Hardware Security Modules

What is it?

Hardware security modules (HSMs) provide secure key storage and efficient cryptographic processing to facilitate secure electronic transactions. We can distinguish among cryptographic (co)processors, cryptographic accelerators, and cryptographic smart cards. Coprocessors and accelerators are in comparison to smart cards typically much bigger add-on cards or external devices that must be installed into so called host devices (e.g., personal computer or some kind of specialized server).

Basic architecture

The basic architecture of HSMs comes from classical von Neumann architecture with the same building blocks. Moreover, the mechanisms of physical protection, special-purpose (co)processors, generator of true random numbers (TRNG), and non-volatile RAM (NVRAM) were added. Clearly, smart cards are using different mechanisms of physical protection than other add-on cards, but the remaining building blocks are the same. The physical protection can be ensured for example by steel shielding surrounding the device, potting in epoxy resin, using wired mesh (alternatively modern conductive membranes), or various kinds of sensors (e.g., light, power glitch, pressure, thermal, and X-Ray sensors). The special-purpose (co)processors are typically used to accelerate symmetric ciphers (mostly DES, 3DES), hash functions (mostly MD5, SHA-1), or modular arithmetic (multiplication, exponentiation) that is used in many asymmetric cryptosystems. Hardware TRNGs are critical part of all HSMs – necessary for generation high-quality (i.e., perfectly random and unpredictable) cryptographic keys, initializing vectors, padding values, or algorithmic counter-measurements against side channel attacks. Battery powered NVRAM then serves as a secure storage of highly sensitive data (e.g. master keys) – while other keys can be securely stored outside HSM protected by master key(s). On the contrary, the amount of I/O circuits is reduced to achieve simpler and easily verifiable design.

Evaluation

Security requirements for HSMs are specified in FIPS 140-2 and cover areas related to the design and implementation of a cryptographic module. These areas include cryptographic module specification; module ports and interfaces; roles, services, and authentication; finite state model; physical security; operational environment; cryptographic key management; electromagnetic interference/electromagnetic compatibility (EMI/EMC); self-tests; and design assurance. A cryptographic module shall be tested against the requirements of each area and shall be independently rated in each area. In addition to receiving independent ratings for each of the security areas, a cryptographic module will also receive an overall rating (from level 1 to the most secure level 4). The overall rating will indicate the minimum of the independent ratings received in the areas. An additional area concerned with the mitigation of other attacks is currently not tested, but the vendor is required to document implemented controls (e.g., differential power analysis, and TEMPEST).

Eracom ProtectServer Orange (CSA 8000)

Basic features

Eracom ProtectServer Orange HSM (internal version) is a FIPS 140-1 (a predecessor of FIPS 140-2) level 3 certified PCI (2.1 compliant interface, 32 bit, 33 MHz, with both 3.3V and 5V support) adapter that is typically employed to provide cryptographic services such as user and data authentication, message integrity, high-speed encryption, secure key storage and key management for e-Commerce, PKI applications, and financial EFT transactions. The evaluation covers the encapsulated processing subsystem and its specialized cryptographic hardware, code loading, tamper detection and response mechanisms, and the cryptographic algorithms: DES, 3DES, RSA, DSA, and SHA-1.

All following cryptographic techniques are also supported: symmetric cryptosystems AES, DES, 3DES, CAST-128, RC2, RC4 (and modes of operation ECB, CBC, OFB64, CFB-8/BCF); asymmetric cryptosystems RSA (up to 4096 bits), DSA, ECDSA (up to 512 bits), Diffie Hellman; message digests (hashing) algorithms MD2, MD5, SHA-1, SHA-256, SHA-384, SHA-512, RIPEMD-128, RIPEMD-160, MDC2; public key certificates PKCS #10 certificate requests, X.509 v3, PKCS #7 decode, PKCS #12 key and certificate import.

The performance of ProtectServer Orange on selected algorithms is follows: SHA-1 (3 MB/s), DES (4 MB/s), RSA 1024 bit key generation (2.3 seconds), RSA encryption with 1024 bit key (0.88 milliseconds), RSA sign with 1024 bit key (4.3 milliseconds).

ProtectServer Orange HSM contains also two RS-232 serial ports and supports smart card readers (Gemplus GemPC410, Towitoko CHIPDRIVE Extern 320 or CHIPDRIVE Micro 120) and smart cards (GPK 4000, GPK 8000, and GPK 16000).

APIs and developer toolkits

The only communication interface between HSM and host application (executed on the host device) is carefully designed application programming interface (API). API is the interface – the names, parameters and return values – of functions that are executed inside the HSM. This set of functions (ideally with some kind of security policy provided) allows write new applications communicating with HSMs. Eracom ProtectServer Orange HSM supports proprietary PKCS #11 API implementation (ProtectToolkit C), Java JCA/JCE provider implementation (ProtectToolkit J), Cryptographic Service Provider (CSP) for Microsoft CryptoAPI (ProtectToolkit M), EFT command set (ProtectToolkit EFT), and also PKCS#11 based OpenSSL engine integration (to accelerate for example Apache web server).

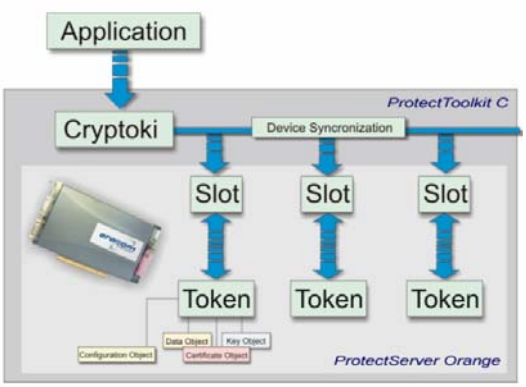
The software development kit (ProtectProcessing) allows an unsurpassed level of flexibility and extensibility. It provides the ability to produce your own custom-specific functionality modules (e.g., new cryptographic algorithms) and allows them to be securely downloaded and executed within the HSM. Development kit supports also a software emulation of HSM that allows easier developing and debugging of such custom-specific functionality modules.

Eracom ProtectToolkit C (based on PKCS #11) overview

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

- PCI mode in conjunction with a compatible Eracom Hardware Security Module (HSM) such as the ProtectServer Orange installed locally.
- Network mode over a TCP/IP network, in conjunction with a compatible Eracom HSM such as the ProtectHost Orange, the ProtectServer Orange or the ProtectServer Orange External.
- Software only mode, on a local machine without access to a hardware security module, for development and testing purposes.

The model for ProtectToolkit C is based on standard PKCS #11 processing as illustrated in the following picture. It demonstrates how an application communicates its requests to a token via the PKCS #11 interface. In this model, a slot represents a device interface and a token represents the actual cryptographic device. For example, a smart card reader would represent a slot and the smart card would represent the token.



ProtectToolkit C supports a number of different slot types: admin slots, user slots, smart card slots:

- The admin slot is designated for the administrator and is used for configuration and administration of the HSM. Each HSM will have a single admin slot.
- User slots are created by the Administrator for each user of the HSM and are designated for standard application usage. Each HSM may have a configurable number of user slots.
- Smart card slots are automatically configured based on the attached smart card readers. Their primary purpose is for key backup and key restoration.

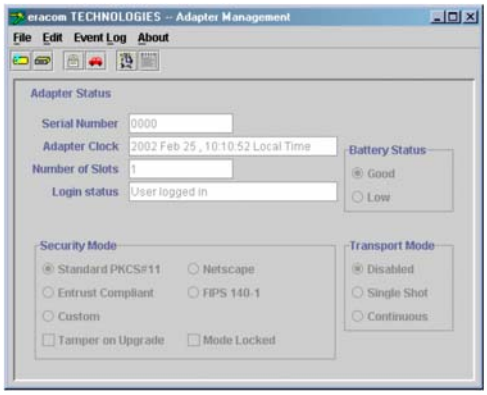
Each token may contain a number of objects. The PKCS #11 standard allows for different types of objects which are classified as follows:

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

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

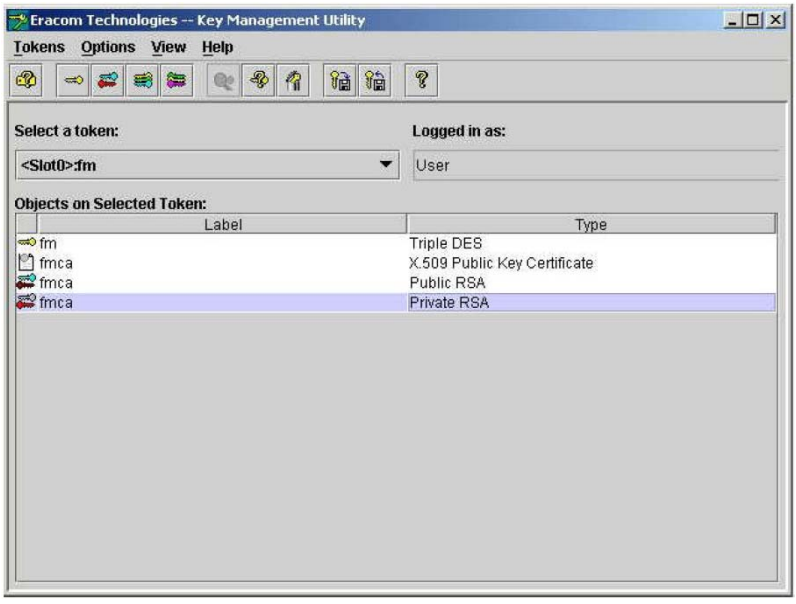
In general both PINs and case-sensitive passwords (between 1 and 32 characters in length) are used to authenticate users and to provide access to secured computer systems.

The administration of Eracom ProtectServer Orange HSM can be done via CLI or GUI. The CLI utilities are for example `ctconf` (configuration utility for administrators used for configure the operating parameters as setting initial admin PIN, security mode, slots, etc.), `ctkmu` (key management utility for key creation, deletion, import, export, as well as PIN change, token initialization and replication, etc.), `ctstat` (used to check the status of a token, determine what state the token is in and what, if any, objects it contains), and `ctperf` (performance reporting utility). Eracom provides also two applications with GUI – the administration utility (GCTADMIN) and the key management utility (KMU).



The java-based GCTADMIN allows management of the HSM hardware using a PKCS #11 sub-system (however, the functionality is identical to the command line utility `ctconf`). After starting GCTADMIN, the utility will check if the HSM hardware has been initialized. If the hardware has not been initialized, the operator will be prompted to initialize the admin token. If the hardware has been initialized, the operator is prompted for entry of the administrator PIN. Following a successful login, the main user interface is displayed (see the picture on the right side). The main interface shows the currently selected HSM and a variety of settings pertaining to the hardware.

The java-based KMU allows management of keys using a PKCS #11 sub-system (however, the functionality is identical to the command line utility `ctkmu`). When the KMU is first started, all toolbar functions are initially disabled. The user must first select a token from the “Select a token” dropdown box, which will list all available tokens. Initialized tokens are displayed by their assigned label name. Uninitialized tokens are displayed as “<Slot n>:<uninitialized token>”. Note that the KMU is unable to initialize tokens. There are other utilities, such as GCTADMIN, which can be used to initialize tokens.



Once a token has been selected the user will be given the option to login. After the PIN is successfully authenticated, a list of keys and other objects contained within the token is displayed in the “Objects on Selected Token” box. Appropriate buttons on the toolbar will now also be enabled as shown on the picture.

Eracom ProtectToolkit C – examples of API functions calls

Development of new applications that use ProtectToolkit C API functions requires the library `cryptoki.dll` and for some extra API functions also libraries `ctextra.dll` and `ctutil.dll`. Hence, the corresponding header files must be included in the C/C++ source file and some minor changes in linking options must be done.

```
//PROTECTTOOLKIT C HEADER FILES
#include<cryptoki.h>
#include<ctextra.h>
#include<ctutil.h>
```

The slot number of a particular user must be known for session initialization and a PIN for user authentication.

```
//VARIABLES FOR SESSION INICIALIZATION
CK_SESSION_HANDLE hSession;                 //Session handle
CK_CHAR userPIN[]={"1234"};                 //User PIN
CK_SLOT_ID slotID=0;                        //Number of slot

//VARIABLES FOR ERROR HANDLING
CK_RV rv;                                   //Return value
CK_BYTE ErrorString[100]="\0";              //Error string
```

The initialization of Cryptoki library is done by calling API function `C_Initialize`.

```
//INITIALIZATION -- this must be the first PKCS #11 call
rv = C_Initialize(NULL_PTR);
if ( rv ) {
       C_ErrorString(rv,ErrorString,sizeof(ErrorString));
       fprintf(stderr, "C_Initialize error %x, %s\n", rv, ErrorString);
       return 1;
}

// CHECK PKCS#11 VERSION
rv = CheckCryptokiVersion();
if ( rv ) {
printf( "Incompatible PKCS#11 version (0x%x)\n", rv );
return -1;
}
```

The session opening is done by calling API function `C_OpenSession`.

```
//OPEN SESSION
rv = C_OpenSession(slotID, CKF_RW_SESSION|CKF_SERIAL_SESSION, NULL, NULL, &hSession);
if ( rv ) return 1;
```

At this point the user/admin login can be performed by calling API function `C_Login`. After successful login, all (i.e., not only public) objects of a particular user or admin are accessible.

```
//USER LOGIN
rv = C_Login(hSession, CKU_USER, userPIN, sizeof(userPIN)-1);
if ( rv == CKR_OK) {
       //.
       //THE CORE OF APPLICATION
       //.
       rv = C_Logout(hSession);
} else {
       printf("Login failed - error %d\n", rv);
}
```

These securely stored objects (e.g., cryptographic keys) can be subsequently used in the core of the application. API function `C_Logout` performs logout of a particular user or admin.

For example, decrypting of ciphertext (by using AES in ECB mode) with the key located inside HSM can be done by using the API functions `C_DecryptInit` and `C_Decrypt`. The first function initializes a decryption operation (hSession is the session's handle; pMechanism points to the decryption mechanism; hKey is the handle of the decryption key) and the second function decrypts encrypted data in a single part.

```
//VARIABLES AND STRUCTURES FOR DECRYPTING
CK_BYTE_PTR p3;
CK_MECHANISM pMechanism = {                 //Structure for cipher mechanism
       CKM_AES_ECB, NULL_PTR, 0
}
CK_BYTE ciphertext[CT_SIZE];
CK_ULONG plaintextdatalen;

//DECRYPTING
rv = C_DecryptInit(hSession, &pMechanism, hMKey);
rv = C_Decrypt(hSession, ciphertext, CT_SIZE, NULL_PTR, &plaintextdatalen);
p3 = (CK_BYTE_PTR) malloc(plaintextdatalen);
rv = C_Decrypt(hSession, ciphertext, CT_SIZE, p3, &plaintextdatalen);
```

API function `C_CloseSession` closes a session between an application and a token and API function `C_Finalize` is called to indicate that an application is finished with the Cryptoki library (it should be the last Cryptoki call made by an application).

```
//CLOSE SESSION
C_CloseSession(hSession);

rv = C_Finalize(NULL_PTR);
if ( rv ) {
       C_ErrorString(rv,ErrorString,sizeof(ErrorString));
       fprintf(stderr, "C_Finalize error %x, %s\n", rv, ErrorString);
}
```

Assignments

1) Use Key Management Utility to create a 256bit AES user key with name/label "LaBAK". This key should not be marked as a sensitive. Then use Cryptoki Token Browser to get the hex value of the key and store this value to the ASCII text file. [2 points]

2) Create a short program that perform login to the software emulator of Eracom HSM (use both, token "labak"/"LABAK"/"LaBAK" and user password "1111" created/initialised on the seminar). [3 points]

3) Extend the previous program to be capable of encrypting by your AES key "LaBAK". You must first get the handle of the key and then perform an encryption with this key (similarly as sketched the decryption in the example above). Then encrypt the message "Everyone Loves Hypno Toad" (for details about this strange animal from old good Futurama see http://elht.ytmnd.com/ :-). [5 points]