SafeNet Authentication Client (Linux)

Developer's Guide
Version 8.0 Revision A

Support

We work closely with our reseller partners to offer the best worldwide technical support services. Your reseller is the first line of support when you have questions about products and services. However, if you require additional assistance you can contact us directly at:

Telephone

You can call our help-desk 24 hours a day, seven days a week:

USA: 1-800-545-6608

International: +1-410-931-7520

Email

You can send a question to the technical support team at the following email address:

support@safenet-inc.com

Website

You can submit a question through the SafeNet Support portal:

http://c3.safenet-inc.com/secure.asp

Additional Documentation

We recommend reading the following SafeNet Token publication:

- SafeNet Authentication Client (Linux) 8.0 Administrator's Guide
- SafeNet Authentication Client (Linux) 8.0 User's Guide
- SafeNet Authentication Client (Linux) 8.0 ReadMe

Table of Contents

Chapter 1

Overview

This chapter provides an overview of SafeNet Authentication Client 8.0

- SafeNet Authentication Client 8.0
- Choosing the Correct API
- Obsolete APIs
- Developing in Non-C/C++ Environments
- A Note on Cryptography
- Password Management
- Supported Token Models

SafeNet Authentication Client 8.0

SafeNet Authentication Client 8.0 installs all the necessary files and token drivers to support token integration with various security applications. It enables operating systems and third party applications to access the token. Installing the SafeNet Authentication Client allows communication with all available token devices and forms the basis for SafeNet's various management solutions, such as Token Management System (TMS); a complete framework for managing all aspects of token assignment, deployment and personalization within an organization.

SafeNet's token based PKI Solutions enable the implementation of strong two-factor authentication using standard certificates. Generic integration with PKCS#11 security interfaces enables interoperability with a variety of security application such as Web Access, VPN Access, Network Logon, PC Protection and Secure eMail. PKI keys and certificates can be securely created, stored and used from within the token.

When used with eToken PRO/Smartcard or eToken NG-OTP the PKI Private keys can be generated and operated on board the secure chip.

SafeNet Authentication Client supports the various types of token device in both form factors. This means that only a single PKI installation is required to enable operations of either a traditional Smartcard or a USB token (PRO/ NG-OTP), resulting in easy deployment and cost effective installation in use of token products and solutions.

SafeNet Authentication Client can be deployed and updated using any standard software distribution system such as TMS. In addition, the TMS supports software distribution using the Microsoft GPO system.

Choosing the Correct API

You have the following options when deciding which methodology to use when programming applications that utilize the token:

- PKCS#11
- SAPI

PKCS#11

PKCS#11 is a Public-Key Cryptography Standard (PKCS) for public-key cryptography, developed by RSA Laboratories and includes both algorithm-specific and algorithm-independent implementation standards. It is an industry standard that defines a technology-independent programming interface for cryptographic devices such as smartcards and PCMCIA cards.

This standard specifies an application program interface (API), called Cryptoki (Cryptographic Token Interface), to devices, either physical or virtual, which hold cryptographic information (keys and other data) and perform cryptographic functions.

This API is used across many platforms and is powerful enough for most security-related applications. SafeNet uses PKCS#11 as the main API for token programming.

All vendor-specific token functionality is available either via PKCS#11 API or via proprietary extensions developed to be usable in PKCS#11 applications.

SAPI

SAPI (Supplementary API) was first introduced in eToken RTE 3.60 to overcome the requirement for applications to use obsolete low-level APIs. It provided access to the token-specific capabilities not covered by the PKCS#11 standard. This functionality is available now via PKCS#11 API.

SAPI is supported in SafeNet Authentication Client (with restrictions described later in this document) and may be used by applications. However the new features are not supported via SAPI. You are encouraged to use PKCS#11 for new development (unless the application is required to run on eToken RTE 3.65 as well).

Obsolete APIs

- eToken SDK 3.51 described several low-level APIs, such as:
  - eToken API that provided low-level access to a token

- A partly documented APDU-level APDU that allowed sending a separate APDU to the token
- eTOCX COM-API for low-level token functionality.

- All low-level APIs are considered obsolete and are not supported in PKI Client 5.0.
- The Certificate Store is not supported in PKI Client 5.0. The functionality is now available through the PKI Client taskbar menu (PKI Monitor).

Developing in Non-C/C++ Environments

Using Sun's PKCS#11 Provider in Java

The Java platform defines a set of programming interfaces for performing cryptographic operations. These interfaces are collectively known as Java Cryptography Architecture (JCA) and Java Cryptography Extension (JCE).

For more information, refer to:

http://java.sun.com/j2se/1.5.0/docs/guide/security/index.html

The Sun PKCS#11 provider acts as a bridge between JCA and JCE interfaces and the native PKCS#11 provider. However, working in this manner does not provide full access to all the functionality of PKCS#11. For instance, it is only possible to work with keys and certificates.

This functionality is supported since Java 2 Platform Standard Edition 5 (J2SE 5.0).

Writing Wrapper Objects

The alternate solution (for either programming environment) is to write a wrapper object that answers the particular needs of the application. Depending on these needs, the wrapper object may be a .NET assembly object, an ActiveX object or something else.

A Note on Cryptography

Application developers use token for one or two main purposes:

- Adding strong cryptographic capabilities to the application
- As a secure data repository

Using token for secure data storage is simple and obvious, but cryptography is a complicated subject and requires a few comments.

Cryptography itself is not too complicated to understand, but it can be very complicated to use in the correct manner. There are a number of good references to learn about cryptography (See *Additional Cryptography Information Sources* on page 5) but the first rule is do not reinvent the wheel. In most cases it will not work correctly.

There are many good cryptographic algorithms available, but they need to be used properly. Many of the real world security attacks are the result of improper use of cryptography rather than the use of weak algorithms. Each task requires the correct algorithm to be used.

Wherever possible, try to use standard solutions. When a proprietary solution is required, obtain the assistance of a cryptography specialist before developing the application and ensure that the process being followed is appropriate for the purpose.

Additional Cryptography Information Sources

There are a number of sources for information on cryptography. We recommend reading some or all of the following:

Books:

Applied Cryptography by Bruce Schneier

Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition by Bruce Scneier

Practical Cryptography by Niels Ferguson, Bruce Schneier

Web Sources:

http://theory.lcs.mit.edu/~rivest/crypto-security.html

http://www.cryptovb.com/links/links.html

Password Management

Multi-Language Support

Most applications, irrespective of the selected API, need to transfer the password to the token. Using non-ASCII characters in the password may result in serious compatibility problems with other applications. SafeNet Authentication Client UI converts the password to the UTF-8 encoding (as required by later versions of PKCS#11).

In order to avoid compatibility problems we recommend that the password includes only printable ASCII characters.

Password Policy Management

The PKI Client (since version 4.0) enforces password policy setting:

- The password must be changed (for example, due to expiration), and the application will not work with the token until the password is changed. The exact behavior depends on the API in use:
  - PKCS#11 application will behave according to the v2.20 spec, which means:
    - **`C_Login`** with **`CKU_USER`** will succeed.
    - The subsequent call to **`C_GetTokenInfo`** returns the **`CKF_USER_PIN_TO_BE_CHANGED`** flag set.
    - Any subsequent call requiring the user to be logged fails with **`CKR_PIN_EXPIRED`** until the user changes the password.

- **`SAPI_Login`** not involving the PKI Client UI fails with **`CKR_SAPI_PIN_EXPIRATION`**.
- The logon operation (in any API) involving PKI Client UI will enforce the user to change the password.

- Password change may fail if the new password does not fit the defined password policy.

Supported Token Models

SafeNet Authentication Client supports multiple token types:

- eToken PRO
- eToken PRO Anywhere
- eToken NG-FLASH
- eToken NG-FLASH Anywhere
- eToken NG-OTP
- SafeNet eToken Rescue
- SafeNet eToken Virtual
- eToken PRO Smartcard

In addition SafeNet Authentication Client may be used to perform cryptographic operations in the absence of any token.

Chapter 2

PKCS#11 and Configuration

This section contains information about the SafeNet Authentication Client implementation of the PKCS#11 standard.

For full specifications on the PKCS#11 standard, refer to:

http://www.rsasecurity.com/rsalabs/node.asp?id=2133.

- PKCS#11 Implementation for SafeNet Authentication Client 8.0
- Supported Object types
- Supported Mechanisms
- PKCS#11 Functions
- Major Backward Compatibility Issues of PKCS#11

PKCS#11 Implementation for SafeNet Authentication Client 8.0

SafeNet Authentication Client 8.0 implements PKCS#11 v2.01.

In addition, SafeNet Authentication Client 8.0 provides implementation of some features defined in later versions of the PKCS#11 standard.

Note:

- PKCS#11 requires the presence of a token to perform any cryptographic operation (even if no key object is created on the token). SafeNet Authentication Client 8.0 allows the application to perform cryptographic operations in the absence of a token.This so-called virtual session is explained later in this document.
- Calling PKCS#11 functions from DLL detach is forbidden. (For more information, see the Note in *C_Finalize* on page 14)

Using libeTPkcs11

`libeTPkcs11` is a shared library (**`libeTPkcs11.so`**) and is installed at the following locations:

- `/usr/lib/` on 32-bit machine
- `/usr/lib64/` and `/usr/lib32/` on Ubuntu 64-bit
- `/usr/lib64/` and `/usr/lib/` on Centos 64-bit and RHEL 64-bit.

We strongly recommend the following:

- Use only explicit dynamic linking when linking with **`libeTPkcs11`**, and use the full path to load the shared library to minimize load times.

- Use the **dlsym()** to obtain the address of **C_GetFunctionList()**. However, to obtain the table of other function pointers in the shared library you should use **C_GetFunctionList()** and not use **dlsym()**.
- **libeTPkcs11** maps a PKCS #11 Slot to any reader that is on the system when the PKCS #11 library is first loaded. A token is present in this slot whenever a token is inserted into that reader.

Supported Object types

- **CKO_DATA**
- **CKO_CERTIFICATE (CKC_X_509** only**)**
  - Regular certificate **(CKA_CERTIFICATE_CATEGORY=0)**
  - CA certificate **(CKA_CERTIFICATE_CATEGORY=2)**
- **CKO_PRIVATE_KEY** (**CKK_RSA** only)
- **CKO_PUBLIC_KEY** (**CKK_RSA** only)
- **CKO_SECRET_KEY**
  - **CKK_DES**
  - **CKK_DES2**
  - **CKK_DES3**
  - **CKK_HOTP**
  - **CKK_AES**
  - **CKK_RC4**
  - **CKK_GENERIC_SECRET**
- **CKO_HW_FEATURE.**

Note:

- **CKK_HOTP** is not supported on token models without OTP capabilities. SafeNet eToken Virtual simulates the OTP capacity and so supports **CKK_HOTP**.
- **CKK_AES, CKK_RC4** and **CKK_GENERIC_SECRET** are supported for eToken Pro (with Format 5), eToken Pro (Java), SafeNet eToken Virtual and virtual sessions.
- **CKO_HW_FEATURE** was introduced in later versions of the PKCS#11 standard to explore various token capabilities. Feature objects supported by SafeNet Authentication Client 8.0 are covered later in this document.

Supported Mechanisms

These mechanisms are supported by all token types:

- **CKM_DES_MAC**
- **CKM_DES_MAC_GENERAL**
- **CKM_DES3_MAC_GENERAL**
- **CKM_DES_ECB**
- **CKM_DES_CBC**
- **CKM_DES_CBC_PAD**
- **CKM_DES3_ECB**
- **CKM_DES3_CBC**
- **CKM_DES3_CBC_PAD**
- **CKM_RSA_PKCS_KEY_PAIR_GEN**
- **CKM_RSA_PKCS**
- **CKM_RSA_X_509**
- **CKM_MD5_RSA_PKCS**
- **CKM_SHA1_RSA_PKCS**
- **CKM_DES_KEY_GEN**
- **CKM_DES2_KEY_GEN**
- **CKM_DES3_KEY_GEN**
- **CKM_PBE_SHA1_DES3_EDE_CBC**
- **CKM_PBE_SHA1_DES2_EDE_CBC**
- **CKM_PBA_SHA1_WITH_SHA1_HMAC**
- **CKM_PBE_MD5_DES_CBC**
- **CKM_PKCS5_PBKD2**
- **CKM_MD5_HMAC_GENERAL**
- **CKM_MD5_HMAC**
- **CKM_SHA_1_HMAC_GENERAL**
- **CKM_SHA_1_HMAC**
- **CKM_MD5**
- **CKM_SHA_1**
- **ETCKM_PBA_LEGACY** (this is a proprietary mechanism described later in this document)

These mechanisms are supported by eToken Pro(with Format 5), eToken Pro (Java), SafeNet eToken Virtual and virtual sessions:

- **CKM_AES_MAC**
- **CKM_AES_MAC_GENERAL**
- **CKM_RC4**
- **CKM_AES_ECB**
- **CKM_AES_CBC**
- **CKM_AES_CBC_PAD**

- **CKM_RC4_KEY_GEN**
- **CKM_AES_KEY_GEN**
- **CKM_PBE_SHA1_RC4_128**
- **CKM_PBE_SHA1_RC4_40**
- **CKM_GENERIC_SECRET_KEY_GEN**
- **CKM_AES_ECB_ENCRYPT_DATA**
- **CKM_AES_CBC_ENCRYPT_DATA**

This mechanism is not supported on tokens without OTP capability:

- **CKM_HOTP**

Note:
SafeNet eToken Virtual simulates the OTP capability, so supports **CKM_HOTP**.

PKCS#11 Functions

This section covers details of SafeNet Authentication Client PKCS#11 implementation.

Note:
On hardware tokens, except for eToken Pro (Format 5) and eToken Pro (Java Card), the application must perform user logon to create, change or remove public objects.

C_Initialize

According to the specifications, the subsequent call to **C_Initialize** should fail with **CKR_CRYPTOKI_ALREADY_INITIALIZED** error. SafeNet Authentication Client returns **CKR_OK** and increments the 'init'-counter. This is done to overcome the problem of PKCS#11-enabled applications calling third-part code that also uses PKCS#11 API.

C_Finalize

According to the specifications, the subsequent call to **C_Finalize** should fail with **CKR_CRYPTOKI_NOT_INITIALIZED** error. SafeNet Authentication Client will return this code only when the number of subsequent **C_Finalize** calls overtakes the number of previous **C_Initialize** calls.

Note:
Calling PKCS#11 functions from DLL detach is forbidden.

For more information see:
http://msdn2.microsoft.com/en-us/library/ms682583.aspx

According to the MSDN description of DllMain, calling functions that require DLLs other than Kernel32.dll may result in problems that are difficult to diagnose. For example, calling User, Shell, and COM functions can cause access violation errors, because some functions load other system components. Conversely, calling functions such as these during termination can cause access violation errors because the corresponding component may already have been unloaded or uninitialized.

C_GetInfo

The information returned by this function may vary between different versions of PKI Client. For SafeNet Authentication Client 8.0 the following information is returned:

- cryptokiVersion = 2.01
- manufacturerID = "SafeNet, Inc."
- libraryDescription = "eToken PKCS#11"
- libraryVersion = 8.0

C_GetSlotList

According to the PKCS#11 v2.01 standard, the number of slots returned to an application once cannot change until **C_Finalize**. In reality, this may be problematic. For instance, the user may connect/disconnect the smartcard readers. Later versions of the standard allow returning different number of slots once the function is called again with a NULL value of the **pSlotList** parameter. SafeNet Authentication Client does not do this for several reasons. One of the major reasons is problem of PKCS#11-enabled applications using third-part code which also uses PKCS#11. The application may not be aware of **C_GetSlotList** call done by third-part code, while SafeNet Authentication Client cannot distinguish the source of call (in the context of the same process). Therefore the SafeNet Authentication Client 8.0 behaves as following:

- The number of slots reported by the **C_GetSlotList** function does not change after the **C_Initialize** call.
- The number of reported slots (built from the number of SafeNet eToken Virtual slots and PC/SC slots for smartcard readers and hardware token devices) is configurable.
- The PC/SC readers are mapped to the PKCS#11 slots. Once mapped, a slot cannot be used for another reader until **C_Finalize** is called (to avoid confusing of applications using slot description).

C_GetSlotInfo

The information returned by this function depends on the slot in use and may vary between PKI Client versions.

The SafeNet Authentication Client 8.0 returns the following information:

- slotDescription is
  - <smartcard reader name> for PC/SC slots.
  - <file path> for software slots (cut to the field size).
  - <empty> for reserved slots
- manufacturerID = "SafeNet, Inc."
- flags
  - **CKF_REMOVABLE_DEVICE**

- **CKF_HW_SLOT** except for SafeNet eToken Virtual
- **CKF_TOKEN_PRESENT** if the token is present in the slot

- hardwareVersion
  - 0.0 for PC/SC slots in absence of virtual device
  - 1.0 for token hardware devices and SafeNet eToken Virtual managed by SafeNet Authentication Client drivers (That is, any token device except for token Card and CCID-compliant tokens)
  - 2.0 for smartcard readers and CCID-compliant token.
- firmwareVersion = 0.0

Note: More information may be received via vendor-specific extensions.

C_GetTokenInfo

The returned information depends on the token in use. In some minor details it may vary between PKI Client versions. The information returned by this function has been extended according to latest PKCS#11 version.

For SafeNet Authentication Client 8.0 the following information will be returned:

- label - token label (UTF-8)
- manufacturerID = "SafeNet, Inc."
- model = "eToken"
- serialNumber = token serial number
- flags
  - **CKF_RNG** - except for token initialized as FIPS
  - **CKF_LOGIN_REQUIRED** - except for token initialized as one-factor, an empty token, or SafeNet eToken Virtual.
  - **CKF_USER_PIN_INITIALIZED** if **C_InitPIN** has been issued
  - **CKF_DUAL_CRYPTO_OPERATIONS**
  - **CKF_TOKEN_INITIALIZED** if **C_InitToken** has been called
  - **CKF_USER_PIN_COUNT_LOW** if user PIN retry counter < 3

- **CKF_USER_PIN_LOCKED** whenever user PIN is locked. It may be not possible to know that the PIN is locked till unsuccessful **C_Login trial**.
- **CKF_USER_PIN_TO_BE_CHANGED** if user PIN must be changed (enforced on initialization or due to expiration). This flag is guaranteed to be set only after successful **C_Login** with **CKU_USER**.

- **ulMaxSessionCount** = **CK_EFFECTIVELY_INFINITE**
- **ulSessionCount** = actual number of sessions that this application currently has open with the token
- **ulMaxRwSessionCount** = **CK_EFFECTIVELY_INFINITE**
- **ulRwSessionCount** = actual number of read/write sessions that this application currently has open with the token
- **ulMaxPinLen** = 255
- **ulMinPinLen** depends on current password policy
- **ulTotalPublicMemory**
  - taken from token for hardware tokens
  - **CK_EFFECTIVELY_INFINITE** for software tokens
- **ulFreePublicMemory**
  - taken from token for hardware tokens
  - **CK_EFFECTIVELY_INFINITE** for software tokens
- **ulTotalPrivateMemory** - same as **ulTotalPublicMemory**
- **ulFreePrivateMemory** - same as **ulFreePublicMemory**
- hardwareVersion
  - taken from token (except for Token Card and SafeNet eToken Virtual)
  - 0.0 for Token Card and SafeNet eToken Virtual
- firmwareVersion
  - taken from token (except for Token Card and SafeNet eToken Virtual)
  - 0.0 for Token Card and SafeNet eToken Virtual
- **utcTime** - empty

Note:
More information may be received through vendor-specific extensions.

C_WaitForSlotEvent

PKCS#11 does not provide a mechanism to cancel the blocked **C_WaitForSlotEvent** call except for **C_Finalize**. The SafeNet Authentication Client provides more flexible mechanism through vendor-specific extensions.

To cancel **C_WaitForSlotEvent** function execution, the tracker handle should be created in advance.This can be done using the function **ETC_CreateTracker**.

After the tracker handle creation, its handle should be passed to **C_WaitForSlotEvent** using the **pReserved** parameter. In this case, the execution of **C_WaitForSlotEvent** can be canceled calling the function **ETC_DestroyTracker**.

See also: **ETC_CreateTracker** (page 64), **ETC_DestroyTracker** (page 65).

C_InitToken

The behavior of C_InitToken in PKI Client 5.0 is not compatible with earlier versions of eToken RTE. The results of **C_InitToken** may be affected by PKI Client settings. Look for vendor-specific extensions for more details.

C_SetPIN

- The function **C_SetPIN** may be called with NULL as old and new PIN. In this case the SafeNet Authentication Client UI will be launched.
- If the user (**CKU_USER**) PIN cannot be changed due to password policy restrictions of the token, **CKR_INVALID_PIN** will be returned.
- If the new user (**CKU_USER**) PIN does not fit the password policy requirements of the token, **CKR_INVALID_PIN** or **CKR_PIN_LEN_RANGE** will be returned.

C_Login

- If the user PIN must be changed, the **C_Login** will succeed, but the subsequent calls to **C_GetTokenInfo** will return **CKF_USER_PIN_TO_BE_CHANGED** flag set. Any further call requiring the user to be logged in will fail as long as **C_SetPIN** is not issued.
- The function **C_Login** may be called with NULL as a PIN. In this case the SafeNet Authentication Client UI will be launched. It will also handle the issue of password change (according to the password policy of the token).
- In Single Logon mode, the user can log on without entering a PIN value, if the PIN value was supplied to the PKI Client during a previous log on. To enable this, the PIN parameter must contain a predetermined value. You can get the predetermined value using **ETC_SingleLogonGetPin**. See **ETC_SingleLogonGetPin**, on page 68.

C_CopyObject

The function **C_CopyObject** does not allow objects to be copied between tokens, as it is not possible for hardware tokens to copy private RSA keys.

C_GetObjectSize

According to the PKCS#11 specifications, the **C_GetObjectSize** function returns an approximate object size and may be slightly inaccurate. As a result, when deleting or creating an object (for example using **C_CopyObject**), the reported number of bytes may not be completely accurate and should be monitored.

C_GetAttributeValue

SafeNet Authentication Client may return particular attributes of private objects even after **C_Logout**.

The following attributes may be returned:

- **CKA_TOKEN**

- **CKA_PRIVATE**
- **CKA_CERTIFICATE_TYPE**
- **CKA_KEY_TYPE**
- **CKA_ALWAYS_AUTHENTICATE**
- **CKA_MODULUS**
- **CKA_PUBLIC_EXPONENT**

C_GenerateKeyPair

SafeNet Authentication Client 8.0 allows passing NULL as a **phPublicKey** parameter. In this case only a private key will be generated.

Token devices based on CardOS version 4.01 ignore the public exponent value passed by an application.

C_DeriveKey

The function **C_DeriveKey** returns **CKR_FUNCTION_NOT_SUPPORTED**.

C_SeedRandom

The function **C_SeedRandom** returns success (**CKR_OK**).

C_CreateObject

The boolean attribute **ETCKA_DESTROYABLE** is used to create non-deletable PKCS#11 objects. This attribute is given in **C_CreateObject** function in the caller-defined object's template. To create non-deletable PKCS#11 objects (RSA keys, certificates, data objects and so on) the **ETCKA_DESTROYABLE** attribute must be equal to **CK_FALSE**. The objects defined as non-deletable can't be changed, except by initializing the token.

The default value is **CK_TRUE**. This means the new object will be deletable by default if its creation template has no **ETCKA_DESTROYABLE** attribute.

X.509 Attribute Tolerance

The following conditions should be met when creating a certificate:

- The certifcate must be created in the DER encoded X.509 format.
- The serial number, subject and issuer objects of the internal certificate (**CKA_VALUE**) must match the external attributes (**CKA_SERIAL_NUMBER**, **CKA_SUBJECT**, and **CKA_ISSUER**).

If the above conditions are not met, the `TolerantX509Attributes` attribute must be set to 1(True). If set to 0(False), the above conditions must be met.

TolerantX509Attributes was enabled by default in eToken PKI Client versions earlier than 4.0.

Major Backward Compatibility Issues of PKCS#11

The following compatibility issues relate to eToken RTE 3.65. There has been no change in these issues between etoken PKI Client 4.0 and 5.0.

- Behavior of functions **`C_InitToken`** and **`C_InitPIN`** has changed.
- Previous versions might display pop up windows in various situations. For instance, if token requires user to be logged in to change the public object, the UI could appear to ask for the user password. PKI Client 4.0, 4.5 and 5.0 do not display a UI except that covered by this document (which may be predicted and eliminated by the application).
- Information returned by **`C_GetSlotInfo`** and **`C_GetTokenInfo`** varies.
- PKI Client now enforces password policy mechanisms as described in later versions of the PKCS#11 standard.
- The implementation of **CryptAcquireContext** deals with flags (**CRYPT_NEWKEYSET | CRYPT_SILENT**) in different ways in PKI Client 4.55 and 5.00.

  In the previous versions of PKI those flags lead to function failure, while in the current version of PKI the function succeeds.

CryptAcquireContext with CRYPT_NEWKEYSET is a first step in the two-step process of RSA key generation or import. The entire process requires that the user enters a token password. Since the flag CRYPT_SILENT contradicts this, the process of the key generation or import will fail.

The difference between previous and current implementations is that in the previous implementations, the failure occured in the first step (CryptAcquireContext), while the new implementation failure occurs at the second step (CryptGenKEy or CryptImportKey) only if the user didn't log in.

Chapter 3

Token-Specific PKCS#11 Extensions

SafeNet Authentication Client 8.0 provides a wide functionality that cannot be expressed in terms of the PKCS#11 standard. It is provided through special extensions to PKCS#11.

In this chapter:

- General Overview
- Vendor-Specific Information
- Reference

General Overview

Understanding Token-Specific PKCS#11 Extensions

The extensions are as follows:

- Extensions to the future standard versions. The SafeNet Authentication Client implements PKCS#11 v2.01. However, it supports many features introduced in the later versions of the PKCS#11 standard. They are covered in this document. The examples of such extensions are:
  - OTP support;
  - Distinguishing between user and CA certificates;
- Return codes and flags related to the password policy.
- Vendor-specific attributes that can be implemented in the regular PKCS#11 objects. The example of such attribute is duration of OTP value presentation on the token.
- Feature objects: PKCS#11 v2.20 introduces hardware feature objects (**CKO_HW_FEATURE**) as PKCS#11 objects representing various device features. The application does not create such objects. Neither has it found them in the typical **C_FindObjectsInit** call unless the particular feature type was explicitly specified in the search template. SafeNet Authentication Client uses feature objects to represent and manage various token characteristics. The most useful feature objects are token object (**ETCKH_TOKEN_OBJECT**) representing the whole token and Password Policy object (**ETCKH_PIN_POLICY**) representing PIN policy settings of the token. Feature objects are not considered storage objects, so they do not have a **CKA_PRIVATE** attribute. Unless specified differently, they behave as follows:
  - They may be read without authentication.
  - They have attribute **CKA_MODIFIABLE** attribute specifying if the object may be changed in principle. Some attributes cannot be changed by program: they represent the real status of the object and are changed only as a part of token operations (such as last password change date).
  - They have attribute **ETCKA_OWNER (CK_ULONG)** defining who may change the object. The token object has not such attribute. The supported values are:

- **CKU_USER**
- **CKU_SO**
- If **CKA_MODIFIABLE** is FALSE, this attribute should be ignored

- Vendor-specific functions are used whereas some SafeNet Authentication Client functionality cannot be expressed properly via existing PKCS#11 functions. The application uses function **ETC_GetFunctionListEx** to get the structure of pointers to the vendor-specific functions (just as it uses **C_GetFunctionList** to get the structure of pointers to the standard functions).

The following sections describe the functionality provided by the token-specific PKCS#11 extensions. The next section contains the formal reference.

Encoding of Text Attributes, Fields and Parameters

UNICODE Support

Some functions have parameters that may be treated as text strings. The typical example is filename of SafeNet eToken Virtual. Whenever it is not mentioned, UTF-8 is assumed. A function that creates SafeNet eToken Virtual or associates it with the slot should get a UTF-8 string. In non-UNICODE applications, passing a null-terminated ASCII string is enough.

Note:
A particular token may have restrictions on the storing of non-ASCII information for backward compatibility. We therefore recommend using ASCII-only.

Null-termination of strings

Usually, the PKCS#11 spec does not use null-terminators for strings:

- For object attributes, the attribute length is passed explicitly.
- Structure fields have a fixed size (and are usually blank-padded).

SafeNet Authentication Client extensions, in most cases, follow a more friendly convention. Unless specified otherwise:

- The returned attributes are null-terminated (the attribute length takes into account the null-terminator).
- The SafeNet Authentication Client is ready to get the incoming attributes with and without a null-terminator.

SafeNet Authentication Client

Many aspects of the SafeNet Authentication Client behavior are configurable. Typical examples of configurable behavior are:

- Password quality settings to be used when initializing the token.
- 2048-bit RSA key support.

The configurable aspects of the SafeNet Authentication Client vary in their flexibility:

- Some of them may be modified by user, while others may be modified only by the computer administrator.
- Some of them may be configurable permanently on a per-application basis.
- Some of them may even be modified per-process in the run time.

The SafeNet Authentication Client provides the following functions to deal with properties:

- **`ETC_GetProperty`** returns the current setting of particular property. Depending on the property the same value may be or may be not returned for all processes.
- **`ETC_SetProperty`** sets the new value for the property. If an invalid value is passed, the function may not fail, but the result may not be as expected by the application. The property may be changed for the entire process or for the current thread only, depending on the passed **`ETCKF_PROPERTY_THREAD`** flag. Passing NULL value effectively resets the effect if the all previous calls to **`ETC_SetProperty`**.

Slot/Token IOCTL

Some rarely used operations are not presented as separated functions. They are joined to multi-purpose IOCTL functions instead. There are two such functions:

- **ETC_DeviceIOCTL** gets slot ID as a parameter. This function is necessary since standard PKCS#11 provides almost no functions that can be applied to the slot. It is dedicated to operations targeted to a particular slot (possibly even in the absence of the token). An example of such operation is getting the filename associated with SafeNet eToken Virtual.
- **ETC_TokenIOCTL** gets session and object handles as parameters. This function is necessary for operations that cannot be reasonably impressed via existing PKCS#11 functions (like getting or setting particular attributes).

Extensions Related to Operations with Slots and Tokens

SafeNet eToken Virtual

The SafeNet eToken virtual is a file in a special format that acts as a token in terms of PKCS#11. Private objects in this token are encrypted with the user password. It is mostly used as a part of employee on the road solution.

- Use **ETC_DeviceIOCTL** with **ETCK_IODEV_SOFTWARE_TOKEN_PLUGIN** to plug-in the SafeNet eToken virtual to the free slot.
- Use **ETC_DeviceIOCTL** with **ETCK_IODEV_FULL_NAME** to get the file name of the file associated with the slot.
- Use **ETC_DeviceIOCTL** with **ETCK_IODEV_SOFTWARE_TOKEN_PLUGOUT** to disconnect the SafeNet eToken Virtual. The file is not removed on this stage; the application may remove it manually.

Note:
SafeNet eToken virtual does not support **CKU_SO**.

Transactions

The SafeNet Authentication Client ensures that no other application will use the same token during a single application call to any API function. However it is possible to lock a token for longer period of time. The function **ETC_BeginTransaction** is used to lock the token for the exclusive use of the application. The function **ETC_EndTransaction** unlocks the token.

Note the following:

- Transactions are usable when application prepares its data on the token. It ensures that the token state is not changed by any other application during the transaction.
- Transactions are usable when application performs the sequence of operations with the token. It may increase the performance significantly.
- Transactions do not guarantee a log of changes done to the token. If the token is removed or the application crashes during transaction, the data on the token will not be rolled back. It is more similar to the smartcard locking, not to the database transaction.
- As long as the transaction is opened, other applications will not be able to access the token, even for the most innocent operations (such as **C_GetSlotList**). Do not keep the transaction opened longer that it is needed.
- Do not use any interactive UI during transactions.

Notifications

A major problem with the standard event tracking mechanism in PKCS#11 (**C_WaitForSlotEvent**) is that it cannot be stopped. It will continue listening till **C_Finalize** call. Sometimes this behavior is too restrictive. The proposed solution uses the last reserved parameters of the function. The application should:

- Create the tracker by calling proprietary function **ETC_CreateTracker**.
- Call **C_WaitForSlotEvent**, passing the pointer to the tracker as a parameter in order to start listening.
- Destroy the tracker by calling the proprietary function **ETC_DestroyTracker** to stop listening.

Token-Less Operations

PKCS#11 cryptographic operations are performed within a session opened with a token. So, in the absence of a token, cryptographic operations cannot be launched (even if it would use only session objects). SafeNet Authentication Client overcomes this restriction with the function **`ETC_CreateVirtualSession`**. This allows the creation of a session that is not associated with any token. Multiple calls to this function create multiple independent sessions. The virtual session is used to perform token-less operations, such as cryptographic operations used for token unblocking. The function **`C_GetSessionInfo`** for a virtual session will fail with error code **`CKR_FUNCTION_FAILED`**.

Password Quality Settings

The SafeNet Authentication Client enforces the password quality settings when working with the token. The password quality settings may include:

- Minimum password length
- Maximum password usage period
- Minimum password usage period
- Warning period before password expiration
- Password history size
- Minimum character repetitions in a password
- Password complexity rules
  - Numerals, upper-case letters, lower-case letters or special characters permitted, mandatory or forbidden.

In addition, the administrator may enforce users to change the password upon the first use.

The password quality settings for the particular token is defined during the token initialization and may be changed later. If no password quality setting is stored on the token (it may be done intentionally or the token may be initialized with a previous version of eToken PKI Client), the current settings of the SafeNet Authentication Client on the machine will be used. They are defined by the corresponding SafeNet Authentication Client.

Note:
The password policy mechanism implemented in the eToken PKI Client 4.0, 4.5 and 5.0 is not backward compatible with the former versions of the eToken PKI Client.

The mechanism operates as described in the PKCS#11 v2.20:

- If user must change the PIN (due to administrator enforcement or expiration), the **C_Login** with **CKU_USER** will succeed.
- The subsequent call to **C_GetTokenInfo** will return **CKF_USER_PIN_TO_BE_CHANGED** flag set.
- Until **C_SetPIN** is done, any subsequent PKCS#11 call which requires user to be logged in will fail with the error code **CKR_PIN_EXPIRED**.
- If the newly supplied PIN cannot be accepted due to the current password policy, the function **C_SetPIN** will fail with the error code either **CKR_PIN_LEN_RANGE** (for too short password) or **CKR_PIN_INVALID** (for any other failure).

Extended features in eToken PKI Client 5.0

The eToken PKI Client 5.0 extends the standard capabilities with the following features:

- Using **C_SetPIN** with NULL passed as both old and new PIN, the application invokes the eToken PKI Client UI. It manages automatically all aspects of the password policy, performing all necessary checks.
- Similarly using **C_Login** with NULL in the case when the user password change is enforced will switch you automatically to the password change process.
- The password policy settings of the particular token may be examined and controlled by using the special hardware feature object **ETCKH_PIN_POLICY**. This object may be created during the token initialization.

- The current settings of the PKI Client itself may be examined by reading the corresponding properties.
- By issuing **C_GetSessionInfo** after **C_Login** or after unsuccessful **C_SetPIN**, the application may get more detailed information about particular need to change the password or about particular reason of failure to change it. It is returned in the **ulDeviceError** field.
- The **IOCTL ETCK_IOCTL_PIN_EVALUATE** may be issued to check the acceptance of the new password without a real attempt to change it.

Additional Notes

- The password policy is applied only to the user PIN.
- When administrator sets the user PIN (**C_InitPIN** or **ETC_InitPIN**) functions, the password policy is not applied.
- When the using PIN is unlocked by using challenge-response authentication (**ETC_UnlockPIN**) the password policy is applied but the minimal password age is ignored.

Special Token Capabilities

OTP

SafeNet Authentication Client supports One Time Passwords (OTP) as defined in PKCS#11 v2.20 Amendment 1 (PKCS#11 Mechanisms for One-Time Password Tokens). The only currently supported mechanism is **CKM_HOTP**.

When using OTP functionality, the following should be kept in mind:

- Not all tokens support OTP functionality. The application should check whether the token supports the **CKM_HOTP** mechanism.
- The supported key length may vary between tokens. It should be checked by application by using the **C_GetMechanismInfo** function.
- The hardware tokens supporting OTP have some restrictions. In particular:
  - Only one OTP key per token is supported.

- The flag **CKF_NEXT_OTP** is not supported for hardware tokens (for security reasons).
- The flag **CKF_EXCLUDE_COUNTER** is supported, but on some older models of eToken NG-OTP the operation may fail.
- The following attributes are not supported for hardware tokens:
  - **CKA_OTP_USER_IDENTIFIER**
  - **CKA_OTP_SERVICE_IDENTIFIER**
  - **CKA_OTP_SERVICE_LOGO**
  - **CKA_OTP_SERVICE_LOGO_TYPE**
- The following vendor-specific attributes are supported for the hardware tokens:
  - **ETCKA_OTP_DURATION** - duration of OTP value representation on the LCD, in seconds. May be passed during creation. May be possible to change later, depending on **ETCKA_MAY_CHANGE_DURATION**.
  - **ETCKA_MAY_SET_DURATION** - defines whether it is possible to change duration after object creation. Cannot be changed after object creation.

- It is possible to use the **CKM_HOTP** mechanism in the virtual session (to perform server-side OTP validation).
- **CKA_OTP_TIME** attribute is not supported as it has no sense for **CKM_HOTP**.
- The newly introduced notification (**CKN_OTP_CHANGED**) is not supported by the SafeNet Authentication Client.
- For SafeNet eToken virtual, the OTP key object behaves according to the **CKA_PRIVATE** attribute. For hardware tokens, **CKA_PRIVATE** attributes is ignored as assumed to be FALSE. However object creation, deletion or changing the display duration requires **CKU_USER** to be logged in.

Other attributes defined in PKCS#11 have the following restrictions:

- **CKA_OTP_FORMAT** - only **CK_OTP_FORMAT_DECIMAL** is supported.
- **CKA_OTP_LENGTH** - must be 6 (according to HOTP spec).
- **CKA_OTP_CHALLENGE_REQUIREMENT** - should be **CK_OTP_PARAM_IGNORED**.
- **CKA_OTP_TIME_REQUIREMENT** - should be **CK_OTP_PARAM_IGNORED**.
- **CKA_OTP_COUNTER_REQUIREMENT** - should be **CK_OTP_PARAM_OPTIONAL** for virtual sessions and **CK_OTP_PARAM_IGNORED** otherwise.

- **CKA_OTP_PIN_REQUIREMENT** - should be **CK_OTP_PARAM_IGNORED**.
- **CKA_OTP_COUNTER** (byte array). It shall be 8-bytes in big-endian form. The default value upon creation is 0. This value is non-modifiable (except for virtual session).

Vendor-Specific Information

Vendor-specific APIs are defined in the eTPkcs11.h header file.

Vendor-Specific OTP Key Attributes

The following vendor-specific attributes are defined for OTP keys:

- **ETCKA_OTP_DURATION** - duration of OTP value representation on the LCD, in seconds. May be passed during creation. May be possible to change later, depending on **ETCKA_MAY_CHANGE_DURATION**.
- **ETCKA_MAY_SET_DURATION** - defines whether it is possible to change duration after object creation. Cannot be changed after object creation.

Secondary authentication

Understanding Secondary Authentication

The SafeNet Authentication Client may support the additional level of protection for RSA private keys. In this mode an additional password must be supplied each time when the cryptographic operation with the RSA private key is performed. This feature is named secondary authentication, meaning that in order to use the particular RSA key, the application should pass an additional (secondary) authentication (the usual user login is considered as a primary authentication).

The secondary authentication is controlled by combination of two factors: the key object attributes passed during the RSA key object creation and the special hardware feature object **ETCKH_2AUTH**.

Most PKCS#11 applications are not aware of the secondary authentication mechanism (as it was added only in v2.20 of the PKCS#11 spec). The special hardware feature object allows configuration of the default behavior of the token, giving more flexibility and power to the off-the-shelf applications.

Creation of the Protected RSA Key

The later versions of PKCS#11 standard introduces the new attribute for the RSA private key object: **CKA_ALWAYS_AUTHENTICATE**. Being set to TRUE during object creation, this attribute defines that the PIN presentation is required each time when the key is used. The standard does not define whether it is the regular user PIN or the separate PIN. The SafeNet Authentication Client uses separate PIN for each such key. Note that this PIN is not synchronized with the regular user PIN (is not changed together with it, is not subject to any PIN policy and cannot be unlocked). This key is required later for each cryptographic operation and is not required for any other operation with the key (such as changing of particular attributes or object deletion). According to the standard, this attribute cannot be changed after object creation.

The PKCS#11 spec does not address the passing of the initial PIN value during key creation. The SafeNet Authentication Client introduces for this purpose the proprietary attribute **ETCKA_2NDAUTH_PIN**, which contains a UTF-8 encoded PIN value. This attribute may be used only during object creation. If this attribute is not passed, the SafeNet Authentication Client will pop up the special window, asking user to enter the password. If it is not supplied (user cancels the operation), the behavior depends on **ETCKH_2AUTH** object, described below.

Supplying Special PIN to the RSA Private Key Operation

The PIN for the secondary authentication is provided according to the v2.20 of the PKCS#11 spec:

- The application issues a PKCS#11 call starting the cryptographic operation (such as **C_SignInit**) with the RSA key protected with the secondary authentication PIN (having **CKA_ALWAYS_AUTHENTICATE** attribute set to TRUE).

- The application issues a **C_Login** call within the same session, passing **CKU_CONTEXT_SPECIFIC** instead of the user type and proper PIN.
- The application completes the cryptographic operation.

SafeNet Authentication Client extends this behavior to support secondary authentication protected keys in the legacy applications in the following way: if **C_Login** with **CKU_CONTEXT_SPECIFIC** was not issued for the secondary authentication protected key, the SafeNet Authentication Client will pop up the window asking user to supply the password.

The SafeNet Authentication Client does not provide an interface to change the per-key PIN after creation.

Configuring Secondary Authentication for the Token

The special hardware feature object **ETCKH_2AUTH** controls the behavior of the secondary authentication feature on a per-token basis. It controls the creation of the new RSA private key objects (once the particular key object has been created, its behavior cannot change anymore). This feature object may be created during the token initialization session (otherwise it is created automatically).

Token initialization

Why Extensions are Needed for Initialization

The initialization functions provided in PKCS#11 (**C_InitToken** and **C_InitPIN**) are not flexible enough. This is because tokens from different vendors are so different, it is almost impossible to cover them all by the same standard API. Below are just several features of SafeNet Authentication Client that cannot be covered by the standard functions.

- It is impossible to define error retry counters for user and SO PIN.
- If the token is initialized in several modes (for instance, FIPS or one-factor), there is no way to address it.
- If the token has special capabilities that should be explicitly enabled (such as OTP support) there is no way to do it.

Proprietary Initialization Functions

The token-initialization is two-or-more steps process. In order to manage this process the following proprietary functions are introduced:

- **`ETC_InitTokenInit`** - this function starts the process of token initialization.
- **`ETC_InitTokenFinal`** - this function completes the process of token initialization.
- **`ETC_InitPIN`** - this function initializes the user PIN (providing more flexibility than the standard function **C_InitPIN**).

Initialization Flow

The flow of initialization is as follows:

1. The application explores token capabilities, such as the ability to support OTP, and pre-requisites required for initialization (need for the old PIN) in the regular session, using the token hardware feature object.
2. The application closes all open sessions with the token. This is because PKCS#11 requires that no sessions are open during **`C_InitToken`** call.
3. **`ETC_InitTokenInit`** starts the initialization process and opens the special session. This session serves as the context of initialization and will be closed in **`ETC_InitTokenFinal`** call. The application may close it by **`C_CloseSession`** call to cancel the initialization. It is important to mention that the token will remain in unpredictable state. SO PIN is ommited in the following cricumstances.
   a. Initializing the token without SO. This will happen if **`C_InitPIN`** or **`ETC_InitPIN`** is called during the initialization session.
   b. Initializing one-factor token or empty token (depending on subsequent calls).
4. If the token requires old password to be presented in order to re-initialize the token, the application should issue the proper **`C_Login call`**.

5. Issue **C_CreateObject** to create the token object. Unless you initialize the token as empty, this is a mandatory step. Optionally issue more **C_CreateObject** calls to create other feature objects. Typically, the password policy object is created here. Note that you create feature objects here. This is allowed only in an initialization session. Any attempt to create an object that makes no sense for the initialization process may fail (or **ETC_InitTokenFinal** may fail).
6. Issue **C_InitPIN** or **ETC_InitPIN**. This function initializes the user PIN. **ETC_InitPIN** gives you more flexibility. These functions are not subject to the password policies. If you do not issue this call:
   a. If the token has SO PIN, the token will be initialized, but the **CKF_USER_PIN_INITIALIZED** flag will be reset in the token info. The application will not be able to perform user login, until the SO initializes the user PIN.
   b. If the token has no SO PIN, one-factor or empty token is assumed (depending on the attributes of the created token object.
7. The **ETC_InitFinal** call completes the token initialization process and closes the initialization session.

Notes:

- The SafeNet Authentication Client may perform part of the job during calls issued by an application or just collect the information and perform the entire process during **ETC_InitFinal**. This is implementation specific and may change between SafeNet Authentication Client versions. The application cannot assume the exact behavior.
- The valid or invalid values of attributes or parameters may depend on the entire process. For example, passing non-NULL label in the **ETC_InitTokenInit** function is correct except when the token object defines the token as empty following creation. Similarly, the passing of a password-policy object assumes that the initialized token contains a user password.
- Some of parameters depend on the token type. For example, many parameters will not make sense for a software token.
- Only special objects controlling the flow of initialization may be created in the initialization session.

Here are several examples of the initialization flow:

To create an empty token:

```
ETC_InitTokenInit(SO=NULL, Label=NULL)
ETC_InitTokenFinal()
```

To create a one-factor authentication token (no user PIN)

```
ETC_InitTokenInit(SO=NULL)
C_CreateObject(ETCKH_TOKEN_OBJECT with
ETCKA_ONE_FACTOR=TRUE)
ETC_InitTokenFinal()
```

To create a token without SO

```
ETC_InitTokenInit(SO=NULL)
optional C_CreateObject(ETCKH_TOKEN_OBJECT)
optional C_CreateObject(ETCKH_PIN_POLICY)
C_InitPIN() or ETC_InitPIN()
ETC_InitTokenFinal()
```

To create a token with SO

```
ETC_InitTokenInit(SO is not NULL)
optional C_CreateObject(ETCKH_TOKEN_OBJECT)
optional C_CreateObject(ETCKH_PIN_POLICY)
C_InitPIN() or ETC_InitPIN() // may be postponed
ETC_InitTokenFinal()
```

Controlling Initialization Parameters

For all attributes defined here, unless stated otherwise:

- If an attribute is not passed, the corresponding property setting is looked for in the registry.
- If there is no such setting, SafeNet Authentication Client has some internal behavior (described together with an attribute). In rare cases this internal behavior may be token-dependent.

Creation of Token Object

The token object is used to define the major characteristics of the newly initialized token. The following attributes may be set when creating this object during initialization:

- **CKA_CLASS** - Mandatory, must be **CKO_HW_FEATURE**
- **CKA_HW_FEATURE_TYPE** - Mandatory, must be **ETCKH_TOKEN_OBJECT**
- **ETCKA_FORMAT_VERSION** - If not passed, the default value will depend on the token type. For all CardOS based token devices the default value is taken from the property **LEGACY-FORMAT-VERSION**.
- **ETCKA_ONE_FACTOR** - TRUE if the token should be initialized as one-factor. Default: FALSE.
- **ETCKA_FIPS** - TRUE if token should be initialized as FIPS-compliant. Default: FALSE.
- **ETCKA_HMAC_SHA1** - TRUE if **HMAC_SHA**1 support is required. This algorithm is essential for OTP support. It is ignored for SafeNet eToken Virtual Default: TRUE for all eToken NG models.
- **ETCKA_RSA_2048** - TRUE if the token is required to support 2048-bit RSA keys. It is ignored for SafeNet eToken Virtual and for all token models having on-board support for 2048-bit RSA (CardOS 4.20b based tokens). Default: TRUE for all eToken Pro models using CardOS 4.20 and higher.
- **ETCKA_RSA_AREA_SIZE** - size of area (in bytes) to be reserved for RSA keys. CardOS-based tokens use this parameter to reserve the place for RSA keys. The RSA keys may be created only within this area (and it cannot be used to store any other data). Passing 0 prevents creation of RSA keys on the token (effectively leaving more place for data). Default: RTE computes it based on card EEPROM size.

Creation of Password Policy Object

Typically, the process of token initialization includes creation of the password policy object (unless a one-factor or empty token is initialized). If this object is not created during token initialization, there will be no on-token password policy management and SafeNet Authentication Client will use the current password policy of the computer each time (which may cause the token to behave differently on different computers). The following attributes may be set when creating the object:

- **CKA_CLASS** - Mandatory, must be **CKO_HW_FEATURE**
- **CKA_HW_FEATURE_TYPE** - Mandatory, must be **ETCKH_PIN_POLICY**
- **ETCKA_PIN_POLICY_TYPE** - Mandatory, must be **ETCKPT_GENERAL_PIN_POLICY**
- **ETCKA_OWNER** - Default: **CKU_SO** for token with SO, **CKU_USER**
- **CKA_MODIFIABLE** - Default: TRUE if **ETCKA_OWNER** = **CKU_SO**

All other attributes of the password policy object may be passed, except for **ETCKA_PROXY**. Any missing attribute is taken from the corresponding SafeNet Authentication Client. The typical way is to create the password policy object with mandatory attributes only, enforcing the default settings from the SafeNet Authentication Client properties to be taken.

Note:
The application is responsible for the consistency of the attributes, even if some of them are passed explicitly and others are taken from the properties. For example, if the minimal password age is set equal to or higher than the password expiration period the operation will fail.

Token Initialization Keys

Token initialization keys may be set to ensure that the token may be initialized only by the proper entity. This is required only for CardOS-based tokens.

The old key is assumed to be the current one. The new key is assumed to be switched-to after initialization. Absence of one of them means that the default key is used. So, if an application needs to use the initialization key differing from the default (but not to change it) it should create two objects with the same key value.

Note:
If the wrong initialization key is passed, the initialization will fail. By continuous failures the application may lock the key and it will not be possible to re-initialize the token.

Typically the tokens are shipped already initialized with the default key. So, if application does not need to gain more control over the initialization process, there is no need to create these objects.

If created, the key objects should have the following attributes:

- **CKA_CLASS** = **CKO_SECRET_KEY**
- **CKA_KEY_TYPE** = **CKK_DES2**
- **CKA_LABEL** = **OLDKEY** or **NEWKEY**
- **CKA_VALUE** = 16-bytes key value

Note:
PKI Client versions previous to version 4.0 performed MD5 calculations on passed values before actually using it with the token. This was done to serve applications supplying data from human input. PKI Client 4.0, 4.5 and 5.0 take the passed value as the key material without any additional conversion. This option is useful for big deployments where initialization keys may vary for each token and are kept in a database. If your application wants to use the same inputs as in former RTE versions, compute MD5-hash of your data and use the output as the key value. This is exactly what SAPI does to ensure backward-compatibility.

Other Hardware Feature Objects

The initialization process is the only opportunity to create other hardware feature objects (such as **ETCKH_2NDAUTH** and **ETCKH_PRIVATE_CACHING**). If not created explicitly by application, they will be created automatically by the SafeNet Authentication Client.

PIN Initialization

The user PIN may be initialized by **C_InitPIN** function (as described in PKCS#11 standard). There is proprietary function **ETC_InitPIN** that:

- Provides the retry counter for newly created PIN.
- Forces the user to change the PIN upon the first login.

Both functions may be called in the same conditions, either in the SO session or in the special initialization session described previously.

Behavior of Standard C_InitToken and C_InitPIN Functions

The standard way to initialize tokens in PKCS#11 is by issuing a **C_InitToken** call followed by a **C_InitPIN** call. It is very restrictive and provides no flexibility. The functions are supported by SafeNet Authentication Client are as follows:

- **C_InitToken** treats the passed PIN as new SO PIN.
- If the token was initialized without SO and required user password for re-initialization, the operation will fail.
- No password policy object is created (proxy behavior).
- Software token cannot be initialized via **C_InitToken** (since it does not support SO).
- All decisions in regard to customizable parameters are taken automatically by SafeNet Authentication Client and may vary in the future versions. Some of them may be affected by various property settings.
- The token has the **CKF_USER_PIN_INITIALIZED** flag reset, until the **C_InitPIN** (or **ETC_InitPIN**) function is be called.

Backward Compatibility Issues

Tokens Initialized by Earlier PKI Client Versions

If a token formatted in a previous PKI Client version had the **CKF_USER_PIN_INITIALIZED** flag reset, the token will behave differently now. In previous PKI Client versions, PKCS#11 would fail to login with such a token. In SafeNet Authentication Client 8.0 the **CKF_USER_PIN_INITIALIZED** flag is not set for such a token, but password policy enforces password change on the first login.

Using Tokens Initialized by PKI Client 4.0, 4.5 or 5.0 in Earlier Versions

The application should consider whether to initialize the token to be backward compatible. Keeping token backward compatible may have significant performance penalty. Tokens are not expected to work slower than in older PKI Client versions in most real-life scenarios, but some performance gain will be adversely affected.

Using some features (such as one-factor authentication) prevent token from being backward compatible.

Earlier PKI Client versions will ignore password policy settings of the token, as they used a different mechanism.

The way to control during initialization whether the token will be backward compatible is by setting the **ETCKA_FORMAT_VERSION** attribute (explicit or via property) in the token object. If the token is initialized to be backward compatible, **C_InitPIN** or **ETC_InitPIN** must be called during initialization session.

Enforcement of password change for backward-compatible CardOS tokens will be displayed in earlier eToken RTE versions as "PKCS#11 PIN not initialized".

Backward Compatibility of C_InitToken/C_InitPIN

In PKI Client versions earlier than 4.0. there were complicated recommendations of how to re-initialize the token. These recommendations are now irrelevant. Vendors are strongly encouraged to use the new mechanism described in this document.

Special Authentication Features

Single Logon Mode

SafeNet Authentication Client supports the Single Logon mode, where one application actually performs login. Below is the description of this mechanism:

- Until the application performs **`CKU_USER`** login, it cannot access private objects.
- The application may issue proprietary call to the function **`ETC_SingleLogonGetPin`**. If the Single Logon mode is disabled or if the password was not entered in this user session, the function fails.
- If the password is available, the function will return with some printable data string. The application may then use this string in the consequent **`C_Login`** call. The string does not contain the user password. It is just a marker enforcing the SafeNet Authentication Client to use the password entered in another application.
- The SafeNet Authentication Client supports a timeout setting for Single Login mode. This timeout defines how long since the password was really supplied to SafeNet Authentication Client, as it may be used by other applications. After the timeout occurs, the consequent calls to **`ETC_SingleLogonGetPin`** fail as long as the password is not supplied again by an application in the same user session. Reaching timeout does not affect the login state of the applications.
- The Single Logon mechanism fits user password only (not SO).

One-Factor Authentication

The SafeNet Authentication Client allows a token to be initialized without requiring a user password. We usually recommend eliminating the use of this feature, as it decreases the security dramatically. Still, some organizations may prefer using token as single-factor authentication solution.

If token has been initialized as one-factor, the flag **CKF_LOGIN_REQUIRED** in the token information structure will not be set. The **CKA_PRIVATE** attribute will be effectively ignored and all objects will be created as public objects. Such token does not require **C_Login** call. However, there are two cases when the **C_Login** call is successful:

- **C_Login** with NULL will succeed. It is done to simplify applications working with RTE UI.
- **C_Login** with some pre-defined (undocumented) value will succeed. This is done to support a future Password Management application.

Any other call to **C_Login** will fail.

The one-factor mode is supported only for hardware tokens (that is, where there is another factor of authentication except for the user PIN - the token presence).

User PIN Unlocking

If a user forgets the PIN, the administrator (SO) logon is required in order to reset the user PIN to a known value. This may be problem in a distributed environment, where the administrator is physically located at a different site. Administrators will usually not reveal their password to the user, so the standard mechanism of PIN unlocking may not be usable. The SafeNet Authentication Client supports an alternate mechanism based on challenge-response authentication:

- The application issues **ETC_UnlockChallenge** call getting the cryptographic challenge from the token.
- The application passes this challenge to the administrator in any way (network connection, email, phone, Web helpdesk site and so on) and gets back the cryptographic response (based on the challenge and the administrator password). The token itself is not necessary for a computing response.
- The application issues **ETC_UnlockResponse** call, passing both the cryptographic response and the new password.

Notes:

- To check whether the particular token supports challenge-response mechanism for the user PIN unlocking, the application should check presence of the hardware feature object **ETCKH_SO_UNLOCK** This mechanic is available for all token devices initialized with administrator password except for tokens initialized as FIPS.
- No operations with the token should be performed between **ETC_UnlockChallenge** and **ETC_UnlockResponse** calls, otherwise the authentication will fail (while decrementing the SO PIN retry counter).
- The server application should be able to convert the SO PIN into the cryptographic key. Different tokens may use different password-derivation mechanisms (such as those defined in PKCS#5 and PKCS#12 standards). All current token models use the proprietary algorithm. The description of this algorithm may be reached from Aladdin Knowledge Systems and is out of the scope of this document. If the server application uses SafeNet Authentication Client (virtual session), it may use **ETCKM_PBA_LEGACY** mechanism.

Using SafeNet Authentication Client UI

The SafeNet Authentication Client allows passing of NULL as PIN in **C_Login/C_SetPin** functions. This will enforce SafeNet Authentication Client to pop-up the window for login or password change correspondingly.

Miscellaneous Features

CA Certificates

SafeNet Authentication Client 8.0 supports management of CA certificates on token (in addition to the regular user certificates). The CA certificates are distinguished by the attribute. **CKA_CERTIFICATE_CATEGORY** (introduced in the later versions of the PKCS#11 spec). This attribute value should be 2 for CA certificate (for user certificates it is 0).

Use this attribute in templates of object search or creation.

Private Data Caching

Since the tokens are relatively slow devices (in comparison to the desktop operations), the SafeNet Authentication Client uses data caching to improve the performance. The caching of the private data (private objects) is configurable per token by using the special hardware feature object **ETCKH_PRIVATE_CACHING**. By default the private data caching is enabled. RSA private keys do not leave the hardware token (are not cached) regardless of the token settings.

Reference

Common Information

All system-specific definitions (such as structure packing and pointers) are done according to PKCS#11 H-files on the same platform.

Constants

ETCKF_PROPERTY_THREAD

This constant is used solely in flags of **ETC_SetProperty function**. The application should pass this flag if the newly set property value should apply only to the subsequent calls done from the same thread (by default it will apply to the subsequent calls done from any thread of the process).

ETCKO_SHADOW_PRIVATE_KEY

This class is used instead of CKO_PRIVATE_KEY when looking for RSA private key objects without being logged on.

ETCKA_OWNER

This attribute (**CK_USER_TYPE**) is part of some of the feature objects (**ETCKH_PIN_POLICY, ETCKH_PRIVATE_CACHING and ETCKH_2NDAUTH**). It defines who is able to modify the object (assuming that **CKA_MODIFIABLE** is TRUE). It may have either value **CKU_USER** or **CKU_SO**. If **CKA_MODIFIABLE** is FALSE, the corresponding object cannot be changed regardless to **ETCKA_OWNER** value. This attribute may be set only during object creation (that is, during token initialization process since it relates to hardware feature objects).

ETCKA_2NDAUTH_PIN

This attribute may be passed for newly created keys with secondary authentication (assumes **CKA_ALWAYS_AUTHENTICATE** to be TRUE). If not passed, the SafeNet Authentication Client will pop-up the window asking for the password.

ETCKA_OTP_DURATION (CK_ULONG)

This is vendor-specific attribute of the OTP key object, defining the duration (in seconds) of the OTP value representation by the token. It makes sense only for hardware tokens. If not passed during object creation, it will have the default value set by the SafeNet Authentication Client. Depending of **ETCKA_OTP_MAY_SET_DURATION** may or may not be modified later by **C_SetAttributeValue function** (the change will require **CKU_USER** to be logged on).

ETCKA_OTP_MAY_SET_DURATION (CK_BBOOL)

This is vendor-specific attribute of the OTP key object defines whether the value of **ETCKA_OTP_DURATION** may be changed by **C_SetAttributeValue**.

These flags define possible PIN policy issues. They are reported in the **ulDeviceError** field of the structure returned by the **C_GetSessionInfo** function. This function should be called just after functions involving PIN policy (such as **C_Login**, **C_SetPIN** or **ETC_TokenIOCTL** with **ETCK_IOCTL_PIN_EVALUATE**). Some may return only from particular functions. More than one flag may be set.

- **ETCKF_PIN_MIN_LEN** - the newly supplied PIN is too short.
- **ETCKF_PIN_MIX_CHARS** - the newly supplied PIN does not meet mixed-characters criteria.

- **ETCKF_PIN_MAX_AGE** - the current PIN should be changed since it is expired.
- **ETCKF_PIN_MIN_AGE** - the current PIN cannot be changed since the minimally required number of days since the last PIN change didn't pass yet.
- **ETCKF_PIN_WARN_PERIOD** - the current PIN will be expired soon (i.e. it is not expired yet, but within the 'warning' period).
- **ETCKF_PIN_HISTORY** - the newly supplied PIN cannot be accepted since it repeats one of the lastly used PIN values.
- **ETCKF_PIN_MUST_BE_CHANGED** - the current PIN is enforced (by SO) to be changed.

Data Types

```
typedef CK_ULONG ETCK_TRACKER_HANDLE;

typedef ETCK_TRACKER_HANDLE CK_PTR
ETCK_TRACKER_HANDLE_PTR;
```

The **ETCK_TRACKER_HANDLE** represents the tracker data type. The tracker is created by the function **ETC_CreateTracker** and a pointer to it may be passed to **C_WaitForSlotEvent**. The advantage of the tracker is that it may be destroyed (effectively canceling the wait process) by the function **ETC_DestroyTracker** while the regular **C_WaitForSlotEvent** may be cancelled only by issuing **C_Finalize** call.

```
typedef struct tag_ETCK_FUNCTION_LIST_EX
{
  CK_VERSION                    version;  /* Cryptoki extension
version */
    unsigned short                  flags;
    CK_ETC_GetFunctionListEx        ETC_GetFunctionListEx;
    CK_ETC_DeviceIOCTL              ETC_DeviceIOCTL;
    CK_ETC_TokenIOCTL               ETC_TokenIOCTL;
    CK_ETC_CreateTracker            ETC_CreateTracker;
    CK_ETC_DestroyTracker           ETC_DestroyTracker;
    CK_ETC_BeginTransaction         ETC_BeginTransaction;
    CK_ETC_EndTransaction           ETC_EndTransaction;
    CK_ETC_GetProperty              ETC_GetProperty;
    CK_ETC_SetProperty              ETC_SetProperty;
    CK_ETC_CreateVirtualSession     ETC_CreateVirtualSession;
    CK_VOID_PTR                   pReserved;
    CK_ETC_SingleLogonGetPin      ETC_SingleLogonGetPin;
    CK_ETC_InitTokenInit            ETC_InitTokenInit;
    CK_ETC_InitTokenFinal           ETC_InitTokenFinal;
    CK_ETC_InitPIN                  ETC_InitPIN;
    CK_ETC_UnlockGetChallenge       ETC_UnlockGetChallenge;
    CK_ETC_UnlockComplete           ETC_UnlockComplete;
} CK_FUNCTION_LIST_EX ;
typedef ETCK_FUNCTION_LIST_EX CK_PTR
ETCK_FUNCTION_LIST_EX_PTR;
typedef ETCK_FUNCTION_LIST_EX_PTR CK_PTR
ETCK_FUNCTION_LIST_EX_PTR_PTR;
```

This structure is used to reach addresses of the vendor-specific PKCS#11 Extension functions. It is returned by the function **`ETC_GetFunctionListEx`** function. The structure contains the following fields:

- Version - Version of the structure. Future versions of the structure may contain more functions. Currently returned version is 1.0.
- Flags - currently 0, reserved for future use.
- All other fields are pointers to the corresponding functions.

Objects

The SafeNet Authentication Client introduces several hardware feature objects covered in this chapter.

Token Feature Object (ETCKH_TOKEN_OBJECT)

The **`ETCKH_TOKEN_OBJECT`** is used to obtain detailed information about token capabilities (that is not covered in standard information returned by the **`C_GetTokenInfo`** function) and for defining the settings of the newly initialized token.

Attribute	Type	Meaning
CKA_CLASS	CK_OBJECT_CLASS	CKO_HW_FEATURE
CKA_HW_FEATURE_TYPE	CK_HW_FEATURE_TYPE	ETCKH_TOKEN_OBJECT
CKA_LABEL	CK_UTF8CHAR_PTR	Token label
ETCKA_PRODUCT_NAME	CK_UTF8CHAR_PTR	Token product name.
ETCKA_MODEL	CK_UTF8CHAR_PTR	Token model
ETCKA_PRODUCTION_DATE	CK_DATE	Token production date
ETCKA_CASE_MODEL	CK_ULONG	Token case model
ETCKA_CARD_TYPE	CK_ULONG	Token card type
ETCKA_CARD_VERSION	CK_ULONG	CK_VERSION
ETCKA_RETRY_USER	CK_ULONG	Current retry counter for user PIN
ETCKA_RETRY_SO	CK_ULONG	Current retry counter for SO PIN
ETCKA_RETRY_USER_MAX	CK_ULONG	Maximal retry counter for user PIN
ETCKA_RETRY_SO_MAX	CK_ULONG	Maximal retry counter for SO PIN
ETCKA_HAS_LCD	CK_BBOOL	TRUE if token has LCD (i.e. may be used for off-line OTP computation)
ETCKA_HAS_SO	CK_BBOOL	TRUE if SO may be logged in for this token (i.e. it has been initialized with SO PIN)

Attribute	Type	Meaning
ETCKA_FIPS	CK_BBOOL	TRUE if token is initialized to be FIPS-compliant
ETCKA_FIPS_SUPPORTED	CK_BBOOL	TRUE if token can be initialized as FIPS-compliant
ETCKA_INIT_PIN_REQ	CK_BBOOL	TRUE if authentication (either user or SO) is needed to re initialize the token.
ETCKA_RSA_2048	CK_BBOOL	TRUE if token supports 2048-bit RSA keys
ETCKA_RSA_2048_SUPPORTED	CK_BBOOL	TRUE if token may be initialized with 2048-bit RSA keys support
ETCKA_HMAC_SHA1	CK_BBOOL	TRUE if token supports HMAC-SHA1
ETCKA_HMAC_SHA1_SUPPORTED	CK_BBOOL	TRUE if token may be initialized with HMAC-SHA1 support
ETCKA_MAY_INIT	CK_BBOOL	TRUE if the token may be re initialized with SafeNet Authentication Client
ETCKA_MASS_STORAGE_PRESENT	CK_BBOOL	TRUE if the token has built-in mass-storage device
ETCKA_ONE_FACTOR	CK_BBOOL	TRUE if the token is initialized as one-factor authentication device
ETCKA_RSA_AREA_SIZE	CK_ULONG	Amount of bytes reserved to store RSA keys
ETCKA_FORMAT_VERSION	CK_ULONG	Defines the token format version
ETCKA_USER_PIN_AGE	CK_ULONG	User PIN age (in days), i.e. how long ago the user PIN has been changed.

The attribute **CKA_LABEL** contains the token label (the same one as is returned by the **C_GetTokenInfo** function) in the form of null-terminated **CK_UTF8CHAR** string. This is the only attribute that can be changed by using **C_SetAttributeValue** function. It requires **CKU_USER** to be logged on.

The attribute **ETCKA_PRODUCT_NAME** represents the token product name (such as "eToken NG-OTP).

The attribute **ETCKA_MODEL** is a printable string representing the exact token model (hardware/firmware version etc.). It is used solely for support purposes. The following constants represent supported values of the **ETCKA_CASE_MODEL** attribute of the **ETCKH_TOKEN** object.

- **ETCK_CASE_NONE** - the case model is not relevant. This value may be returned for the SafeNet eToken Virtual or for the etoken smartcard.
- **ETCK_CASE_CLASSIC** - the classic case model of eToken Pro.
- **ETCK_CASE_NG1, ETCK_CASE_NG2** - eToken NG-OTP
- **ETCK_CASE_NG2_NOLCD** - eToken NG-FLASH

The attribute **ETCKA_PRODUCTION_DATE** contains the date of token production. It is available only for particular token models. Whenever it is not available, zero bytes will be returned for this attribute.

The attribute **ETCKA_CASE_MODEL** may be used to associate the token with particular icon in the application (different tokens may have different views).

The attribute **ETCKA_CARD_TYPE** encodes the smartcard type used in the particular token model (listed in the Constants section). The following constants represent supported values of the **ETCKA_CARD_TYPE:**

- **ETCK_CARD_NONE** - the token has no smartcard (it may be returned, for instance, for SafeNet eToken Virtual).
- **ETCK_CARD_OS** - the token contains Siemens CardOS smartcard.

The attribute **ETCKA_RSA_SIZE** represents the amount of memory reserved during token initialization for RSA keys. If this is irrelevant for certain tokens (that is, the storage should not be specially reserved for RSA keys), the returned value may be **CK_EFFECTIVELY_INFINITE** or **CK_UNAVAILABLE_INFORMATION**.

The attribute **ETCKA_FORMAT_VERSION** defines the version of the token format in use. The valid values may vary between token models. The format version defines backward compatibility and ability to support such or other features.

The following constants represent supported values of the **ETCKA_FORMAT_VERSION** attribute for tokens using CardOS smartcard (when **ETCKA_CARD_TYPE** is equal to **ETCK_CARD_OS**).

- **ETCK_FORMAT_VERSION_LEGACY** - the format version compatible with prior versions of eToken PKI Client.

- **ETCK_FORMAT_VERSION_4_0** - the format version supported since eToken PKI Client 4.0.
- **ETCK_FORMAT_VERSION_5_0** - for Java Cards. Introduced in PKI CLient 4.5.

The token object is create during token initialization process and cannot be changed afterwards (except for the **CKA_LABEL** attribute). Only a subset of attributes may be passed during token initialization.

PIN Policy Feature Object

Attribute	Type	Meaning
CKA_CLASS	CK_OBJECT_CLASS	CKO_HW_FEATURE
CKA_HW_FEATURE_TYPE	CK_HW_FEATURE_TYPE	ETCKH_PIN_POLICY
CKA_MODIFIABLE	CK_BBOOL	TRUE if the object can be changed
ETCKA_OWNER	CK_USER_TYPE	The required logon type for changing the object
ETCKA_PIN_POLICY_TYPE	CK_ULONG	PIN Policy type
ETCKA_PIN_MIN_LEN	CK_ULONG	Minimal PIN length
ETCKA_PIN_MIN_AGE	CK_ULONG	Minimal amount of days after that should pass since PIN change until it will be allowed to change it again by using C_SetPIN function
ETCKA_PIN_MAX_AGE	CK_ULONG	PIN expiration period (in days)
ETCKA_PIN_WARN_PERIOD	CK_ULONG	PIN change warning period (in days)
ETCKA_PIN_HISTORY_SIZE	CK_ULONG	PIN history size
ETCKA_PIN_PROXY	CK_BBOOL	TRUE if no real PIN policy information is stored on the token
ETCKA_RETRY_USER_MAX	CK_ULONG	Maximal retry counter for user PIN
ETCKA_PIN_MAX_REPEATED	CK_ULONG	Maximum number of characters that can be repeated in sequence.
ETCKA_PIN_NUMBERS	CK_ULONG	Determines if numbers are permitted, mandatory or forbidden in the PIN

Attribute	Type	Meaning
ETCKA_PIN_UPPER_CASE	CK_ULONG	Determines if uppercase letters are permitted, mandatory or forbidden in the PIN
ETCKA_PIN_LOWER_CASE	CK_ULONG	Determines if lowercase letters are permitted, mandatory or forbidden in the PIN
ETCKA_PIN_SPECIAL	CK_ULONG	Determines if special characters are permitted, mandatory or forbidden in the PIN

The **ETCKH_PIN_POLICY** object is used to manage the user PIN policy of the token.

The attribute **ETCKA_PIN_POLICY_TYPE** defines the supported attributes and behavior of the PIN policy. It will allow different PIN policy schemes to be supported in the future. The only currently supported value is **ETCKPT_GENERAL_PIN_POLICY.** This attribute cannot be changed after object creation.

The attribute **ETCKA_PIN_MIX_CHARS** (if set to TRUE) means that from the following categories of characters (English uppercase letters, English lowercase letters, digits, and all the rest) at least 3 should be presented in the PIN.

The attribute **ETCKA_PIN_WARN_PERIOD** is a recommendation for the applications: how many days before real password expiration the user should be warned.

The attribute **ETCKA_PIN_HISTORY_SIZE** defines how many old PIN values should be prevented from using for the new PIN. The token may have an upper restriction for this attribute since it is storage-consuming.

The attribute **ETCKA_PIN_PROXY** is set to TRUE if during token initialization the PIN policy object has not been created (that is no PIN policy settings are stored on token). In this case the token will behave according to the per-machine settings of the SafeNet Authentication Client. The PIN policy hardware feature object is still returned to the application to simplify the application programming. This attribute cannot be changed after token initialization.

The attributes ETCKA_PIN_NUMBERS, ETCKA_PIN_UPPER_CASE, ETCKA_PIN_LOWER_CASE and ETCKA_PIN_SPECIAL enable additional options when setting the manual complexity requirements in SafeNet Authentication Client. These attributes support the following values:

ETCK_PIN_DONTCARE (0-default) - allows usage of the characters determined by the attribute

ETCK_PIN_FORBIDDEN (1) - prohibits usage of the characters determined by the attribute

ETCK_PIN_ENFORCE (2) - forces usage of the characters determined by the attribute

Challenge-Response Unlocking Capability Feature Object

The **ETCKH_SO_UNLOCK** object represents the challenge-response unlocking capability of the token (see **ETC_UnlockGetChallenge/ETC_UnlockComplete functions**). If such an object is not found, the token's PIN cannot be unlocked by using challenge-response even in presence of SO PIN. It is not enough to know the SO PIN on the server side to perform the cryptographic response computation. The application should also how to convert the SO PIN into the cryptographic key. It may be PKCS#5 or PKCS#12 or any other password conversion scheme. **ETCKH_SO_UNLOCK** describes the mechanism used by particular token and its parameters.

Attribute	Type	Meaning
CKA_CLASS	CK_OBJECT_CLASS	CKO_HW_FEATURE
CKA_HW_FEATURE_TYPE	CK_HW_FEATURE_TYPE	ETCKH_SO_UNLOCK
ETCKA_PBA_MECHANISM	CK_MECHANISM_TYPE	Mechanism of password derivation
ETCKA_PBA_ITERATION	CK_ULONG	Iteration counter
ETCKA_PBA_SALT	Byte array	Salt value

The attribute **ETCKA_PBA_MECHANISM** represents the cryptographic mechanism used for derivation of the cryptographic key from the password. All tokens supporting challenge-response are currently using the proprietary mechanism **ETCKM_PBA_LEGACY**. No additional parameters are needed for this mechanism.

The attributes **ETCKA_PBA_ITERATION** and **ETCKA_PBA_SALT** are reserved for future use. For PKCS#5 and PKCS#12 based password derivation they would represent the parameters of mechanism to be used. For **ETCKM_PBA_LEGACY** they have no sense.

Private Data Caching Feature Object

The **ETCKH_PRIVATE_CACHING** object defines the rules applied to the caching of the private data coming from the token by the SafeNet Authentication Client.

Attribute	Type	Meaning
CKA_CLASS	CK_OBJECT_CLASS	CKO_HW_FEATURE
CKA_HW_FEATURE_TYPE	CK_HW_FEATURE_TYPE	ETCKH_PRIVATE_CACHING
CKA_MODIFIABLE	CK_BBOOL	TRUE if the object can be changed
ETCKA_OWNER	CK_USER_TYPE	The required logon type for changing the object
ETCKA_CACHE_PRIVATE	CK_ULONG	Caching policy

The **ETCKA_CACHE_PRIVATE** may have the following values:

- **ETCK_CACHE_OFF** - the caching of private data is disabled.

- **ETCK_CACHE_LOGIN** - the caching of private data is enabled as long as the user is logged on.
- **ETCK_CACHE_ON** - the private data caching is enabled. The data will be kept in the cache (but not available to application) even when user is not logged on. This is the default behavior.

Note:
Regardless of the private data caching mode for hardware tokens, the RSA private keys never leave the token.

Secondary Authentication Policy Feature Object

The **ETCKH_2AUTH** object represents the secondary authentication settings of the token.

Attribute	Type	Meaning
CKA_CLASS	CK_OBJECT_CLASS	CKO_HW_FEATURE
CKA_HW_FEATURE_TYPE	CK_HW_FEATURE_TYPE	ETCKH_2AUTH
CKA_MODIFIABLE	CK_BBOOL	TRUE if the object can be changed
ETCKA_OWNER	CK_USER_TYPE	The required logon type for changing the object
ETCKA_2NDAUTH_CREATE	CK_ULONG	Secondary authentication policy

The attribute **ETCKA _2NDAUTH_CREATE** defines how RSA private keys are created. It may have one of the following values:

- **ETCK _2NDAUTH_PROMPT_NEVER** (default): the newly created RSA private keys will be regular private objects (that is, they will not be protected with an additional password). The **CKA_ALWAYS_AUTHENTICATE** attribute will be ignored.

- **ETCK _2NDAUTH_PROMPT_CONDITIONAL -** the newly create RSA private keys is protected with secondary authentication only if the **CKA_ALWAYS_AUTHENTICATE** attribute has been set to TRUE during the object creation. If the user didn't supply the password, the window will appear. Cancelling the operation creates the RSA private key with **CKA_ALWAYS_AUTHENTICATE** equal to FALSE.
- **ETCK _2NDAUTH_PROMPT_ALWAYS** behaves similarly to **ETCK _2NDAUTH_PROMPT_CONDITIONAL** except that the UI will appear regardless of the initial value of the **CKA_ALWAYS_AUTHENTICATE** attribute.
- **ETCK _2NDAUTH_PROMPT** mandatory behaves similarly to **ETCK _2NDAUTH_PROMPT_ALWAYS,** except that the creation of keys not protected with the secondary authentication is prohibited. Cancelling of operation by the user will finish the operation with the failure.

Functions

ETC_GetFunctionListEx

```
CK_DECLARE_FUNCTION(CK_RV, ETC_GetFunctionListEx)
(
  ETCK_FUNCTION_LIST_EX_PTR_PTR ppFunctionListEx /* receives
pointer to extention functions list */
);
```

ETC_GetFunctionListEx obtains a pointer to the data structure containing pointers to all PKCS#11 Extensions functions. **ppFunctionListEx** points to a value which will receive a pointer to the library's **ETCK_FUNCTION_LIST** structure, which in turn contains function pointers for all the PKCS#11 Extensions routines in the library. The pointer thus obtains may points into memory which is owned by the SafeNet Authentication Client, and which may or may not be writable. No attempt should be made to write to this memory.

ETC_DeviceIOCTL

```
CK_DECLARE_FUNCTION(CK_RV, ETC_DeviceIOCTL)
(
  CK_SLOT_ID slotId,
  CK_ULONG code,
  CK_VOID_PTR pInput,
  CK_ULONG ulInputLength,
  CK_VOID_PTR pOutput,
  CK_ULONG_PTR pulOutputLength
);
```

ETC_DeviceIOCTL is used to perform various slot-level operations not covered by the PKCS#11 spec. The main difference between **ETC_DeviceIOCTL** and **ETC_TokenIOCTL** is that **ETC_DeviceIOCTL** is directed to the entire slot and in most of cases does not even assume a ready-to-work token to be plugged.

ETC_DeviceIOCTL directs the command with function code to the slotID. The meaning of the input parameter **pInput** of **ulInputLength** and the output buffer **pOutput** of **pulOutputLength** depends on particular operation to be performed.

ETCK_IODEV_SOFTWARE_TOKEN_PLUGIN

ETCK_IODEV_SOFTWARE_TOKEN_PLUGIN connects the SafeNet eToken Virtual file to the free token slot. The SafeNet eToken Virtual may be connected only to the soft token slot which has no token connected to it. The **pInput** should point to the SafeNet eToken Virtual file name (UTF-8 encoded).

Note the following:

- The corresponding file should exist and keep a valid SafeNet eToken Virtual. SafeNet Authentication Client does not check the file presence and correctness, but future versions of SafeNet Authentication Client may do this. Also, various applications (such as SafeNet Authentication Client) may automatically unplug the SafeNet eToken Virtual if the corresponding file is unavailable or corrupted.
- The file should be achievable (for the same reasons). It is therefore not recommended to plug-in the SafeNet eToken Virtual located on the removable storage (or network) device.
- The file should be available for read/write access; otherwise the subsequent operations with the SafeNet eToken Virtual may fail.

- All applications (including one who issued the **ETC_DeviceIOCTL**) will get notification about the slot event (via **C_WaitForSlotEvent**).
- The token will remained plugged in as long as it will not be plugged out by some application.

ETCK_IODEV_SOFTWARE_TOKEN_PLUGOUT

ETCK_IODEV_SOFTWARE_TOKEN_PLUGOUT disconnects the SafeNet eToken Virtual from the slot. No input/output parameters are defined for this function. The slotID is the only necessary information. Upon plug out, all applications (including the one that issued the **ETC_DeviceIOCTL**) will get notification about the slot event (via **C_WaitForSlotEvent**). The function does not remove the file physically, it just disconnects it from the slot.

ETCK_IODEV_FULL_NAME

ETCK_IODEV_FULL_NAME returns the full name for particular slot. It is used to overcome PKCS#11 restriction of 64 bytes for slot description. The pOutput is buffer where the name will be returned. The function will return the following:

- For smartcard reader - full reader name.
- For software slot with connected SafeNet eToken Virtual - the filename of the SafeNet eToken Virtual.

ETCK_IODEV_SOFTWARE_GET_EMULATE

ETCK_IODEV_SOFTWARE_GET_EMULATE checks whether the software slot is in the smartcard reader emulation mode. Only one slot (with SafeNet eToken virtual connected) may be in such mode in any period of time. The **pOutput** is considered as **CK_BBOOL_PTR**. TRUE or FALSE will be returned, depending on the slot state.

The following constants represent the currently supported slot IOCTL codes. See description of **ETC_DeviceIOCTL** function for details of use.

ETCK_IODEV_SOFTWARE_SET_EMULATE

ETCK_IODEV_SOFTWARE_SET_EMULATE is used to turn the smartcard reader emulation mode on or off for the particular slot. The **pInput** is considered as **CK_BBOOL_PTR**, pointing to the input parameter. If TRUE is passed, the emulation will be turned on. If FALSE is passed, the emulation will be turned off.

ETCK_IODEV_CHECK_NAME

ETCK_IODEV_CHECK_NAME is used to check whether the particular name is associated with the passed slotID. The **pInput** is considered as **CK_UTF8CHAR_PTR** pointing to the name, and **pOutput** is **CK_BBOOL_PTR** pointing to the result to be returned. Usually, the function will return TRUE if the passed name is the same one as returned by **ETCK_IODEV_FULL_NAME**. The only exception is that software slot in emulation mode will also return TRUE if the name of associated smartcard reader is passed.

ETC_TokenIOCTL

```
CK_DECLARE_FUNCTION(CK_RV, ETC_TokenIOCTL)
(
  CK_SESSION_HANDLE hSession,
  CK_OBJECT_HANDLE hObject,
  CK_ULONG code,
  CK_VOID_PTR pInput,
  CK_ULONG ulInputLength,
  CK_VOID_PTR pOutput,
  CK_ULONG_PTR pulOutputLength
);
```

ETC_TokenIOCTL is used to perform various operations that are not covered by the PKCS#11 spec. The **hSession** and **hObject** define the session and object in use and the code defines particular operation to be performed. The meaning of the input parameter **pInput** of **ulInputLength** and the output buffer **pOutput** of **pulOutputLength** depends on particular operation to be performed.

ETCK_IOCTL_PIN_EVALUATE

ETCK_IOCTL_PIN_EVALUATE is used to check whether the PIN meets the PIN policy requirements, without actually changing the PIN.

The parameters are:

- **hSession** - session opened with the requested token.
- **hObject** - is ignored by the SafeNet Authentication Client 8.0. It is recommended however to pass the handle to the PIN policy hardware feature token (for forward compatibility).
- **code** - **ETCK_IOCTL_PIN_EVALUATE**.
- **pInput, ulInputLength** - the new PIN value to be evaluated. If NULL is passed, the function will check only if the PIN change is already allowed (according to the minimal PIN age setting).
- **pOutput, pulOutputLength** - the output parameter is of type **CK_ULONG**. It returns some estimation about the passed PIN (according to PIN policy) as the number between 0 and 100. The application may use this number for user interface purposes. No value of this parameter should be understood as acceptance or rejection of the passed PIN value. If **pInput** is NULL, **pOutput** should be NULL too.

The function returns **CKR_OK** if the PIN is acceptable, or alternatively, **CKR_INVALID_PIN or CKR_PIN_LEN_RANGE** (any other code is considered as an error of the function). More information may be received by subsequent call to the **C_GetSessionInfo** function.

Note the following:

- The function may not check whether the PIN is equivalent to one of the old PIN values. So, even if the function succeeded, the subsequent **C_SetPIN** may fail due to the PIN history.
- If the application intends to use **ETC_UnlockComplete** rather than **C_SetPIN**, the operation ignores the minimal password age. In this case, even in the case of **CKR_INVALID_PIN** being returned, the application should check the reason for failure. If minimal PIN age is the only failure, the **ETC_UnlockComplete** will still succeed.

ETC_CreateTracker

```
CK_DECLARE_FUNCTION(CK_RV, ETC_CreateTracker)
(
  ETCK_TRACKER_HANDLE_PTR pTracker,
  CK_VOID_PTR param
);
```

ETC_CreateTracker creates the tracker object that may later be used in the **C_WaitForSlotEvent** function. **pTracker** is the pointer for the tracker handle to be returned, **param** is reserved for future use and must be NULL. The reason to use trackers is that the tracker may be destroyed later (effectively breaking the wait and unlocking the waiting thread) without stopping the entire application activity with **C_Finalize**.

ETC_DestroyTracker

```
CK_DECLARE_FUNCTION(CK_RV, ETC_DestroyTracker)
(
  ETCK_TRACKER_HANDLE hTracker
);
```

ETC_DestroyTracker destroys **hTracker** previously created by the **ETC_CreateTracker** function.

ETC_BeginTransaction

```
CK_DECLARE_FUNCTION(CK_RV, ETC_BeginTransaction)
(
  CK_SESSION_HANDLE hSession
);
```

ETC_BeginTransaction locks the token for using from other processes or threads until **ETC_EndTransaction** is performed. Locking the token ensures that no other application (or thread) will work with the token at that time. Using transactions for long series of operations with the token allows the application to ensure consistency and improve performance.

Note:

- Locking the token means only preventing other applications from using it. There is no database-like transaction (If the application crashes before finishing a transaction, there is no rollback).
- If the application finishes without ending the transaction, it will be ended automatically. However, if the thread finishes without releasing the transaction, the behavior is unpredictable: the transaction may be ended immediately or only after finishing the application. So, if another thread of the same application tries to access the token, the deadlock is possible.
- The transaction with some token may block even 'innocent' operations working with multiple slots (such as **C_GetSlotList**).
- Do not keep open a transaction longer than necessary. In particular, do not perform any user interaction during opened transaction.

ETC_EndTransaction

```
CK_DECLARE_FUNCTION(CK_RV, ETC_EndTransaction)
(
  CK_SESSION_HANDLE hSession
);
```

ETC_EndTransaction ends the transaction opened by the **ETC_BeginTransaction** call.

ETC_GetProperty

```
CK_DECLARE_FUNCTION(CK_RV, ETC_GetProperty)
(
  CK_UTF8CHAR_PTR name,
  CK_VOID_PTR pBuffer,
  CK_ULONG_PTR pulSize,
  CK_VOID_PTR pReserved /* NULL */
);
```

ETC_GetProperty returns the current setting of particular property. **pReserved** is reserved for future use and must be NULL.

ETC_SetProperty

```
CK_DECLARE_FUNCTION(CK_RV, ETC_SetProperty)
(
  CK_UTF8CHAR_PTR name,
  CK_VOID_PTR pBuffer,
  CK_ULONG ulSize,
  CK_ULONG flags,
  CK_VOID_PTR pReserved /* NULL */
);
```

ETC_SetProperty modifies the property setting. Only some properties may be set in the by application. The change affects only current process. The only currently defined flag is **ETCKF_PROPERTY_THREAD**. When passed, the change will affect only the particular thread; otherwise it will affect the entire process. Passing NULL value will effectively reset the property value to its initial value (that taken place on application launch).

pReserved is reserved for future use and must be NULL.

ETC_CreateVirtualSession

```
CK_DECLARE_FUNCTION(CK_RV, ETC_CreateVirtualSession)
(
  CK_SESSION_HANDLE_PTR phSession
);
```

ETC_CreateVirtualSession creates the virtual session, that is, the session that is not associated with any token. It is useful for server-side operations, such as:

- Cryptographic operations based on any keys known to the application.
- OTP verification.
- Computation response for the user token unlocking.

The application may call **ETC_CreateVirtualSession** many times, all created sessions are independent.

ETC_SingleLogonGetPin

```
CK_DECLARE_FUNCTION(CK_RV, ETC_SingleLogonGetPin)
(
  CK_SESSION_HANDLE hSession,
  CK_CHAR_PTR       pPin,
  CK_ULONG_PTR      ulPinLen
);
```

ETC_SingleLogonGetPin returns a pseudo-PIN value that may be later used in a **C_Login** call. If the SafeNet Authentication Client is currently not in the Single Login Mode the function will fail. If the user PIN was not entered yet, the function will fail.

The data returned by the **ETC_SingleLogonGetPin** is not related in any way to the user PIN value. It is just some data that will be later understood by **C_Login** as an attempt to reuse PIN value (if already known to the SafeNet Authentication Client in the current user session).

ETC_InitTokenInit

```
CK_DECLARE_FUNCTION(CK_RV, ETC_InitTokenInit)
(
  CK_SLOT_ID             slotID,
  CK_UTF8CHAR_PTR        pPin,
  CK_ULONG               ulPinLen,
  CK_ULONG               ulRetryCounter,
  CK_UTF8CHAR_PTR        pLabel,
  CK_SESSION_HANDLE_PTR  phSession
);
```

ETC_InitTokenInit opens the initialization session with the token located in the **slotId** and returns in **phSession** the handle to this session. The initialization session will later be used for creation of hardware feature objects. It is closed by **ETC_InitTokenFinal** or by **C_CloseSession**. The parameters **pPin** and **ulPinLen** represent the SO PIN. NULL should be passed if the token will have no SO PIN. **ulRetryCounter** is the retry counter for the SO PIN. If zero is passed, the SafeNet Authentication Client will use the default value.

ETC_InitTokenFinal

```
CK_DECLARE_FUNCTION(CK_RV, ETC_InitTokenFinal)
(
  CK_SESSION_HANDLE hSession
);
```

ETC_InitTokenFinal performs the actual token initialization and closes the initialization session opened by **ETC_InitTokenInit**.

ETC_InitPIN

```
CK_DECLARE_FUNCTION(CK_RV, ETC_InitPIN)
(
  CK_SESSION_HANDLE hSession,
  CK_UTF8CHAR_PTR   pPin,
  CK_ULONG          ulPinLen,
  CK_ULONG          ulRetryCounter,
  CK_BBOOL          toBeChanged
);
```

ETC_InitPIN initializes the user PIN. It extends the functionality of the standard function **C_InitPIN**. The function should be called when SO is logged in or during the initialization session opened by **ETC_InitTokenInit**.

If the function is called during the initialization session, **ulRetryCounter** defines the retry counter for the user password. If zero is passed, the SafeNet Authentication Client will use the default value. If the function is called after token initialization, it is recommended to pass zero (so that the retry counter will not be changed), since not all token models may be able to change retry counter without token re-initialization.

The parameter **toBeChanged** allows to enforce the user to change the PIN upon the first login.

ETC_UnlockGetChallenge

```
CK_DECLARE_FUNCTION(CK_RV, ETC_UnlockGetChallenge)
(
  CK_SESSION_HANDLE hSession,
  CK_VOID_PTR       pChallenge,
  CK_ULONG_PTR      pulChallengeLen
);
```

ETC_UnlockChallenge returns in **pChallenge** buffer the cryptographic challenge. The application should compute response and call with it the function **ETC_UnlockComplete** in order to unlock the user PIN.

ETC_UnlockComplete;

```
CK_DECLARE_FUNCTION(CK_RV, ETC_UnlockComplete)
(
  CK_SESSION_HANDLE hSession,
  CK_VOID_PTR       pResponse,
  CK_ULONG          ulResponse,
  CK_UTF8CHAR_PTR   pPin,
  CK_ULONG          ulPinLen,
  CK_ULONG          ulRetryCounter,
  CK_BBOOL          toBeChanged
);
```

ETC_UnlockComplete completes the process of user PIN unlocking. **pResponse** should be the cryptographic response computed by application for the challenge returned from the function **ETC_UnlockChallenge**. The newly passed user PIN (pPin) should meet the PIN Policy settings of the token. It is recommended that the **ulRetryCounter** will be the same one that the password had before or zero (in case of zero the SafeNet Authentication Client keeps the old counter). Other values may be or may not be supported by the particular token. The parameter **toBeChanged** may be used to force the user to change the PIN upon the first login.

Mechanisms

ETCKM_PBA_LEGACY

This is vendor-specific mechanism of key derivation from the PIN. It is used to convert the SO PIN to the Triple-DES MAC key. The PIN value serves as the only mechanism parameter. In other words, the **CK_MECHANISM** structure for key generation from the SO PIN should be filled as follows for this mechanism:

- mechanism - **ETCKM_PBA_LEGACY**.

- **pParameter** - pointer to the SO PIN
- **ulParameterLen** - length of the SO PIN

Properties

The SafeNet Authentication Client SDK provides Set and Get property functions. The properties are defined by the SafeNet Authentication Client module.

In versions earlier than eToken PKI Client 5.0, some properties did not allow the Set function. In SafeNet Authentication Client 8.0, properties are arranged according to a hierarchy, and the Set function is available according to the location of the property in this hierarchy:

- Policy (Machine) - requires Administrator permissions
- Policy (User) - requires Administrator permissions
- User- requires User permissions
- Machine - requires User permissions
- Current User - requires User permissions

The Set function is enabled for properties that are located in the areas accessible with User permissions.

For more information, see the *Configuration Settings* chapter in the *SafeNet Authentication Client Administrator's Guide*.

General

Property name	Explanation
TolerantX509Attributes	TolerantX509Attributes determines if the following conditions must be met when creating a certificate: ■ The certificate must be created in the DER encoded X.509 format. ■ The serial number, subject and issuer objects of the internal certificate must match the external attributes. For more information see *X.509 Attribute Tolerance* on page 21

Password Policy

Property name	Explanation
pqMinLen	Default value for ETCKA_PIN_MIN_LEN attribute (see Password Policies)
pqMixChars	Default value for ETCKA_PIN_MIX_CHARS attribute (see Password Policies)
pqMaxAge	Default value for ETCKA_PIN_MAX_AGE attribute (see Password Policies)
pqMinAge	Default value for ETCKA_PIN_MIN_AGE attribute (see Password Policies)
pqWarnPeriod	Default value for ETCKA_PIN_WARN_PERIOD attribute (see Password Policies)
pqHistorySize	Default value for ETCKA_PIN_HISTORY_SIZE attribute (see Password Policies)
pqNumbers	Default value for ETCKA_PIN_NUMBERS attribute
pqLowerCase	Default value for ETCKA_PIN_LOWER_CASE attribute
pqUpperCase	Default value for ETCKA_PIN_UPPER_CASE attribute
pqSpecial	Default value for ETCKA_PIN_SPECIAL attribute
pqMaxRepeated	Default value for ETCKA_PIN_MAX_REPEATED attribute

Initialization

Property name	Explanation
HMAC-SHA1	Default value for ETCKA_HMAC_SHA1 attribute (see token initialization). If not set, there is no default value, the logic of SafeNet Authentication Client may vary depending on the token model.
RSA-2048	Default value for ETCKA_RSA_2048 attribute (see token initialization). If not set, there is no default value, the logic of SafeNet Authentication Client may vary depending on the token model.
LEGACY-FORMAT-VERSION	Default value for ETCKA_FORMAT_VERSION attribute for CardOS-based eToken models (see token initialization).
RSA-AREA-SIZE	Default value for ETCKA_RSA_AREA_SIZE attribute (see token initialization). If not set, there is no default value, the logic of SafeNet Authentication Client may vary depending on the token model.

Token Model Specifications

	FIPS	HMAC*	2048	One Factor	RSA Area Size
Java Card	No	Built in	Controlled by FW CFG block	Yes	Not relevant
SafeNet eToken Virtual	No	Yes	Yes	No	Set the area size attribute to ZERO to disable RSA use.
eToken Pro/ eToken-NG Card OS 4.20.B	Yes, with format version 5	Yes, through external package	Yes	Yes, with format version 4 and 5 (non FIPS)	Applies to formats 0 and 4. Set the area size attribute to ZERO to disable RSA use.
eToken Pro/ eToken-NG Card OS 4.20	No	Yes, through external package	Yes, through external package	Yes, with format version 4 and 5	Applies to formats 0 and 4. Set the area size attribute to ZERO to disable RSA use.
eToken Pro/ eToken-NG Card OS 4.01	Yes, with format version 0 (compatible with RTE 3.65)	No	No	Yes, with format version 4 (non FIPS)	Applies to formats 0 and 4. Set the area size attribute to ZERO to disable RSA use.

* OTP depends on the HMAC algorithm.

Chapter 4

SAPI

This chapter describes SAPI as implemented in SafeNet Authentication Client 8.0, differences in behavior and backward compatibility issues.

- Introduction
- Common Description of SAPI
- Data Types
- Error Codes
- SAPI Objects
- Functions
- Major Backward Compatibility Issues of SAPI

Introduction

SAPI (Supplementary API) was introduced in SDK 3.60 to provide developers with access to token functionality not covered at that stage by PKCS#11 API, such as:

- Obtaining extended information about the version and capabilities of a particular token device.
- Initializing of token in a more flexible manner than provided by the **`C_InitToken`** PKCS#11 function.
- Secure unblocking of a user PIN.
- Managing the OTP (one-time password) functionality of token devices supporting OTP.

In SafeNet Authentication Client 8.0, all this functionality is available through PKCS#11 API (including vendor-specific extensions described in this document). However SAPI continues to be supported for reasons of backward compatibility. However, new functionality will be provided only through PKCS#11. Therefore, we recommend using PKCS#11 for future development, while using SAPI for maintenance purposes.

Common Description of SAPI

When working with SAPI functions:

- The application may call SAPI functions in one of two ways:
  - Statically linking `libeTSapi.so` by using the import library supplied with the SDK.
  - Dynamically loading `libeTSapi.so` by using the **`LoadLibrary`** function and then locating API functions by using **`GetProcAddress`**.
- The application should call the PKCS#11 **`C_Initialize`** function prior to any usage of SAPI functions.
- The application should not call the **`C_Finalize`** function as long as it continues to use SAPI functions.

This section contains a brief description of the proposed SAPI functions.

For each function, the following applies:

- Function returns **CK_RV** as the return value as in all PKCS#11 functions. Several vendor specific return codes are defined by SAPI.

  Note:
  Error codes returned by the same function may vary between versions of PKI Client.

- If the function operates with a particular slot, it gets either a slot ID or session handle as a parameter. A session handle is used if it is expected that the token must be properly initialized before the function call.
- Whenever possible data types and structures defined in PKCS#11 are used. SAPI defines several more data types.
- Parameter template means **CK_ATTRIBUTE_PTR** which points to the attributes array and **CK_ULONG** containing the array size. The handling of this parameter is similar to other PKCS#11 functions.
- Although SAPI describes new object classes, this API does not really use PKCS#11 object-related functions (such as **C_FindObjects**). These object classes are used only to group the relevant attributes, while separate API functions are used to operate with each feature.

The API is defined in the eTSAPI.h header file.

OTP Functionality

Some token devices, such as eToken NG-OTP and SafeNet eToken Virtual, support one-time password (OTP) functionality. OTP-capable tokens should be initialized with some OTP secret. The user may then generate a new OTP value by pressing the button without even connecting the token to the computer and use the token in various password-protection schemes (certainly, proper back end support is needed).

Generally speaking the token may be able to support in the future multiple OTP secrets and support multiple OTP algorithms. But SAPI supports addressing of only one OTP secret per token is supported, it is referenced indirectly by all OTP functions.

Similarly, currently only one OTP algorithm is supported. This is an HMAC-SHA1 based HOTP algorithm.

For more information, refer to:

http://tools.ietf.org/html/draft-mraihi-oath-hmac-otp-04

The token should be initialized with HMAC-SHA1 support to use HOTP functionality.

The application may use:

- **`SAPI_OTP_GetMechanismList`** and **`SAPI_OTP_GetMechanismInfo`** to examine the token's OTP capabilities.
- **`SAPI_OTP_Create`** to create instances of the OTP object.
- **`SAPI_OTP_Destroy`** to delete it (the OTP object).
- **`SAPI_OTP_GetAttributeValue`** to examine attributes of the existing OTP object.
- **`SAPI_OTP_SetAttributeValue`** to change the presentation duration of the OTP object on the token.
- **`SAPI_OTP_Execute`** to perform OTP calculations on the token.
- **`SAPI_Server_OTP_Calculate`** to perform OTP calculations without the token, passing the OTP secret as a parameter. This function is used by the server application to validate user authentication.

Miscellaneous Functionality

Supplementary API provides application developers with additional functionality that may be usable for some applications.

The application may use:

- **`SAPI_GetLibraryInfo`** to obtain information about the library version.
- **`SAPI_GetSlotInfo`** to obtain information about the particular slot.
- **`SAPI_GetTokenInfo`** to obtain full information about the token model, version and capability. The function returns the required information in the template of attributes that describes "token object".
- **`SAPI_InitToken`** to initialize the token. The function gets the initialization parameters in the template of attributes that describe "token object" (many of these attributes are common with the **`SAPI_GetTokenInfo`** function).

- **SAPI_FindTokens** or **SAPI_LocateToken** to find the needed token between all tokens currently inserted in the computer.
- **SAPI_UnblockPIN**, **SAPI_UnblockPINEx** and **SAPI_Server_Unblock** to unblock the user PIN remotely. Unlike the **C_InitPIN** function of PKCS#11, it allows unblocking of the user PIN without the Security Officer PIN (SO PIN) being known by the client application by using a challenge-response mechanism. The function **SAPI_UnblockPIN** is run on the client machine (where the token is inserted), while the function **SAPI_Server_Unblock** cooperates on the server side, computing the proper cryptographic response.
- **SAPI_Login** and **SAPI_SetPIN** allow applications to use UI and password policy mechanisms of the SafeNet Authentication Client (something that comes automatically since PKI Client 4.0, but not in previous versions).

Data Types

SAPI uses data types as defined in PKCS#11 and in addition defines several more. These are described here:

CK_INIT_CALLBACK

```
typedef CK_CALLBACK_FUNCTION (CK_RV, CK_INIT_CALLBACK)
(CK_VOID_PTR pContext, CK_ULONG progress);
```

This callback function is used with the **SAPI_InitToken** function to let the application show its progress as percentages.

The parameters are:

- **pContext** - the context that has been passed to SAPI_InitToken
- **progress** - the formatting progress in percentage.

This function should return CKR_OK to continue the initialization or any other return value to stop it.

CK_UNBLOCK_CALLBACK

```
typedef CK_CALLBACK_FUNCTION (CK_RV, CK_UNBLOCK_CALLBACK)
(CK_SESSION_HANDLE hSession, CK_VOID_PTR pChallenge,
CK_VOID_PTR pResponse);
```

This callback function is for the **SAPI_UnblockPIN** function and is used to complete the challenge-response calculation.

The parameters are:

- **hSession** - the session handle that has been passed to the SAPI_UnblockPIN function
- **pChallenge** - the cryptographic challenge as created by the token
- **pResponse** - the response to the challenge

CK_UNBLOCK_CALLBACK_EX

```
typedef CK_CALLBACK_FUNCTION (CK_RV, CK_UNBLOCK_CALLBACK_EX)
(CK_VOID_PTR pContext, CK_VOID_PTR pChallenge, CK_VOID_PTR
pResponse);
```

This callback function is for the **SAPI_UnblockPINEx** function and is used to complete the challenge-response calculation.

The parameters are:

- **pContext** - application context that has been passed to the SAPI_UnblockPIN function
- **pChallenge** - the cryptographic challenge as created by the token
- **pResponse** - the response to the challenge

SAPI_PIN_POLICY_INFO

```
typedef struct tagSAPI_PIN_POLICY_INFO
{
    CK_RV warning;
    CK_ULONG days;
    CK_ULONG warningPeriod;
    CK_ULONG expiryPeriod;
}SAPI_PIN_POLICY_INFO;
```

This structure is used in function **SAPI_Login** to return the information about need to change password.

CK_SAPI_OTP_MECHANISM_INFO

```
typedef struct tagCK_SAPI_OTP_MECHANISM_INFO
{
    CK_ULONG mechanism;// CK_SAPI_OTP_HMAC_SHA1_DEC6
    CK_ULONG minKeyLen;
    CK_ULONG maxKeyLen;
    CK_ULONG OTPLen;// 6
    CK_ULONG defDuration;
    CK_ULONG flags;
}CK_SAPI_OTP_MECHANISM_INFO,
*CK_SAPI_OTP_MECHANISM_INFO_PTR;
```

This structure describes information about the OTP mechanism as returned from the function **SAPI_OTP_GetMechanismInfo**.

The fields of the structure are:

- **mechanism** - OTP mechanism (only the **CK_SAPI_OTP_HMAC_SHA1_DEC6** mechanism is currently supported)
- **minKeyLen** - the minimum key length of the OTP secret
- **maxKeyLen** - the maximum key length of the OTP secret
- **OTPLen** - the size of the produced OTP value (excluding the final zero-character)
- **defDuration** - the duration of the OTP presentation in seconds
- **flags** - this can be a combination of one or more flags that describe the mechanism's capabilities. They can be:

- **CK_SAPI_OTP_CURRENT_SUPPORTED** - the token supports retrieving the current OTP value. See *SAPI_OTP_Execute* on page 109.
- **CK_SAPI_OTP_ZERO_SUPPORTED** - the token supports retrieving the "zero" OTP value. See *SAPI_OTP_Execute* on page 109.
- **CK_SAPI_OTP_CUSTOM_DURATION** - the token supports customization of the OTP display duration.
- **CK_SAPI_OTP_CTL_DURATION** - the ability to change the OTP display duration can be restricted at the time of OTP object creation.
- **CK_SAPI_OTP_BUTTON SUPPORTED** - token supports the use of an OTP button when the token is connected to the computer.

These flags are kept for backward compatibility. In SafeNet Authentication Client 8.0 all these flags are set.

Error Codes

Usually, SAPI functions return the standard PKCS#11 error codes. The following table shows some of the more common error codes likely to be returned. In addition to these, refer to the PKCS#11 documentation for a complete list of the error codes:

http://www.rsasecurity.com/rsalabs/node.asp?id=2133

General PKCS#11 Error Codes

Name	Description
CKR_TEMPLATE_INCOMPLETE	A mandatory attribute is not passed.
CKR_TEMPLATE_INCONSISTENT	Certain passed attributes make no sense together.

Name	Description
CKR_ARGUMENTS_BAD	The required operation cannot be performed with the passed parameters
CKR_USER_NOT_LOGGED_IN	The user is required to be logged in.
CKR_DEVICE_ERROR	An object is damaged.

eTSAPI functionality allows for certain actions that are not covered by PKCS#11 error codes and for these actions SAPI specific codes are required. The instances of this are limited. The following table shows SAPI specific error codes.

SAPI Specific Error Codes

Name	Description
CKR_SAPI_OBJECT_DOES_NOT_EXIST	The object asked about in the operation does not exist.
CKR_SAPI_OBJECT_ALREADY_EXISTS	An object already exists. This may be returned from functions like SAPI_BI_Create and SAPI_OTP_Create
CKR_SAPI_NOT_SUPPORTED_BY_TOKEN	The token does not support the requested feature.
CKR_SAPI_PIN_QUALITY	The newly supplied PIN does not meet requirements of password policy
CKR_SAPI_PIN_DEFAULT	The default PIN must be changed. This error code will never return in SafeNet Authentication Client 8.0 (due to changes in the password policy mechanisms).
CKR_SAPI_PIN_EXPIRATION	The PIN is expired.
CKR_SAPI_PIN_CHANGE_NOT_ALLOWED	The PIN change is currently not allowed
CKR_SAPI_CANCELLED	The UI operation is cancelled by user.

Note:

- The specific error codes returned by functions in the case of such or other failure may sometimes vary between PKI Client versions.
- Due to historical reasons SAPI introduces several error codes similar to error codes introduced in later versions of PKCS#11.

SAPI Objects

As mentioned previously, SAPI does not have concept of objects as it is introduced by PKCS#11. You cannot operate with SAPI objects by using PKCS#11 functions. However, similar to PKCS#11, SAPI uses templates of attributes to identify the entities it works with. They are called 'objects' as long as we speak about SAPI.

Slot Object

The slot object is used to represent the token characteristics that are not available via the **`C_GetSlotInfo`** function.

CKA_SAPI_SLOT_NAME (CK_UTF8CHAR_PTR)

This attribute is a null terminated string. For PC/SC readers it contains the full reader name while for SafeNet eToken Virtual contains the file name (empty string is returned if no file is associated with the SafeNet eToken virtual slot). See *ETCK_IODEV_FULL_NAME* on page 62.

CKA_SAPI_SLOT_TYPE (CK_ULONG)

This attribute contains a constant that defines the slot type thereby distinguishing virtual slots from real slots and tokens from smartcards.

These are the possible values:

`CK_SAPI_SLOT_SC_READER` - PKCS#11 slot corresponds to the real smart card reader (For example, Athena reader).

`CK_SAPI_SLOT_SC_VIRTUAL` - PKCS#11 slot corresponds to the SafeNet eToken Virtual reader (named ordinaly "AKS ifdh 0", ..1, ..2 and so on).

`CK_SAPI_SLOT_FILE` - PKCS#11 slot corresponds to the 'software' token (that is, the binary file).

Token Object

This object is used to:

- Represent token characteristics not available via the **`C_GetTokenInfo`** function.
- Learn whether the token has some special capabilities (like OTP).
- Perform token initialization.

Note:
Some token attributes may not be allowed during initialization, while others are allowed only during initialization.

In SafeNet Authentication Client 8.0 most of this information is represented via the special hardware feature object E**`TCKH_TOKEN_OBJECT.`**

CKA_SAPI_CARD_ID (CK_BYTE_PTR)

This is the smartcard's unique ID. It is unique for cards from a particular OS vendor (in conjunction with **`CKA_SAPI_CARD_TYPE`**). Cardless tokens (for example, SafeNet eToken Virtual) return an empty byte array as the smartcard ID.

This is a read only attribute.

See also ETCKA_CARD_ID, in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_CARD_TYPE (CK_ULONG)

This distinguishes cards from different vendors with **`CK_SAPI_CARD_NONE`** meaning No Smartcard and **`CK_SAPI_CARD_OS`** meaning Siemens CardOS.

This is a read only attribute.

See also ETCKA_CARD_TYPE in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_CARD_VERSION (CK_VERSION)

This is the OS version of the smartcard.

This is a read only attribute

See also ETCKA_CARD_VERSION in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_CASE_MODEL (CK_ULONG)

This refers to constants that state how the casing looks:

CK_SAPI_CASE_NONE	Smartcard or SafeNet eToken Virtual
CK_SAPI_CASE_CLASSIC	Classic shape (eToken PRO)
CK_SAPI_CASE_NG1	"NG1" shape (eToken NG-OTP)
CK_SAPI_CASE_NG2	"NG2" shape (eToken NG-OTP)
ETCK_CASE_NG2_NOLCD	"NG2" shape (eToken NG Flash)

This is a read only attribute.

See also ETCKA_CASE_MODEL in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_COLOR (CK_ULONG)

This provides information on the token color where it has been burnt in or on a smartcard. See also ETCKA_COLOR in ETCKH_TOKEN_OBJECT. For tokens that keep no information about token color PKI Client 4.0 sets this attribute to "Unknown" (0xFFFFFFFF).

CKA_SAPI_FIPS (CK_BBOOL)

When used in the **SAPI_GetTokenInfo** function, this attribute states whether the token is currently initialized as FIPS-compliant or not. When passed to the **SAPI_InitToken** function, this attribute defines whether the token should be initialized as FIPS-compliant or not.

See also ETCKA_FIPS in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_FIPS_SUPPORTED (CK_BBOOL)

This attribute states whether the token can be initialized as a FIPS token.

This is a read only attribute.

See also ETCKA_FIPS_SUPPORTED in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_HAS_LCD (CK_BBOOL)

This indicates whether or not the token has an LCD display by means of a true or false answer.

This is a read only attribute.

See also ETCKA_HAS_LCD in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_HAS_SO (CK_BBOOL)

This indicates whether or not the token has a Security Officer by means of a true or false answer.

This is a read only attribute.

See also ETCKA_HAS_SO in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_HAS_USER (CK_BBOOL)

This attributes determines whether the token is initialized or empty.

This is a read only attribute.

SafeNet Authentication Client 8.0 returns this information also via flag **`CKF_USER_PIN_INITIALIZED`** in the **`CK_TOKEN_INFO`** structure.

CKA_SAPI_HMAC_SHA1 (CK_BBOOL)

When used in the **SAPI_GetTokenInfo** function, this attribute states whether the token currently supports the HMAC SHA1 algorithm or not. When passed to the **SAPI_InitToken** function, this attribute defines whether you need the token to support the HMAC SHA1 algorithm.

See also ETCKA_HMAC_SHA1 in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_HMAC_SHA1_SUPPORTED (CK_BBOOL)

This attribute states whether the token can be initialized with HMAC SHA1 algorithm support.

This is a read only attribute.

See also ETCKA_HMAC_SHA1_SUPPORTED in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_INIT_PIN_REQ CK_BBOOL)

This attribute states whether the **CKA_SAPI_PIN_CURRENT** attribute is required for token initialization.

This is a read only attribute.

CKA_SAPI_MAY_INIT(CK_BBOOL)

This attribute states whether or not it is possible to initialize the token via **SAPI_InitToken**.

This is a read only attribute.

CKA_SAPI_MODEL (CK_CHAR_PTR0)

This produces a character string describing the product and includes information on the hardware version. This information may be displayed by an application to get a token description. This description is not informative for user applications, but may be helpful for support reasons including troubleshooting.

The content of this attribute may change as a result of a token FW upgrade.

This is a read only attribute.

See also ETCKA_MODEL in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_NEW_KEY (CK_BYTE_PTR)

This attribute defines the secret key that will be set on the token for further initialization. Absence of this attribute means that the default key will be used. This attribute may be used only when calling **`SAPI_InitToken`**.

CKA_SAPI_OLD_KEY (CK_BYTE_PTR)

This attribute defines the secret key used for token initialization. Absence of this attribute means that the default key will be used. This attribute may be used only when calling **`SAPI_InitToken`**.

CKA_SAPI_PIN_CURRENT (CK_CHAR_PTR)

This is the current password of the user or SO. This is supplied only if the token is re initialized after having been initialized in FIPS mode. This attribute may be used only when calling **`SAPI_InitToken`**.

CKA_SAPI_PIN_SO (CK_CHAR_PTR)

This is used only for token initialization and the administrator password is provided to the token. If no password is supplied, the token will not have an administrator. This attribute may be used only when calling **`SAPI_InitToken`**.

CKA_SAPI_PIN_USER (CK_CHAR_PTR0

This is used only for token initialization and the user password is provided to the token. If no password is supplied, the token is initialized as empty. This attribute may be used only when calling **`SAPI_InitToken`**.

CKA_SAPI_PRODUCT_NAME (CK_CHAR_PTR)

This is a product name like eToken PRO or eToken NG-OTP and contains the token type encoded as a string.

This is a read only attribute.

See also ETCKA_PRODUCT_NAME in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_PRODUCTION_DATE (CK_DATE)

This is the date on which the token was produced. This attribute may be zeroed for tokens that do not store the production date. If this information is not available, the size of the returned attribute will be 0 bytes

This is a read only attribute.

See also ETCKA_PRODUCTION_DATE in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_REAL_COLOR(CK_BBOOL)

This indicates whether the color information returned by the **`CKA_SAPI_COLOR`** attribute is burned onto the token during production. If this attribute is FALSE, the SafeNet Authentication Client 8.0 will return "Unknown" (0xFFFFFFFF) as the color value (earlier versions could return an arbitrary value).

This is a read only attribute.

See also ETCKA_REAL_COLOR in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_RETRY_SO (CK_ULONG)

This is the current number of failed log attempts remaining before the Security Officer (SO) PIN is locked. It should be noted that when log in is successful, the counter automatically reverts to the maximum for future attempts.

This is a read only attribute.

See also ETCKA_RETRY_SO in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_RETRY_SO_MAX (CK_ULONG)

This attribute defines the maximum number of log on attempts a user can make with incorrect passwords before the SO PIN is locked. When initializing the token, this attribute applies only if a SO PIN (**`CKA_SAPI_SO_PIN`**) is also supplied.

See also ETCKA_RETRY_SO_MAX in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_RETRY_USER (CK_ULONG)

This is the current number of failed log on attempts remaining before the user PIN is locked. When the logon is successful, the counter automatically reverts to the maximum for future attempts.

This is a read only attribute.

See also ETCKA_RETRY_USER in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_RETRY_USER_MAX (CK_ULONG)

This attribute defines the maximum number of log in attempts a user can make with incorrect passwords before the user PIN has been locked. When initializing the token, this attribute applies only if a user PIN (**`CKA_SAPI_USER_PIN`**) is also supplied.

See also ETCKA_RETRY_USER_MAX in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_RSA_KEYS (CK_ULONG)

This attribute is used only in the **`SAPI_InitToken`** function to define the amount of space reserved for RSA keys during token initialization. It is defined in terms of the number of 1024-bit RSA keys that may be created. If this parameter is omitted, the default value will be used. If 0 is passed, no place will be allocated for RSA keys.

See also ETCKA_RSA_AREA_SIZE in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_RSA_2048 (CK_BBOOL)

When used in the **`SAPI_GetTokenInfo`** function, this attribute states whether the token currently supports RSA 2048 keys or not. When passed to the **`SAPI_InitToken`** function, this attribute defines whether you need the token to support RSA 2048 keys.

See also ETCKA_RSA_2048 in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_RSA_2048_SUPPORTED (CK_BBOOL)

This attribute states whether the token can be initialized with RSA 2048 key support.

This is a read only attribute.

See also ETCKA_RSA_2048_SUPPORTED in the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_SERIAL (CK_CHAR_PTR)

This is a unique token identifier. This field should be used by applications to refer to the particular token. It is guaranteed to be unique and compatible with the corresponding field in the CK_TOKEN_INFO structure in PKCS#11.

This is a read only attribute.

CKA_SAPI_TOKEN_ID (CK_BYTE_PTR)

This is a unique ID for each USB token. The tokens that have no meaningful token ID (such as smartcards) return an empty byte array as the token ID.

This is a read only attribute.

See also ETCKA_TOKEN_IDin the *Token Feature Object (ETCKH_TOKEN_OBJECT)* table on page 51.

CKA_SAPI_USER_PIN_INITIALIZED (CK_BBOOL)

This attribute defines the user PIN as being initialized. The default value for this attribute is TRUE. This attribute may be used only when calling **SAPI_InitToken**.

OTP Object

The OTP object represents the OTP secret and corresponding data (such as counter), stored and operated by the token. Generally speaking, the token may support multiple OTP algorithms as well as multiple objects implementing the same OTP algorithm. Only one OTP object per token is currently supported by SAPI.

Attributes

When the token is initialized, these attributes control how the token will behave. When the attributes are simply being read, they inform the application about how the token behaves. These attributes cannot be changed.

CKA_SAPI_OTP_COUNTER (CK_ULONG)

This determines the current value of the moving factor. If not supplied to the function **SAPI_OTP_Create**, 0 will be used as the default value. This attribute may not be changed by an application after creation.

See also CKA_OTP_COUNTER on page 33.

CKA_SAPI_OTP_CURRENT_ALLOWED (CK_BBOOL)

This attribute defines whether the last OTP value may be received by using the **SAPI_OTP_Execute** function. This attribute is TRUE for all currently supported tokens. SafeNet Authentication Client 8.0 always returns TRUE for this attribute.

This is a read only attribute.

CKA_SAPI_OTP_CUSTOM_DURATION_ALLOWED (CK_BBOOL)

This indicates whether, after the object was created, the duration can be changed.

See also ETCKA_OTP_MAY_SET_DURATION in PKCS#11 CKO_OTP object on page 48.

CKA_SAPI_OTP_DURATION (CK_ULONG)

This determines for how long (in seconds) the OTP value appears on the token when the button is pressed.

See also ETCKA_OTP_DURATION in PKCS#11 CKO_OTP object on page 32.

CKA_SAPI_OTP_MECHANISM (CK_MECHANISM_TYPE)

This identifies the particular OTP mechanism.

CKA_SAPI_OTP_VALUE (CK_BYTE_PTR)

This attribute contains the value of the OTP secret that should be passed to the **`SAPI_OTP_Create`** function. This is a sensitive attribute in PKCS#11 terms.

The OTP secret value range is different for Java Card & CardOS OTP devices:

- Java Card OTP MinKeySize = 20
- Java Card OTP MaxKeySize = 24
- CardOS OTP MinKeySize = 20
- CardOS OTP MaxKeySize = 32

CKA_SAPI_OTP_ZERO_ALLOWED (CK_BBOOL)

This attribute defines whether the OTP computation based on a zero-counter is allowed. In SafeNet Authentication Client 8.0 it should be passed as TRUE.

Refer to the *SAPI_OTP_Execute* function on page 109 for more information.

Functions

Note:
In this section the term 'object' means SAPI pseudo-objects, unless specified differently.

Common Functionality

SAPI_GetLibraryInfo

This function returns version information about the currently installed SAPI and underlying SafeNet Authentication Client.

```
CK_RV SAPI_GetLibraryInfo (
    CK_VERSION_PTR pSapiVersion,
    CK_VERSION_PTR pRteVersion
);
```

Parameters

pSapiVersion

[out] Pointer to the structure that receives the current SAPI API version number. SafeNet Authentication Client 8.0 returns 1.2 in this field.

pRTEVersion

[out] Pointer to the structure that receives the currently installed PKI Client version number. The SafeNet Authentication Client returns 8.0 in this field.

Slot/Token Functionality

SAPI_GetSlotInfo

This function returns information about a particular slot. The template may include any attributes defined for the slot object.

```
CK_RV SAPI_GetSlotInfo (
    CK_SLOT_ID slotId,
    CK_ATTRIBUTE_PTR pTemplate,
    CK_ULONG ulCount
);
```

Parameters

slotId

[in] Slot identifier for which information is requested.

pTemplate

[in/out] Pointer to the attributes array that receives the requested information.

ulCount

[in] Count of attributes in the **`pTemplate`** array.

SAPI_GetTokenInfo

This function returns information about a particular token. The template may contain any attributes defined for the token object.

```
CK_RV SAPI_GetTokenInfo (
    CK_SLOT_ID slotId,
    CK_ATTRIBUTE_PTR pTemplate,
    CK_ULONG ulCount
);
```

Parameters

slotId

[in] identifier of the slot where the token is located

pTemplate

[in/out] Pointer to the attributes array that receives the requested information.

ulCount

[in] Count of attributes in the *pTemplate* array.

SAPI_SetTokenName

This function is used to set the token name without re-initializing it. Since it is currently the only function that sets a token property and since this property is available via the **`C_GetTokenInfo`** structure, it does not use the template as a parameter.

```
CK_RV SAPI_SetTokenName (
    CK_SESSION_HANDLE hSession,
    CK_CHAR_PTR label
);
```

Parameters

hSession

[in] Session handle opened for given token.

label

[in] Zero terminated string of the new token name that will be set.

Remarks

The application should pass proper authentication to use this function. The token label is subject to size restriction as defined in CK_TOKEN_INFO structure and in PKCS#11-part of this document.

SAPI_InitToken

This function is used to initialize the token. Token attributes are passed to customize the initialization. The callback function may be used to report the status of initialization.

```
CK_RV SAPI_InitToken (
    CK_SLOT_ID slotId,
    CK_ATTRIBUTE_PTR pTemplate,
    CK_ULONG ulCount,
    CK_VOID_PTR pContext,
    CK_INIT_CALLBACK_p Callback
);
```

Parameters

slotId

[in] identifier of the slot where the token is located

pTemplate

[in] Pointer to the attributes array that contains the initialization parameters.

ulCount

[in] Count of attributes in the **`pTemplate`** array.

pContext

[in] User provided parameter to be passed to **`pCallback`** function.

pCallback

[in] Callback function that lets the application show the progress of the token initialization process. The callback is optional (NULL may be passed).

SAPI_FindTokens

This function returns a list of slots containing tokens that comply with search criteria. The search criteria may contain any token attributes except those that may be used only for initialization.

Note:
We do not recommend calling this function with an empty template, as the behavior may vary between PKI Client versions.

```
CK_RV SAPI_FindTokens (
    CK_SLOT_ID_PTR pSlots,
    CK_ULONG_PTR pSlotCount,
    CK_ATTRIBUTE_PTR pTemplate,
    CK_ULONG ulCount
);
```

Parameters

pSlots

[out] Pointer to the array that will be filled by found slot identifiers.

pSlotCount

[in/out] Pointer to the variable that:

- On input defines the size of the **`pSlots`** array
- On output receives the actual number of found slots

pTemplate

[in] Pointer to the attributes array containing the parameters of the requested tokens.

ulCount

[in] Count of attributes in the **`pTemplate`** array.

SAPI_LocateToken

This function finds the slot where a particular token is located.

```
CK_RV SAPI_LocateToken (
    CK_VOID_PTR unique,
    CK_ULONG size,
    CK_SLOT_ID_PTR pSlotId
);
```

Parameters

unique

[in] Pointer to the buffer containing the serial number (CKA_SAPI_SERIAL) of the token to be located.

size

[in] Size of the unique buffer.

pSlotId

[out] Pointer to the variable that receives the located token's slot identifier.

SAPI_UnblockPIN

This function is used to unblock the user PIN (using a challenge-response mechanism). The token should have an Security Officer(SO) PIN, but the SO should not be logged on.

PKCS#11 has the function **`C_InitPIN`** that may be used by the SO to unblock the user password. However, it requires the application to log on firstly with the SO password. This approach may not be applicable for some real-world applications when the token is located on the user's site because the administrator will not be ready to reveal the SO password to the user.

The function **SAPI_UnblockPIN** uses a challenge-response mechanism instead of the password to authenticate the administrator. The cryptographic challenge is received from the token and passed to the **pCallback** function provided by the application. The application has to compute a response in order to complete authentication to the token. The **SAPI_Server_Unblock** function may be used for this purpose. Since this function does not require a token presence it may be performed on the server site.

```
CK_RV SAPI_UnblockPIN (
    CK_SESSION_HANDLE hSession,
    CK_CHAR_PTR pNewPin,
    CK_ULONG ulNewPinLen,
    CK_UNBLOCK_CALLBACK pCallback
);
```

Parameters

hSession

[in] Session handle opened for the given token.

pNewPin

[in] Pointer to the new user PIN.

pNewPinLen

[in] Size of **pNewPin**.

pCallback

[in] Callback function that is expected to compute the response for the given challenge.

SAPI_UnblockPINEx

The function is equivalent to **SAPI_UnblockPIN**, except that it gets one more parameter (**pContext**), which is passed to the callback function.

```
CK_RV SAPI_UnblockPIN (
    CK_SESSION_HANDLE hSession,
    CK_CHAR_PTR pNewPin,
    CK_ULONG ulNewPinLen,
    CK_UNBLOCK_CALLBACK_EX pCallback,
    CK_VOID_PTR pContext
);
```

Parameters

hSession

[in] Session handle opened for the given token.

pNewPin

[in] Pointer to the new user PIN.

pNewPinLen

[in] Size of **pNewPin**.

pCallback

[in] Callback function that is expected to compute the response for the given challenge.

pContext

[in] Context that will be passed to the callback function.

SAPI_Login

The function extends **C_Login** functionality by UI and password policy management.

```
CK_RV SAPI_Login (
    CK_SESSION_HANDLE hSession,
    CK_USER_TYPE userType,
    CK_CHAR_PTR pPin,
    CK_ULONG ulPinLen,
    SAPI_PIN_POLICY_INFO* pPolicyInfo
);
```

Parameters

hSession

[in] Session handle opened for the given token.

userType

[in] **CKU_USER** or **CKU_SO**.

pPin

[in] PIN.

ulPinLen

[in] PIN length.

pPolicyInfo

[out] Information about password expiration.

Remarks

- The function takes in account the password policy settings even for prior versions.
- If NULL is passed as **pPin** and 0 is passed as **ulPinLen**, the SafeNet Authentication Client UI will appear prompting for user or SO login.
- If **pPolicyInfo** is passed, the function will return valuable information about password expiration. It returns information about expiration and warning period and number of days remained till password expiration. If the password change is required, the function will fail with the proper error code. If the password will expire soon, the function will success and will return the expected (future) error in the warning-field.
- **pPolicyInfo** cannot be used together with SafeNet Authentication Client UI.
- Password policy is applied only to the user PIN.

SAPI_SetPIN

The function extends **C_SetPIN** functionality by UI and password policy management.

```
CK_RV SAPI_SetPIN (
    CK_SESSION_HANDLE hSession,
    CK_CHAR_PTR pOldPin,
    CK_ULONG ulOldPinLen,
    CK_CHAR_PTR pNewPin,
    CK_ULONG ulNewPinLen
    );
```

Parameters

hSession

[in] Session handle opened for the given token.

pOldPin

[in] Old PIN.

ulOldPinLen

[in] Old PIN length.

pNewPin

[in] New PIN.

ulNewPinLen

[in] New PIN length.

Remarks:

- If NULL is passed as old and new PIN, the SafeNet Authentication Client UI will appear prompting for user or SO password change.
- Password policy is applied only to the user PIN.

OTP Functionality

Theoretically a single token may support multiple OTP algorithms and keep more than one OTP object. However SAPI makes the following assumptions:

- Only one OTP object currently exists on the token. Therefore no special mechanism is proposed to address the particular OTP object instance on the token.
- The supported OTP algorithms are counter-based (only **CK_SAPI_OTP_HMAC_SHA1_DEC6** is supported, which is equivalent to **CKM_HOTP**). It is not mentioned explicitly across the API, but is implied from the set of attributes defined for the OTP object.
- The token is properly initialized in order to operate with OTP.

SAPI_OTPGetMechanismList

This function returns a list of available OTP mechanisms. Depending on OTP-support on the token, either **CK_SAPI_OTP_HMAC_SHA1_DEC6** or zero-length list will be returned.

```
CK_RV SAPI_OTP_GetMechanismList (
    CK_SLOT_ID slotId,
    CK_ULONG_PTR pMechanismList,
    CK_ULONG_PTR pCount
```

);

Parameters

slotId

[in] identifier of the slot where the token is located

pMechanismList

[out] Pointer to the array that will be filled by OTP mechanism identifiers.

pCount

[in/out] Pointer to the variable that:

- On input defines the size of the **`pMechanismList`** array
- On output receives the actual number of found mechanisms

SAPI_OTP_GetMechanismInfo

This function returns information about a specific OTP mechanism.

```
CK_RV SAPI_OTP_GetMechanismInfo (
    CK_SLOT_ID slotId,
    CK_ULONG mechanism,
    CK_SAPI_OTP_MECHANISM_INFO_PTR pMechanismInfo
);
```

Parameters

slotId

[in] identifier of the slot where the token is located

mechanism

[in] OTP mechanism identifier for which information is requested.

pMechanismInfo

[out] Pointer to the structure that receives the OTP mechanism information.

SAPI_OTP_Create

This function creates an OTP object. User login should be performed prior to this operation.

```
CK_RV SAPI_OTP_Create (
    CK_SESSION_HANDLE hSession,
    CK_ATTRIBUTE_PTR pTemplate,
    CK_ULONG ulCount
);
```

Parameters

hSession

[in] Session handle opened for given token.

pTemplate

[in] Pointer to the attributes array containing the parameters of the created OTP object.

ulCount

[in] Count of attributes in the **pTemplate** array.

Remarks

Mandatory attributes to be provided are **CKA_SAPI_OTP_MECHANISM** and **CKA_SAPI_OTP_VALUE**.

If no specific attribute is provided, then zero will be used as the default value for these attributes.

SAPI_OTP_GetAttributeValue

This function returns the characteristics of an existing OTP object. The key value is sensitive and will not be returned.

```
CK_RV SAPI_OTP_GetAttributeValue (
    CK_SESSION_HANDLE hSession,
    CK_ATTRIBUTE_PTR pTemplate,
    CK_ULONG ulCount
);
```

Parameters

hSession

[in] Session handle opened for given token.

pTemplate

[in/out] Pointer to the attributes array that receives the requested information.

ulCount

[in] Count of attributes in the **`pTemplate`** array.

SAPI_OTP_SetAttributeValue

This function is used to change the OTP object parameters. Only the display duration may be changed.

```
CK_RV SAPI_OTP_SetAttributeValue (
    CK_SESSION_HANDLE hSession,
    CK_ATTRIBUTE_PTR pTemplate,
    CK_ULONG ulCount
);
```

Parameters

hSession

[in] Session handle opened for given token.

pTemplate

[in] Pointer to the attributes array containing the new parameters of the OTP object.

ulCount

[in] Count of attributes in the **`pTemplate`** array.

SAPI_OTP_Destroy

This function deletes an existing OTP object. User login should be performed prior to this operation.

```
CK_RV SAPI_OTP_Destroy (
    CK_SESSION_HANDLE hSession
);
```

Parameters

hSession

[in] Session handle opened for given token.

SAPI_OTP_Execute

This function computes an OTP value.

```
CK_RV SAPI_OTP_Execute (
    CK_SESSION_HANDLE hSession,
    CK_ULONG mode,
    CK_CHAR_PTR pResult,
    CK_ULONG_PTR pSize
);
```

Parameters

hSession

[in] Session handle opened for given token.

mode

[in] Execution mode. (See Remarks)

pResult

[out] Pointer to the buffer which receives the executed value.

pSize

[in/out] Pointer to the variable that:

- On input defines the size of the **`pResult`** array
- On output receives the actual length of the **`pResult`** array.

Remarks

- Mode **`CK_OTP_CURRENT`** returns the last OTP computation without moving the counter.

- Mode **CK_OTP_RELEASE** is kept only for backward compatibility. In prior versions the application was supposed to call the **SAPI_OTP_Execute** function with this mode after getting a new OTP value in order to release the token for future OTP operations. This mode is deprecated now and behaves the same as **CK_OTP_CURRENT**.
- **Mode CK_OTP_ZERO** performs an OTP computation for the zero-counter. This mode may be used when the token and the server are unsynchronized in order to resynchronize them.

SAPI_Server_OTP_Calculate

This function calculates the OTP value on the server for a given mechanism, key and counter.

```
CK_RV SAPI_Server_OTP_Calculate (
    CK_ATTRIBUTE_PTR pTemplate,
    CK_ULONG ulCount,
    CK_CHAR_PTR pResult,
    CK_ULONG_PTR pSize
);
```

Parameters

pTemplate

[in] Pointer to the attributes array containing the OTP object parameters.

ulCount

[in] Count of attributes in the **pTemplate** array.

pResult

[out] Pointer to the buffer which receives the calculated value.

pSize

[in/out] Pointer to the variable that:

- On input defines the size of the **pResult** array
- On output receives the actual length of the **pResult** array.

Remarks

The template should contain the following attributes:

- **CKA_MECHANISM**
- **CKA_COUNTER**
- **CKA_VALUE**

SAPI_Server_BI_EstimateValue

This function is deprecated.

SAPI_Server_BI_EstimateRetainDays

This function is deprecated.

SAPI_Server_Unblock

This function computes the proper response for the challenge-response mechanism used during user PIN unblocking.

```
CK_RV SAPI_Server_Unblock (
    CK_CHAR_PTR pPin,
    CK_ULONG ulPinLen,
    CK_VOID_PTR pChallenge,
    CK_VOID_PTR pResponse
);
```

Parameters

pPin

[in] Pointer to the Security Officer (SO) PIN string.

ulPinLen

[in] Length of SO PIN.

pChallenge

[in] Pointer to the 8 byte length challenge buffer received from the token.

pResponse

[out] Pointer to the 8 byte length response buffer provided to unblock the user PIN.

Major Backward Compatibility Issues of SAPI

- Many SAPI functions are deprecated (especially battery indication).
- Errors returned by the SAPI functions may vary between versions.
- Prior to PKI Client 4.0 it was enough to redistribute only `libeTSapi.so` to use server-only functionality. Since PKI Client 4.0 it is not true; PKI Client must be installed.
- Some OTP attributes are taken as fixed.

Chapter 5

Samples

This chapter describes the current SafeNet Authentication Client 32-bit samples.

- Sample Overview
- Compiling the Samples
- PKCS#11 Samples
- PKCS#11 Token-Specific Extensions Samples
- SAPI Samples

Sample Overview

This chapter provides a description of the samples. The samples themselves are available on the token website. For details contact Support (See *Support* on page iii)

All samples are in C/C++.

The samples cover different programming techniques to show the multiple options available. Different samples may achieve similar results, but through different methodologies.

The samples are not always fully functional applications. Certain details, such as error handling, may be omitted for the sake of brevity.

The following samples are described in this chapter:

API	Sample
PKCS#11	■ Info Test ■ InitToken ■ Password Policy ■ CACert
PKCS#11 Token Specific Extensions	■ Initiating ■ UnlockToken
SAPI	■ InitOTP ■ InitToken ■ TokenInfo

Compiling the Samples

You can compile the samples in one of the following ways:

- Use SlickEdit: open the relevant sample's SlickEdit project (vpw).
- Run Makefile: go to the relevant sample and run the make command.

PKCS#11 Samples

Read the following introductory remarks before starting to work with these samples:

- We recommend loading the library dynamically.
- Do not use hard-coded function names. According to the standard, you should call the function **`C_GetFunctionList`** to reach the addresses of other functions. This is the only function name that is guaranteed to you.
- In many cryptographic applications you have to deal with X.509 certificates. The X.509 certificate has a complicated data structure encoded in a special way (DER-encoding of ASN.1). There is no simple way for applications to deal with this data structure. SafeNet Authentication Client does not provide special helper functions for this purpose. However, you may use Microsoft helper functions described in MSDN. They are used widely in the samples.

CACert

This sample demonstrates how to import a CA certificate into a token.

The sample performs the following actions:

- Reads the CER file.
- Extracts the subject of the CER file.
- Searches for the first connected token.
- Opens a session and performs login to the token.
- Imports the CA certificate into the token.

ClearToken

This sample gets the token password and a certificate label. It then runs over all the token's objects deleting them, excluding the object with the given label.

Info Test

This sample prints general information about the PKCS#11 library and waits for any slot events (token insertion or removal). Upon token insertion, it prints information about the token and its certificates.

The sample continues this operation in an endless loop. It can be stopped by ending the process. No parameters are passed to this sample.

This sample demonstrates the following techniques:

- Dynamic loading and initialization of the PKCS#11 library (all other samples follow the same technique) and reaching addresses of functions via **`C_GetFunctionList`**.
- Obtaining information about the library, slots and tokens via functions **`C_Info`**, **`C_GetSlotInfo`**, **`C_GetTokenInfo`**.
- Obtaining the supported RSA key size with the function **`C_GetMechanismInfo`**.
- Finding objects that satisfy a particular pattern (set of attributes); the sample shows how to find a X.509 certificate object.
- Retrieving object attributes.
- Using Microsoft helper functions to extract information from the certificate.

InitToken

The initialization of token is complicated by the fact that some important issues, such as the need for authentication, are purposely kept out of the scope of the PKCS#11 standard. This sample demonstrates how to initialize the token using PKCS#11. The function that is used in this sample for initialization is **`C_InitToken`** (and not one of PKI Client's extended functions for initialization).

This sample also demonstrates how to generate an RSA key pair on the token.

The sample has the following command line parameters:

- Reader name.
- Token formatting password:
  - For a non-initialized token – use the SO password that is set after initialization (see *Behavior of Standard C_InitToken and C_InitPIN Functions* on page 42).
  - For a token initialized with an administrator password, use the administrator password.
  - For a token initialized without an administrator password, use the current user password.
- User password to be set after initialization.

This sample demonstrates that a flow does not require a UI for any kind of token (not initialized, initialized with or without administrator password).

The suggested flow is:

1. Map the reader name to the slot ID.
2. Open the session with the token. SafeNet Authentication Client will allow the session to open even with a non-initialized token.
3. Login as the SO with the token formatting password.
4. Close the session.

   You should close the session since it cannot perform **`C_InitToken`** if at least one session with the token is open. Your token will be logged out, but SafeNet Authentication Client will keep the password for a period, preventing login UI (for the next step).
5. Perform **`C_InitEToken`**.

   The same formatting password is passed as the parameter.

6. Open a new session.
7. Login as SO with the formatting password.
8. Perform **`C_InitPin`** to initialize the user password.
9. Perform **`C_Logout`**. Your token is initialized now.

   To demonstrate successful token initialization, the sample then logs in as the user and generates an RSA key pair.

Note:
Treatment of the **`C_InitToken`** parameter varies between versions ofPKCS#11. Later versions (v2.11, v2.20) define it as:

- **For an empty token:** New SO password
- **For already initialized token:** Current SO password

Such a definition is much more useful. When future versions of the SafeNet Authentication Client support later versions of PKCS#11, this approach will be applied.

Password Policy

This sample demonstrates how to manipulate the token password policy object. After the token is initialized we change the current password policy object.

This sample demonstrates the use of the following functions, to find the PIN policy object:

- **`C_FindObjectsInit`**
- **`C_FindObjects`**
- **`C_FindObjectsFinal`**

The sample then uses the following to update the current PIN Policy Object:

- **`C_SetAttributeValue`**

The sample then demonstrates a the use of some token functions that verify the new updated settings.

PKCS#11 Token-Specific Extensions Samples

Initiating

This sample demonstrates how to initialize the token using PKCS#11 token-specific extensions.

The sample loads the PKCS# 11 and the PKCS11 Token-Specific extensions function lists and then gets information from the token to decide if the token supports the 2048 key size. The initialization process then begins.

The sample also demonstrates the initialization of the Pin Policy Object.

The sample uses the following functions:

- **`ETC_InitTokenInit`** - create the session (start the transaction)
- **`ETC_InitPIN`** - same as **C_InitPin** but with the addition of two new variables:
  - User retry counter
  - Flag to change the password
- **`ETC_InitTokenFinal`** - end the transaction and physically initialize the token

UnlockToken

Takes a token and unlocks it using PCKS#11 Extensions functions. The sample shows the use of the Challenge-Response process.

Note:
This sample is supported in PKI Client 4.55 or later.

The sample performs the following

- Opens session with the token.
- Prints the pointers of the PCKS#11 Extended functions.
- Checks **`ETC_ UnlockGetChallenge`**.
- Calls PCKs#11 standard functions and prints the user retry counter from the token.
- Performs unlocking of the token:
  - PCKS#11 Extensions **`ETC_ UnlockGetChallenge`**
  - SAPI **`SAPI_Server_Unblock`**
  - PCKS#11 Extensions **`ETC_UnlockComplete`**

SAPI Samples

InitOTP

This sample demonstrates the OTP functionally of the token using SAPI. This sample demonstrates the OTP functionality of the token. It is assumed that token has OTP capabilities and that the token firmware supports the OTP calculation in the online mode (that is, when the token connected). This information is received from the **`SAPI_OTP_GetMechanismInfo`** function.

The sample performs the following:

- Opens the session with the token
- Logs in the user (it is necessary to create an OTP object)
- Destroys old OTP objects (if any)
- Creates a new OTP object (In a real application, the same key should be stored on the server as well)
- Demonstrates an online OTP calculation (it is rarely used in real applications)
- Demonstrates a server-side calculation

InitToken

This sample demonstrates how to initialize a token using SAPI.

TokenInfo

This sample demonstrates how to use SAPI to obtain more detailed information about a particular token.

Index

D

E

I

K

M

N

O

P

S

T

U

V

W