptographic processing is performed on the standard Cryptoki library. The Cryptoki Library interfaces to local HSMs via the PCI driver and remote HSMs via the Netserver Driver.

ExtToken achieves secure external storage by using two master keys. These master keys are used to transparently wrap and unwrap the external objects. One of these keys is for protecting public objects, and the other is for protecting private objects. These master keys are DES2 keys and will be stored in slot 0. The token in slot 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.

As token objects are created via the ExtToken library, they 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 to 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 utilized 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 this limit is reached, then the least used object is unloaded from cache.

Key back up can be achieved simply by storing the master keys on a smart card and compressing the files utilized by the Secure Object Store using SafeNet ProtectPack.

Performance

Performance overhead is introduced when storing objects externally. The overhead is introduced by the need to unwrap objects. The CTPERF utility (provided with the PTK-C installation) can be used to determine performance on individual systems. As an indication, 109 keys/ second can be unwrapped using a DES3 key. As previously discussed, the HSM utilizes internal cache memory to store the most recently utilized token objects in an effort to reduce this processing overhead. The environment variable ET_PTKC_MAXLOADED allows the user to configure the maximum number of token objects stored in cache. However, there is also a processing overhead involved in managing the keys stored in cache. This overhead increases linearly as the number of items stored in cache increases. Systems must be individually tuned for maximum performance depending upon patterns of key usage by the host application and by taking into consideration the tradeoff between the processing overhead involved in unwrapping keys and the processing overhead involved in managing the cache.

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.

There are two files per 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 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 the CKA_EXTRACTABLE attribute 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 - that is 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 the application performance. This negative effect can be countered by using as many attributes from the above list as possible, 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 the upper limit supported by SafeNet HSMs. One example of such an application is ctkmu. This means that it is possible to have so many objects on an external token that it is not possible to back them up. This can be rectified by adjusting the value of ET_PTKC_EXTTOKEN_MAXLOADED to a value which suits your application/environment.

The SafeNet implementation of JCA/JCE (ProtectToolkit J) uses one session per KeyStore. An application which uses the same KeyStore to access a large number of keys runs the risk of consuming all HSM resources (see point 7). A work-around is to use a new KeyStore object when locating keys. This does not introduce a significant performance overhead.

Smartcards and the Admin Token cannot be used as external tokens.

External Key Storage cannot be utilized in conjunction with WLD.

Configuration

There are a number of files named cryptoki.dll provided as a part of the PTK-C installation. This design was utilized so that PCKS#11 applications always link to a library called cryptoki.dll. The following table shows 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\ProtectToolkit C Runtime	Cryptoki library used for runtime applications	1
C:\Program Files\SafeNet\ProtectToolkit C Runtime\extToken	ExtToken library used for runtime applications	2
C:\Program Files\SafeNet\ProtectToolkit C SDK\bin\logger	Logger library used during application development	3
C:\Program Files\SafeNet\ProtectToolkit C SDK\bin\hsm	Cryptoki library used during application development when communicating to HSMs	4
C:\Program Files\SafeNet\ProtectToolkit C SDK\bin\ExtToken	ExtToken library used for application development	5
C:\Program Files\SafeNet\ProtectToolkit C SDK\bin\sw	Cryptoki library used during application development in software only mode	6

As both the ExtToken library and a Cryptoki library are named cryptoki.dll and utilized in an implementation that requires external key storage, environment variables are used to indicate which library is linked to which software component. As indicated in ***Figure 8***, PKCS#11 applications link to the ExtToken library. The ExtToken Library in turn links to a Cryptoki library. The following steps detail how this can be achieved.

Configuration for Application Development

1. Locate the current cryptoki.dll in use.

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

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

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

3. Configure the ET_PTKC_EXTTOKEN_PKCS11LIB environment variable.

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

   Typically this should be achieved by:

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

  To specify the PTK-C SDK (PCI and network modes) Cryptoki library (Id 4), typically, ET_PTKC_EXTTOKEN_PKCS11LIB should be set to *C:\Program Files\SafeNet\ProtectToolkit C SDK\bin\hsm\cryptoki.dll*.

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

Checking the Configuration

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

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

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

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

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

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

Configuration for Runtime Operation

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

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

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

Checking the Configuration

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

Creating Externally Stored Objects

The utilities typically located in either Program Files\SafeNet\ProtectToolkit C Runtime (for runtime installation) or C:\Program Files\SafeNet\ProtectToolkit C SDK\bin (for SDK installation) can now be utilized to create keys that are stored externally. 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.

Back-up and Restore

1. The simplest method for backing up keys is to zip the secure external storage files and to export the objects keys utilized in the ExtToken mechanism. SafeNet's ProtectPack can be used to compress the files. As the files are already in an encrypted format encryption need not be applied when compressing the files. These files are located in the folder indicated by the ET_PTKC_EXTTOKEN_PATH environment variable and have the extensions .ort and .ods.

2. To access the objects used 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\ ProtectToolkit C Runtime\hsm (if the previous configuration steps were followed). For SDK, this is typically C:\Program Files\SafeNet\ProtectToolkit C SDK\bin\hsm.

3. At a DOS command prompt, enter the command `ctkmu l –s0`. Three objects should be listed with the label ExtToken. The first object is a Data object containing information relating to the configuration of ExtToken. Two secret keys are created; one key is for private objects and the other key is for public objects.

4. Export the objects in slot 0 onto Multiple Custodian smartcards. The following example illustrates how to do this with two smartcards, where the smartcard reader is located in slot 1.
   `ctkmu x –s0 –c1`

5. When restoring keys, the Cryptoki Library (Id 1or 4) must be used (see step2). To restore keys after tampering the HSM, uncompress the secure external storage files into the folder indicated by the ET_PTKC_EXTTOKEN_PATH environment variable. Import the secret keys from the smartcards. The following example illustrates how import the keys if exported in the manner described above.
   `ctkmu i –s0 –c1`

6. To make use of the ExtToken Library the system must be re-configured to use the ExtToken Library for Application Development or for Runtime Operation as discussed previously.

Real Time Clock

The HSMAdmin API is provided which allow applications to access the Real Time Clock (RTC). The API gives an application the capability to access and adjust the time, the RTC status and access statistics regarding the effective adjustment and the number of times the RTC has been adjusted.

The CTCONF utility provides the capability for an administrator to configure adjustment access control for the RTC. This gives an administrator the capability to control the delta amount and the number of times the RTC can be adjusted within a configurable period of time. The CTCONF utility does this via two command line options: one that sets the rule for adjustment access control and the second that enables/disables adjustment access control. The CTCONF utility description provides full 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 be failed.

The following examples illustrate how to use the utility to set guard parameters. The table below indicates the parameter settings as follows.

Parameter	Meaning
secs	Total amount of deviation (in no. of seconds) within a guard duration. Range 1≤ secs ≤ 120
count	Total number of adjustments that can be made within the guard duration. Range 0≤ count. 0 indicates an unlimited number of adjustments
days	The guard duration in number of days. Range 1≤ days ≤ 12

Example 1:

For example, if applications accessing the RTC should 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 could be used to set the rule for RTC Adjustment Access Control.

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

Example 2:

For example, if this rule should then be modified so that the number of days in the guard duration should be extended to 4, the following command could be used so that 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

RTC Adjustment Access Control can be enabled once the RTC Adjustment Access Control Rule has been set. 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.

Example 1:

The following command line enables 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 5
SECURITY POLICIES AND USER ROLES

Overview

This chapter discusses different aspects that administrators must consider when selecting and setting a security policy for the ProtectToolkit C environment. There are numerous areas that can affect operational security and it is important to develop an understanding of the various security features and how they 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 from a security perspective. For example, whether PINs may be passed across the host interface in an unencrypted form or whether a soft tamper (erase all internal secure memory) should occur as part of a firmware upgrade.

Organizations are free to create unique security policies to satisfy their own needs or they may adopt policies, defined externally by standards bodies or other organizations, to meet their security requirements.

A number of security settings offered as a part of ProtectToolkit C can be used to implement *typical security policies* that meet certain standards or satisfy application integration requirements. These can be utilized where appropriate or any other custom security policy can be implemented. The options available are fully discussed in this chapter and implementation instructions given.

If you are implementing a security policy to satisfy application integration requirements other relevant information may be available. See the section SafeNet Application Integration Guides in Chapter 1 for further details.

It should be noted that the level of compliance with the PKCS #11 standard will vary from policy to policy. In general, a greater level of compliance equates to a lower level of security. See the section PKCS #11 Compliance and Security below for further information.

How security policy changes affect users will, in some cases, also depend upon the roles to which they have been assigned. This is because some security policy settings have effects that are user-role-specific. Please consult the User Roles section later in this chapter for more information.

ProtectToolkit C security policies are implemented by setting or clearing *security flags* to switch on or off particular functionality. A security policy might be implemented by setting a single security flag. In other cases more than one flag must be set.

PKCS #11 Compliance and Security

ProtectToolkit C may be configured to be highly compliant with the PKCS #11 standard. This is achieved by using the security policy *PKCS #11 Compatibility Mode.* If a greater level of security is required then an alternate standard or custom security policy may be adopted. These and all other typical security policies are discussed in the next section.

By default (after initial HSM installation or following a tamper event) the ProtectToolkit C security policy applying is *SafeNet Default Mode.* This mode offers a greater level of security than is afforded when operating in PKCS #11 Compatibility Mode while at the same time affording a greater level of compliance with the PKCS #11 standard over other possible security policy implementations.

For further information about how the SafeNet Default Mode differs from PKCS #11 Compatibility Mode and the related security issues, see the section PKCS #11 Compatibility Mode on page 47.

As a general guide, the above discussion is summarized in the following table.

Security Policy	**PKCS #11 Compliance Level**	**Security Level**
PKCS #11 Compatibility Mode	High	Low
SafeNet Default Mode	Medium	Medium
Other security policies	Low	High

Typical Security Policies

Overview

A number of *typical security policies* that are designed to meet standards or satisfy application integration requirements are offered as a part of ProtectToolkit C and can be utilized where appropriate. In this section the following typical security policies are described.

- PKCS #11 Compatibility Mode
- SafeNet Default Mode
- FIPS Mode
- Entrust Compliant Modes
- Netscape Compliant Mode and
- Restricted Mode

The *ctconf* command line utility is used to implement the policies by setting *security flags*. The specific commands to be used are also given in each case.

Security flags are discussed in detail in the Security Flags section later in this chapter.

Optionally with some of the typical security policies, security setting flags may be changed to change security behavior without invalidating the policy. See the section Security Policy Options later in this chapter for further information.

For the complete ctconf command reference see the CTCONF section in Command Line Utilities Reference.

PKCS #11 Compatibility Mode

Allows the following mechanisms to behave 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

Warning

Use of this security policy compromises security. If a skilled attacker manages to introduce software into the host system they can exploit vulnerabilities that these mechanisms, when operating in PKCS #11 Compatibility Mode, allow.

ctconf Command

```
ctconf -fp
```

SafeNet Default Mode

By default (after initial HSM installation or following a tamper event) the ProtectToolkit C security policy applying is *SafeNet Default Mode.* This mode offers a greater level of security than is afforded when operating in PKCS #11 Compatibility Mode while at the same time affording a greater level of compliance with the PKCS #11 standard over other possible security policy implementations.

For further information about how the SafeNet Default Mode differs from PKCS #11 Compatibility Mode and the related security issues, see the section PKCS #11 Compatibility Mode above.

ctconf Command

```
ctconf -f0
```

FIPS Mode

ProtectToolkit C and the ProtectServer HSM have been certified to Federal Information Processing Standard (FIPS) 140-1 level 3. The FIPS certification assures users that an independent third party has verified that the product meets the high levels of security specified by the standard.

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 implies that cryptographic services cannot be performed by unauthenticated users.

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

FIPS Mode Operational Restrictions

All RSA operations performed under FIPS mode will only be carried out if the specified key has a modulus of 1024 bits or greater. Any attempt to use or create an RSA key smaller than 1024 bits while running in FIPS mode will result in a CKR_KEY_SIZE_RANGE error. The following mechanisms will be affected by the key size limitation when running in FIPS mode:

CKM_RSA_PKCS_KEY_PAIR_GEN
CKM_RSA_X9_31 KEY_PAIR_GEN
CKM_KEY_WRAP_SET_OAEP
CKM_RSA_PKCS
CKM_RSA_PKCS_OAEP
CKM_RSA_X_509
CKM_SHA1_RSA_PKCS
CKM_SHA224_RSA_PKCS
CKM_SHA256_RSA_PKCS
CKM_SHA384_RSA_PKCS
CKM_SHA512_RSA_PKCS
CKM_SHA1_RSA_PKCS_TIMESTAMP

All DSA operations performed under FIPS mode will only be carried out if the specified key has a modulus of 1024 bits or greater. Any attempt to use or create a DSA key smaller than 1024 bits while running in FIPS mode will result in a CKR_KEY_SIZE_RANGE error. The following mechanisms will be affected by the key size limitation when running in FIPS mode:

CKM_DSA_KEY_PAIR_GEN
CKM_DSA_PARAMETER_GEN
CKM_DSA
CKM_DSA_SHA1
CKM_DSA_SHA1_PKCS

ctconf Command

`ctconf -fF` (equivalent to `ctconf -faclntu`)

Entrust Compliant Modes

Entrust Compliant Mode 1

The *Entrust Compliant Mode 1* may be used to implement the specific security profile required by Entrust Authority version 5.x software.

Ctconf Command

```
ctconf –fe
```

Entrust Compliant Mode 2

The *Entrust Compliant Mode 2* may be used to implement the specific security profile required by Entrust Authority version 6.x and Entrust Security Manager version 7.x software.

Ctconf Command

```
ctconf –fc
```

Netscape Compliant Mode

ProtectToolkit C is compatible with the Netscape/iPlanet range of products. In particular, 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

Placing the HSM in this mode is achieved by setting the *No Public Cryptography* flag to enabled.

Ctconf Command

```
ctconf –fc
```

Restricted Mode

In *Restricted Mode* the HSM requires all users to identify themselves before cryptographic services are 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.

Ctconf Command

```
ctconf –fcnl
```

Security Flags

Overview

Once a policy has been selected it is implemented in ProtectToolkit C by configuring *security flags.*

Security flags control particular security settings. One or more of these flags can be set to create custom security policies or to implement the typical security policies described in the previous section.

Configuring Security Flags

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

The command syntax is as follows:

ctconf –**f***flags*

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

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

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

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

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

Each of these flags is fully described in the next section.

For the complete *ctconf* command reference, see the CTCONF section in.Command Line Utilities Reference.

Security Flag Descriptions

The security settings indicated by each of the security flags are described below. A mapping of security flags to the typical security policies described in this manual is given in the next section Security Policy Options.

Auth Protection

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

By enabling this setting applications will operate in a more secure manner, however this will also have the effect of decreasing HSM performance. This is due to the increased operations required to sign and verify each request message.

DES Keys Even Parity Allowed

The Des Keys Even Parity Allowed permits creation of DES, DES2 and DES3 keys that have even parity. Creation of a DES key and DES key components with even parity is permitted if this flag is set.

Entrust Ready

The *Entrust Ready* (Entrust Compliant) flag, when set, indicates that:

- When a mechanism is queried that does not exist an empty mechanism structure is returned
- When a token is initialized with the C_InitToken command the SO PIN is not required
- A user that 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 then it defaults to true so wrapping is allowed

FIPS Algorithms Only

The *FIPS Algorithms Only* (Only Allow FIPS Approved Algorithms) flag, when set, indicates that non-FIPS approved algorithms are disabled.

The algorithms that are 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, above, for further information.

NOTE: For the list of 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, indicates that the following composite flags are set.

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

Instead of specifying each of these flags individually when using the *ctconf* utility to put a HSM into FIPS Mode, the *FIPS Mode* flag can be specified as a shortcut.

Refer to the entries for the composite flags and the *Typical Security Policies* section FIPS Mode for further information.

Full Secure Messaging Encryption

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

- User PINs or other sensitive information cannot be passed across the host interface in an unencrypted form.
- Secure messaging encryption is enabled so that every message is encrypted in both directions between the application and the HSM.
- 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 that every message is encrypted in both directions between the application and the HSM. The key used for the message encryption is generated using the PKCS #3 Diffie-Hellman Key Agreement Standard.

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

Full Secure Messaging Signing

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

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

By enabling this setting applications will operate in a more secure manner, however this will also have the effect of decreasing HSM performance. This is due to the increased operations required to sign and verify each request and response message.

Increased Security Level

The *Increased Security Level* flag, when set, indicates 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, indicates that this flag (or any other security flag) cannot be modified. A new security policy can only be implemented after a tamper operation has been performed.

No Clear PINs

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

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

No Public Crypto

The *No Public Crypto* flag, when set, indicates 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 to indicate 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: This flag does not imply that public key cryptography is not allowed. Setting this flag will not prevent RSA processing.

Pure PKCS11

The *Pure PKCS11* flag, when set, indicates that the following mechanisms will behave 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

Warning

Setting this flag compromises security. If a skilled attacker manages to introduce software into the host system they can exploit vulnerabilities that these mechanisms, when operating with this flag set, allow.

Tamper Before Upgrade

The *Tamper Before Upgrade* flag, when set, indicates 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*, when set, indicates that ECC Public and Private keys may be generated and stored within the HSM which have Domain Parameters other then the set of named curves built into the HSM.

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	×	×	✓ ×	✓ ×	✓ ×	✓	✓ ×	✓ ×	✓ ×	✓×
SafeNet Default Mode	×	×	x	×	×	×	×	×	×	×	×	×	✓×
FIPS Mode	✓	✓	x	×	✓ ×	✓	✓	✓ ×	✓ ×	✓	✓	✓ ×	✓×
Entrust Compliant Mode 1 [1]	×	×	x	✓	✓ ×	✓ ×	×	×	✓ ×	✓ ×	×	×	✓×
Entrust Compliant Mode 2 [2]	×	✓	x	×	✓ ×	✓ ×	×	×	✓ ×	✓ ×	×	×	✓×
Netscape Compliant Mode	×	✓	x	×	✓ ×	✓ ×	×	×	✓ ×	✓ ×	×	×	✓×
Restricted Mode	×	✓	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

c No Public Crypto

d DES Keys Even Parity Allowed

e Entrust Ready

i Increased Security Level

l Mode Locked

✓ The security flag must be set. If cleared the security policy is invalidated.

× The security flag must be cleared. If set the security policy is invalidated.

✓× Optional. Setting or clearing the security flag will not invalidate the security policy.

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:

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

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

- Set or Change Real Time Clock (RTC) value
- Read the System Event Log
- Purge a full System Event Log
- Configure the Transport Mode feature
- Specify the Security Policy of the HSM
- Create new 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 is allowed to HSMs. 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 security policy implemented.

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 either the *hsmreset* or *e8kreset* utility

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.

The procedural steps described herein assume that ProtectToolkit C has been installed and configured for normal runtime usage. This chapter also frequently refers to various command line utilities that can be referenced for further detail in the Command Line Utilities Reference chapter.. Many of these functions can also be achieved with the GUI based tools, which are detailed in the GUI Utilities Reference chapter.

Changing a User or Security Officer PIN

At certain stages of ProtectToolkit C usage it may 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, the command line utility ctkmu is used.

Example:

```
ctkmu p –s2 –O
```

The above will attempt to change the SO PIN on slot 2.

The current SO PIN will be prompted for and the new PIN will have to be entered and confirmed.

```
ctkmu p –s1
```

The above will attempt to change the token user PIN on slot 1.

The current user PIN will be prompted for and the new PIN will have to be entered and confirmed.

NOTE: This command is also used to initialize the User PIN. When this command is executed and the User PIN is un-initialized the SO PIN will be prompted for and the initial User PIN may be entered.

Secure Key Backup and Restoration

For the purpose of conveniently transferring sensitive keys to other machines or for off-site storage and subsequent recovery, ProtectToolkit C allows for keys to be backed up to disk files or smart cards. Encrypted parts may also be written to the screen.

Determining Backup Requirements

There are no set rules within ProtectToolkit C that dictate which keys should be backed up. The task of deciding which keys require backup is up to the individual key owner.

As a guideline, keys that would generally be backed up are those that cannot be re-created or easily reconstructed by other means. These may include generated key values or long keys that may have been 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 the *Key Attributes* section later in this chapter.

Available Backup and Recovery Methods

There are two different methods that may be used to backup a key. These are:

- The *multiple custodians method* where a key is split into multiple shares and then distributed to multiple custodians. The shares are encrypted (wrapped) by a second key called the wrapping key which is selected at random.
- The *single custodian method* where the key to be backed up is encrypted (wrapped) by a second key called the wrapping key that is specifically chosen.

Once encrypted, the key is then stored on the backup medium.

Key Splitting Scheme Selection

If it is decided to split a key into multiple shares, then the scheme to split the key must be selected. It is possible to split the key in such a way that the original key may only be recovered with the co-operation of either:

- all the custodians using the *standard scheme*, or
- any of the custodians, with a user specified minimum number being required, using the *N of M scheme*[1]

If the *N of M* scheme is used then not all the custodians are required to recover the key. An additional advantage is that should a smart card become corrupted the key can still be recovered. When a corrupted card is encountered during an import operation it will be rejected. A prompt will then display for another card to enable the import operation to continue.

A Typical Key Backup and Recovery Scheme

An overall backup scheme would typically combine the *multiple custodians* and *single custodian* methods. In this scenario, a wrapping key (KWRAP) would be generated and then 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 ***Figure 9***.

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

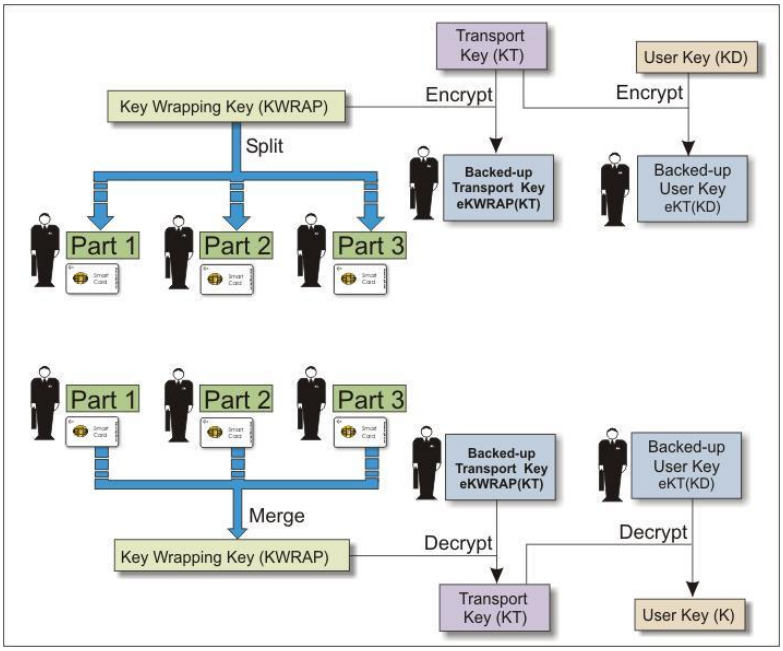


Figure 9 - A typical key backup and recovery scheme

Key Attributes

As mentioned previously, it is important to ensure that appropriate attributes are selected for keys so that these keys may be used for their desired purpose, while remaining compatible with the key backup scheme that is to be implemented.

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

To combat this situation, ProtectToolkit C introduces an alternate key backup method. This extension allows for the backup key to have the CKA_EXPORT attribute set to true and the key to be backed up to have the CKA_EXPORTABLE attribute set to true. In this case the wrapping key is known as an *export key* and the key to be backed up as an *exportable key*. The difference between this pair of attributes and the previous pair is that only the Security Officer may set the CKA_EXPORT attribute to true.

The key backup procedures described below work equally for the PKCS #11 standard attributes or the extension ProtectToolkit C attributes. It is, however, recommended that the extension method be implemented in order to mitigate the threat described above.

NOTE: When the export/exportable procedure is used with split custodian the token Security Officer must be present in order to create the custodian export keys.

Also, 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 and the user should create a new export key to create a new backup batch of exportable keys.

Prior to key backup, it must be ensured that the keys selected for backup have the correct attributes set in order to allow the export. The following table details the key attribute settings required. Also shown are the attribute settings required to enable use of a key as a key wrapping key.

Attribute	Key Wrapping Key	Key designated for backup
CKA_MODIFIABLE	FALSE	-
CKA_SENSITIVE	TRUE	-
CKA_WRAP	TRUE	-
CKA_EXPORT	TRUE	-
CKA_UNWRAP[3]	TRUE	-
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 attributes. There are two pairs of attributes that must match, CKA_EXPORT and CKA_EXPORTABLE and/or CKA_WRAP and CKA_EXTRACTABLE for the Key Wrapping Key and Key designated for backup, respectively.

[2] Wrapping keys should not be available for decryption; otherwise the wrapped key may be decrypted directly exposing the sensitive key material.

[3] Note that the CKA_IMPORT attribute can be used in place of the CKA_UNWRAP attribute. CKA_IMPORT 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.

The command line tool ctkmu and the GUI tool gctkmu can be used for both creating keys and changing their attributes. Further details on these utilities and key generation may be found in Chapter 7 and Chapter 8 respectively. In addition, details regarding key attributes may be found in Appendix B.

Key Backup Procedure

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

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

The rules applying to key backup are as follows:

- Attempting a key backup without specifying a wrapping key will result in a multiple custodian backup using a random key to 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 that was used to export it.
- When using the *ctkmu* command line utility, the *standard scheme* will be used by default for multiple custodian key backups unless the "-M" parameter is specified. When "-M" is specified, the *N of M scheme* is used.

See the Available Backup and Recovery Methods section above for further information regarding the methods mentioned here.

Key backup and restore is accomplished using either the command line utility *ctkmu* or the GUI utility *gctkmu*. These utilities can backup and restore keys from either a disk file or one or more smart cards. Please refer to Chapter 7 and Chapter 8 for the complete *ctkmu* and *gctkmu* references respectively.

The following examples use the *ctkmu* command line utility. See the section Exporting Keys within the Key Management Utility (KMU) section in GUI Utilities Reference for the procedure when using the KMU GUI utility.

Example 1: Using a Wrapping Key

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

In this example, the 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.

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

```
ctkmu x –s2 –nMyWRAP1 –c4
```

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*. This means that the key is split in such a way that the original key may only be recovered with the co-operation of all the custodians.

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

```
ctkmu x –s2 –nMyWRAP1 –c4 –M
```

This example is the same as Example 2 with the addition of the "-M" parameter. Specifying this parameter causes the *N of M scheme* to be used. This means that the key is split in such a way that the original key may be recovered with the co-operation of any of the custodians with a user specified, minimum number of custodians (N) being required out of the total (M).

In addition to the prompts described in Example 2, an additional prompt displays 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 the Available Backup and Recovery Methods section above for further information regarding the N of M scheme if required.

Key Restore Procedure

The task of restoring keys that have been backed up is essentially a reversal of the above-mentioned procedures. When attempting to restore or import key data, note the following:

- When restoring a key held by multiple custodians, all custodians (if the standard key slitting scheme was used) 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 re-combined to form the original key.
- When restoring a key held by a single custodian, the same wrapping key must first be available on the token that was used to encrypt the key.

Example 1: Single Custodian Key Recovery

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

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.

Example 2: Multiple Custodian Key Recovery

```
ctkmu i –s2 –c4
```

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

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 for further information regarding the schemes.

Re-initializing a Token

The *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 re-used for a different application.

The 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.

Example:

```
ctkmu t –s1
```

This example will re-initialize the token on slot 1 and initialize it with a new User PIN and a new label. This operation will prompt for the slot SO PIN.

Adding and Removing Slots

The task of adding or removing slots to or from ProtectToolkit C is the Administrator's responsibility. To accomplish this, the administration utility ctconf is used.

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.

Example:

```
ctconf –c2
```

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

After the addition of slots, all smart card and the admin slot numbers will automatically re-adjust. Each token in the newly created slots will also require initialization as described in Chapter 4.

Removing Slots

When removing slots from ProtectToolkit C, consideration has to be paid to the fact that either the user or some other entity may be 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.

Example:

```
ctconf –d2
```

This example will permanently remove slot 2 from ProtectToolkit C. The Administrator PIN will be prompted for.

Connecting and Removing Smart Card Readers

NOTE: The smart card slot detection behavior is changed in ProtectToolkit C version 3.10 and newer.

In order to speed up the application start up, the last detected smart card reader information is cached. If a change occurs in the attached smart card readers, the user must tell the system to query for changes in peripheral devices explicitly. This is valid for both connecting and removing smart card readers.

The command "ctconf -q" or "ctconf --query" can be used to perform a full device scan on all available serial ports. Alternatively, a system reboot, or a HSM reset (using the HSM specific tooling – for example, "e8kreset" for the ProtectServer) will cause ProtectToolkit C to initiate a full detection cycle on the next application startup.

Using Transport Mode to Avoid a Board Removal Tamper

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

It is the Administrator's responsibility to set the required transport mode on the HSM.

To accomplish this, the command line utility ctconf is used with the –m option.

Example:

ctconf –m2

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

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

NOTE: The transport mode does not disable the tamper response mechanism entirely. Any attempt to physically attack the HSM will still result in a tamper response.

Adjusting the HSM Clock

Due to host system and HSM timing differences, such as clock drifts, it may become necessary, at certain stages, to adjust the internal time on the HSM hardware. Only the Administrator can perform this task.

Note that 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 system host time as per your host operating system.

2. From a command prompt, type the following:

```
ctconf -t
```

Changing Secure Messaging Mode

See *Changing Secure Messaging Mode* in the Operating Mode Setup section.

Managing Session Key Rollover

See Configuring Session Key Rollover in ADH Mode in the Operating Mode Setup section.

Using the System Event Log

Viewing and Interpreting the Event Log

ProtectToolkit C maintains a system event log in order to provide a means of tracking serious hardware or consistent operational faults, tamper events and self-test error information.

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 chronologically. If the date and time of a later entry in the log is stating an earlier time than an entry preceding it, it indicates that the real time clock or audit information has been altered.

See Appendix A for a complete list of the possible error code values generated by HSM firmware that may be recorded in the event log.

To view the event log:

From a command prompt, type the following:

```
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.

Note that the event log cannot be purged until it is full.

To purge the event log:

From a command prompt, type the following:

```
ctconf -p
```

Updating Firmware

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

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

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

Prior to performing a firmware upgrade, ensure that you have performed the following:

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

The upgrade of the HSM firmware is performed by typing the following at a command line prompt:

```
ctconf –gfilename
```

where *filename* refers to the name of the firmware upgrade file. Entry of the Administrator password will be required during this operation.

Success or failure of the firmware upgrade will be displayed on screen.

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

Tampering the HSM

The tampering of the HSM may be necessary at the end of its lifecycle or any other security sensitive event that requires all stored data to be immediately destroyed.

A tamper formats the secure memory of the HSM and thereby erases all configuration and user data.

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

To tamper the HSM:

From a command prompt, type the following:

```
ctconf –x
```

The Administrator will then be prompted for their PIN and to confirm the action.

NOTE: The above action cannot tamper the HSM while other applications are active. This command will indicate if the tamper operation was successful.

Installing a Functionality Module

The SafeNet Protect Series HSMs support development of custom functionality which can affect how the internal processing of the hardware is performed.

Functionality modules can be developed with the aid of a Functional Module development kit, which can be purchased separately from SafeNet.

Functionality modules are distributed in conjunction with a certificate so that their 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, you can upgrade it by contacting SafeNet.

To install a functionality module:

Assuming you have the verification certificate for the fm in a file called *certname.cert* and the fm in a file called *fmfile.fm*...

From a command prompt, type the following:

```
ctfm   i -lcertname –ffmFile.fm
```

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 7
COMMAND LINE UTILITIES REFERENCE

CTCERT

Certificate Management Utility for the ProtectToolkit C environment.

Synopsis

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

Description

The **ctcert** utility provides basic support for the creation of X.509v3 certificates using the ProtectToolkit C product. With this tool it is possible to:

- Generate both self signed certificates and certificates signed with a specified CA key.
- Generate PKCS #10 certificate requests.
- List certificates, certificate requests, and key objects that exist in a specified slot.
- Import certificates (PEM format).
- Export certificates (PEM format).
- Mark certificates as trusted

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 the Operation in WLD Mode and Operation in HA Mode sections for further details.

Commands

Command	Description	
c	**Generate Certificate**	This command is used to generate X.509v3 certificates. When generating certificates with ctcert, a number of different approaches can be used. These approaches and the minimum options required are listed below: *Generate new keys and self sign* **ctcert c** -k -l<*label*> [options ...] *Generate new keys and sign with a CA key* **ctcert c** -c<*label*> -k -l<*label*> [options ...] *Use existing keys and self sign* **ctcert c** -l<*label*> [options ...] *Use existing keys and sign with a CA key* **ctcert c** -c<*label*> -l<*label*> [options ...] When using the Generate Certificate command, one of the above combinations of the options '-c', '-k', '-l' must be used. All other options are optional as indicated and facilitate finer control over the default actions of ctcert. Please refer below for detailed description of each option. When the -l<*label*> option is used without the -k (generate new key pair) option, the <*label*> refers to 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 if found uses it to generate the certificate. If a certificate request does not exist, ctcert searches for a public key with a matching label and if found uses it to generate a certificate. Otherwise ctcert reports an error.
i	**Import Certificate or Certificate Request**	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 that the object is PEM encoded, the first line in the text file should contain one of the following strings. "----BEGIN CERTIFICATE----" "----BEGIN CERTIFICATE REQUEST----" The -l<*label*> option specifies the label for the certificate or certificate request object once it has been imported.
l	**List Certificates, Certificate Requests, and Keys**	This command will list the certificates, certificate requests and keys that exist on the specified token.

Command	Description	
r	**Generate Certificate Request**	This command is used to generate a certificate request from either an existing key pair or a 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 using the *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 then ctcert will prompt for this information. If a new key pair is being generated then the -l<*label*> option specifies the label for the new key pair and ctcert will prompt for the certificate request objects subject distinguished name. The new certificate request objects label will also be set to <*label*>.
t	**Set Trusted Certificate**	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 command will prompt for the current SO PIN of the token.
x	**Export Certificate or Certificate Request**	This command is used to export 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. The -l<*label*> option specifies the object to export. If a certificate object with a matching label exists, it will be exported, otherwise a search will be made for a certificate request with a matching label.

Options

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

Option	Description
-b <*YYYYMMDDhhmmss[Z]*>	**--cert-begin**=<*YYYYMMDDhhmmss[Z]*> Specifies the begin time (notBefore) for a Certificate. The *Z* implies the time specified is GMT, otherwise Local time is assumed and converts to GMT. This option is only valid for the Generate Certificate Command (c), and must be used with either the -b<*YYYYMMDDhhmmss[Z]*> or -d<*duration*<*h*|*d*|*m*|*y*>> option.
–**c**<*label*>	**--ca-label=**<*label*> Specifies a label that identifies a CA (private) key used to sign a newly generated certificate. The <*label*> can be a label for a certificate that is associated with the CA key, or it can be the label of the private key itself. This option is only valid for the Generate Certificate Command (c).
-C<*curve_name*>	--**curve-name**=<*label*> Specifies which curve to use. Valid values are: • P-192 (also known as prime192v1 and secp192r1) • P-224 (also known as secp224r1) • P-256 (also known as prime256v1 and secp256r11) • P-384 (also known as secp384r1) • P-521 (also known as secp521r1) • c2nb191v1 • c2tnb191v1e • or any valid Domain Parameters object label • If -tec is specified, the –C parameter must be included in the command otherwise ctcert will exit with an error message.
-d <*duration*<*h*|*d*|*m*|*y*>>	**--cert-duration**=<*duration*<*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 provided it defaults to start from now. This option is only valid for the Generate Certificate Command (c).
-e <*YYYYMMDDhhmmss[Z]*>	**--cert-end**=<*YYYYMMDDhhmmss[Z]*> Specifies the end time (notAfter) for a Certificate. The *Z* implies the time specified is GMT, otherwise Local time is assumed and converts to GMT. This option is only valid for the Generate Certificate Command (c), and must be used with the –b<*YYYYMMDDhhmmss[Z]*> option.

Option	Description
–**f**< *name*>	**--import-file=**< *name*> Specifies a text file that contains a PEM encoding of a certificate or certificate request object. This option is only valid with the Import Command (i), Export Command (x) or Generate Certificate Request Command (r).
–**h, -?**	**--help** Display usage information
–**i**<*slot*>	**--ca-slot=**<*slot*> Specifies the slot containing the CA signing key identified by the –c<*label*> option. If the –i<*slot*> option is not used then the CA key is assumed to be in the slot identified by the –s<*slot*> option. If this latter option is not used then the CA key is assumed to exist in slot 0. If the CA signing key has the CKA_SIGN attribute set to FALSE and the CKA_SIGN_LOCAL_ATTRIBUTE set to TRUE, then the CA signing key must reside in the same slot as the certificate it is signing. This option is only valid when the –c option is used.
–**k**	**--key-gen** Specifies that a new key pair be generated. The -l<*label*> option specifies the label for the new keys. A key pair with the same label must not already exist.
–**l**<*label*>	**--label=**<*label*> Depending on the command and other options, this option specifies a label for a new or an existing certificate request or public key object. Refer to the description of each command for more details.
–s<*slot*>	**--slot=**<*slot*> Specifies the slot: • in which, a new key pair and a certificate or certificate request will be generated; or • into which, a certificate or certificate request will be imported; or • from which, keys, certificates and certificate requests will be listed; or • that contains the certificate or certificate request to be exported.
-S<*mechanism*>	--sig-hash-alg=<rsa_sign_alg> Specifies the RSA signing algorithm for certificate request or certificate generation. Valid mechanisms are: • SHA1 • SHA224 • SHA256 • SHA384 • SHA512 The default is SHA1. If this option is applied to a DSA or EC key pair and is not SHA1, ctcert will exit with an error message. **NOTE:** ECDSA is used for a certificate and EC is used for a key pair.

Option	Description
–**t**<*type*>	**--type=**<*type*> Use with the **–k** option to specify the key type that should be generated. The valid key types are *rsa, rsax931*, ec, and *dsa*. The default is *rsa*. If –tec is specified, the –C parameter must be included in the command, otherwise ctcert will exit with an error message.
–**x**< *name*>	**--attribute-file=**< *name*> Specifies a text file that contains certain certificate attributes and extensions. For details on the attributes and extensions supported and the format of this file refer to the Certificate Attribute/Extension File section below. This option is only valid with the Generate Certificate command (c).
–**z**<*bits*>	**--size=**<*bits*> Use with the **–k** option to specify the new key size in bits. The default key size is 1024 bits.

Certificate Attribute File

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

- Certificate label
- Certificate serial number
- Certificate issuer distinguished name
- Certificate subject distinguished name
- Certificate policies extension with support for a certification practice statement (CPS) or a 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.

Option	Description
Tag – *label*	This tag defines the certificate's label and is different to the label specified by the -l<*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 -l<*label*> option. The label can be any string of ASCII characters. If the label contains multiple words then 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. *label { Test Certificate }*
Tag – *serialnumber*	This tag defines the certificates serial number. However, to ensure uniqueness, it is only used if the signing key does not have the usage count attribute defined. If this attribute is defined then 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 for this information. The following example illustrates the correct use of this tag: *serialnumber { 999999 }*
Tag – *issuer* **Tag - *subject***	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: *issuer | subject* *{* *CN=<string> ,* *OU=<string> ,* *O=<string> ,* *C= <string>* *}* The meaning of each name component in each name/value pair is as follows: CN – Common Name OU – Organizational Unit Name O – Organization Name C – Country Name The following example illustrates a well formed issuer distinguished name: *issuer* *{* *CN=any string ,* *OU=Testing ,* *O=safenet ,* *C=AU* *}*

Option	Description
	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. For the **subject** distinguished name tag there exist two special strings that can be assigned to the CN (Common Name) component of the subject distinguished name. These special strings are "**serialno**" and "**unique**". When the **serialno** string is used, ctcert will replace the **serialno** string with the HSMs serial number. This can be used to distinguish certificates that belong to specific HSMs. 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.
Tag – *certificatepolicies*	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: *certificatepolicies* *{* *oid=oid_string ,* *[critical | noncritical ,]* *[unotice="<string>" ,]* *[cps="<string>"]* *}* 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: id-ce-keyusage OBJECT IDENTIFIER ::= { 2.5.29.15 } The critical / noncritical keywords are used to indicate whether this certificate policy extension is critical or not. By default the certificate policy extension is not marked critical. Multiple certificate policy extensions may be defined in the certificate attribute/extension file. The following example illustrates a well formed certificatepolicies extension: *certificatepolicies* *{* *oid=1.2.3.45.6.8 ,* *unotice=Test string,* *critical* *}*

Option	Description
Tag – *keyusage*	This extension is used to restrict the usage of the public key in the certificate. The format of the keyusage entry is: *keyusage* *{* *<key usage string> ,* *[<key usage string> ,]* *[critical | noncritical ,]* *}* The *<key usage strings>* conform to those defined in RFC2459 and acceptable values are: • digitalSignature • nonRepudiation • keyEncipherment • dataEncipherment • keyAgreement • keyCertSign • cRLSign • encipherOnly • decipherOnly The critical / noncritical keywords are used to indicate whether this key usage extension is critical or not. By default the key usage extension is marked critical. An example of a well formed keyusage extension is: *keyusage* *{* *digitalSignature ,* *keyCertSign* *}*

Examples

Example 1	Generate new DSA keys with label "Test" and self sign. This command will prompt for the subject distinguished name. **ctcert c -k -l**Test **-t**dsa
Example 2	Generate new RSA keys with label "Test" and key size 512 bites, sign with a key that has the label "CA Key". **ctcert c -c**"CA Key" **-k -l**Test **–z**512
Example 3	Generate a new ECDSA certificate, a EC key pair, and self-sign using the P-192 curve: **ctcert c –tec –CP-192 –lecdsa**Cert1 **–k** **NOTE:** Use ECDSA for a certificate and EC for a key pair.
Example 4	Use existing keys with label "Test" and use certificate attribute file. **ctcert c -l**Test **-x** certificate_file.txt
Example 5	Use existing keys with label "Test" and sign with a key with that has the label "CA Key". **ctcert c -c**"CA Key" **-l**Test
Example 6	Use existing certificate request with a label "Test Cert" and sign with a key that has the label "CA Key". **ctcert c -c**"CA Key" **-l**"Test Cert"
Example 7	To create a new certificate request with the label "User" and generate new keys (RSA is default) **ctcert r -k -l**User
Example 8	To export a previously generated certificate request as a PEM object and store this in the file name *mycert.txt*. **ctcert x –l**User **-f**mycert.txt

CTCHECK

SafeNet Cryptoki provider status enquiry utility.

Synopsis

ctcheck	[**-a**, **--all**] [**-b** *string*, **--device-details**=*string*] [**-d** *device*, **--device**=*device*] [**-f** *x* | *s*, **--format**=*x* | *s*] [**-g** *string*, **--global-details**=*string*] [**-h**, **--help**] [**-n**, **--number**] [**-N**, **--noglobals**] [**-s** *char*, **--separator**=*char*] [**-V**, **--version**]

Description

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

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 then the device driver package must be installed and running (check it with the **hsmstate/e8kstate**(1m) command). If a remote Cryptoki is used then its IP address must be given with the **CT_SERVER** environment parameter.

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

```
serialnumber~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 **--separator** option)

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 the Operation in WLD Mode and Operation in HA Mode sections for further details.

Options

The following options are supported:

Option	Description
-a	--all Print all device information (overrides -b options)
-b *string*	--device-details=*string* *string* specifies what device information to output. *string* is a ~ (tilde) delimited list of parameters to output. Enclose the string in "inverted commas" or 'apostrophes' to avoid interpretation of the separator characters by the shell. Parameters available: • 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'

Option	Description
-d *device*	**--device**=*device* Just print details for device number *device* (the first device is number 0)
-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. *string* is a **~** (tilde) delimited list of parameters to output. Enclose the string in "inverted commas" or 'apostrophes' to avoid expansion by the shell. Parameters available: • devicecount - Number of active devices. • applicationcount - Number of applications currently using Cryptoki or 'UNAVAILABLE' • totalsessioncount - Number of sessions open on all devices.
-h	**--help** Display usage information.
-n	**--number** Just print the number of devices
-N	**--noglobals** Don't print the global information
-s *char*	**--separator**=*char* Separator for output (default is **~** (tilde))
-V	**--version** Print the program version

Diagnostics

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

Examples

The default case:

```
ctcheck
# global info: devicecount~applicationcount~totalsessioncount~
1~1~0~
# device info: serialnumber~model~devicerevision~firmwarerevision...
1267~8000:PL450~G~1.35.02~3.10~1~1~1046528~1025264~11291712~
```

Default XML output:

```
ctcheck -fx
<?xml version="1.0" encoding="UTF-8"?>
<cryptoki>
        <devicecount>1</devicecount>
        <applicationcount>UNAVAILABLE</applicationcount>
        <totalsessioncount>0</totalsessioncount>
        <devicecount>1
        <applicationcount>1
        <totalsessioncount>0
        <device>
                <serialnumber>1267</serialnumber>
                <model>8000:PL450</model>
                <devicerevision>G</devicerevision>
                <firmwarerevision>1.35.02</firmwarerevision>
                <ptkcrevision>3.10</ptkcrevision>
                <deviceinitialised>1</deviceinitialised>
                <slotcount>1</slotcount>
                <totalpublicmemory>1046528</totalpublicmemory>
                <freepublicmemory>1025264</freepublicmemory>
                <freememory>11291712</freememory>
                <securitymode>Default (No flags set)</securitymode>
                <transportmode>None</transportmode>
                <batterystatus>GOOD</batterystatus>
                <eventlogfull>0</eventlogfull>
                <fmsupport>1</fmsupport>
        </device>
</cryptoki>
```

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

```
ctcheck -Nf x -b  "serialnumber~batterystatus"
<?xml version="1.0" encoding="UTF-8"?>
<cryptoki>
        <device>
                <serialnumber>1267</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, then notices, warning or alarms can be raised as appropriate. 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.

Synopsis

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]

Description

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 utilized to view the configuration. Any changes to the configuration should be made when operating in NORMAL mode. Refer to the Operation in WLD Mode and Operation in HA Mode sections for further details.

Options

The following options are supported:

Option	Description
–**a***device*	**--device-number=**<*device*> Use the admin token on the specified *device*
–**b***name*	**--fm-cert=**<*name*> FM validation certificate
–**c***slots*	**--create-slots=**<*slots*> Create *slots* new User slots
–**d***slot*	**--delete-slot=**<*slot*> Delete and remove User slot with ID *slot*. (You cannot delete the admin slot.)
–**e**	**--event-log** Prints the event log on stdout
–**f***flags*	Configures security flags. Security flags are used to implement security policies. • Multiple flags may be set simultaneously. For example the command: *ctconf –ftu* would set both the t and the u flags. • When flags are set, any flags set previously are cleared. Setting *flags* = 0 Clears all the flags and places the device in *SafeNet Default Mode* (Default <No flags set>). This security policy is described in the *Typical Security Policies* section SafeNet Default Mode. Use other *flags* values to set flags as follows: a FIPS Algorithms Only c No Public Crypto d DES Keys Even Parity Allowed e Entrust Ready F FIPS Mode (equivalent to –faclntu) i Increased Security Level l Mode Locked n No Clear PINs N Full Secure Messaging Encryption p Pure PKCS11

Option	Description
	t Tamper Before Upgrade u Auth Protection U Full Secure Messaging Signing E User Specified EC Parameters allowed Each of these flags is fully described in the Security Flag Descriptions section.
–g*file*	**--upgrade-fw**=<*file*> Upgrade firmware with *file*
–**h**	**--help** Display usage information
–i*file*	**--integrity-fw**=<*file*> Verify the authenticity/integrity of a firmware file by specifying its filename.
–j*file*	--**download-fm=**<*file*> Download FM module *file*
–k*file*	--**validate-fm=**<*file*> Validate FM module *file*
–l*fmid*	--**delete-fm** **--disable-fm** --fmid=<*fmid*> Disable/delete an FM module. <*fmid*> specifies 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** – will tamper the HSM if it is removed from the PCI Bus. This is the default mode. **1** **Single Transport Mode** – HSM will not be tampered by being removed from the PCI Bus. HSM will automatically change to No Transport Mode the first time an application calls C_Initialize() after an HSM reboot. **2** **Continuous Transport Mode** – HSM will not be tampered by being removed from the PCI Bus.
–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
–**t**	**--time-set** Sets the HSM clock to the same value as the host system. This command is only valid when the RTC Status is either HSMADM_RTC_UNINITIALIZED or

Option	Description
	HSMADM_RTC_STAND_ALONE. Refer to the *ProtectToolkit C Programmers Guide* (in the Appendix titled *HSMAdmin.h Library Reference*) for further details.
–v	**--verbose** Display extended status information
–x	**--tamper** This will cause the Key Store memory on the HSM to be erased (as if tampered) and made ready for re-initialization. The **–x** option is only available on hardware-based ProtectToolkit C implementations.
--rtc-adj-access-control-rule=[secs]:[count]:[days]	This option sets 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. **secs** total amount of deviation (in no. of seconds) within a guard duration. Range 1≤ secs ≤ 120. **count** total number of adjustment that can be made within the guard duration. Range 0≤count. 0 denotes that unlimited adjustments can be made. **days** the guard duration in number of days. Range 1≤ days ≤ 12. The separator ':' is a compulsory argument. However, the values for secs, count and days can be NULL. A NULL equates to no modification. For example: ctconf --rtc-adj-access-control-rule=12:0:1 ctconf --rtc-adj-access-control-rule=12:: ctconf --rtc-adj-access-control-rule=::4 The current settings for the RTC Adjustment Access Control Rule are displayed via the CTCONF –v command.
--rtc-adj-access-control=0|1	RTC Adjustment Access Control can be enabled once the RTC Adjustment Access Control Rule has been set. 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. 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 RTC Adjustment Access Control The current settings for RTC Adjustment Access Control are displayed via the ctconf –v command.

CTFM

Functionality Module Management utility for the ProtectToolkit C environment.

Synopsis

ctfm d [**-a**<*device*> | **-A**]

ctfm	**i**	[**-a**<*device*> | **-A**] [**–c**<*certFile*>] **-l**<*certLabel*> **-f**<*fmFile*>
ctfm	**q**	[**-a**<*device*> | **-A**]
ctfm	**v**	[**-a**<*device*> | **-A**] [**–c**<*certFile*>] **-l**<*certLabel*> **-f**<*fmFile*>

Description

The **ctfm** utility is designed for the ProtectToolkit C administrator and is used to manage functionality modules on Protect Server or Protect Host devices (HSMs).

With this tool it is possible to:

- Load a new FM (if an FM is already loaded then it is overwritten)
- Delete an FM so it becomes inactive
- Query the status of an FM (if any)
- Verify an FM file is correctly signed

In each case the operation may apply to all HSMs or an individually specified HSM.

By default, **ctfm** will report the FM state for the first device found.

The device Administrator PIN and Admin SO PIN must be initialized in order for these commands to run. The **ctfm** utility will prompt the operator for new PINs if it detects the PINs are not initialized.

When the commands are executed they may require the Admin PIN or Admin SO PIN of the HSM. When they are required the utility will prompt the operator for the values (unless the values have already been previously entered during the execution of the same command).

Event log entries are created when FMs are loaded or disabled. In order to create event logs correctly the HSM RTC should be initialized. If the **ctfm** utility detects that the RTC is not initialized, after alerting the operator and gaining approval to do so, it will initialize the HSM RTC to match the system clock.

In order 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 already present in the Admin token the utility will import the certificate from the file into the device's Admin token and set it to *Trusted*.

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 the Operation in WLD Mode and Operation in HA Mode sections for further details.

Commands

Command	Description
d	**Delete FM** This command is used to delete an FM so it becomes inactive on all or on an individual HSM.
i	**Import FM** This command is used to Load a new FM onto all or an individual HSM (if an FM is already loaded then it is overwritten). Existing FMs do not need to be disabled

Command	Description
	prior to executing this command. The command looks on the Admin Token of the device for a certificate label equal to the *certLabel* parameter. If the certificate object is present then the utility will ensure the certificate is set to *Trusted*. If the certificate object is not present then the utility will attempt to create a Trusted certificate from the contents of the *certFile*. If the *certFile* parameter is not provided the utility will assume the filename is the *certLabel* with .cert appended. For example, if the certificate label is *myfm* then the utility will search for a file named *myfm.cert.* The device Administrator PIN (and possibly the Admin SO PIN) will be required.
q	**Query FM Status** This is the default command and is used to query the status of an FM (if any) on all or an individual HSM. Use this command to obtain the name, version information and disable status of an FM or to see if an FM is loaded at all. No PINs are required to perform this operation.
v	**Verify an FM Signature** This command is used to verify that an FM file has been signed correctly without attempting to download the FM. The device Administrator PIN will be required. The behavior of the *certLabel* and *certFile* parameters is the same as is described for the *Import FM* command above.

Options

The following options are supported:

Option	Description
–**a***device*	--**device-number=**<*device*> Use the admin token on the specified *device* The first device is numbered 0. If this option is absent then the first device is implied.
-A	--**all-devices** Apply command to all available devices.
–**c***certFile*	--**fm-cert-file**=<*certFile*> FM validation certificate filename.
–**f***fmFile*	--**fm-file**=<*fmFile*> Name of file holding a new FM.
-h,**-?**	--**help** Display usage information.
–**l***certLabel*	--**fm-cert-label**=<*certLabel*> FM validation certificate object label.

CTIDENT

CTIDENT is a utility for establishing and maintaining trust between devices within the ProtectToolkit C environment.

Synopsis

ctident gen	[**-b**] [**-f**] [**-o** <*pin*>] <*targets*>
ctident trust	[**-b**] [**-f**] [**-o** <*pin*>] <*targets*> <*peers*>
ctident remove	[**-b**] [**-o** <*pin*>] <*targets*> <*peers*>
ctident list	[**-b**] [**-t** <*types*>] [**-a**] <*targets*>
ctident check	[**-b**] <*targets*>

Description

The **ctident** utility is used for establishing trust between devices. This includes operations performed by the administrative token **SO** to establish the trust as well as operations that can be performed by any user to verify trust relationships.

A device trusts another peer when the device holds the HSM Identity public-key of the peer in its administrative token.

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 the Operation in WLD Mode and Operation in HA Mode sections for further details.

Commands

When specifying the command, the user need only supply the minimum number of characters to uniquely distinguish the command.

Command	Description
check	The **check key command 'check'** is used to check HSM Identity keys for consistency on the devices specified by the **<*targets*>** parameter. Any anomalies will be reported. This command ensures that the peer keys match the device private key they represent, and ensures that all key objects have been created with appropriate security attributes.
gen	The **generate key command 'gen'** is used to generate the HSM Identity key-pair on the devices specified by the **<*targets*>** parameter. If a device already has an identity key a key will not be generated and a warning will be issued, unless the **-f** parameter is used to force key re-generation. When a key is re-generated, the existing key is destroyed BEFORE the new key has been generated to avoid any inconsistencies that could occur with multiple keys. 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.
list	The **list key command 'list'** is used to list summary information for HSM Identity

Command	Description
	keys located on the devices specified by the **<*targets*>** parameter. The **–t** parameter restricts the types of keys listed. By default all HSM Identity key types are listed. The **–a** parameter lists all of the non-sensitive attributes for each key.
remove	The **remove key command 'remove'** is used to remove HSM Identity keys from the devices specified by the **<*targets*>** parameter. 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. 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. 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.
trust	The **trust key command 'trust'** is used to add peer HSM Identity public-keys to the devices specified by the **<*targets*>** parameter. The <***peers***> parameter specifies one or more peer devices to trust. 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. 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. 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.

Parameters

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.

Option	Description
-b	**--batch** Batch mode. Do not prompt for anything, including PINS. If the required information was not supplied on the command line **ctident** will report an error.
-f	**--force** Force the command, even if the key already exists.
-o <*pin*>	**--so-pin=**<*pin*> Specifies the security officer (**SO)** PIN. Use of this operation is a security risk due to the tools command line being visible in the systems process list.
-t <*types*>	**--type=**<*types*> Specifies a comma separated list of key types. The available key types are: • **priv** — local private keys • **pub** — local public keys • **peer** — peer public keys • **all** — all key types

Exit Status

The **ctident** utility will return a zero(0) exit status when successful. A non-zero exit status is returned on an error. Warnings are **not** treated as errors.

CTLIMITS

CTLIMITS is a utility for establishing and managing usage limits on cryptographic keys within the ProtectToolkit C environment.

Synopsis

ctlimits ct	[-**U**<usertype>] -**k** <keyspec> [-**m**<message>] -**S**<serial_no> -**t**<tok_label> -**l**<target_label> -**i**<key_id> [-**d**<days>] [-**L**<limit>] [-**s**<date>] [-**e**<date>] [-**c**<cert_file_name>] filename
ctlimits pt	[-**U**<usertype>] [-O <objtype>] -**k** <keyspec> [-**i**<key_id>] filename
ctlimits up	[-**U**<usertype>] [-O <objtype>] -**k** <keyspec> [-**i**<key_id>] [-**C**<count>] [-**L**<limit>] [-**s**<date>] [-**e**<date>] [-**c**<cert_file_name>]
ctlimits vk	[-**U**<usertype>] [-O <objtype>] -**k** <keyspec> [-**i**<key_id>]

Description

The **ctlimits** utility is used to set and/or modify the usage limitation attributes of cryptographic objects within the ProtectToolkit C environment.

The utility will gracefully recognize older firmware and report meaningful error message.

Options

Option	Description

Option	Description
-U <*user*>	**--usertype**=<user> User type creating ticket – may be either 'SO' or 'USER' (default)
-k <*keyspec*>	**--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
-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_file_name*>	**--cert**=cert_file_name 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

Commands

Command	Description
ct	**Create ticket from offline specification** This command creates an SET ATTRIBUTES ticket in the file *filename*. This ticket may be presented to a PTK C HSM using the ctlimits pt command. The ticket is signed with the authority of the user type specified by -U option (or the CKU_USER if no –U option is provided). • 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.
pt	**Present ticket to HSM** 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. 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.
up	**Apply limit attributes directly** This command sets or updates attributes on the target object directly without making an intermediate ticket file. The object must be modifiable. 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. 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. After the command sets the new attributes it will lock the object by setting the CKA_MODIFIABLE to False (in a C_CopyObject operation). 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.
vk	**View key attributes** This command displays the current limits attributes of an object. To identify the target object the *–k* option must be provided. To further identify the target object the *-i* option may be specified. 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.

CTKMU

Key Management Utility for the ProtectToolkit C environment.

Synopsis

ctkmu c	[-**s**<*slot*>] [-**z**<*size*>] [-**g**] [-**k**<*numb*>] [-**p**] [-C <curve_name>] -**a**<*attribute*> -**n**<*name*> -**t**<*type*>
ctkmu d	[-**s**<*slot*>] -**n**<*name*>
ctkmu e	-**c**<*slot*>
ctkmu i	[-**s**<*slot*>] [-**2**] [-**w**<*name*>] (-**y** [-**m**] -**a**<*attribute*> -**n**<*name*> -**t**<*type*> | -**c**<*slot*> | <*filename*>)
ctkmu idp	**[-s<slot>]** -**n**<*name*> -**t**<*type*> **[-a**<*attribute*>**] <*filename*>**
ctkmu it	[-**s** <*slot*>] <*filename*>
ctkmu j	[-**s**<*slot*>] -**a**<*attribute*> -**n**<*name*> <*filename*>
ctkmu l	[-**s**<*slot*>] [-v] [-**n**<*name*>]
ctkmu m	[-**s**<*slot*>] -**a**<*attribute*> -**n**<*name*>
ctkmu p	[-**s**<*slot*>] [-**O**]
ctkmu rt	[-**s** <*slot*>] -**d** <*slotlist*>
ctkmu s	-**c**<*slot*>
ctkmu t	[-**s**<*slot*>] [-**l**<*label*>]
ctkmu x	[-**s**<*slot*>] [-3] [-**n**<*name*>] [-**w**<*name*>] (-**y** [-**m**] | -**c**<*slot*> [-**M**] | <*filename*>)
ctkmu x	[-**s**<*slot*>] [-**n**<*name*>] [-**w**<*name*>] [-**c**] [filename] [-**y**] [-**m**][-**M**] [-**j**] [--pkLabel] [--keyCertLabel] [--pkalgo] [--certalgo]
ctkmu xt	[-**s** <*slot*>] -**S** <*serial*> <*filename*>

Description

The **ctkmu** utility is used for ProtectToolkit C token management. This includes the 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 the Operation in WLD Mode and Operation in HA Mode sections for further details.

Commands

Command	Description
c	**Create Key command** 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. Appendix B 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.
d	**Destroy Key command** This command is used to delete a key on the specified token. This command will permanently destroy the key with the label specified with the –**n** parameter.
e	**Erase Smart Card command** This command is used to erase a smart card in the specified slot and will leave the smart card in an un-initialized state.
i	**Import Key command** This command is used to import keys previously exported with the export command (see below).
idt	**Import Domain Parameters command** This command is used to store Domain Parameters objects onto a Token. • The **–s** option indicates the slot e.g. –s1 for slot 1 – default is slot 0. • The **–n** option indicates the label of the new object. • The **–t** option specifies the key type, it may be 'ec' or 'dsa' or 'dh' but only 'ec' is supported. • The **–a** option allows attributes to be specified. Only the 'P' private and 'M' Modifiable attributes are allowed. The default attribute if –a option is missing is CKA_PRIVATE=false and CKA_MODIFIABLE=false. The filename option specifies a test file that contains the information required to construct the domain parameters. See appendix G for examples of how to construct a parameter file.
it	**Import Token command** This command is used to import a token image into the specified token. The **-s** parameter identifies the token that will be replaced with the imported token image, by default slot 0 is used. The <*filename*> parameter specifies the token image file to import. To complete this operation, **ctkmu** will prompt for the user PIN of the destination token. 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.
j	**Import Private Key** This command is used to import a Private Key and a Certificate from a PKCS #12 file format.
l	**List Information** This command is used to display information on the objects stored on the token in the specified slot. This command will list the actual keys, certificates and other objects, or, if the token is a smart card token previously used with the key export function information on

Command	Description
	that key backup set.
m	**The Modify Attributes command 'm'** is used to toggle the specified attributes. That is, change from TRUE to FALSE and vice versa or add the attribute if it does not exist.
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.' 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.
rt	**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. 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. To complete this operation, **ctkmu** will prompt for the user PIN of the source token. When replicating to an un-initialized token, **ctkmu** will prompt for the SO PIN of the destination token. If the device is running in FIPS mode, **ctkmu** will prompt for the device administrator PIN of the destination token.
s	**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.
t	**The Initialize/Reset Token command 't'** allows for existing tokens to be initialized or re-initialized. If the specified token contains an initialized token the current SO PIN will be prompted for before a new Token label may be specified and the token re-initialized. If the token is un-initialized this command will only operate if the 'No clear PINs' flag is not specified for the HSM (otherwise only the Administrator may initialize tokens with the ctconf utility). In this case the new SO PIN and label may be specified. Once the token has been reset or initialized a new user PIN may also be set.
x	**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. 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 PTK based HSMs. 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. 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. 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. • If the private key being exported is marked CKA_EXPORTABLE=TRUE and CKA_EXTRACTABLE=FALSE, the toolkit will prompt for Security Officer (SO) to login to perform the export operation.

Command	Description
	• 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 to be identical. OpenSSL, however, will extract Key and Certificate exported by **ctkmu** using two different passwords. The users need to decide which password policy best suit 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.
xt	**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.

Parameters

Option	Description
-a<*attributes*>	**--attributes** =<*attributes*> Specifies attributes for an object / key. Valid attributes are: **P** CKA_PRIVATE=1 **M** CKA_MODIFIABLE=1 **T** CKA_SENSITIVE=1 **W** CKA_WRAP=1 **w** CKA_EXPORT=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 **I** CKA_IMPORT=1
-c<*slot*>	**--sc-slot-num =**<*slot*> Specifies the Smart Card slot to export to or import from.
-C <curve_name>	**--curve-name** =<label> Specifies which curve to use. Valid values are: • P-192 (also known as prime 192v1 and secp192r1) • P-224 (also known as secp224r1) • P-256 (also known as prime 256v1 and secp256r1) • P-384 (also known as secp384r1) • P-521 (also known as secp521r1) • c2nb191v1

Option	Description
	• c2tnb191v1e • or any valid Domain Parameters object label If *-tec* is specified, the *–C* parameter must be included in the command otherwise *ctcert* will exit with an error message.
-d <*slotlist*>	**--dest =**<*slotlist*> 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.
-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 *N of M scheme* 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 any 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.
-P	**--pinpad** Use a supported pin pad device for entering key components.
-s<*slot*>	**--slot-num =**<*slot*> Specifies the slot to operate on. Default is 0 (zero), however must be specified when using the **l** command and **–v** option for Slot 0.
-S <*serial*>	**--serial =**<*serial*> Specifies the device serial number.
-t<*type*>	**--type=**<*type*> The type of key to create. Options are: aes | des | des2 | des3 |rc2 |rc4 |cast| idea| seed | rsa | dsa | ec.
-v	**--verbose** Displays the attributes that *ctkmu* may change.
-w<*name*>	**--wrap-key =**<*name*> Name of the key used to wrap or unwrap.
-y	**--console** Import/Export using the console.
-z<*size*>	**--size=**<*size*> Size of the key to create/import (for aes, rc2, rc4, cast, rsa, dsa and generic secret).
-2	**--Cprov2** Import keys from a Cprov 2 formatted file. This is used when migrating keys from an older Cprov 2 key format to the current format (see Appendix D).
-3	**--PTKC3** Generate export to smart card and file using the PTK C version 3 format. Used when exporting keys to be sent to older style HSMs.

Exit Status

The *ctkmu* utility will return a zero(0) exit status when successful. A non-zero exit status is returned on an error. Warnings are not treated as errors.

CTPERF

Performance reporting utility.

Synopsis

ctperf	[**-h**] [**-b**<bytes>] [**-c**] [**-e**] [**-i**<*count*>] [**-k**] [**-m**< *bits*>] [**-n**<*mechanism*>] [**-o**<*mechanism*>] [**-p**] [**-q**] [**-r**] [**-R**] [**-s**<*slot*>] [**-t**<*seconds*>] [**-v**] [**-x**] [**-z**<name>]

Description

The **ctperf** utility reports on the performance of PKCS #11 cryptographic operations.

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

Options

The following options are supported:

Option	Description
-b<*bytes*>	**--block-size=**<*bytes*> Specify the block size to use for the symmetric cipher tests. For example, -b8 specifies 8 bytes, -b8k specifies 8 kilobytes. Default size is 4 kilobytes.
-c	**--strict** Strict PKCS #11.
-C<*curve_name*>	**--curve-name**=<*label*> Specifies which curve to use. Valid values are: • P-192 (also known as prime 192v1 and secp192r1) • P-224 (also known as secp224r1) • P-256 (also known as prime 256v1 and secp256r1) • P-384 (also known as secp384r1) • P-521 (also known as secp521r1) • c2nb191v1 • c2tnb191v1e • or any valid Domain Parameters object label If a curve name is not specified, the default P-192 is used.
-e	**--EMC** Runs tests suitable for EMC testing purposes.
-h,-?	**--help** Display usage information.
-i<*count*>	**--iterations=**<*count*> The number of iterations of the performance tests to run. Default is 1, use -1 to specify an infinite count.
-k	**--keygen** Generation random keys (default uses fixed keys).
-m<*bits*>	**--modulus=**<*bits*> Modulus bit length.
-n<*mechanism*>	**--exc-mechanism=**<*mechanism*> Mechanisms to exclude from the test. This option may be repeated with additional mechanisms to specify more than one. See the **–o** option for a list of mechanisms. Default is no mechanisms.
-o<*mechanism*>	**--inc-mechanism=**<*mechanism*>

Option	Description
	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 *ProtectToolkit C Programmers Guide*.) The following mechanism tests are supported: **-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 **-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

Option	Description
	-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: dsa_sign dsa_verify **-o sym** all symmetric algorithm mechanisms **-o asym** all asymmetric algorithm mechanisms **-o ec** all EC DSA operations. That is: ecdsa_kg ecdsa_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 **-o extra** the following: 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 **-o mc** reading of the monotonic counter object
-p	**--dsa-params** Parameters generated for DSA.
-q	**--quick** Quick Keygen (key generation tests not performed).
-r	**--random** Execute a random selection of the performance tests.
-R	**--Random** Seed the random number generator. This option should be used with the **-r** option to generate a unique sequence of tests, otherwise the same pseudo random sequence will be repeated.
-s<*slot*>	**--slot-num=**<*slot*> Specify the slot number to perform test on.

Option	Description
-t<*seconds*>	**--time-period=**<*seconds*> Specify the measurement period. Default is 5 seconds.
-v	**--verbose** Verbose (provide more information).
-x	**--csv** Create a CSV (comma-separated variable) file.
-z<*name*>	**--cryptoki-module**=<*name*> Optionally, specify a different cryptoki module to use. May include full path.

CTSTAT

Show status for ProtectToolkit C tokens and objects.

Synopsis

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

Description

The **ctstat** utility is used to check the status of a token, determine what state the token is in and what, if any, objects it contains. With no arguments, **ctstat** will provide a summary report of all tokens found.

NOTE: When operating in WLD/HA mode, this utility should only be utilized to view the configuration. Any changes to the configuration should be made when operating in NORMAL mode. Refer to the Operation in WLD Mode and Operation in HA Mode sections for further details.

Options

The following options are supported:

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*>

Option	Description
	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).

CHAPTER 8
GUI UTILITIES REFERENCE

This chapter outlines the graphical user interface utilities supplied with the ProtectToolkit C software.

Key Management Utility

The Key Management Utility (KMU) provides a graphical user interface to functions that allow management of keys using a PKCS #11- sub-system. The functionality is identical to the command line utility "ctkmu" (Chapter 7).

NOTE: The KMU application is a Java based application. Thus it is necessary to have a working Java runtime installed which supports the Swing user interface. This application has been tested with JDK 1.2, JDK 1.3 and JDK 1.1 (with Swing installed). In addition the screen shots throughout this manual may vary from platform to platform.

NOTE: When operating in WLD mode, this utility should only be used to view the configuration. Any changes to the configuration should be made when operating in NORMAL mode. Refer to the Operation in WLD Mode section for further details.

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, consult the Key Management section in the ProtectToolkit J Reference Manual.

Please contact SafeNet for further details on its 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 PTK 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 PTK C v4.0 and V3.x HSMs. So, if you are exporting keys from a system running PTK C 4.1 or above for import to an older system then use the ctkmu and the -3 option.

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

Using KMU with the ProtectServer Blue

This version of the KMU is not compatible for use with a ProtectServer Blue HSM. If you require a KMU for use with this type of HSM please contact SafeNet for further details.

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 first 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. Un-initialized tokens are displayed as **"<Slot *n*>:<uninitialized token>**".

NOTE: The KMU is unable to initialize tokens. There are other utilities, such as "gctadmin", which can be used to initialize tokens.

Once a token has been selected, the user is given the option to login (***Figure 11***). The PIN is authenticated, and a list of keys and other objects within the token are displayed in the *Keys on Selected Token* box. Appropriate buttons on the toolbar are enabled as shown in ***Figure 10***.

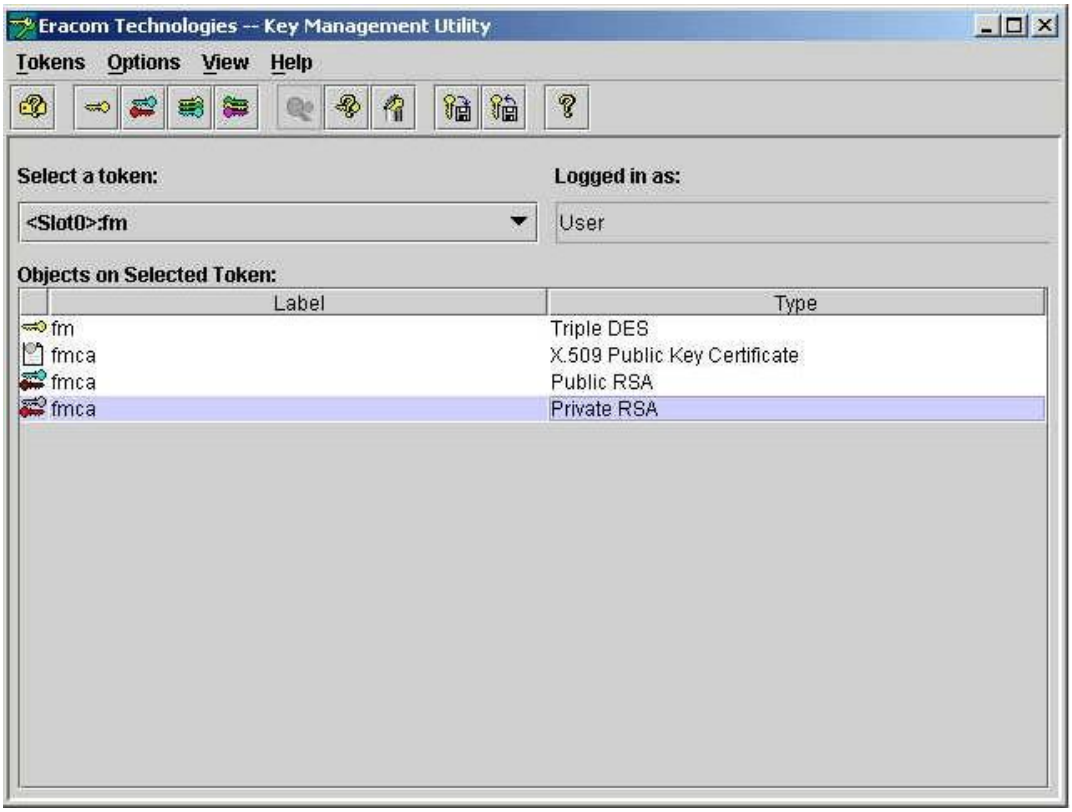
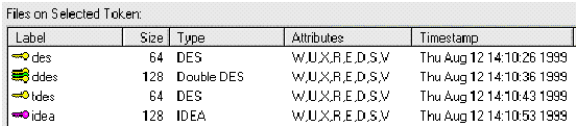


Figure 10 - Key Management Utility Main Interface

Token and Key Selection

The selection boxes are as follows.

Selection Box	Description
	Token Selection drop-down
	Key Selection box

The *Key Selection* box displays the objects currently stored on the selected token. This list displays the label and the type of each object. The entries in this list may be selected in order to perform the various functions.

NOTE: More than one key may be selected by drag-selecting to choose a range or SHIFT-LBUTTON to add/remove items to a selection. Operations that can accept more than one key will process all selected keys.

Toolbar Buttons

The buttons on the toolbar correspond to the following commands.

Button	Description	Button	Description
	Token Info		Delete Key
	Create Random Secret Key		Display Key Check Value
	Create Key Pair		Import Key
	Create Key & Components		Export Key
	Enter Key from Components		
	Edit Key Attributes		

The toolbar can be enabled or disabled from within the View menu. A check mark beside the selected toolbar item indicates the current view status. For example, to hide the toolbar menu from view, select ***View | Toolbar***, and make sure that there is no check mark beside the toolbar name.

Logging into a Token

1. Select an initialized token.
2. Select a user type and enter the PIN corresponding to the selected token (***Figure 11***).

   NOTE: Make sure that the CAPS lock is not on if the password contains lower case 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, however only public objects will be visible if this option is used (in addition, depending on the security policy for the token, various operations might not be possible – for example, key generation).

Figure 11 - Token Password Entry screen.

Logging Out from a Token

1. To log out from the current token, select **Tokens | Logout From Token** from the menu bar.

Setting the User's PIN

When the User's PIN is un-initialized it may be set by the security officer.

1. Choose **Tokens | Set User Pin** from the menu bar. The Set User PIN dialog appears (***Figure 12***).

   NOTE: The Security Officer must be logged in before being able to set the User PIN.

2. Enter the User Password into the appropriate fields. The User Password must be re-entered for validation.

3. Press **OK** to confirm your entry or **Cancel** to reject your input.

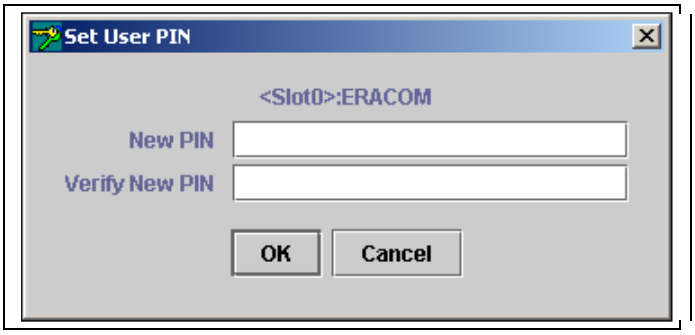


Figure 12 - Set User Password

The values for each password must be between 4 and 32 characters long and alphanumeric. All input fields echo characters with an asterisk (*).

Changing the PIN of the Logged on User

1. Choose ***Tokens | Change Pin*** from the menu bar. The Change Pin dialog appears (***Figure 13***).
2. Enter the Old User PIN and the New User PIN into the appropriate entry fields. The new User PIN must be re-entered for validation.
3. Press **OK** to accept or **Cancel** to reject your input.

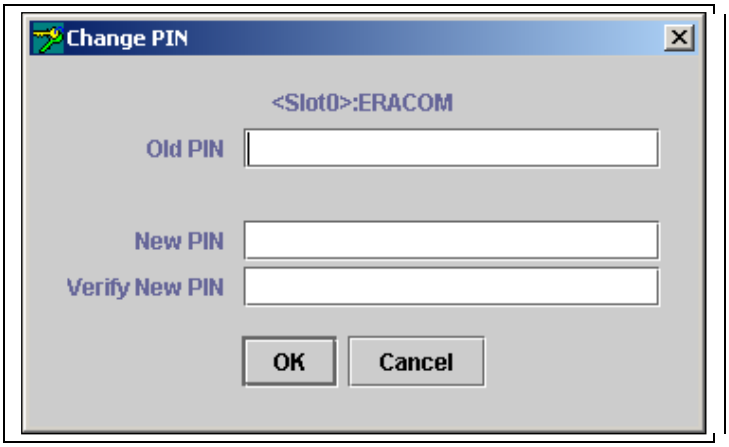


Figure 13 - Change PIN Dialog

The values for each password must be between 4 and 32 characters long and alphanumeric. All input fields echo characters with an asterisk (*).

Retrieving Information about a Token

1. Click the "Token Info" button n the toolbar, or choose ***Tokens*** **|** ***Token Info*** from the menu bar. The Token Info dialog will be shown (***Figure 14***).

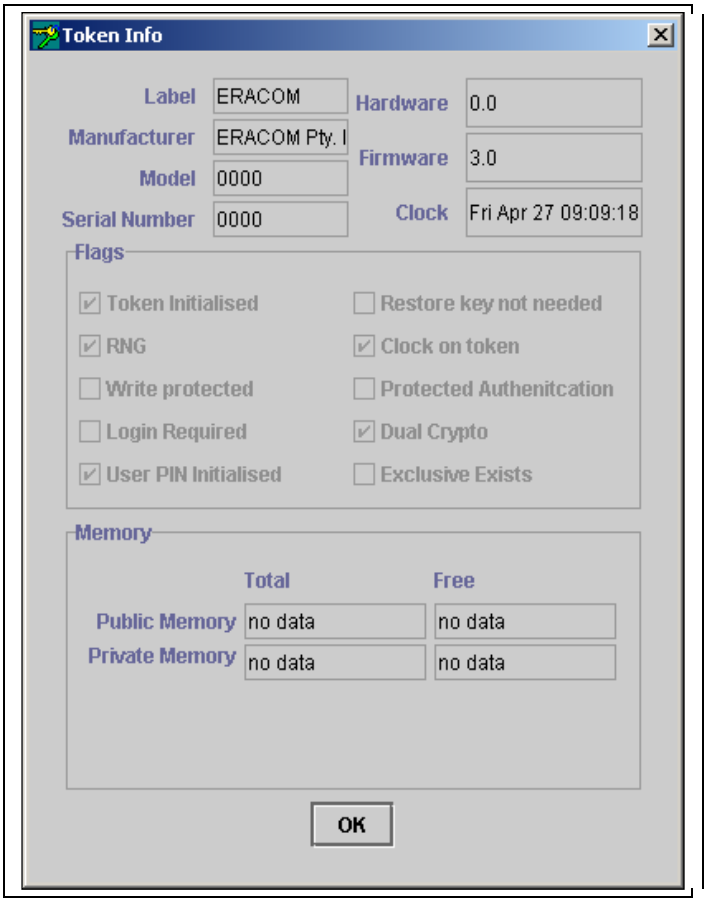


Figure 14 - Token Info dialog

For a more information on the items shown in this dialog, please refer to the PKCS #11 standard document.

Creating Keys

The KMU supports four key creation functions. These are:

- Creating a Random Secret Key
- Creating a Random Key Pair (for example, RSA public and private keys)
- Creating Key Components
- Entering a Key from Components

NOTE: To refresh the key information that is displayed on screen at any time, select **Options | Refresh** from the menu bar. The display is just a representation of what KMU has found on that token. If the token is modified by any other process or if for any reason the KMU is out of sync with the token. In such a case choosing this menu option will refresh the display.

The KMU also supports key export and import for the purposes of key backup and / or key escrow. This feature uses the PKCS #11 concept of key wrapping using high security key encryption keys (KEK) to wrap other key encrypting keys and / or data keys. The KEK is a special key that is created with the "wrap" attribute allowing it to be used for this purpose. KEKs are usually created as split custodian keys because of their high security nature.

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 entry is an important feature of this software since it allows the distribution of key material, in parts, across multiple trusted custodians for the highest level of security assurance where keys must be managed this way. To reconstruct any of the key material, all custodians must combine their components so that the key parts may be recombined into the original key(s). Key custodians may use smart cards for key component and authentication PIN data storage, or just use a disk file for key component storage.

Available Keys

The following different key types are available when selecting a key operation. A list of available key types is as follows:

Single Key Types	Key Pair Types
DES	RSA (Public)
Double DES	RSA (Private)
Triple DES	DSA (Public)
AES (16, 24, or 36 bytes)	DSA (Private)
IDEA	DH (Public)
CAST128 (1 to 16 bytes)	DH (Private)
RC2 (1 to 128 bytes)	EC (Public)
RC4 (1 to 256 bytes)	EC (Private)
SEED	

Key Attribute Types

You can specify what attributes a key will have when it is created. The following table describes the attributes which you can set when creating a key using the KMU.

Attribute	Description
Persistent	Stores the object on non-volatile memory. Persistent objects can be accessed after session termination.
Private	Defines whether the user PIN protects the object. A private object is only accessible to an application that has supplied the user PIN.
Sensitive	If a key is sensitive, the key's value cannot be revealed in plain text. Once a key becomes Sensitive it cannot be modified to be non-sensitive.
Modifiable	Indicates whether or not the object is modifiable, that is, if the object's attributes may be modified after creation.
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 *Token Selection* 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 (***Figure 15***).

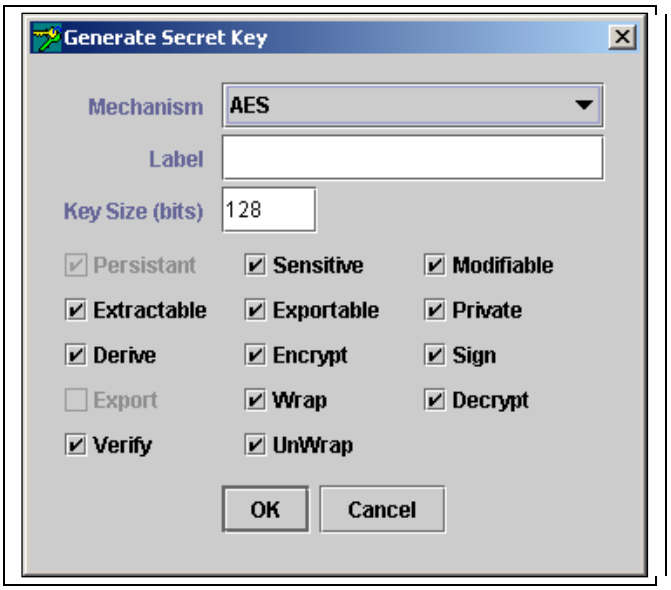


Figure 15 - Generate Secret Key Dialog

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 Key label for the key into the Label input field.

   The group of checkboxes shown in this dialog represent the various attributes which can be set for the selected key. There will be a default set of attributes checked for the selected key.
4. 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 "Key Selection" box on the main KMU user interface.

Creating a Random Key Pair

1. Select an initialized token from the *Token Selection* 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 (***Figure 16***).

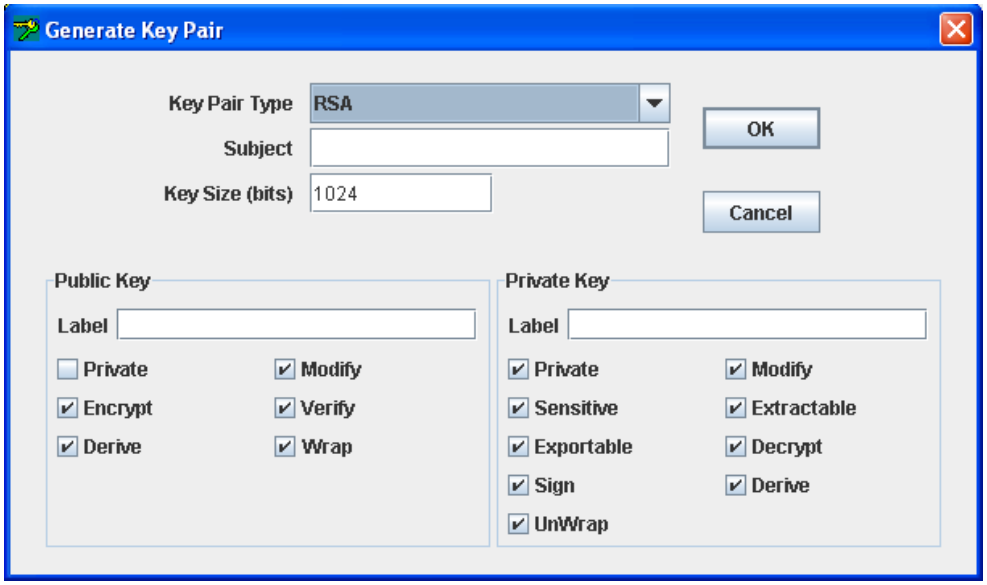


Figure 16 - Generate Key Pair dialog – when RSA selected

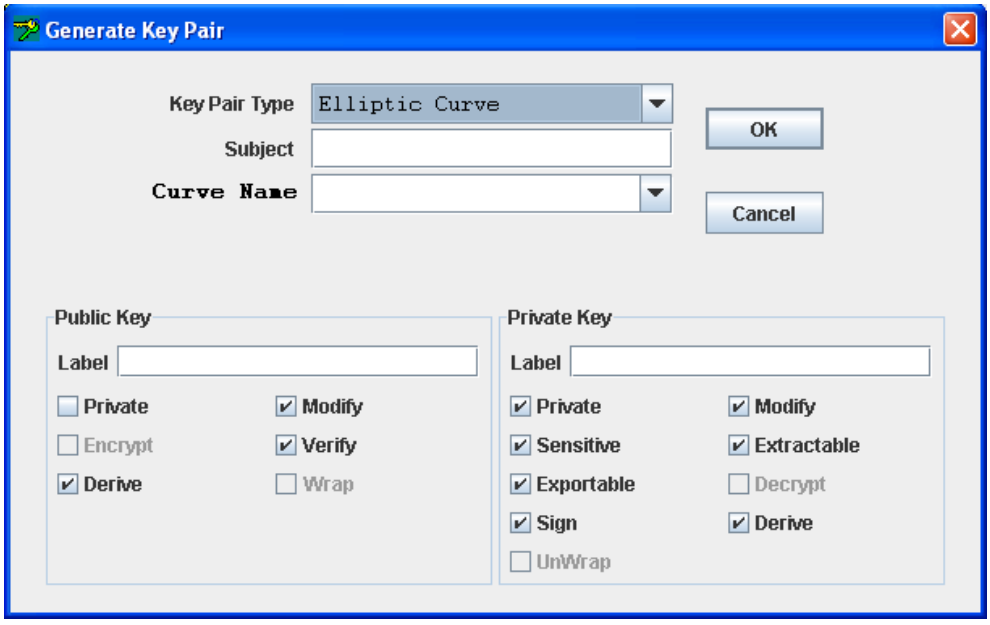


Figure 17 - Generate Key Pair dialog – when Elliptic Curve selected

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

   The "Subject" can be left blank, in which case there will be no X500 certificate information attached to the key pair. If you specify a "Subject", it must be set according to X.500 distinguished name syntax rules. For example,. C=AU, 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 PKCS #10 and X.509 certificates which are derived from these keys.

3. Specify the "Key Size (bits)" or "Curve Name" (only enabled if Key pair type is "Elliptic Curve"). Available curve names are:

   - P-192 (also known as prime192v1 and secp192r1)
   - P-224 (also known as secp224r1)
   - P-256 (also known as prime256v1 and secp256r1)
   - P-384 (also known as secp384r1)
   - P-521 (also known as secp521r1)
   - c2tnb191v1
   - c2tnb191v1e

4. Label both the public key and the private key.

   NOTE: The check boxes are enabled and disabled according to the selected Key Pair Type.

5. Press **OK** to generate the keys, or press **Cancel** to discard your input and return to the previous menu.

   Generated keys will be displayed under the *Key Selection* 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. With this function it is possible to create a key whose value is unknown to any single party. However, by combining the components known by each custodian the key may be regenerated. Each component that is generated is random and in itself does not expose any portion of the final key value.

1. Select an initialized token from the *Token Selection* drop-down.
2. Choose **Options | Create | Key Components** from the menu bar, to open the *Create Key Components* dialog box (***Figure 18***).

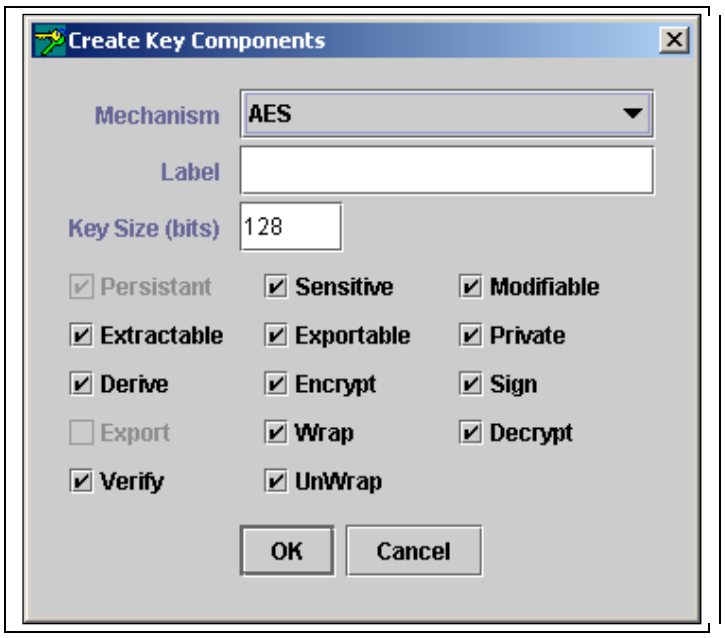


Figure 18 - Create Key Components Dialog

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 in the *Key Size (bits)* field the size of the key that you wish to generate.
6. Decide the attributes for the key 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.

   The number of component screens that display will correspond to the number of components that were specified in step 8.

   An example *Component Generation* dialog is shown in the next figure.

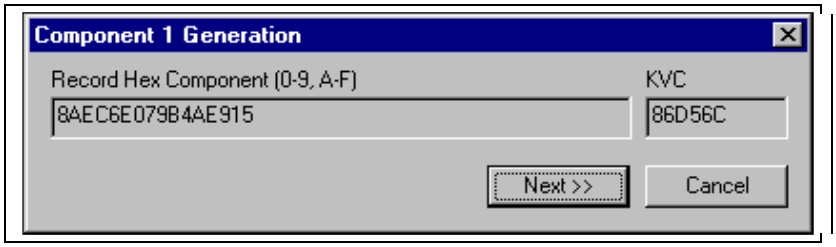


Figure 19 - Component Generation Dialog

10. Record the Component Value and Key Check Value (KCV), both given in hexadecimal, displayed in these dialogs. The KCV for the generated component may be used to verify correct entry of the component when performing manual key component entry.

Entering a Key from Components

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

1. Select an initialized token from the *Token Selection* 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 (***Figure 20***).

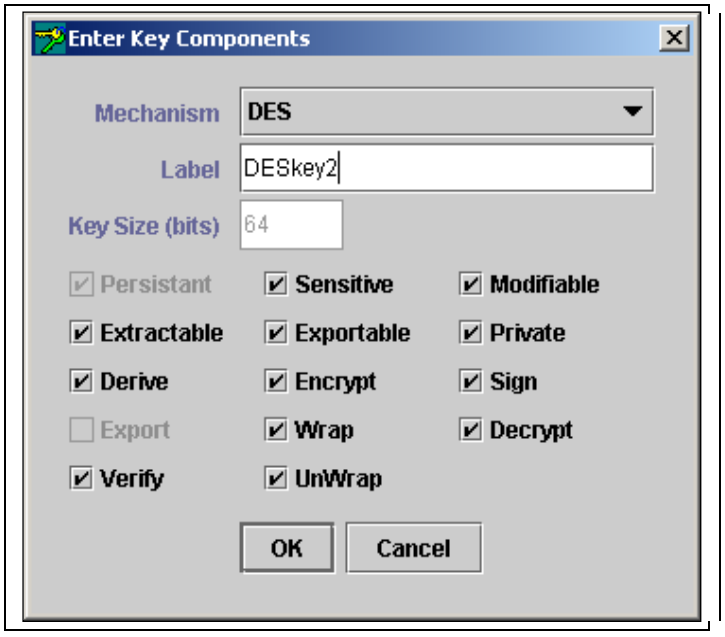


Figure 20 - Enter Key Components Dialog

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 in the *Key Size (bits)* field the size of the key that you wish to generate.

5. Decide the attributes for the key 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 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.
8. Click **OK** to continue and open the ***Ready to accept component*** dialog (***Figure 21***), or **Cancel** to abort this operation

   The number of components screens requiring input, corresponds to the number of components specified in the *Enter Key* dialog.

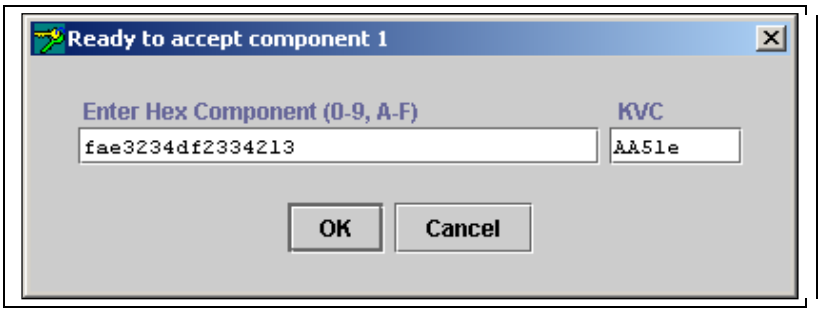


Figure 21 - Component Entry Dialog

NOTE: The KMU will check that the KCV entered matches that of the key components being input. If a mismatch is detectecd, an error is shown.

NOTE: The component entry may be masked by selecting ***Options | Mask Component Entry*** before beginning the operation.

Key check value (KCV) of symmetric keys can be displayed by selecting a key and clicking **View** on the toolbar. Alternatively you may also select ***Options | View*** from the menu bar.

Refer to Appendix B for details on how the KCV is calculated.

Editing Key Attributes

The attributes you can 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.

1. Double-click on the key you want to edit.
2. In the *Edit Attributes* dialog box, check the active boxes for the attributes you want to change (***Figure 22*** and ***Figure 23***).

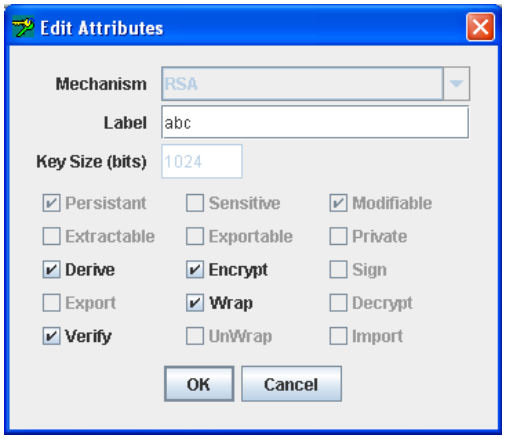


Figure 22 - Edit Attributes Dialog – for RSA keys

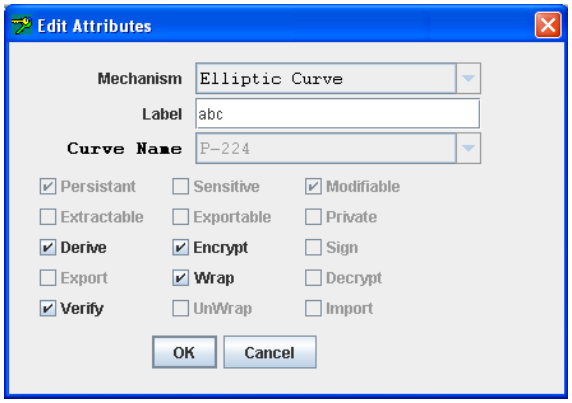


Figure 23 - Edit Attributes Dialog – for EC keys

Available curve names when **Elliptic Curve** is selected are:

- P-192 (also known as prime192v1 and secp192r1)
- P-224 (also known as secp224r1)
- P-256 (also known as prime256v1 and secp256r1)
- P-384 (also known as secp384r1)
- P-521 (also known as secp521r1)
- c2tnb191v1
- c2tnb191v1e

Deleting a Key

1. Select an initialized token from the *Token Selection* drop-down box.
2. Select the key that you want to delete from the *Key Selection* 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 having to reveal anything about the actual key value, by checking the Key Check Value (KCV) for the key.

The KCV is a standard technique for obtaining a fingerprint from a key for identification purposes. The mechanism used is compatible with AS 2805 and is simply the first three hex digits obtained by encrypting binary zeros with the key. Please refer to Appendix C for details of the KCV generation.

To display the KCV for a key (***Figure 24***):

1. Select a key and click the **View** button on the toolbar. Alternatively, select **Options | View** from the menu bar.

Figure 24 - Display KCV Dialog

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 the section Secure Key Backup and Restoration in

Operational Tasks 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.
- If backing up using a wrapping key, created a wrapping key. Refer to Creating Keys for details on how to create keys.

To export a key (or set of keys):

1. From the Key Management Utility main interface (***Figure 10***), under *Select a token*, select the token containing the key(s) to be exported and log on to the token.

   The *Objects on Selected Token* table is populated with values for the selected token.

2. Select one or more keys to export from the *Objects on Selected Token* table.

3. Right click on the selected key(s) or go to the *Options* menu. Then select **Export**. Alternatively, click on the **Export Key** button on the toolbar

   The *Export Keys* dialog box displays (***Figure 25***). Details of selections appear in the *Selected Token* and *Selected Key(s)* fields.

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:**
Multiple custodians method	*<Random key>*
Single custodian method*	The particular wrapping key required This key is then used to encrypt the key (or set of keys) to be exported

   * Note that wrapping keys must have been previously created. Refer to the Creating Keys section for details on how to create keys.

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.

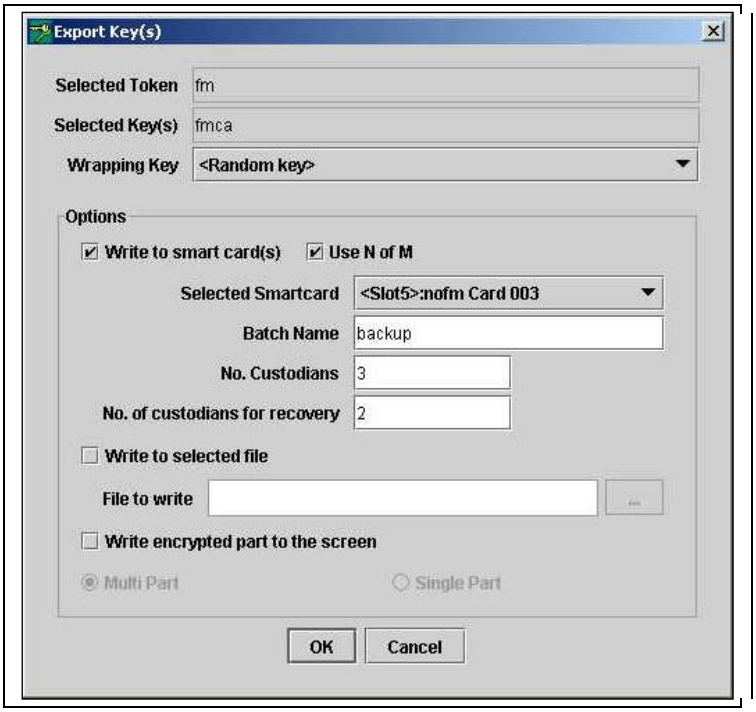


Figure 25 - Export Keys Dialog Box with Use N of M selected

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 (***Figure 26***).

Figure 26 - Smart Card User PIN entry dialog box

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.
8. Click **OK** to move on.
9. If a new PIN was entered, a prompt for the old PIN displays. Enter the old PIN to complete the change.

   If an incorrect smart card PIN is entered more times than the number specified for the card during its initialization, the smart card will become blocked. The card may then only be un-locked by entering the Security Officer PIN. Refer to the smart card initialization section for further details.

   Data is now written to the smart card. If additional key shares are to be written to smart cards then a prompt for the next smart card displays.
10. Remove the smart card from the smart card reader and repeat steps 11 to 14 until all the key shares required have been written to smart cards.

    When the operation is complete, an *Export Successful* message box displays.
11. 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 the section Secure Key Backup and Restoration.

Operational Tasks for all the necessary background information on backup and recovery methods, key splitting schemes and key attributes.

To import a key (or set of keys):

1. From the *Token Selection* drop-down box select the token that is to receive the imported keys and click the *Import Keys* button on the toolbar. Alternatively select **Options | Import** from the menu bar.

   The *Import Key(s)* dialog displays (***Figure 27***).

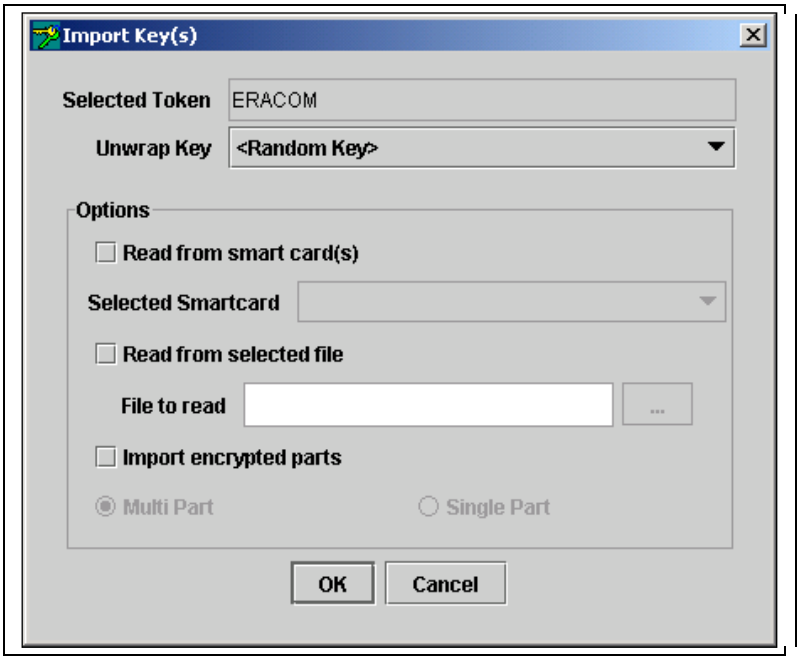


Figure 27 - Import Key(s) dialog box

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 backup 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:
Multiple custodians	*<Random key>*
Single custodian	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.
5. Click **OK** to start the import operation, or **Cancel** to abort.
6. The following dialog box, displaying the current card number and batch name, prompts for the smart card PIN.

Figure 28 - Smart Card Key Import – PIN request dialog box

7. Enter the PIN for the smart card and click **OK**.

   If an incorrect smart card PIN is entered a prompt will display to enable re-entry. If an incorrect smart card PIN is entered more times than the number specified for the card during its initialization, the smart card will become blocked. The card may then only be un-locked by entering the Security Officer PIN. Refer to the smart card initialization section for further details.

   If a smart card is from a different batch is inserted or if the card has already been read it will be rejected. A prompt will display to insert another card.

   Data is now retrieved from the smart card. If additional key shares are required to recover the key(s) then a prompt for the next smart card displays.

8. 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.

9. 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.

Administration Utility (GCTADMIN)

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" (Chapter 7).

NOTE: The GCTADMIN application is a Java based application. Thus it is necessary to have a working Java runtime installed which supports the Swing user interface. This application has been tested with JDK 1.2, JDK 1.3 and JDK 1.1 (with Swing installed). In addition the screen shots throughout this manual may vary from platform to platform.

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

To start the admin utility when using Microsoft Windows, locate the program folder titled "ProtectToolkit c Runtime" 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.

Selecting **Help** from the main menu can retrieve information about the current version of the software.

Logging In

After starting GCTADMIN, the utility will check if the HSM hardware has been initialized.

If the hardware has not been initialized, the operator will be prompted to initialize the admin token. For full details regarding initial configuration, please refer to Chapter 4. The purpose of initialization is 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.

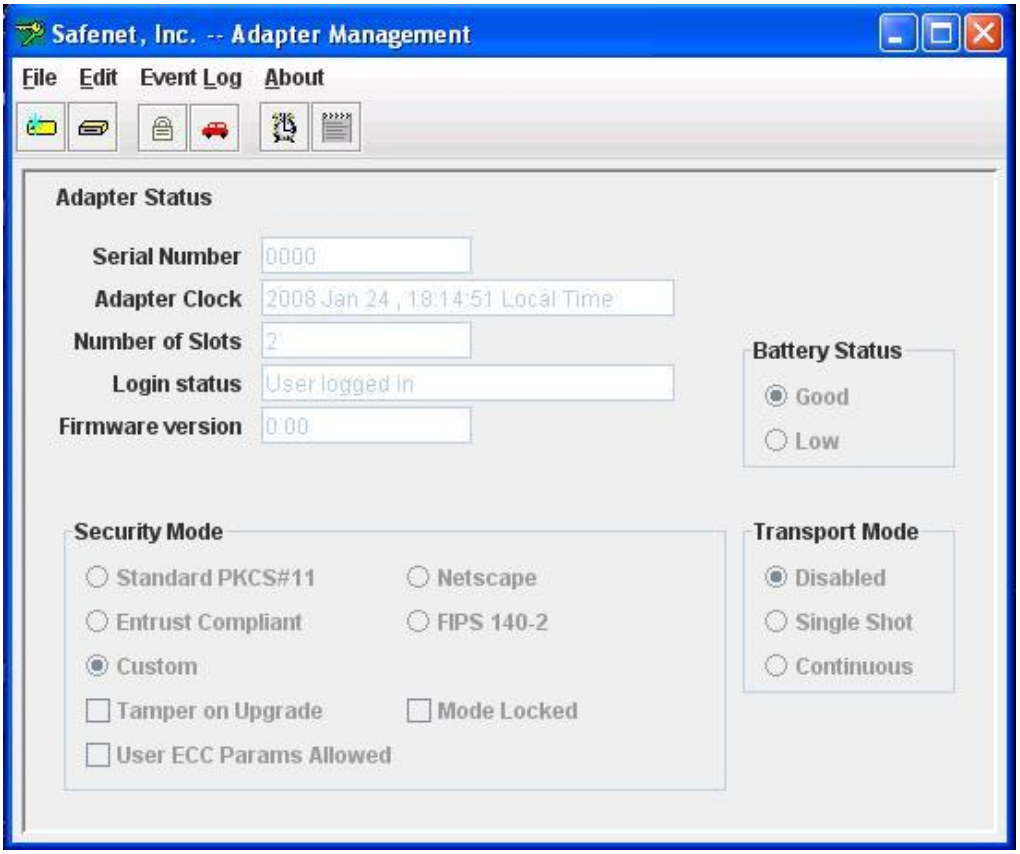
Logging Out

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 (***Figure 29**). The main interface shows the currently selected HSM and a variety of settings pertaining to the hardware.

In a host system which contains multiple HSMs, other HSMs can be selected by opening the **File** menu and choosing the **Select Adapter** option. Choosing a different HSM will require a new login.



\Figure 29 - Main GCTADMIN interface

Toolbar Buttons

The buttons on the toolbar correspond to the following commands.

Button	Description	Button	Description
	Token Configuration		Set Transport Mode
	Create Slots		Synchronize Clock
	Set Security Policy		View Event Log

Slot and Token Management

Creating Slots

1. To create slots on the HSM, select **Create Slot** from the **File** menu, 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.

1. To remove a slot, select **Delete Slots** from the **File** menu. A list of available slots is displayed. Select the slot to delete from the list and click the **Delete** button.

   NOTE: The slot containing the admin token cannot be deleted.

Initializing a Token

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

1. To initialize a token, select **Tokens…** from the **Edit** menu to open the *Manage Tokens* dialog (***Figure 30***).

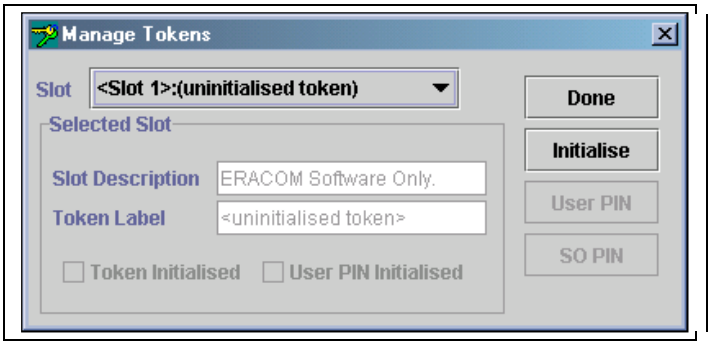


Figure 30 - Manage Tokens dialog

2. Select an un-initialized 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 does not have to be set until an application requires storage on that slot.

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

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

Setting the Token User PIN

1. To set a token User PIN, select **Tokens…** from the **Edit** menu.
2. Select an initialized token from the slot drop-down box, then click **User PIN**. If the selected token does not have a current User PIN, the dialog will prompt for the SO PIN in order to authorize the creation of the new User PIN.

   If the selected token already has an assigned User PIN 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 **Tokens…** from the **Edit** menu.
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.

   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.

1. Select an un-initialized token from the slot drop-down box, and then click **Reset** 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.

   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

It is the Administrator's task to choose the settings, or *Security Policy*, which will ultimately determine how ProtectToolkit C can be used. This is the most important aspect of ProtectToolkit C administration. Therefore, the Administrator is strongly advised to read Security Policies and User Roles which explains in detail how different settings affect the security and performance of the ProtectToolkit C environment.

1. To set the HSM security policy, select **Security Mode** from the **Edit** menu.
2. Select the settings as required.
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. The mode can be set to the following:

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 change to No 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 response.

Clock Drift Correction

Due to host system and HSM timing differences, the clock drift correction is used to synchronize the HSM and host system clock.

NOTE: It is not possible to directly specify a value for the HSM clock. It is only possible to synchronize the HSM clock with the host system clock.

1. To synchronize the HSM clock, select **Clock** from the **Edit** menu.
   The current value of the HSM clock is displayed.
2. Synchronize the host and HSM clock by clicking **Synch**.

Viewing and Purging the System Event Log

ProtectToolkit C maintains a system event log as a means of tracking serious hardware or consistent operational faults. For full details on what the event log stores and how to interpret its data, please refer to the Using the System Event Log section in Operational Tasks.

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

NOTE: The event log cannot be purged until it is full.

1. To view the event log, select **Event Log View** from the **Event Log** menu.

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

2. To purge the event log, select **Event Log Purge** from the **Event Log** menu.

   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.

NOTE: Subject to the security policy, the HSM might do 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 for more information on security policies.

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

1. Before a firmware upgrade, ensure that you have done the following:
   - 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
2. To upgrade the firmware, select **Upgrade Firmware** from the **File** menu.
3. 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, the user is shown a dialog stating the success or failure of the upgrade operation.

Tampering the HSM

The tampering of the HSM may be necessary at the end of its lifecycle or any other security sensitive event that requires all stored data to be immediately destroyed.

A tamper formats the secure memory of the HSM and thereby erases all configuration and user data. To tamper the HSM:

1. Select **Tamper Adapter** from the **File** menu.
2. In response to the prompt, confirm the desired action.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A
EVENT LOG ERROR TYPES

The following table lists the error entries that may be generated by the ProtectServer HSM firmware and written to the HSM's event log.

Name	Description
POST_ERR_SRAM_WRITE	POST Error: Cannot write to SRAM
POST_ERR_SRAM_READ	POST Error: Cannot read from SRAM
POST_ERR_SDRAM_DATA_STUCK	POST Error: SDRAM, bit stuck
POST_ERR_SDRAM_DATA_SHORT	POST Error: SDRAM data bits short Param 1. Bit number Param 2. Value
POST_ERR_SDRAM_ADDR_STUCK	POST Error: SDRAM address bit stuck
POST_ERR_SDRAM_ADDR_SHORT	POST Error: SDRAM address bits short Param 1. Bit number
POST_ERR_SDRAM_BAD_BYTESEL	POST Error: SDRAM bad bytes select
POST_ERR_BAD_SECTOR0	POST Error: POST Sector checksum is not correct
POST_ERR_NOMEM	Cannot allocate memory
POST_ERR_OS_HASH	The OS hash value is incorrect
POST_ERR_KAT	Known answer test failed Param 1. Algorithm Identifier Param 2. Error Code
POST_ERR_RNG	RNG did not pass chi-squared test
POST_ERR_NO_THREAD	Unable to start POST Thread
POST_ERR_SMFS	Secure memory file system error Param 1. Error Number
POST_ERR_RTC	Unable to access RTC
POST_ERR_SER	Unable to access UART
EXCEPT_UNDEF	An undefined instruction has been executed Param 1. Address Param 2. Instruction
EXCEPT_SWI	A software interrupt generated Param 1. Address Param 2. Instruction
EXCEPT_PREFETCH	A Prefetch abort generated Param 1. Address
EXCEPT_DATA	A Data abort generated Param 1. Address
EXCEPT_IRQ	An unhandled IRQ received Param 1. Identifier
ERR_HOT_TAMPER	Hot tamper detected
LOG_FIRST_ENTRY	Initial event entry
LOG_INITIALIZING_SRAM	Initializing the SRAM after a tamper
LOG_EVENT_LOG_PURGED	Event log has been purged
ERROR_ASSERT	Runtime Assertion Param 1. File Param 2. Line
ERROR_INIT_RESOURCE	Out of resources in initialization

Name	Description
	Param 1. File Param 2. Line
ERROR_INIT_PLATFORM	Failed to detect hardware platform Param 1. File Param 2. Line
HEAP_INVALID_ADDRESS	Heap Invalid block address Param 1. Heap number Param 2. Address
HEAP_MEM_FREED_TWICE	Heap: Memory Freed twice Param 1. Address
PCCISES_TIMEOUT	PCCISES: Timeout error on device Param 1. Error
PCCISES_BAD_STAT	PCCISES: Bad device status Param 1. Status
PCCISES_BAD_DATA	PCCISES: Bad input data
PCCISES_RNG_STUCK	PCCISES: Continuous RNG test error Param 1. Value
PCCISES_LNAU_EXCEPTION	PCCISES: Large Number Arith Hardware exception (Unit,0)
PCCISES_FAILED_RESET	PCCISES: Failed to reset
PCCISES_RESOURCES	PCCISES: Insufficient resources to start driver
CPROV_OS_UPGRADED	OS Upgrade performed Param 1. Mod Param 2. Version
CPROV_OS_UPGRADE_FAILED	OS Upgrade failed
PROT_NO_SMPR	PROTECTION: HSM SMPR not found
PROT_CIPHER_ERROR	PROTECTION: Cipher operation failed
KEYGEN_ERR_PAIRWISE	Key generation: Pair-wise consistency failure
FM_OP_DOWNLOAD	FM Download Performed Param 1. Mod Param 2. Version
FM_OP_DISABLE	FM Disabled Param 1. Mod Param 2. Version
FM_MODULE_FAILED	FM failed to load Param 1. Mod Param 2. Version
PTKC_CFG_CHNG	PTK C config change Param 1. New Val Param 2. Old Val

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B
PKCS #11 ATTRIBUTES

Objects as described by PKCS #11 consist of a number of attributes that define both the *object* and its *access policy*. In general the 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.

CKA_LABEL

This attribute specifies a textual label for an object. This label is used to assist in differentiating the various objects stored on a token.

NOTE: Although ProtectToolkit C does not require this attribute to be unique, various tools may do so.

CKA_CLASS

This attribute is assigned by the system when an object is created. There are a number of classes in common use:

- CKO_PUBLIC_KEY
- CKO_PRIVATE_KEY
- CKO_SECRET_KEY
- CKO_CERTIFICATE
- CKO_CERTIFICATE_REQUEST
- CKO_DATA

CKA_KEY_TYPE

This attribute specifies the key type associated with the object. There are many key types supported by ProtectToolkit C. For example:

- CKK_AES, CKK_DES, CKK_DES2, CKK_DES3, CKK_RSA, CKK_DSA
- CKA_ENCRYPT
- CKA_DECRYPT
- CKA_SIGN
- CKA_VERIFY
- CKA_WRAP
- CKA_UNWRAP

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.

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 *security officer* 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 *No Clear PINs* flag is set only sensitive keys may be created on the HSM.

CKA_EXTRACTABLE / CKA_EXPORTABLE

These attributes are used to specify that the key may be extracted from the token in an encrypted (for example, wrapped) form. These attributes determine how the key may be backed up. Please consult the key backup section in Chapter 6 for more information.

APPENDIX C
KMU KEY CHECK VALUE (KCV) CALCULATION

The key management utility calculates and displays keys according to AS 2805.6.3.

Single-length Key KCV

The single-length key check value is a one-way cryptographic function of a key which is 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 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 (***Figure 31***).

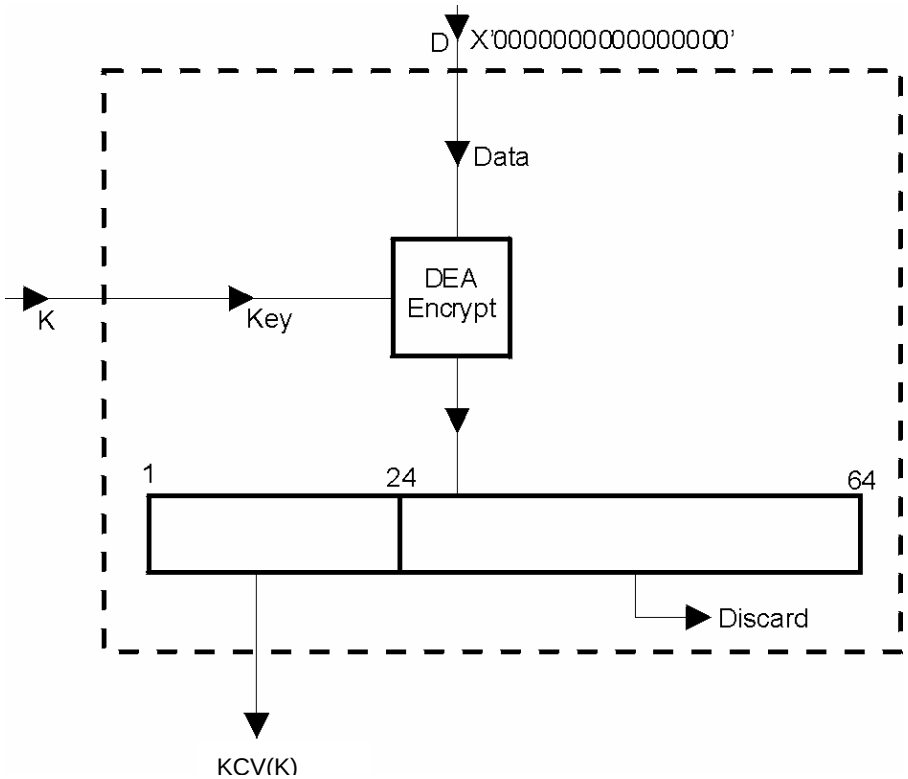


Figure 31 - Single-length Key Check Value KCV(K).

Double-length Key KCV

The single-length key check value is a one-way cryptographic function of a key which is 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 (***Figure 32**).

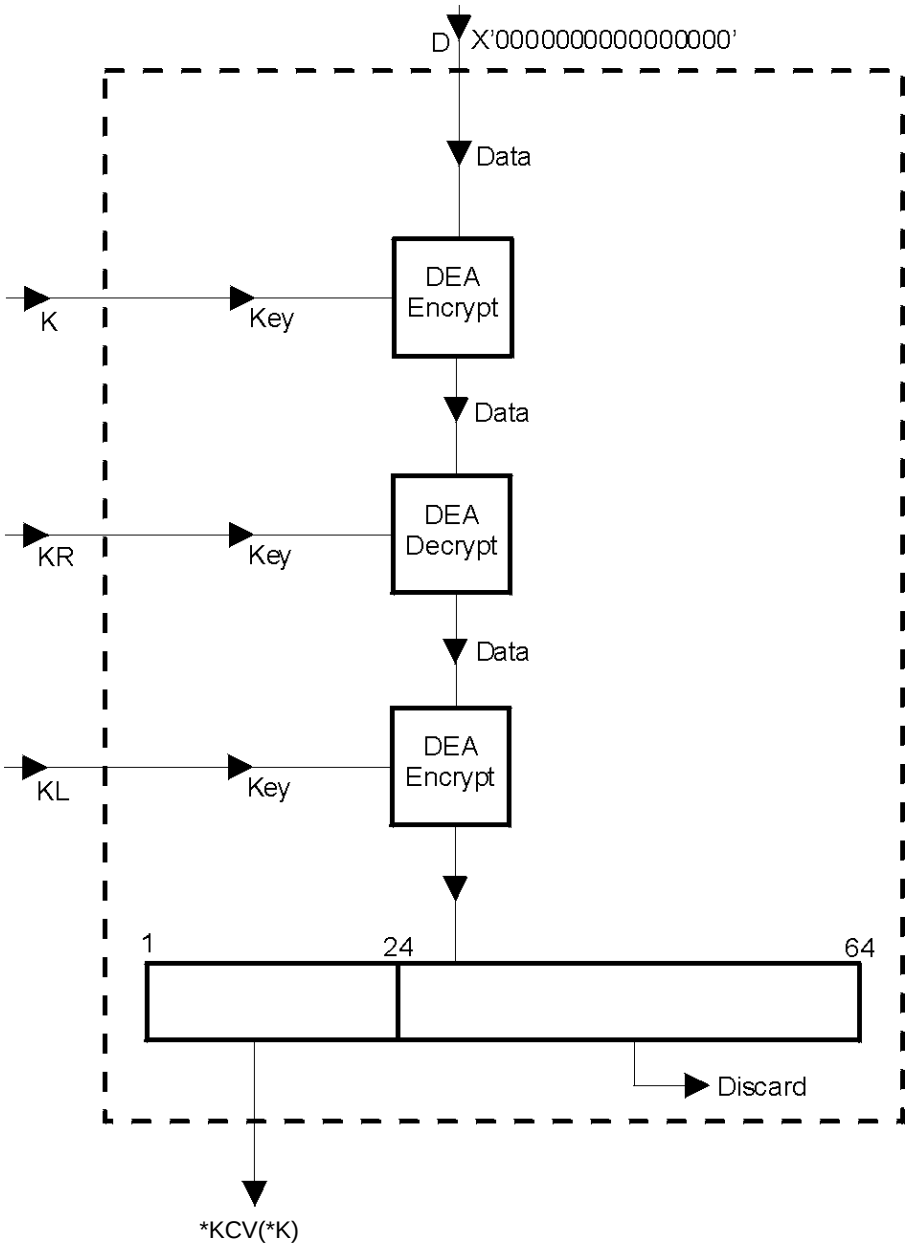


*Figure 32 - Double length Key Check Value *KCV(*K)*

APPENDIX D
KEY MIGRATION FROM PROTECTTOOLKIT C V4.1

Overview

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 by may be disabled of 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.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E
EXTERNAL KEY STORAGE APPLICATION NOTE

The secure memory available on ProtectServer HSMs is limited to 4MB.

Applications in which secure memory requirements exceed those stated above can utilize the External Token Support Library (ExtToken) to overcome this limitation. ExtToken facilitates secure, external-to-the-HSM storage for token objects. ExtToken manages external token support transparently to host applications. Host applications can utilize standard PKCS#11 function calls to access and manipulate token objects as though the token objects were stored on the HSM.

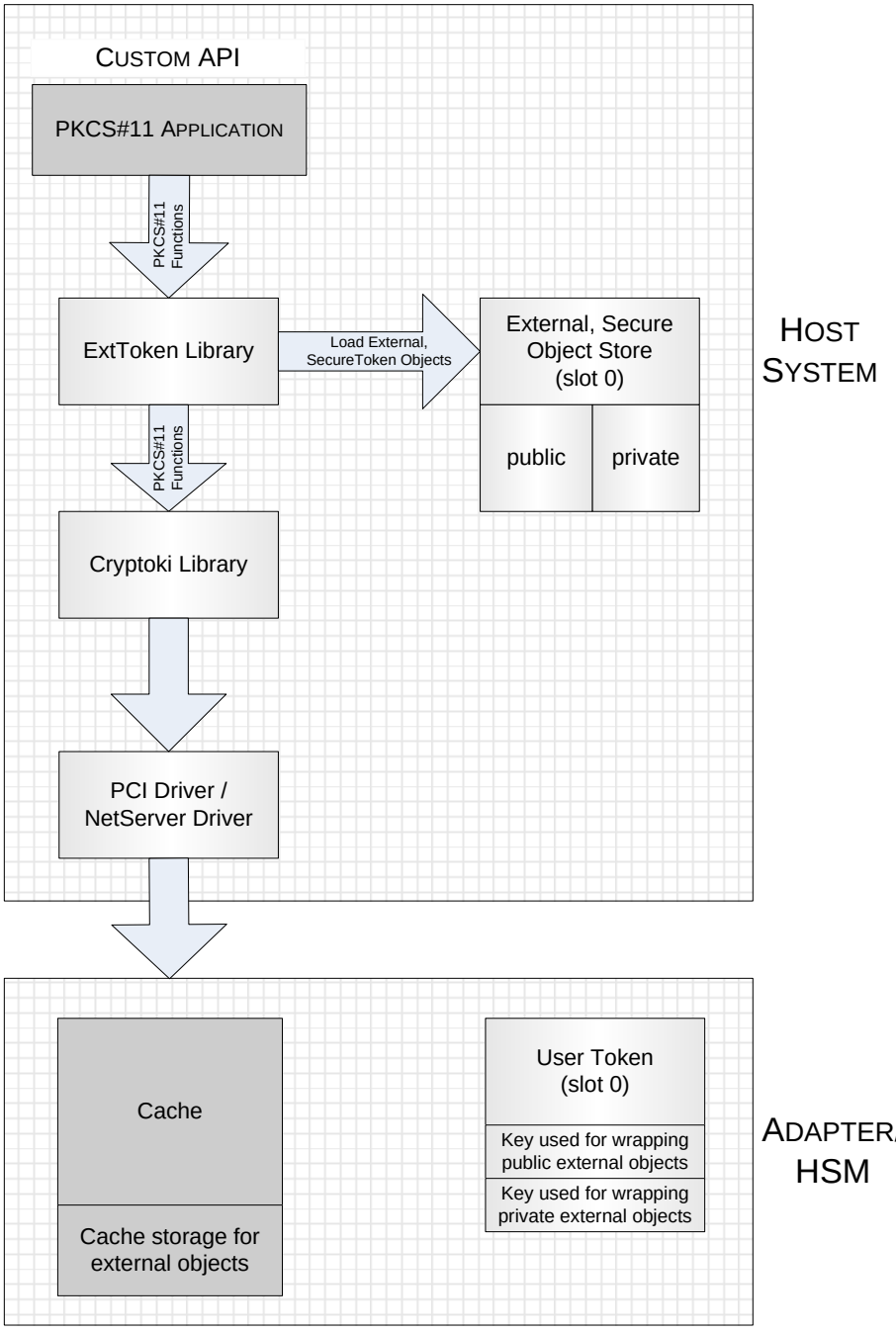
The ExtToken library is available with the ProtectToolkit C product (PTK-C) and is a part of the standard installation of the PTKcprt package incorporated in the PTK-C product release. The ExtToken library is supported on Windows platforms only.

ExtToken supports the secure external storage of token objects for the purpose of RSA signing, checking certificates, DES key exchange and DES encryption of transaction messages, to name a few. To reduce the processing overhead introduced in the secure and external storage of token objects, the HSM utilizes internal cache memory to store the most recently utilized token objects. The number of token objects stored in cache is configurable by the user.

The PSI-E2 and PSE2 HSMs support the use of secure external token object storage and the storage of token objects in user slots simultaneously.

Implementation

The following figure illustrates how external key storage is achieved on the host system and HSM.



PKCS#11 applications interface to the ExtToken library via standard PKCS#11 function calls. The ExtToken library makes use of another PKCS#11 provider, the Cryptoki library. The Cryptoki library is responsible for enforcing security policies and storing all data not related to the token objects of an external token. All cryptographic processing is performed on the standard Cryptoki library. The Cryptoki Library interfaces to local HSMs via the PCI driver and remote HSMs via the Netserver Driver.

ExtToken achieves secure external storage by using two master keys. These master keys are used to transparently wrap and unwrap the external objects. One of these keys is for protecting public objects, and the other is for protecting private objects. These master keys are DES2 keys and will be stored in slot 0. The token in slot 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.

As token objects are created via the ExtToken library, they 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 to 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 utilized 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 this limit is reached, then the least used object is unloaded from cache.

Key back up can be achieved simply by storing the master keys on a smart card and compressing the files utilized by the Secure Object Store using SafeNet's ProtectPack.

Technical Details

Installation

After installing PTK-C, installation instructions for ExtToken are provided the <InstallDir>\extToken\readme.txt file.

Performance

Performance overhead is introduced when storing objects externally. The overhead is introduced by the need to unwrap objects. The CTPERF utility (provided with the PTK-C installation) can be used to determine performance on individual systems. As an indication, 109 keys/ second can be unwrapped using a DES3 key. As previously discussed, the HSM utilizes internal cache memory to store the most recently utilized token objects in an effort to reduce this processing overhead. The environment variable ET_PTKC_MAXLOADED allows the user to configure the maximum number of token objects stored in cache. However, there is also a processing overhead involved in managing the keys stored in cache. This overhead increases linearly as the number of items stored in cache increases. Systems must be individually tuned for maximum performance depending upon patterns of key usage by the host application and by taking into consideration the tradeoff between the processing overhead involved in unwrapping keys and the processing overhead involved in managing the cache.

External Secure Object Store

There are two files per external token:

1. 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 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. Please refer to the ProtectToolkit C Programmers Manual for mechanism details.

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

Known Limitations

1. 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.
2. 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, i.e., 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 the application performance. This negative effect can be countered by using as many attributes from the above list as possible, and limiting such operations to application initialization.
3. Only objects with the CKA_EXTRACTABLE attribute set to TRUE can be imported to an external token.
4. It is not possible to set the SafeNet vendor defined CKA_EXPORT attribute to TRUE on an external token object.
5. It is not possible to set the CKA_TRUSTED attribute to TRUE on an external token object.
6. 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 provided with the PTK-C product to backup all the objects to a file, rename/delete the existing ORT and ODS files, then restore from the backup.
7. 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 the upper limit supported by SafeNet HSMs. One example of such an application is ctkmu. This means that it is possible to have so many objects on an external token that it is not possible to back them up. This can be rectified by adjusting the value of ET_PTKC_EXTTOKEN_MAXLOADED to a value which suits your application/environment.
8. The SafeNet implementation of JCA/JCE (ProtectToolkit J) uses one session per KeyStore. An application which uses the same KeyStore to access a large number of keys runs the risk of consuming all HSM resources (see point 7). A work-around is to use a new KeyStore object when locating keys. This does not introduce a significant performance overhead.
9. Smartcards and the Admin Token cannot be used as external tokens.

APPENDIX F
SAMPLE EC DOMAIN PARAMETER FILES

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    = 40000000000000000000000004A20E90C39067C893BBB9A5
```

brainpoolP160r1

```
#
#This file describes the domain parameters of an EC curve
#
#File contains lines of text. All lines not of the form key=value are
ignored.
#All values must be Hexidecimal numbers except m, k, k1, k2 and k3
which are decimal.
#Lines starting with ';' or '#' are comments.
#
#Keys recognised for fieldID values are -
#prime           - only if the Curve is based on a prime field
#m               - only if the curve is based on a 2^M field
#k               - only if the curve is 2^M field and is Trinomial
basis
#k1, k2, k3      - these three only if 2^M field and Pentanomial
basis
#
#You should have these combinations of fieldID values -
#prime      - if Curve is based on a prime field
#m          - if curve is based on 2^M and Basis is Gaussian normal
basis
#m,k        - if curve is based on 2^M and Basis is Polynomial basis
#m,k1,k2,k3 - if curve is based on 2^M and Basis is Pentanomial basis
#
#These are the values common to prime fields and polynomial fields.
#curveA          - field element A
#curveB          - field element B
#curveSeed       - this one is optional
#baseX           - field element Xg of the point G
#baseY           - field element Yg of the point G
#bpOrder         - order n of the point G
#cofactor        - (optional) cofactor h
#
#
# Curve name brainpoolP160r1

prime      = E95E4A5F737059DC60DFC7AD95B3D8139515620F
curveA     = 340E7BE2A280EB74E2BE61BADA745D97E8F7C300
curveB     = 1E589A8595423412134FAA2DBDEC95C8D8675E58
baseX      = BED5AF16EA3F6A4F62938C4631EB5AF7BDBCDBC3
baseY      = 1667CB477A1A8EC338F94741669C976316DA6321
bpOrder    = E95E4A5F737059DC60DF5991D45029409E60FC09
```

Hexadecimal to Decimal Conversion Table

Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec
00	000	20	032	40	064	60	096	80	128	A0	160	C0	192	E0	224
01	001	21	033	41	065	61	097	81	129	A1	161	C1	193	E1	225
02	002	22	034	42	066	62	098	82	130	A2	162	C2	194	E2	226
03	003	23	035	43	067	63	099	83	131	A3	163	C3	195	E3	227
04	004	24	036	44	068	64	100	84	132	A4	164	C4	196	E4	228
05	005	25	037	45	069	65	101	85	133	A5	165	C5	197	E5	229
06	006	26	038	46	070	66	102	86	134	A6	166	C6	198	E6	230
07	007	27	039	47	071	67	103	87	135	A7	167	C7	199	E7	231
08	008	28	040	48	072	68	104	88	136	A8	168	C8	200	E8	232
09	009	29	041	49	073	69	105	89	137	A9	169	C9	201	E9	233
0A	010	2A	42	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

THIS PAGE INTENTIONALLY LEFT BLANK

GLOSSARY

AES	Advanced Encryption Standard.
API	Application Programming Interface.
ASO	Administration Security Officer.
Bus	One of the sets of conductors (wires, PCB tracks or connections) in an IC.
CA	Certification Authority.
CAST	Encryption algorithm developed by Carlisle Adams and Stafford Tavares.
CMOS	Complementary Metal-Oxide Semiconductor. A common data storage component.
Cprov	ProtectToolkit C - SafeNet's PKCS #11 Cryptoki Provider.
DES	Cryptographic algorithm named as the Data Encryption Standard.
DSA	Digital Signature Algorithm.
FIPS	Federal Information Protection Standards.
HA	High Availability
HSM	Hardware Security Module
IDEA	International Data Encryption Algorithm.
IP	Internet Protocol.
KWRAP	Key Wrapping Key.
PEM	Privacy Enhanced Mail.
PCI	Peripheral Component Interconnect.
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.
RC2/RC4	Ciphers designed by RSA Data Security, Inc.
RNG	Random Number Generator
RSA	Cryptographic algorithm by Ron Rivest, Adi Shamir and Leonard Adelman.
RTC	Real Time Clock.
SO	Security Officer.
TC	Trusted Channel.
TCP/IP	Transmission Control Protocol / Internet Protocol.
URI	Universal Resource Identifier
VA	Validation Authority.
X.509	Digital Certificate Standard.

END OF DOCUMENT