ProtectToolkit C

SDK Programming Guide

© 2000-2014 SafeNet, Inc. All rights reserved.
Part Number 007-008395-006
Version 5.0

Trademarks

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

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 information	
Address	SafeNet, Inc. 4690 Millennium Drive Belcamp, Maryland 21017 USA	
Phone	United States	(800) 545-6608, (410) 931-7520
	Australia and New Zealand	+1 410-931-7520
	China	(86) 10 8851 9191
	France	0825 341000
	Germany	01803 7246269
	India	+1 410-931-7520
	United Kingdom	0870 7529200, +1 410 931-7520
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.
Customer Connection Center	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	27 October 2014	Release 5.0

Table of Contents

Technical Support ... ii

Glossary ... iv

Overview ... 1
Runtime Licensing ... 1
System Requirements ... 1

An Introduction to PKCS#11 ... 2
Overview ... 2
The Cryptoki Model ... 2

Installation ... 4
Installation for Windows Server 2008 ... 4
Installation for Solaris ... 5
Installation for AIX 5.3/6.1 ... 6
Installation for Linux ... 7
Setting Up Your Environment ... 7
Operation ... 9
Configuration / setup ... 9

Sample Programs ... 10
CTDEMO ... 10
FCRYPT ... 10

Additional APIs ... 11
CTUTIL ... 11
CheckCryptokiVersion ... 11
NUMITEMS ... 11
FindTokenFromName, FindKeyFromName ... 11
CT_OpenObject CT_CreateObject CT_RenameObject CT_ReadObject CT_WriteObject ... 12
GetAttr, SetAttr ... 13
calcKvc ... 13
C_ErrorString ... 13
String display functions ... 14
Key Generation / Creation functions ... 14
TransferObject ... 15
CTEXTRA ... 16
NUMITEMS ... 16
Mechanism associations ... 16
Attribute list management ... 17
Attribute lookups ... 17
Attribute list management ... 17
Miscellaneous attribute functions ... 18
Password to validation code / Key functions. ... 18
Miscellaneous ... 18

ProtectToolkit C Development Tips and Techniques ... 19

Extensions ... 20
Attribute Enumeration ... 20
Token Creation ... 20
Additional Object Types ... 20
CKO_CERTIFICATE_REQUEST ... 20
CKO_KG_PARAMETERS ... 21
Additional Attribute Types ... 21
CKA_TIME_STAMP ... 21

CKA_TRUST_LEVEL 21
CKA_USAGE_COUNT 22
CKA_ISSUER_STR 22
CKA_SUBJECT_STR 22
CKA_SERIAL_NUMBER_INT 22
Additional Mechanisms 22
CKM_ENCODE_PKCS_10 23
CKM_ENCODE_X_509 23
CKM_DSA_PARAMETER_GEN 24
CKM_DH_PKCS_PARAMETER_GEN 24
CKM_WRAPKEY_DES3_EBC 25
Key Generation Variations 25
PKCS#11 Interpretations 25
Software-Only Version Specific 25

System Information 26
ProtectToolkit C - Software only 26
ProtectToolkit C - Remote Client 27

Glossary

PKCS#11	Public Key Cryptography Standard # 11. Cryptographic Token Interface Standard (Cryptoki). An RSA Laboratories Technical Note.
Cryptoki	Cryptographic Token Interface Standard. (aka PKCS#11).
ProtectToolkit C	SafeNet's implementation of PKCS#11. ProtectToolkit C represents a suite of products including various PKCS#11 runtimes including software only, hardware adapter, and host security module based variants. A Remote client and server are also available.
JCA	Java Cryptographic Architecture.
JCE	Java Cryptographic Extensions.
ProtectToolkit J	SafeNet's implementation of JCE. Runs on top of ProtectToolkit C
ProtectServer	SafeNet HSM
Slot	PKCS#11 slot which is capable of holding a token.
Token	PKCS#11 token that provides cryptographic services and access controlled secure key storage.
SO	Security Officer for a PKCS#11 token.

THIS PAGE INTENTIONALLY LEFT BLANK

Overview

ProtectToolkit C is the name given to the SafeNet PKCS#11 (CRYPTOKI) software product. ProtectToolkit C is available in three different variants, a software-only version, a hardware version for the ProtectServer hardware adapter (and future hardware platforms), and a remote client version for TCP/IP connection to a remote ProtectToolkit C server. The software-only typically used as a development and testing environment for applications that will eventually use the hardware variant of ProtectToolkit C.

This product conforms to the API definition, produced by RSA Labs, named PKCS #11, and otherwise known as CRYPTOKI. ProtectToolkit C is compliant with PKCS#11 V 2.10.

The API provides a suite of cryptographic services for general-purpose usage and permanent key storage, which may be hosted by a physical token.

Unless specifically noted, the comments below refer to all SafeNet PKCS#11 products including the software only version, and the hardware-based ProtectServer implementation.

Runtime Licensing

Note: All of the run-time software, including all applications, and the software-only ProtectToolkit C run-time supplied with this SDK, is licensed to be used for development and testing purposes only. NO RUNTIME LICENSES ARE INCLUDED. This software or any component of it, therefore, must not be used for production systems. Separate run-time licenses must be purchased for production systems deployed using any ProtectToolkit C support.

System Requirements

No special hardware requirements are necessary for this product.
The supported platforms are listed in the following table.

C=PTK-C component, PKCS #11 v2.10/2.20
M=PTK-M component, MS CSP 2.0 with CNG
J=PTK-J, Java runtime 1.6.x/1.7.x

Operating system	32-bit binary 32-bit O/S		32-bit binary 64-bit O/S		64-bit binary 64-bit O/S	
	PSE2	PSI-E2	PSE2	PSI-E2	PSE2	PSI-E2
Windows Server 2012 R2 x86			-	-	C/M/J	C/M/J
Windows Server 2008 R2 x86	–	–	–	–	C/M/J	C/M/J
Windows 7	C/J	C/J			C/M/J	C/M/J
RedHat Enterprise Linux 6 x86	C/J	C/J			C/J	C/J

An Introduction to PKCS#11

Overview

The PKCS#11 Cryptographic Token Interface Standard is one of the Public Key Cryptography Standards developed by RSA Security. Also referred to as Cryptoki, this standard deals with defining the interface between an application and a cryptographic device. This chapter gives a rudimentary outline of Cryptoki and some of its basic concepts. If unfamiliar with PKCS#11, the reader is strongly advised to obtain a copy of the standard from the RSA website at http://www.rsasecurity.com/rsalabs/pkcs/ to gain further knowledge.

Cryptoki is used as a low level interface to perform cryptographic operations without the requirement for the application to directly interface a device through its driver. Cryptoki represents cryptographic devices using a common model referred to simply as a token. An application can therefore perform cryptographic operations on any device or token, using the same independent command set.

ProtectToolkit C is an Application Programming Interface (API) which is built on the Cryptoki standard.

The Cryptoki Model

The model for Cryptoki can be seen illustrated in Figure 1 and demonstrates how an application communicates its requests to a token via the Cryptoki interface. The term slot represents a physical device interface. For example a smartcard reader would represent a slot and the smartcard would represent the token. It is also possible that multiple slots may share the same token.

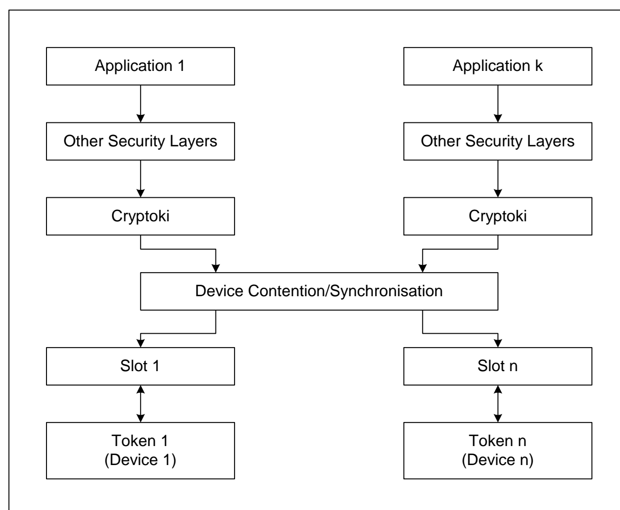

Figure 1 – General Cryptoki Model

Within Cryptoki, a token is viewed as a device that stores objects and can perform cryptographic functions. Objects are defined in one of three classes:

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

Objects within Cryptoki are further defined as either a token object or a session object. Token objects are visible by any application which has sufficient access permission and is connected

to that token. An important attribute of a token object is that it remains on the token until a specific action is performed to remove it.

A connection between a token and an application is referred to as a session. Session objects are temporary and only remain in existence whilst the session is open. In addition, session objects are only ever visible to the application which created them.

Access to objects within Cryptoki is defined by the object type. Public objects are visible to any user or application, whereby private objects require that the user must be logged into that token in order to view them. Cryptoki recognises two types of users, which is either a security officer (SO) or normal user. The security officers only role is to initialize a token and set the normal users access PIN. An important point to note is that the normal user, which manipulates objects and performs most operations, cannot log in until the security officer has set the users PIN.

Installation

The following two sections contain the installation instructions for the ProtectToolkit C SDK package on all supported platforms.

If you intend on using this SDK with the ProtectServer HSM please consult the Installation Guide for the particular hardware (and associated driver software).

Installation for Windows Server 2008

Note: To order to be able to add or remove software the current user must have “Administrator” privileges.

To install the package simply execute the program `PTKcpsdk.msi` and follow the on-screen instructions to install the software.

The installation program will create a new program group named “CProv SDK” and add this to your Start menu. Your will then need to modify your environment settings to include the ProtectToolkit C Cryptoki dynamic link libraries and runtime tools in your path.

To remove the software from your system please go to the “Add/Remove Programs” item in the Control Panel and select the “CProv SDK” item from the list.

Installation for Solaris

The ProtectToolkit C SDK for Solaris is packaged using the standard Solaris packaging software. Under Solaris this package cannot be installed concurrently with any of the ProtectToolkit C runtimes. The package however includes the hardware and software runtimes and it is possible to use either (by default the SDK package will be configured for the software-only runtime).

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

To install the package simply use the **pkgadd** program to add the `PTKcpsdk` package. Once installed, the software will be ready to use under `/opt/safenet/protecttoolkit5/ptk`. To make use of the software you will need to add the `/opt/safenet/protecttoolkit5/ptk/bin` directory to your execution path and `/opt/safenet/protecttoolkit5/ptk/lib` to your library path. The following commands may be used to configure your paths for the **sh** shell (please consult your Solaris manual for other shells):

```
# PATH=/opt/safenet/protecttoolkit5/ptk/bin:$PATH
# export PATH
#
LD_LIBRARY_PATH=/opt/safenet/protecttoolkit5/ptk/lib:$LD_
LIBRARY_PATH
# export LD_LIBRARY_PATH
```

By default the software-only runtime will be selected as the default. To change this simply remove the `libcryptoki.so` soft-link and recreate it to point to the desired runtime library. For example to switch to the hardware library the following shell commands may be used (executed as the super-user).

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

To remove the software from your host system simply use the **pkgrm** program and select the `PTKcpsdk` package for removal.

Installation for AIX 5.3/6.1

The ProtectToolkit C SDK for AIX is packaged using the standard AIX packaging software. Under AIX this package cannot be installed concurrently with any of the ProtectToolkit C runtimes. The package however includes the hardware and software runtimes and it is possible to use either (by default the SDK package will be configured for the software-only runtime).

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

To install the package simply use the **installp** program to add the PTKcpsdk package.

For example:

```
# installp -acgNQqwX -d . PTKcpsdk.rte
```

Once installed, the software will be ready to use under /opt/PTK. To make use of the software you will need to add the /opt/PTK/bin directory to your execution path and /opt/PTK/lib to your library path. The following commands may be used to configure your paths for the **sh** shell (please consult your AIX manual for other shells):

```
# PATH=/opt/safenet/protecttoolkit5/ptk/bin:$PATH
# export PATH
# LIBPATH=/opt/safenet/protecttoolkit5/ptk/lib:$LIBPATH
# export LIBPATH
```

By default the software-only runtime will be selected as the default. To change this simply remove the libcryptoki.so soft-link and recreate it to point to the desired runtime library. For example to switch to the hardware library the following shell commands may be used (executed as the super-user).

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

To remove the software from your host system simply use the **installp** program and select the PTKcpsdk package for removal:

```
# installp –u PTKcpsdk.rte
```

Installation for Linux

The ProtectToolkit C SDK for Linux is packaged using the standard RPM packaging software. Under Linux this package cannot be installed concurrently with any of the ProtectToolkit C runtimes. The package however includes the hardware and software runtimes and it is possible to use either (by default the SDK package will be configured for the software-only runtime).

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

To install the package simply use the **`rpm`** command to add the `PTKcpsdk` package. Once installed, the software will be ready to use under `/opt/safenet/protecttoolkit5/ptk`.

For example:

```
# rpm -i PTKcpsdk-2.21-1.i386.rpm
```

Setting Up Your Environment

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

To set up your environment

1. Go to the PTK software installation directory:

   cd /opt/safenet/protecttoolkit5/ptk

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

   . ./setvars.sh

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

To make use of the software you will need to add the `/opt/PTK/bin` directory to your execution path and `/opt/PTK/lib` to your library path. The following commands may be used to configure your paths for the **`sh`** shell (please consult your Linux manual for other shells):

```
# PATH=/opt/safenet/protecttoolkit5/ptk/bin:$PATH
# export PATH
#
LD_LIBRARY_PATH=/opt/safenet/protecttoolkit5/ptk/lib:$LD_
LIBRARY_PATH
# export LD_LIBRARY_PATH
```

By default the software-only runtime will be selected as the default. The installer provides options for setting the default cryptoki and/or HSM link.

To manually change this simply remove the `libcryptoki.so` soft-link and recreate it to point to the desired runtime library. For example to switch to the hardware library the following shell commands may be used (executed as the super-user).

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

To remove the software from your host system simply use the **rpm** command and select the `PTKcpsdk` package for removal.

For example:

```
# rpm –e PTKcpsdk
```

Operation

- **Win32™**
  ProtectToolkit C is supplied as a WIN32 Dynamic Link Library (CRYPTOKI.DLL) built with Microsoft development tools (MSVC). CRYPTOKI.LIB is an import library that should be linked against applications to resolve function calls into CRYPTOKI DLL.

- **Solaris™**
  Supplied as shared libraries. The hardware based ProtectToolkit C library is stored as the shared library libctcsa.so, the software-only version as libctsw.so and the remote client shared library as libctclient.so. The symbolic link libcryptoki.so should point to the appropriate library. Additionally these libraries must be included in your LD_LIBRARY_PATH.

- **Linux**
  Supplied as shared libraries. , The hardware based ProtectToolkit C library is stored as the shared library libctcsa.so, the software-only version as libctsw.so and the remote client shared library as libctclient.so. The symbolic link libcryptoki.so should point to the appropriate library. Additionally these libraries must be included in your LD_LIBRARY_PATH.

Sample programs, for which source code has been provided, may be compiled and linked against the supplied libraries. The additional libraries "ctextra" and "ctutil" are static libraries that contain additional PKCS#11support and helper functions that are not a part of the PKCS#11standard.

This development kit may be used to build applications for any variant of the ProtectToolkit C runtime including either the software-only ProtectToolkit C, the ProtectServer based ProtectToolkit C, or the ProtectToolkit C remote client.

Configuration / setup

The **cryptoki.ini** file support allows the configuration of global parameters such which debug logging information (if enabled) will be written, etc. Description of .**ini** items can be found in the documentation for the ProtectToolkit C Runtime products.

A setup utility, **ctconf**, is supplied which allows configuration of global parameters. This utility is provided with the various ProtectToolkit C runtime products and has different options related to the different ProtectToolkit C runtime variants. Description of its operation can be found in the documentation for the ProtectToolkit C runtime products.

The **ctreset** utility may be used to free all resources held for programs that are no longer running. This occurs when ProtectToolkit C applications crash or for any other reason do not call C_Finalize(). See ProtectToolkit C runtime manual for more information on **ctreset**.

Sample Programs

Sample programs include a variety of PKCS#11applications. Unless specifically stated, any source code provided with the ProtectToolkit C SDK product may be modified or incorporated into other programs.

CTDEMO

This program sets up a 4 token key profile that may be used for an electronic commerce trading application. Token profiles include a sample customer, merchant, bank and certifying authority. It exchanges public keys between all the tokens and, where CA mechanism extensions are supported, ProtectToolkit C generates certificates for the public keys.

ProtectToolkit C must be configured to have at least 4 slots / tokens for this demo program to operate correctly.

The **ctdemo** program is a console application that takes the following arguments:

ctdemo -s<slotID> **-m**<modulus size> **-q -f -x**

-q	Quick. Does not prompt for values but uses defaults.
-f	Force. Does not warn about overwriting token contents.
-m	Specify modulus size.
-s	First slot number to use.
-x	Extended. Creates more keys.

Defaults :

Security Officer (SO) Pin = 9999

Slot	Token label	Pin
0	Alice	0000
1	NAB	1111
2	Meyer	2222
3	SAFENET	3333

Note: This will overwrite the contents of all of the above tokens.

FCRYPT

The **fcrypt** program is a file encryption program that takes a recipient's public key and sender's private key and uses these to encrypt and sign the file's content. Random triple DES transport keys are generated for the bulk file content encryption. Alternately the Password Based Encryption (PBE) variant can be used so that only the password needs to be shared and no public keys / certificates need be exchanged. The default output file is “file.enc”.

The **fcrypt** program is a console application that takes the following arguments:

fcrypt [-d] -s<sender(pin)/key> -r<recipient(pin)/key> [-o<output file>] file-name

or

fcrypt [-d] -p<password> [-o<output file>] file-name

Additional APIs

CTUTIL

The following additional features do not form part of the standard CRYPTOKI functionality, but are provided by SafeNet as part of the ProtectToolkit C API within the **ctutil.h** library.

CheckCryptokiVersion

Synopsis

```
CK_RV CheckCryptokiVersion(void);
```

Note that this API is implemented as a macro.

Description

Two versions of PKCS#11 are supported by SafeNet - V1.0 and V2.01 that are similar but incompatible. An application compiled for V 1.0 compliance is likely to crash if it links against a V 2.01 compliant DLL and vice-versa.

This function is used to check that the version of CRYPTOKI is correct for the application and will report if an incompatible Cryptoki DLL is loaded. The application should report this fact and terminate.

All the sample applications make this call to check the Cryptoki version they are running with.

NUMITEMS

Synopsis

```
#define NUMITEMS(type)   (sizeof((type))/sizeof((type)[0]))
```

Description

This is a macro that returns the number of elements in an array. Note that only array definitions may be sized by this macro, not pointer definitions.

It is used wherever object templates are defined since the number of items in the template is always passed along with the template address into Cryptoki functions. Use of this macro is preferred to hard coding the number of items in the template that may change with code maintenance.

FindTokenFromName, FindKeyFromName

Synopsis

```
CK_RV FindTokenFromName( char * label, CK_SLOT_ID * pslotID );
CK_RV FindKeyFromName(const char * keyPath, CK_OBJECT_CLASS cl,
     CK_SLOT_ID * phSlot, CK_SESSION_HANDLE * phSession,
     CK_OBJECT_HANDLE * phKey);
```

'keyPath' syntax - "token(pin)/key"

Description

Applications should locate tokens by name rather than by assuming that a particular token will always be found in a particular slot. These functions allow tokens and particular keys to be located easily.

The sample program FCRYPT uses these functions to locate keys.

CT_OpenObject CT_CreateObject CT_RenameObject CT_ReadObject CT_WriteObject

Synopsis

```
CK_RV CT_OpenObject(
      CK_SESSION_HANDLE hSession,
      CK_OBJECT_CLASS cl,
      char * name,
      CK_OBJECT_HANDLE * phObj);

CK_RV CT_CreateObject(
      CK_SESSION_HANDLE hSession,
      CK_OBJECT_CLASS cl,
      char * name,
      CK_OBJECT_HANDLE * phObj);

CK_RV CT_RenameObject(
      CK_SESSION_HANDLE hSession,
      CK_OBJECT_CLASS cl,
      char * oldName,
      char * newName);

 CK_RV CT_ReadObject(
      CK_SESSION_HANDLE hSession,
      CK_OBJECT_HANDLE hObj,
      unsigned char * buf,
      unsigned int len,
      unsigned int * pbr);

CK_RV CT_WriteObject(
      CK_SESSION_HANDLE hSession,
      CK_OBJECT_HANDLE hObj,
      const unsigned char * buf,
      unsigned int len,
      unsigned int * pbr);
```

Description

These functions treat PKCS#11 objects like named files using the `CKA_LABEL` as the file name. They make standard PKCS#11 calls and return standard errors if any lower level PKCS#11 function fails.

The reader is advised to refer to the sample programs included with ProtectToolkit C for examples of their usage.

GetAttr, SetAttr

Synopsis

```
CK_RV GetAttr(
      CK_SESSION_HANDLE hSession,
      CK_OBJECT_HANDLE obj,
      CK_ATTRIBUTE_TYPE type,
      CK_VOID_PTR buf, CK_SIZE len, CK_SIZE_PTR size);

CK_RV SetAttr(
      CK_SESSION_HANDLE hSession,
      CK_OBJECT_HANDLE obj,
      CK_ATTRIBUTE_TYPE type,
      CK_VOID_PTR buf, CK_SIZE len);

CK_ATTRIBUTE * FindAttribute(
      CK_ATTRIBUTE_TYPE attrType,
      CK_ATTRIBUTE_PTR attr,
      CK_COUNT attrCount);
```

Description

These functions allow the caller to easily obtain, or set, a single attribute from a PKCS#11 object. Findattribute( ) locates a particular attribute from within an attribute template.

calcKvc

Synopsis

```
CK_RV calcKvcMech(CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hKey,
            CK_MECHANISM_TYPE mt,
            unsigned char * kvc, int kvclen, int * pkvclen);
CK_RV calcKvc(CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hKey,
            unsigned char * kvc, int kvclen, int * pkvclen);
```

Description

Calculate and return an AS2805 KVC for a key. The key must be capable of doing an encryption operation using the supplied mechanism for this to succeed. It must also have the `CKA_ENCRYPT` attribute set to 1. Note that mechanism parameters will be set to NULL for the actual encrypt operation to generate the KVC.

C_ErrorString

Synopsis

```
CK_RV C_ErrorString(CK_RV ret, char * errstr, unsigned int len);
```

Description

Convert a Cryptoki error code into a printable string. Note that this function is not a part of the PKCS#11 definition.

The sample programs use this extensively to map Cryptoki error numbers to meaningful text to display to the user.

String display functions

Synopsis

```
char * strObjClass(CK_NUMERIC val );
CK_NUMERIC valObjClass( const char * txt );
char * strKeyType(CK_NUMERIC val );
CK_NUMERIC valKeyType( const char * txt );
char * strAttribute(CK_NUMERIC val );
CK_NUMERIC valAttribute( const char * txt );
char * strMechanism(CK_NUMERIC val );
CK_NUMERIC valMechanism( const char * txt );
char * strSesState(CK_NUMERIC val );
CK_NUMERIC valSesState( const char * txt );
char * strError(CK_NUMERIC val );
CK_NUMERIC valError( const char * txt );
```

Description

Convert PKCS#11 definitions to strings and vice versa.

The token browser **ctbrowse** uses these services extensively in the user interface to display selectable PKCS#11 options.

Key Generation / Creation functions

Synopsis

```
CK_RV CreateSecretKey(CK_SESSION_HANDLE hSession, char * txt,
      int tok, int priv,
      CK_KEY_TYPE kt,
      CK_BYTE * keyValue, int len,
      CK_OBJECT_HANDLE * phKey);

CK_RV CreateDesKey(CK_SESSION_HANDLE hSession, char * txt,
      int tok, int priv,
      CK_BYTE * keyValue, int len,
      CK_OBJECT_HANDLE * phKey);

CK_RV BuildRsaCrtKeyPair(
      CK_SESSION_HANDLE hSession, char * txt,
      int tok, int priv,
      CK_OBJECT_HANDLE * phPub, CK_OBJECT_HANDLE * phPri,
      char * modulusStr, char * pubExpStr,
      char * priExpStr, char * priPStr, char * priQStr,
      char * priE1Str, char * priE2Str, char * priUStr);

CK_RV BuildRsaKeyPair(CK_SESSION_HANDLE hSession, char * txt,
      int tok, int priv,
      CK_OBJECT_HANDLE * phPub, CK_OBJECT_HANDLE * phPri,
      char * modulusStr, char * pubExponentStr,
      char * priExponentStr);

CK_RV GenerateRsaKeyPair(CK_SESSION_HANDLE hSession, char * txt,
      int ftok, int priv,
      CK_SIZE modulusBits, int expType,
      CK_OBJECT_HANDLE * phPublicKey,
      CK_OBJECT_HANDLE * phPrivateKey);

CK_RV BuildDsaKeyPair(CK_SESSION_HANDLE hSession, char * txt,
      int tok, int priv,
      CK_OBJECT_HANDLE * phPub, CK_OBJECT_HANDLE * phPri,
      char * prime, char * subprime, char * base,
      char * pub, char * pri);
```

```
CK_RV GenerateDsaKeyPair(CK_SESSION_HANDLE hSession, char * txt,
      int ftok, int priv, int param,
      CK_SIZE valueBits,
      CK_OBJECT_HANDLE * phPublicKey,
      CK_OBJECT_HANDLE * phPrivateKey);

CK_RV BuildDhKeyPair(CK_SESSION_HANDLE hSession, char * txt,
      int tok, int priv,
      CK_OBJECT_HANDLE * phPub, CK_OBJECT_HANDLE * phPri,
      char * prime, char * base, char * pub, char * pri);

 CK_RV GenerateDhKeyPair(CK_SESSION_HANDLE hSession, char * txt,
      int ftok, int priv, int param,
      CK_SIZE valueBits,
      CK_OBJECT_HANDLE * phPublicKey,
      CK_OBJECT_HANDLE * phPrivateKey);
```

Description

Generate and create keys with simple attribute sets.

CreateSecretKey()	- Generates a secret key object
CreateDesKey()	- Creates a DES key object
BuildRsaCrtKeyPair()	- Builds an RSA Certificate Key Pair
BuildRsaKeyPair()	- Builds an RSA Key Pair
GenerateRsaKeyPair()	- Generates an RSA Key Pair
BuildDsaKeyPair()	- Builds a DSA Key Pair
GenerateDsaKeyPair()	- Generates a DSA Key Pair
BuildDhKeyPair()	- Builds a Diffie Hellman Key Pair
GenerateDhKeyPair()	- Generates a Diffie Hellman Key Pair

TransferObject

Synopsis

```
CK_RV TransferObject(
      CK_SESSION_HANDLE sTo,
      CK_SESSION_HANDLE sFrom,
      CK_OBJECT_HANDLE hObj,
      CK_OBJECT_HANDLE * phObj );
```

Description

Shifts an object from one slot / token to another.

This is used by the token browser for Drag and Drop moving of objects and by **ctdemo** to do certificate exchanges from one token to another.

CTEXTRA

The **ctextra** library contains further functionality that may be useful in a PKCS#11 based application.

NUMITEMS

Synopsis

```
#define NUMITEMS(type)  (sizeof((type))/sizeof((type)[0]))
```

Description

This is a macro that returns the number of elements in an array. Note that only array definitions may be sized by this macro, not pointer definitions.

It is used wherever object templates are defined since the number of items in the template is always passed along with the template address into Cryptoki functions. Use of this macro is preferred to hard coding the number of items in the template that may change with code maintenance.

Mechanism associations

Synopsis

```
struct TOK_MECH_DATA {
      CK_MECHANISM_TYPE * pMechanisms;
      unsigned int count;
};
typedef struct TOK_MECH_DATA TOK_MECH_DATA;

int LookupMech(TOK_MECH_DATA * pMech, CK_MECHANISM_TYPE mechType);
void FreeMechData(TOK_MECH_DATA * pMech);

CK_MECHANISM_TYPE * genMechanismFromMechanism(
      CK_MECHANISM_TYPE mt, unsigned int * len);
CK_MECHANISM_TYPE * genMechanismTabFromMechanismTab(
      TOK_MECH_DATA * mTab, unsigned int * len);
CK_MECHANISM_TYPE * mechFromKt(CK_KEY_TYPE kt, unsigned int * len);
CK_KEY_TYPE * ktFromMech( CK_MECHANISM_TYPE mt, unsigned int * len);
CK_MECHANISM_TYPE * mechFromTokKt(
      TOK_MECH_DATA * mTab, CK_KEY_TYPE kt, unsigned int * len);
CK_MECHANISM_TYPE * mechDeriveFromKt(
      CK_KEY_TYPE kt, unsigned int * len);
CK_MECHANISM_TYPE * mechSignFromKt(
      CK_KEY_TYPE kt, unsigned int * len);
CK_MECHANISM_TYPE * mechSignRecFromKt(
      CK_KEY_TYPE kt, unsigned int * len);
CK_MECHANISM_TYPE * hashMech(unsigned int * len);
CK_MECHANISM_TYPE * kgMech(unsigned int * len);
CK_MECHANISM_TYPE * kpgMech(unsigned int * len);

int isGenMech(CK_MECHANISM_TYPE mechType);
```

Description

Obtain mechanism lists from key types etc.

This is used by the **ctbrowse** token browser.

Attribute list management

Synopsis

```
struct TOK_ATTR_DATA {
      CK_ATTRIBUTE * attributes;  /* an array of attribute items */
      CK_COUNT attrCount; /* number of items in 'attributes' */
};
typedef struct TOK_ATTR_DATA TOK_ATTR_DATA;
```

Attribute lookups

```
CK_NUMERIC numAttr(CK_ATTRIBUTE * at);
CK_NUMERIC numAttrLookup(CK_ATTRIBUTE_TYPE atype,
            CK_ATTRIBUTE * attr, CK_COUNT attrCount);
int intAttrLookup(CK_ATTRIBUTE_TYPE atype, CK_ATTRIBUTE * attr,
            CK_COUNT attrCount);
int intAttr(CK_ATTRIBUTE_PTR at);
```

Extract a numeric attribute from an attribute template.

```
CK_RV GetObjectClassAndKeyType(
      TOK_ATTR_DATA * attr,
      CK_OBJECT_CLASS * at_class, CK_KEY_TYPE * kt);
```

Extract the object class and key type from an object. This is a particularly common job when working with CRYPTOKI key objects. Return `CKR_OK` if both attributes were found.

```
CK_ATTRIBUTE * FindAttr(
            CK_ATTRIBUTE_TYPE attrType,
            TOK_ATTR_DATA * attrData);
```

Find an attribute in an attribute template.

Attribute list management

```
TOK_ATTR_DATA * DupAttributes(
      CK_ATTRIBUTE_PTR attr, CK_COUNT attrCount);
TOK_ATTR_DATA * DupAttributeSet( TOK_ATTR_DATA * attrData );
```

Make a copy of an attribute set. Return a pointer to the set. Return `NULL` if list cannot be duplicated.

Note: the new attribute list is dynamically created and should be freed using `FreeAttributeSet`.

```
int TransferAttr(CK_ATTRIBUTE_PTR pTgtTemplate,
      CK_ATTRIBUTE_PTR pSrcTemplate, CK_COUNT attrCount);
```

Copy attributes from one attribute table to another. The target table must have buffers to accommodate all values. Note: No mallocs are used.

```
int MatchAttributeSet( TOK_ATTR_DATA * match, TOK_ATTR_DATA * ad);
```

Do a comparison of two attribute sets. Every attribute in the 'match' set must be found in the 'ad' set. It is OK if 'ad' is a superset of 'match'. Return `TRUE` if all attributes in 'match' were found in 'ad'.

```
CK_RV AddAttributeSets(TOK_ATTR_DATA ** pSum,
```

```
      TOK_ATTR_DATA * base, TOK_ATTR_DATA * user);
```

Add two attribute sets being careful to drop duplicates. The 'base' attributes will override 'user' attributes where duplicates are concerned.

```
void FreeAttributeSet(TOK_ATTR_DATA * attr);
void FreeAttributes(CK_ATTRIBUTE_PTR attr, CK_COUNT attrCount);
```

Free an attribute list

Miscellaneous attribute functions

```
int isBooleanAttr(CK_ATTRIBUTE * na);
int isEnumeratedAttr(CK_ATTRIBUTE * na);
int isNumericAttr(CK_ATTRIBUTE * na);
int isSensitiveAttr(struct TOK_ATTR_DATA * attrData,
      CK_ATTRIBUTE * na);
```

Password to validation code / Key functions.

Synopsis

```
void KeyFromPin(unsigned char * key, unsigned int klen,
      CK_USER_TYPE user,
      const unsigned char * pin, unsigned int pinLen);
void PvcFromPin(unsigned char * key, unsigned int klen,
      CK_USER_TYPE user,
      const unsigned char * pin, unsigned int pinLen);
```

Description

Derive double DES keys (16 bytes) from a password. Uses PKCS#5.

Miscellaneous

Synopsis

```
CK_SLOT_ID slotIDfromSes(CK_SESSION_HANDLE h);
```

Description

Extract a `CK_SLOT_ID` from a `CK_SESSION_HANDLE`. This function only works with SafeNet's Cryptoki product because it includes an encoding of the SLOT id in the session handle. For other PKCS#11 implementations the slot ID can be obtained from the session info C_GetSessionInfo( ) call.

ProtectToolkit C Development Tips and Techniques

The best place to start building a ProtectToolkit C application is with the sample applications that demonstrate how the ProtectToolkit C system should be initialised and used to perform various cryptographic operations. The samples vary quite significantly in complexity however they are all real working ProtectToolkit C utilities and cover all ProtectToolkit C services.

The **ctbrowse.exe** program is particularly useful while building ProtectToolkit C applications since it can be used to build tokens for testing purposes, and also to examine tokens after a ProtectToolkit C application has run. It may also be used to quickly verify the result of a cryptographic operation.

Debug builds of applications should save all keys to the token, rather than using session keys, and make all keys non sensitive. This allows maximum visibility of key data to the browser (**ctbrowse.exe**) which is always helpful when debugging ProtectToolkit C applications. Non-debug builds should make all secret keys sensitive and should use session objects where possible. It is particularly important to make sure the CKA_SENSITIVE attribute is set to true on non-debug builds otherwise secret keys may be less secure.

When developing for the ProtectServer subsystem it is possible to perform all initial development and testing using only the software-only version and delay use of the ProtectServer hardware until the final testing phase. This significantly reduces the development system setup time since no hardware and associated device drivers need to be installed to allow testing and debugging on the development machine.

Note:
Multi-threaded applications must avoid making ProtectToolkit C calls simultaneously from different threads of the same application. It is possible for multiple threads to operate on different tokens simultaneously.

Object attribute templates are difficult to process in applications so application developers are advised to see how this is done in the sample programs using the additional services in **ctutil**. Otherwise application code can become large and messy where it deals with attributes.

Applications can add their own vendor defined attributes to object templates and these will be incorporated into the objects and will not effect normal ProtectToolkit C processing.

The ProtectToolkit C remote server and client may be used to provide remote access to ProtectToolkit C services. For example hardware based ProtectToolkit C services may be installed onto a machine, then the ProtectToolkit C remote server installed to give access to a ProtectToolkit C remote client which may be running on a machine where the hardware based ProtectToolkit C cannot be used directly.

Extensions

All extensions are provided in a manner that should be compatible with applications that expect fully compliant behavior. Applications relying on the extensions may be incompatible with a fully compliant PKCS#11 implementation that does not support the SafeNet extensions.

Note that no new entry points have been added, just modifications to the internal behavior depending on parameters given to existing functions.

Attribute Enumeration

Attribute enumeration is supported as follows.
First call `C_GetAttributeValue` as follows to initialize the enumeration.

```
CK_ATTRIBUTE at;
rv = C_GetAttributeValue(hSession, hObject, &at, 0);
```

then to get all the attributes loop as follows

```
for (;;) {
     at.type = CKA_ENUM_ATTRIBUTE;
     at.pValue = 0;
     rv = C_GetAttributeValue(hSession, hObject, &at, 1);
     if ( rv == CKR_ATTRIBUTE_TYPE_INVALID )
          break; /* got all the attributes */ }
```

Sensitive attributes will be returned with the type information but an empty value, and will also return a result value of `CKR_ATTRIBUTE_SENSITIVE`. On implementations, where this extension is not supported, the calls to `C_GetAttributeType` are likely to fail with the `CKR_ATTRIBUTE_TYPE_INVALID` error code.

With a result code of `CKR_OK` or `CKR_ATTRIBUTE_SENSITIVE` the `CK_ATTRIBUTE` structure will have the `type` and `valueLen` fields set appropriately for the next attribute, however the `pValue` field will be `NULL_PTR`. To retrieve the actual value of the attribute it is necessary to allocate the necessary room for the value and then make a second call to `C_GetAttributeValue`.

Token Creation

To allow token creation, `C_InitToken` may be called on a slot that does not contain a token. This operation would normally fail on standard PKCS#11's which require a physical token to be inserted by some other means, e.g. by hand.

Additional Object Types

CKO_CERTIFICATE_REQUEST

This object type is used to hold a PKCS#10 certificate request. There are mechanisms included to generate a Certificate Request object from an RSA public key (see `CKM_ENCODE_PKCS_10`) or generate a Certificate from a Certificate Request (see `CKM_ENCODE_X_509`).

Attribute	Data Type	Meaning
CKA_SUBJECT	Byte array	DER-encoding of the certificate request subject name
CKA_VALUE	Byte array	DER-encoding of the certificate request
KEY_TYPE	CK_KEY_TYPE	Type of public key in request

CKO_KG_PARAMETERS

This object type is used to hold DSA or DH key generation parameters.

The CKA_KEY_TYPE attribute indicates which type of parameters it is holding.
Where the key type is CKK_DSA the attributes should be as follows :

Attribute	Data Type	Meaning
CKA_KEY_TYPE	CK_KEY_TYPE	Type of key. Must be CKK_DSA
CKA_PRIME	Big integer	Prime
CKA_SUBPRIME	Big integer	Prime
CKA_BASE	Big integer	Prime

Where the key type is CKK_DH the attributes should be as follows :

Attribute	Data Type	Meaning
CKA_KEY_TYPE	CK_KEY_TYPE	Type of key. Must be CKK_DH
CKA_PRIME	Big integer	Prime
CKA_BASE	Big integer	Prime

Additional Attribute Types

CKA_TIME_STAMP

Every object created with ProtectToolkit C will be assigned a value for the `CKA_TIME_STAMP` attribute. This value is always read-only and may not be included in a template for a new object. However when an object is duplicated using the `C_CopyObject` function or the object is a key derived using the `C_DeriveKey` the new object will inherit the same creation time as the original object.

The value of this attribute will be a text string encoding of the time. The encoding format is "YYYYMMDDHHMMSS00".

ProtectToolkit C ignores the value of this attribute. Previous versions of ProtectToolkit C used this date when doing key wrapping services however this feature has been dropped.

CKA_TRUST_LEVEL

This attribute may be included in a template for the creation of a Certificate object. It may be used to indicate whether or not the certificate is *trusted* by the application. Once set the value of this attribute may not be modified.

The following values are defined for this attribute:

```
TRUST_TRUSTED
TRUST_VALIDATED
TRUST_INVALID
```

The value of TRUST_TRUSTED may only be set when the Security Officer is currently logged in. That is, the session state must be CKS_RW_SO_FUNCTIONS. Once a Certificate object has the CKA_TRUST_LEVEL attribute equal to the TRUST_TRUSTED value the Certificate is considered a “trusted root certificate”. The certificate validation code will stop once it reaches a trusted root certificate.

The value TRUST_VALIDATED may only be set by the adapter. When a Certificate object is used as the key handle in a C_VerifyInit call the library will attempt to verify the Certificate according to the certificate validation algorithm. If this algorithm indicates the certificate should be trusted then the CKA_TRUST_LEVEL attribute of the certificate will be modified to TRUST_VALIDATED, other wise the attribute will be set to

TRUST_INVALID and the C_VerifyInit function will return CKR_CERT_NOT_VALIDATED.

The certificate validation algorithm will locate the certificate's issuer by searching for a Certificate object with the CKA_SUBJECT attribute equal to the issuer's distinguished name. If located it will then verify the signature on the certificate. If the signature is invalid it will return false, otherwise it will check the CKA_TRUST_LEVEL attribute on the issuer's certificate, if it is not equal to TRUST_TRUSTED it will search for the issuer of that certificate. The algorithm will continue until a trusted certificate is found, a signature verification fails or the certificate chain is broken.

CKA_USAGE_COUNT

The value of this attribute maintains a count of the number of usages of the given key object. It is possible to set the value of this attribute for a key, after which ProtectToolkit C will automatically increment the value each time the key is used in a Cryptoki initialisation routine (i.e. C_SignInit).

Additionally this attribute may be used in place of the `CKA_SERIAL_NUMBER` attribute when generating Certificate objects with the `CKM_ENCODE_X_509` mechanism. The usage count will be used if the serial number is not included in the template for the new certificate.

CKA_ISSUER_STR

CKA_SUBJECT_STR

CKA_SERIAL_NUMBER_INT

These attributes mirror the standard attribute (without the _STR or _INT suffix) but present that attribute as a printable value rather than a DER encoding.

For the distinguished name attributes the string will be encoded in the form:
C=Country code, O=Organisation, CN=Common Name, OU=Organisational Unit, L=Locality name, S=State name.

Additional Mechanisms

Additional mechanisms include:

Mechanism	Description
`CKM_NVB`	Message Digest
`CKM_DES_BCF`	Byte cipher feedback
`CKM_DES3_BCF`	Byte cipher feedback, Triple DES
`CKM_DES3_ X919_MAC`	X 9.19 Triple DES MAC
`CKM_DES3_ X919_MAC_GENERAL`	X 9.19 Triple DES MAC
`CKM_ENCODE_X_509`	Derives X.509 certificate
`CKM_DECODE_X_509`	Derives a public key from a X.509 certificate
`CKM_ENCODE_PKCS_10`	Derives PKCS#10 certificate request
`CKM_XOR_BASE_AND_KEY`	XOR two keys together
`CKM_DH_PKCS_PARAMETER_GEN`	Diffie Hellman parameter generation
`CKM_DSA_PARAMETER_GEN`	DSA parameter generation
`CKM_DSA_SHA1_PKCS`	DSA with SHA1 signature generation
`CKM_WRAPKEY_DES3_EBC`	Wrap a key value with attributes
`CKM_ENCODE_PUBLIC_KEY`	Wrap a public key but with no encryption

Where possible PKCS #11 Version 2.10 (Reference B) header constants were used for the additional mechanisms, otherwise values in the vendor-defined range have been used. Please consult the header file `cryptoki.h` for the actual constant values.

CKM_ENCODE_PKCS_10

This mechanism is used with the C_DeriveKey() function to derive a PKCS#10 certification request from a public key. Either an RSA or DSA public key may be used with this function. The PKCS#10 certificate request could then be sent to a Certificate authority for signing.

From PKCS#10:

A certification request consists of a distinguished name, a public key, and optionally a set of attributes that are collectively signed by the entity requesting certification. Certification requests are sent to a certification authority, who will transform the request to an X.509 public-key certificate.

Usage:

- Use `CKM_RSA_PKCS_KEY_PAIR_GEN` to generate a key.
- Add a `CKA_SUBJECT` attribute to the public key, containing the subject's distinguished name.
- Initialize the signature mechanism to sign the request. Note that a digest / sign mechanism must be chosen.
  E.g. `CKM_SHA1_RSA_PKCS`
- Call `C_DeriveKey with the CKM_ENCODE_PKCS_10 mechanism` to perform the generation.
- On success, an object handle for the certificate request will be returned.
- The object's `CKA_VALUE` attribute contains the PKCS#10 request.

For an example on how this mechanism may be used see the source code for the CTDEMO program.

CKM_ENCODE_X_509

This mechanism is used with the C_DeriveKey() function to derive an X.509 certificate from a public key or a PKCS#10 certification request. This mechanism creates a new X.509 certificate based on the provided public key or certification request signed with a CA key.

The new certificate validity period will be based on the CKA_START_DATE and CKA_END_DATE attributes on the base object. If the start date is missing the current time is used and if the end date is missing the certificate will be valid for one year. These dates may be specified as relative values by adding the + character at the start of the date value. The start date is relative to 'now' and the end date is relative to the start date if relative times are specified. Negative relative times are not allowed. If the either the start or end date are invalid then the error CKR_TEMPLATE_INCONSISTENT will be returned.

The certificate's serial number will be taken from the template's CKA_SERIAL_NUMBER, CKA_SERIAL_NUMBER_INT or the signing key's CKA_USAGE_COUNT in that order. If none of these values is available CKR_WRAPPING_KEY_HANDLE_INVALID error will be returned.

To determine the Subject distinguished name for the new certificate if the base object is a public key the algorithm will use the CKA_SUBJECT_STR, CKA_SUBJECT from the template or the base key (in that order). If none of these values is available CKR_KEY_HANDLE_INVALID will be returned.

If the base object is a Certification request or a self signed certificate the subject will be taken from the object's encoded subject name.

Currently this mechanism supports generation of RSA or DSA certificates.

On success, a handle to a new `CKO_CERTIFICATE` object will be returned. The certificate will include the `CKA_ISSUER`, `CKA_SERIAL_NUMBER` and `CKA_SUBJECT` attributes as well as a `CKA_VALUE` attribute which will contain the DER encoded certificate.

Usage:

- Create a key-pair using the `CKM_RSA_PKCS` mechanism (this is the key-pair for the new certificate), or
- Create a `CKO_CERTIFICATE_REQUEST` object (with the object's `CKA_VALUE` attribute set to the PKCS#10 data)
- This object is the "base-key" used in the C_DeriveKey function
- Initialise the signature mechanism to sign the request. Note that a digest / sign mechanism must be chosen.
  E.g. CKM_SHA1_RSA_PKCS
- Call `C_DeriveKey with CKM_ENCODE_X_509` to perform the generation

The new key's template may contain:

`CKA_ISSUER_STR` `CKA_ISSUER`	The distinguished name of the issuer of the new certificate. If this attribute is not included the issuer is taken from the signing key's `CKA_SUBJECT` attribute. CKA_ISSUER is the encoded version of this attribute.
`CKA_SERIAL_NUMBER_INT` `CKA_SERIAL_NUMBER`	The serial number for the new certificate. If this attribute is not included the serial number is set to the value of the `CKA_USAGE_COUNT` attribute of the signing key. CKA_SERIAL_NUMBER is the encoded version of this attribute.
`CKA_SUBJECT_STR` `CKA_SUBJECT`	If the base key (i.e. the input object) is a public key the either template must contain this attribute or the public key must have a CKA_SUBJECT attribute. This attribute contains the distinguished name of the subject. When the base key is a PKCS#10 certification request the `CKA_SUBJECT` information is taken from there. CKA_SUBJECT is the encoded version of this attribute.
`CKA_START_DATE` `CKA_END_DATE`	These attributes will be used to determine the new certificate's validity period. If the start date is missing the current date is used, if the end date is missing the date will be one year from the start date. Relative values may be specified (see above).

For an example on how this mechanism may be used see the source code for the CTDEMO program.

CKM_DSA_PARAMETER_GEN

This mechanism is used with the C_GenerateKey() function to derive a set of DSA parameters for subsequent DSA key generation. The resulting object is not a key and should only be used to extract the parameters into a DSA public key template for key generation purposes.

CKM_DH_PKCS_PARAMETER_GEN

This mechanism is used with the C_GenerateKey() function to derive a set of DH parameters for subsequent key generation. The resulting object is not a key and should only be used to extract the parameters into a DH public key template for key generation purposes.

CKM_WRAPKEY_DES3_EBC

This mechanism is used to wrap a key value plus all of its attributes so that the entire key can be reconstructed without a template at the destination. The key value is encrypted using triple DES and all key attributes are MACed in the encoding. The Wrapping key is supplied as normal to the C_Wrap and C_Unwrap() cryptoki functions.

The C_Unwrap() operation will fail with CKR_SIGNATURE_INVALID if any of the key's attributes have been tampered with while the key is in transit.

Key Generation Variations

DSA and DH key generation is a two step process, where generation parameters produced in step one may be used repeatedly for key pair generation in step two. SafeNet PKCS#11 specifies that step one is outside the API while step two, generation of the actual key pair, is inside. This implementation allows step one to be done inside the library. The support is invoked by not supplying the required “parameters” values in the key templates. Under these circumstances a fully compliant PKCS#11 implementation would return `CKR_TEMPLATE_INCOMPLETE`.

Note that the DSA and DH parameters may be generated separately using the other extension CKM_xxx_PARAMETER_GEN making this extension unnecessary. The use of the alternative mechanism (CKM_xxx_PARAMETER_GEN) is recommended.

PKCS#11 Interpretations

a) The handle for an object may change over the lifetime of the token or object. The handle is allocated to the object when it is read from the token.

b) `C_GetObjectSize` reports the sum of the sizes of all the attributes combined for the object. This gives a good indication of the amount of memory committed to the object although there would be some storage overhead for persistent objects.

c) Certain key wrapping restrictions are not observed. For example, wrapping a multi DES key with a single DES key is not prevented.

d) All key sizes for secret key algorithms, as reported by C_GetMechanismInfo, are reported in bytes not bits.

Software-Only Version Specific

a) Token serial numbers are all fixed as “0”.

b) Token removal processing has not been supported since software tokens can not really be removed in the normal sense. The token can actually be removed by deleting, or renaming the “token” directory found in the “slot” directory, but detection of this by the implementation has not been implemented.

c) File system errors are typically reported as `CKR_DEVICE_ERROR`.

d) Debug version supports error logging to a log file named in the **cryptoki.ini** file

System Information

ProtectToolkit C - Software only

SW only **cryptoki.dll** reads **cryptoki.ini** on startup and processes the following items :

```
[SAFENET_SwOnly]
    LogFile = "c:\cryptoki.log"
    SlotDir = "c:\cryptoki"
```

The values shown are the current defaults.

The value of “SlotDir” will be used as the root directory for the slots and will be created if it does not exist. The software will also write a README.TXT file to this directory to indicate where this directory came from and what it is for.

Directories for the slots will be created as subdirectories in the Cryptoki root directory. The slot directory names will be SlotX where X is the number of the slot starting from 0. This number will also be returned as the SlotID value for the slot.

The software will look for a subdirectory named “token” in each slot directory. If the directory exists then the slot will be reported as containing a token. Otherwise it will report that the slot is empty. Any other files or directories in the slot directory will be ignored.

Note that the “token” directory will be created automatically if `C_InitToken` is called on an empty slot. This is a non-standard PKCS#11 extension included with ProtectToolkit C.

The token directory should be empty for an uninitialised token. On initialisation it will contain the following files.

label - Contains the label specified when the token was initialised.
so.pvc - Contains the security officer PIN verification code.

When the user PIN is initialised the following file appears:

user.pvc - Contains the user’s PIN verification code.

Each token object is stored in a file in one of the following formats:

PubX.obj - Public object, where X represents the object handle (hex).
UsrX.obj - User object.

To prevent tampering of these files while on disk, the contents of private objects are always encrypted using a key derived from the user’s PIN. The contents of all files, public and private, are MACed using keys derived from login PIN values. This protects the confidentiality and integrity of the token’s data while outside of the library’s control. Sensitive attributes are triple DES encrypted for additional safety. Public objects cannot use keys derived from PINs since their contents must be available when no user is logged in. Their entire contents are not encrypted, but are MACed and sensitive attributes are encrypted under static keys.

ProtectToolkit C - Remote Client

Remote client **cryptoki.dll** reads **cryptoki.ini** on startup and processes the following items :

```
[SAFENET_ Remote]
    LogFile = "c:\cryptoki.log"
    ServerName = 127.0.0.1
    ServerPort = 12396
```

The values shown are the current defaults.

End of Document