Gemalto .NET Smart Cards PKCS#11 Library for Linux

User Guide

gemalto
security to be free

GEMALTO, B.P. 100, 13881 GEMENOS CEDEX, FRANCE.

Tel: +33 (0)4.42.36.50.00 Fax: +33 (0)4.42.36.50.90

Printed in France.

Document Reference: DOC117584D

January 29, 2010

www.gemalto.com

security to be free

Contents

List of Figures

List of Tables

This document introduces you to the PKCS#11 Library for Gemalto .NET Smart Cards and provides information about the installation, use and integration of this library.

Who Should Read This Book

This guide is intended for system integrators who want to integrate the software with other applications and for end-users.

It is assumed that users are familiar with .NET smart cards/tokens and smart card reader technology, as well as computer hardware and software.

It is assumed that the user of PKCS#11 for .NET Smart Cards has:

- an understanding of the basic operations in a computer OS.
- administrative privileges for the computer on which PKCS#11 for .NET Smart Cards will be installed.

Documentation

The PKCS#11 Library for Gemalto .NET Smart Cards is delivered with the following documentation:

- *Gemalto .NET Smart Cards PKCS#11 Library for Linux User Guide* (this document) - PKCS#11_Libs_.NET__Linux_User_Guide.pdf.
- A *Release Notes* file. This contains any relevant information about the installation and the complete version history.

Both documents are located in

- /usr/share/doc/libgemaltodotnetp11/ (Ubuntu)
- /usr/share/doc/libgtop11dotnet-2.1.0/ (RedHat)
- /usr/share/doc/packages/libgtop11dotnet/ (SUSE)
- /usr/share/doc/libgemaltodotnetp11/ (Debian)

Conventions

The following conventions are used in this document:

Typographical Conventions

PKCS#11 for .NET Smart Cards documentation uses the following typographical conventions to assist the reader of this document.

Convention	Example	Description
`Courier`	`transaction`	Code examples.
Bold	Type **myscript.dll**	Actual user input or screen output.
>	Select File > Open	Indicates a menu selection. In this example you are instructed to select the **“Open**” option from the “**File”** menu.

Note: Example screen shots of the software are provided throughout this document to illustrate the various procedures and descriptions. These screen shots were produced with the PKCS#11 Library for Gemalto .NET Smart Cards running on Ubuntu 7.10.

Additional Resources

For further information or more detailed use of the PKCS#11 Library for Gemalto .NET Smart Cards, additional resources and documentation are available on the following web site:

www.gemalto.com/products/dotnet_card

Contact Our Hotline

If you do not find the information you need in this manual, or if you find errors, contact the Gemalto hotline at http://support.gemalto.com/.

Please note the document reference number, your job function, and the name of your company. (You will find the document reference number at the bottom of the legal notice on the inside front cover.)

Overview

Welcome to the PKCS#11 Library for Gemalto .NET Smart Cards. You have made a wise investment by purchasing this product as a safeguard for secure network services.

.NET Smart Cards

Gemalto .NET is the first ever implementation of a .NET Framework for Smart Cards. It puts state of the art technology to the service of organizations committed to take their IT Security and Identity & Access infrastructure to the next level.

Two-factor authentication (2FA) solutions can help secure your company's digital assets from end to end. Gemalto .NET comes equipped with support for 2 different 2FA technologies: One Time Passwords (OTP) and X509 Certificates (PKI). Choose the one that suits you best, or combine both at once for different uses.

With Gemalto .NET you will also benefit from unparalleled level of integration with Microsoft's platforms and solutions: Support for the Gemalto .NET Smart Cards and Tokens is built into Windows Vista and Windows Server 2008, and available as a Windows Update for Windows XP and Server 2003. Gemalto .NET is also fully compatible with Forefront Edge, Microsoft's Identity Lifecycle Manager, Active Directory Domain Services and Certificate Services. With Gemalto .NET implementation of Two Factor Authentication, Encryption and Digital Signature services becomes easier than ever.

Gemalto .NET cards empower developers to build services that take advantage of the enhanced programming and communication capabilities of the .NET Framework and the advanced security and cryptographic services that are the foundation of Gemalto Smart Cards. Combined with the award winning SConnect technology, Smart Cards and Tokens can now communicate with all kind of Web Services, and hefty client based solutions can be replaced with zero footprint web based solutions.

.NET smart cards incorporate a .NET framework in a smart card. In fact the smart card can take several different forms:

- traditional smart card
- SIM plug
- converged badge
- connected and unconnected USB token

Cryptoki

Cryptoki is an application programming interface (API) with devices that hold cryptographic information and perform cryptographic functions. It is specified in the RSA standard *PKCS#11 v2.20: Cryptographic Token Interface Standard*.

The specification of the Cryptographic Token Interface Standard (PKCS #11) is available at http://www.rsa.com/rsalabs/node.asp?id=2133.

The PKCS#11 library is Gemalto's implementation of Cryptoki, that grants cryptographic applications access to the .NET smart card. The interface is compliant with a coherent subset of the *PKCS#11 v2.20* standard.

PKCS#11 does not implement the entire Cryptoki standard. For example, the Gemalto .NET smart card cannot perform direct symmetric key encryption operations such as the Data Encryption Standard (DES) and Rivest's Cipher (RC2). For a full list of the methods that are supported, please refer to "PKCS#11 Methods Supported" on page 5.

Note: The PKCS#11 library is fully compliant with *PKCS#11 v2.10* and partially with *PKCS#11 v2.20*.

PKCS#11 Library

This library is a cryptographic library that manages simple access to corporate networks while maintaining the highest level of security. It is for individual users, who want to use .NET smart cards and compatible PC/SC card readers to protect information and transactions made via computers, including stand-alone workstations and Citrix client-server environments.

By default, cryptographic support for .NET cards is provided by Base CSP (Microsoft's default software library). Applications that support CSP architecture, such as Microsoft Word, only need the CSP mini-driver that comes automatically with Microsoft Vista and is available as a Windows Utility for Windows XP. They do not need this PKCS#11 library. Digital certificates are stored on smart cards according to the CSP architecture.

The PKCS#11 library is an extension to CSP, to provide cryptographic support to applications (such as Mozilla Firefox) and operating systems (such as Mac and Linux) that do not support a CSP architecture. In this way, the applications can use the digital certificates stored on the card. For more details about which applications use CSP and which need the PKCS#11 library, please refer to "Supported Platforms and Applications" on page 3.

With this PKCS#11 library you can the digital certificates stored on .NET smart cards to:

- Sign electronic documents.
- Open and verify signed documents.
- Send and receive secure e-mail.
- Connect securely with a Web server.
- Authenticate yourself when accessing desktop, network, and Web applications
- Log on to a computer securely.
- Lock and unlock a computer.
- Use secure screen savers

The PKCS#11 library is implemented as a set of C language function calls supplied as a C header file and Dynamic Link Libraries (DLLs).

Function calls are used to build smart card applications that require medium level cryptography, such as digital signatures and secure messaging applications. For example, C_SetPIN allows the card user to change the PIN number of the card in the reader.

Application developers can use PKCS#11 to:

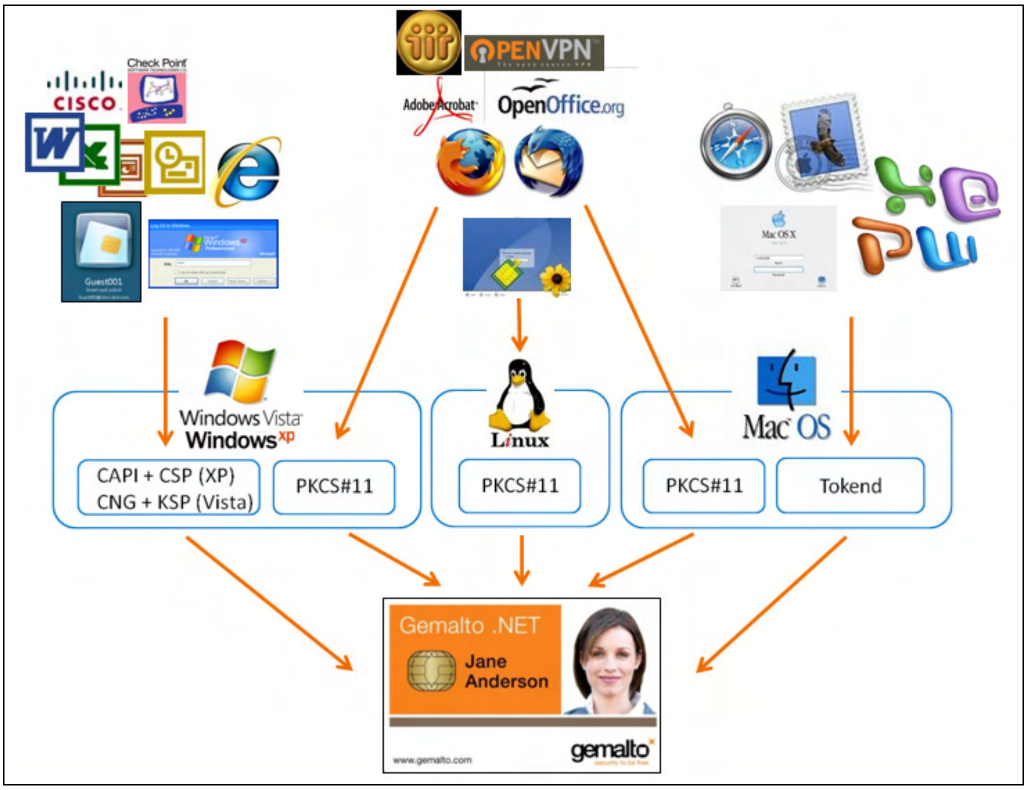
- Load and generate RSA Keys
- Create RSA digital signatures
- Manage PIN codes
- Store certificates
- Store miscellaneous data specific to an application, for example, user profiles and bookmarks
- Generate secure random numbers.

Supported Platforms and Applications

The PKCS#11 library allows .NET smart cards to work on Windows, Mac and Linux operating systems (OS).

“Figure 1” on page 3 shows which cryptographic security modules are used by which applications for these three different OS.

Figure 1 - Smart Card Cryptography Support for Different Platforms and Applications



Windows

Microsoft applications such as Internet Explorer, Outlook, Office, MS VPN, EFS, Windows Logon, ILM use the Microsoft CAPI/Base CSP to provide cryptographic services via Smart Cards. It is also used by a growing number of third party applications, such as Passlogix v-Go., Evidian ESSO, Quest QSSO, Checkpoint & Cisco VPN, Verisign, to name just a few.

The PKCS#11 module used to be the standard CSP and is still used by many major third party applications running in Windows, such as Mozilla Firefox & Thunderbird, Lotus Notes, Adobe Acrobat, Utimaco Safeguard, Evidian, and Avencis.

Gemalto’s PKCS#11 library is the PKCS#11 security module.

Linux

On Linux OS, there is no Microsoft CAPI/CSP, so all applications providing smart card cryptographic services use the PKCS#11 module.

Mac

On Mac OS, there is no Microsoft CAPI/CSP. All native Apple applications (Mac Logon, Safari, mail client and so on) use a Mac proprietary cryptographic architecture called Tokend. This is also used by Microsoft's Office package for Mac OS. Several other third party applications, such as Adobe Acrobat, Mozilla Firefox & Thunderbird use the PKCS#11 security module.

For the Mac version, Gemalto's PKCS#11 library is the PKCS#11 security module + Tokend.

Applications Tested

For information about the applications that have been tested and validated with your version of PKCS#11 for .NET Smart Cards, please refer to the *Release Notes* that accompany it. For the latest information about the applications that have been tested, please refer to the product catalog at www.gemalto.com.

PKCS#11 Specifics

This chapter presents some detailed information about the PKCS#11 library and .NET cards.

Key Sizes Supported

The V2 .NET smart card supports 1024-bit and 2048-bit keys.

The V2+ .NET smart card supports key lengths from 512 bits to 2048 bits in steps of 128 bits (512,640,768,896,1024,1152,1280,1408,1536,1664,1792,1920,2048).

Number of Simultaneous Reader Connections Supported

The PKCS#11 library for .NET cards can manage up to 16 smart card reader connections. Beyond this, a CKR_HOST_MEMORY error occurs when you call the C_GetSlotList function.

PKCS#11 Methods Supported

Not all the PKCS#11 interface methods specified in the PKCS#11 v2.20 are implemented in the PKCS .NET library. The following table lists them all and indicates which ones are not supported.

Table 1 - PKCS#11 Methods Supported

Category	**Function**	**Description**
General purpose functions	C_Initialize	Initializes Cryptoki
	C_Finalize	Cleans up miscellaneous Cryptoki-associated resources
	C_GetInfo	Obtains general information about Cryptoki
	C_GetFunctionList	Obtains entry points of Cryptoki library functions

Table 1 - PKCS#11 Methods Supported (continued)

Category	**Function**	**Description**
Slot and token management functions	C_GetSlotList	Obtains a list of slots in the system
	C_GetSlotInfo	Obtains information about a particular slot
	C_GetTokenInfo	Obtains information about a particular token
	C_WaitForSlotEvent	Waits for a slot event (token insertion, removal, etc.) to occur
	C_GetMechanismList	Obtains a list of mechanisms supported by a token
	C_GetMechanismInfo	Obtains information about a particular mechanism
	C_InitToken	Initializes a token
	C_InitPIN	Initializes the user PIN
	C_SetPIN	Modifies the PIN of the current user
Session management functions	C_OpenSession	Opens a connect ion between an application and a particular token or sets up an application callback for token insertion
	C_CloseSession	Closes a session
	C_CloseAllSessions	Closes all sessions with a token
	C_GetSessionInfo	Obtains information about the session
	C_GetOperationState	NOT SUPPORTED
	C_SetOperationState	NOT SUPPORTED
	C_Login	Logs into a token
	C_Logout	Logs out from a token
Object management functions	C_CreateObject	Creates an object
	C_CopyObject	NOT SUPPORTED
	C_DestroyObject	Destroys an object
	C_GetObjectSize	Obtains the size of an object in bytes
	C_GetAttributeValue	Obtains an attribute value of an object
	C_SetAttributeValue	Modifies an attribute value of an object
	C_FindObjectsInit	Initializes an object search operation
	C_FindObjects	Continues an object search operation
	C_FindObjectsFinal	Finishes an object search operation
Encryption functions	C_EncryptInit	Initializes an encryption operation
	C_Encrypt	Encrypts single-part data
	C_EncryptUpdate	Continues a multiple-part encryption operation
	C_EncryptFinal	Finishes a multiple-part encryption operation

Table 1 - PKCS#11 Methods Supported (continued)

Category	**Function**	**Description**
Decryption functions	C_DecryptInit	Initializes a decryption operation
	C_Decrypt	Decrypts single-part encrypted data
	C_DecryptUpdate	Continues a multiple-part decryption operation
	C_DecryptFinal	Finishes a multiple-part decryption operation
Message digesting functions	C_DigestInit	Initializes a message-digesting operation
	C_Digest	Digests single-part data
	C_DigestUpdate	Continues a multiple-part digesting operation
	C_DigestKey	NOT SUPPORTED
	C_DigestFinal	Finishes a multiple-part digesting operation
Signing and MACing functions	C_SignInit	Initializes a signature operation
	C_Sign	Signs single-part data
	C_SignUpdate	Continues a multiple-part signature operation
	C_SignFinal	Finishes a multiple-part signature operation
	C_SignRecoverInit	NOT SUPPORTED
	C_SignRecover	NOT SUPPORTED
Functions for verifying signatures and MACs	C_VerifyInit	Initializes a verification operation
	C_Verify	Verifies a signature on single-part data
	C_VerifyUpdate	Continues a multiple-part verification operation
	C_VerifyFinal	Finishes a multiple-part verification operation
	C_VerifyRecoverInit	NOT SUPPORTED
	C_VerifyRecover	NOT SUPPORTED
Dual-purpose cryptographic functions	C_DigestEncryptUpdate	NOT SUPPORTED
	C_DecryptDigestUpdate	NOT SUPPORTED
	C_SignEncryptUpdate	NOT SUPPORTED
	C_DecryptVerifyUpdate	NOT SUPPORTED
Key management functions	C_GenerateKey	Generates a secret key
	C_GenerateKeyPair	Generates a public-key/private-key pair
	C_WrapKey	NOT SUPPORTED
	C_UnwrapKey	NOT SUPPORTED
	C_DeriveKey	NOT SUPPORTED
Random number generation functions	C_SeedRandom	Mixes in additional seed material to the random number generator
	C_GenerateRandom	Generates random data

Table 1 - PKCS#11 Methods Supported (continued)

Category	Function	Description
Parallel function management functions	C_GetFunctionStatus	NOT SUPPORTED
	C_CancelFunction	NOT SUPPORTED
Callback function		NOT SUPPORTED

Accessing Objects According to Session Type

Cryptoki requires that an application open one or more sessions with a token to gain access to the token's objects and functions. A session provides a logical connection between the application and the token.

Session Types

A session can be a read/write (R/W) session or a read-only (R/O) session. Read/write and read-only refer to the access to token objects, not to session objects (see “Object Types”).

Sessions where no user has authenticated him or herself with the device are referred to as public sessions (R/O Public or R/W Public). Once the user who owns the token authenticates him/herself with the token, the session is referred to as a user session (R/O User or R/W User). Sessions where the Security Officer has authenticated him/herself with the token are referred to as R/W SO sessions (R/O SO sessions are not possible).

Cryptoki supports multiple sessions on multiple tokens. An application may have one or more sessions with one or more tokens. In general, a token may have multiple sessions with one or more applications. A particular token may allow an application to have only a limited number of sessions-or only a limited number of read/write sessions-- however.

Object Types

Objects that reside on the token are referred to as token objects. Objects that exist only for the duration of a session are referred to as session objects.

When a session is closed, any session objects which were created in that session are destroyed. This holds even for session objects which are “being used” by other sessions. That is, if a single application has multiple sessions open with a token, and it uses one of them to create a session object, then that session object is visible through any of that application's sessions. However, as soon as the session that was used to create the object is closed, that object is destroyed.

Authentication

In public sessions, an application has R/O access to all public objects (token and session). After it opens a session, an application has access to the token's public objects. All threads of a given application have access to exactly the same sessions and the same session objects. No private objects can be accessed.

If the Security Officer authenticates him/herself, the application has Read/Write access to all public objects (token and session). Private objects still cannot be accessed.

To access private objects (token and session), the normal user must log in and be authenticated.

Note: Creating or deleting an object requires read/write access to it, for example, a “R/O User Functions” session cannot create or delete a token object. Creating or deleting an object is not allowed from any Public session. The user must be logged to create or delete public and private token objects.

The following table summarizes the kind of access each type of session has to each type of object.

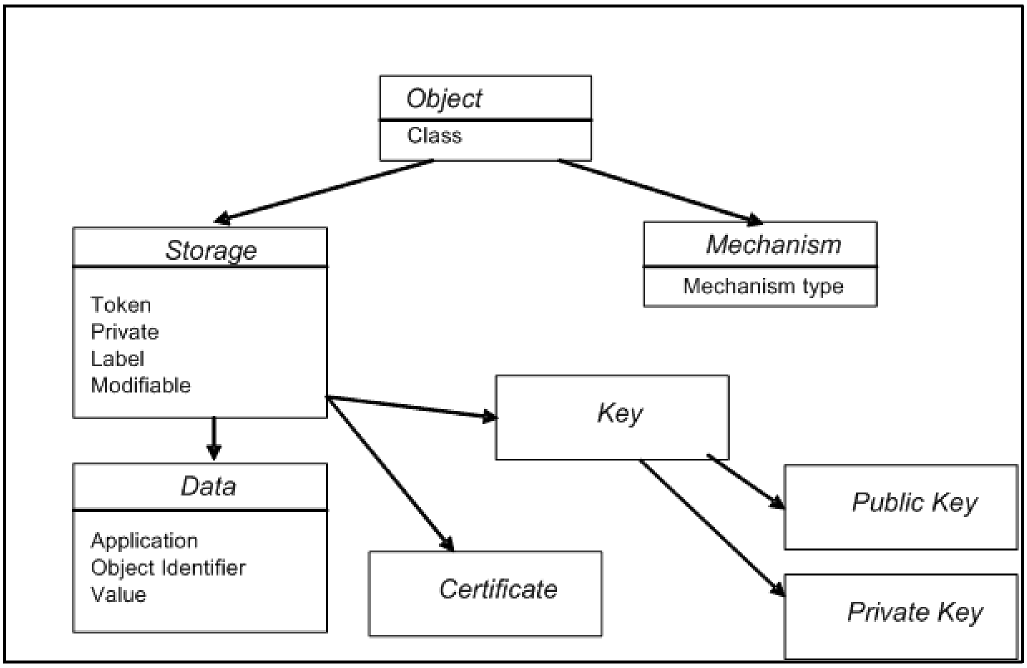
Table 2 - Access to Objects According to Session Type

	Type of session				
Type of object	**R/O Public**	**R/W Public**	**R/O User**	**R/W User**	**R/W SO**
Public session object	R/O	R/O	R/W	R/W	R/W
Private session object			R/W	R/W	
Public token object	R/O	R/O	R/O	R/W	R/W
Private token object			R/O	R/W	

Supported PKCS#11 Objects and Attributes

Cryptoki recognizes a number of classes of objects, as defined in the CK_OBJECT_CLASS data type. An object consists of a set of attributes, each of which has a given value. Each attribute that an object possesses has precisely one value. The following figure illustrates the high-level hierarchy of the Cryptoki objects and some of the attributes they support:

Figure 2 - Supported PKCS#11 Objects and Attributes



The .NET PKCS#11 library does not support **Hardware Feature** objects and **Domain Parameters** objects. Theses objects were introduced in the 2.20 version of the RSA standard. The .NET PKCS#11 library only deals with the object hierarchy specified in the 2.10 version.

The .NET PKCS#11 library does not support any kind of Secret Key object.

Regarding **Certificates**, only X.509 public key certificate objects (certificate type CKC_X_509) are supported.

Regarding the attributes of the objects, several new attributes were introduced in the 2.20 version of the RSA standard. The .NET PKCS#11 library only deals with the attributes specified in the 2.10 version.

Regarding **Private Key** objects, the value of the key is never extractable.

Cryptographic Mechanisms Supported

Cryptographic Algorithms

The PKCS#11 supports the RSA algorithm only. It supports the following mechanism types defined in PKCS#11 v2.20.

- CKM_RSA_PKCS for sign, verify, encrypt and decrypt operations
- CKM_RSA_X_509 for sign, verify, encrypt and decrypt operations
- CKM_SHA256_RSA_PKCS for sign and verify operations
- CKM_MD5_RSA_PKCS for sign and verify operations
- CKM_SHA1_RSA_PKCS for sign and verify operations

Symmetric keys are not supported.

Hash Algorithms

The PKCS#11 library supports the MD5, SHA-1 and SHA-256 hash algorithms, that is, the following mechanism types defined in PKCS#11 v2.20.

- CKM_MD5
- CKM_SHA_1
- CKM_SHA256

Reading the Card Serial Number

The 12-byte .NET card serial number (CSN) can be read either by using Gemalto's .NET utilities or via the .NET PKCS#11 library, however the two interpret the CSN differently.

.NET Utilities

.NET utilities reads the CSN directly from the card and returns it as 24 digits as defined in the Microsoft Minidriver specification. The minidriver also returns a GUID property (or card ID file content). This GUID is displayed by .NET utilities on 16-bytes as:

4-byte random number || CSN (12 bytes)

For more information about .NET utilities go to:

http://www.netsolutions.gemalto.com/utilities.aspx

.NET PKCS#11

The .NET PKCS#11 library performs an MD5 hash of the CSN provided by the minidriver giving a result of 16-bytes. As the CK_TOKEN_INFO string is only 16 bytes it can display 16 characters of the hash result only in ASCII format. These 16 characters are the 8 MSB (leftmost bytes) of the hash result.

Example:

1. The .NET PKCS#11 library requests the 12-byte Card Serial Number (CSN) and the card returns:

   0x57 0x01 0x13 0x51 0x26 0xC7 0xD6 0x10 0x29 0x27 0xFF 0xFF
2. .NET PKCS#11 performs an MD5 hash on the CSN giving a 16-byte result:

   0x05 0xCB 0x00 0x3D 0x76 0xD3 0xE9 0x4F 0x74 0x13 0xD8 0x74 0x38 0x8C 0xBF 0xB4
3. The .NET PKCS#11 transforms the hash into an ASCII string.
4. Finally it fills the serialNumber field of the TokenInfo structure (on 16 bytes) with the first 16 characters of the ASCII string: “05CB003D76D3E94F” corresponding to the 8 MSB of the hash.

Product Limitations

The PKCS#11 library can manage up to 15 key pairs. There is no limit for the number of data objects.

Nevertheless, the real limitation of the PKCS#11 library is the amount of free memory available on the card. This amount can vary from card to card depending on the number of applications present in the card.

Performances

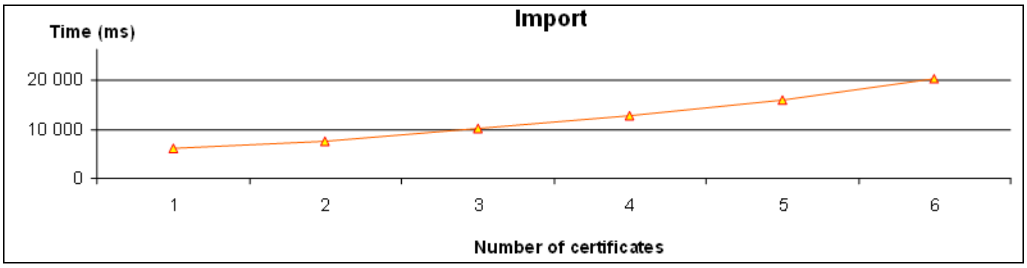
This section shows the results of tests performed in the Gemalto test laboratory on the latest Gemalto .NET cards (V2+) for three common use cases of the PKCS#11 library.

Import Operation

For this test, a 2048-bit certificate was imported into the blank smart card. The operation was repeated until the smart card was full.

“Figure 3” shows the time spent by the PKCS#11 library to create the certificate and the key pair PKCS#11 objects in the card against the number of certificates present in the card.

Figure 3 - Import Operation Test Results

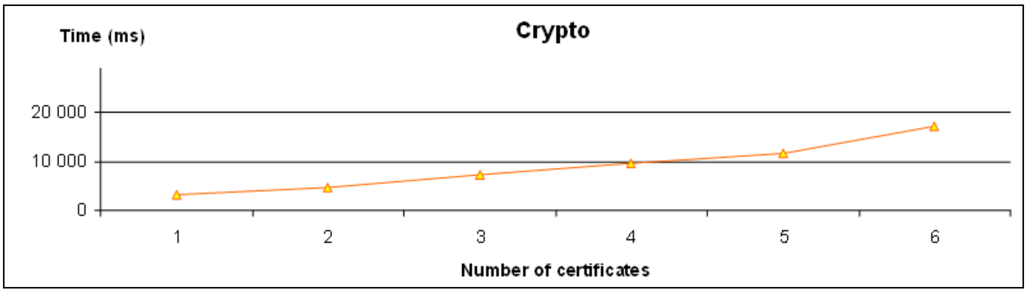


Signature Operation

For this test, a smart card with a certificate and 2048-bit key pair performed a signature and the time taken was recorded. Further certificates and their key pair were imported in the card until the card was full, a signature being performed and timed after each import.

“Figure 4” shows the time spent by the PKCS#11 library for each signature against the number of certificates present in the card.

Figure 4 - Signature Operation Test Results

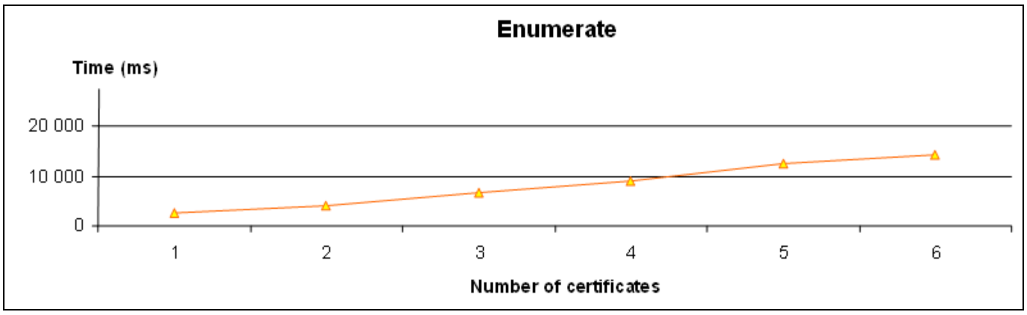


Find All Objects Operation

For this test, a smart card with a certificate and 2048-bit key pair performed a **Find All Objects** operation and the time taken was recorded. Further certificates and their key pair were imported in the card until the card was full, a **Find All Objects** operation being performed and timed after each import.

“Figure 5” shows the time spent by the PKCS#11 library for each **Find All Objects** operation against the number of certificates present in the card.

Figure 5 - Find All Objects Operation Test Results



The Security Officer PIN

The PKCS#11 specification defines the Security Officer (SO) PIN as the PIN that unblocks a user’s PIN. It is also often known as the Administrator’s PIN or the Unblock PIN (PUK).

The concept of an SO PIN does not exist in the .NET smart card specification. Instead, the specification defines a 24-byte administrator key. During mutual authentication between the middleware on the computer and the .NET smart card, the middleware asks for a 16-byte random challenge from the .NET card. It then computes a 16-byte response using the 24-byte Administration key and sends this response to the card. The card compares the response with the response it has calculated itself and if they agree, authenticates the middleware.

For this reason and to be compliant with the PKCS#11 specification, the SO PIN is the same key as the 24-byte administrator key. The .NET PKCS#11 library manages the challenge-response exchange using this key to perform all cryptographic operations requiring the SO PIN.

When blank Gemalto .NET smart cards are delivered, this value is 24 bytes with the value 0x00.

The following sample code, provides an example of how you can call the C_InitToken Method using the SO PIN.

```
#include <memory.h>
#include <stdio.h>
#include <string>

#include "cryptoki.h"


int main( int argc, char* argv[ ] )
{
   // Initialize the cryptoki
   CK_RV rv = C_Initialize( NULL_PTR );
   if( CKR_OK != rv )
   {
      printf( "\n## ERROR - C_Initialize failed <%#02x> ##\n", rv );
      printf( "Press enter to exit..." );
      getchar( );
      return 1;
   }
   printf( "\n== Cryptoki initialized ==\n" );

   // Display the cryptoki information
   CK_INFO info;
   memset( &info, 0, sizeof( CK_INFO ) );
   rv = C_GetInfo( &info );
   if ( CKR_OK != rv )
   {
      printf( "\n## ERROR - C_GetInfo failed <%#02x> ##\n", rv );
      printf( "Press enter to exit..." );
      getchar( );
      return 1;
   }
   printf( "\n== Cryptoki Information ==\n" );
   printf( "C_GetInfo - cryptokiVersion <%d.%d>\n",
info.cryptokiVersion.major, info.cryptokiVersion.minor );
   printf( "C_GetInfo - manufacturerID <%.*s>\n", 32, info.manufacturerID
);
   printf( "C_GetInfo - flags <%ld>\n", info.flags );
   printf( "C_GetInfo - libraryDescription <%.*s>\n", 32,
info.libraryDescription );
   printf( "C_GetInfo - libraryVersion <%d.%d>\n\n",
info.libraryVersion.major, info.libraryVersion.minor );

   // Retreive all the available slots (reader with smartcard inside)
   CK_SLOT_ID aSlotList[ 10 ];
   memset( aSlotList, 0, sizeof( aSlotList ) );
   CK_ULONG ulCount = sizeof( aSlotList ) / sizeof( CK_SLOT_ID );
   rv = C_GetSlotList( TRUE, aSlotList, &ulCount );
   if ( CKR_OK != rv )
   {
      printf( "\n## ERROR - C_GetSlotList failed <%#02x> ##\n", rv );
      printf( "Press enter to exit..." );
      getchar( );
      return 1;
   }
   if( 0 == ulCount )
   {
      printf( "\n## ERROR - No slot available. Insert a smartcard into a 
reader. ##\n" );
      printf( "Press enter to exit..." );
      getchar( );
      return 1;
   }

   printf( "\n== Slot Information ==\n" );
   // Display the slot information
   for( size_t i = 0 ; i < ulCount ; i++ )
   {
      CK_SLOT_INFO slotInfo;
      memset( &slotInfo, 0, sizeof( CK_SLOT_INFO ) );
      rv = C_GetSlotInfo( aSlotList[ i ], &slotInfo );
      if ( CKR_OK == rv )
      {
         printf( "slot[ %d ] - slotDescription <%.*s>\n", aSlotList[ i ], 
64, slotInfo.slotDescription );
        printf( "slot[ %d ] - manufacturerID <%.*s>\n", aSlotList[ i ], 32, 
slotInfo.manufacturerID );
         printf( "slot[ %d ] - flags <%ld>\n", aSlotList[ i ], 
slotInfo.flags );
         printf( "slot[ %d ] - hardwareVersion <%d.%d>\n", aSlotList[ i ], 
slotInfo.hardwareVersion.major, slotInfo.hardwareVersion.minor );
        printf( "slot[ %d ] - firmwareVersion <%d.%d>\n\n", aSlotList[ i ], 
slotInfo.firmwareVersion.major, slotInfo.firmwareVersion.minor );
      }
   }

   // Take the first available slot
   CK_SLOT_ID slotId = aSlotList[ 0 ];

   // Prepare the ADMIN key as PKCS#11 PIN SO
   CK_CHAR aPinSo[ 24 ] = {   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 
0x00, 0x00, 0x00, 
                            0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 
0x00, 0x00, 
                              0x00, 0x00, 0x00, 0x00 };
   CK_ULONG ulPinSoLen = (CK_ULONG) sizeof( aPinSo );

   // Prepare the new token label
   std::string s = "My label for the token";
   CK_CHAR aLabel[ 32 ];
   memset( aLabel, ' ', sizeof( aLabel ) );
   size_t l = ( sizeof( aLabel ) <= s.length( ) ) ? sizeof( aLabel ) : 
s.length( );
   memcpy( aLabel, s.c_str( ), l );

   // Init token
   rv = C_InitToken( slotId, aPinSo, ulPinSoLen, aLabel );
   if ( CKR_OK != rv )
   {
      printf( "\n## ERROR - C_GetSlotList failed <%#02x> ##\n", rv );
      printf( "Press enter to exit..." );
      getchar( );
      return 1;
   }
   printf( "\n== Token initialized ==\n" );

   // Release the cryptoki
   rv = C_Finalize( NULL_PTR );
   if ( CKR_OK != rv )
   {
      printf( "\n## ERROR - C_Finalize failed <%#02x> ##\n", rv );
      printf( "Press enter to exit..." );
      getchar( );
      return 1;
   }
   printf( "\n== Cryptoki released ==\n" );

   printf( "Press enter to exit..." );
   getchar( );

   return 0;
}
```

Installation

This chapter discusses information related to the installation of the PKCS#11 Library for Gemalto .NET Smart Cards, such as:

- The hardware and software you need to use the library.
- How to install the library on your computer.

System Requirements

The following sections describe the hardware, operating systems, peripherals and software you need to use the PKCS#11 Library for Gemalto .NET Smart Cards. You must have administrator rights to the computer on which you are installing the library.

Computer

The workstation must meet the normal system requirements to run the version of the OS installed.

Operating Systems

For a list of the operating systems supported by PKCS#11 for .NET Smart Cards, please refer to the *Release Notes*.

Peripherals

PKCS#11 for .NET Smart Cards requires the following peripherals:

- A PC/SC compatible reader. Depending on the type of reader, you may also need a USB port on the computer to connect it.

You can find details about Gemalto’s smart card readers at

http://store.gemalto.com/is-bin/INTERSHOP.enfinity/eCS/Store/en/-/EUR/BrowseCatalog-Start;sid=Kc5AIHjNFzlAMTGo27iAiB3JIxliIw71FOw=?CatalogCategoryID=ovcKBQCcLWUAAAD0%2ei6_1fgy&BuyerClass=

Smart Cards

The following types of .NET smart card are supported:

- .NET V2
- .NET V2+
- All devices containing any of the above 2 smart cards:
  - .NET Key
  - .NET Dual
  - Smart Enterprise Guardian (SEG)

Note: Although .NET v2.0 smart cards are supported, the PKCS#11 library for Gemalto.NET has been designed to take advantage of the latest enhancements introduced in the .NET v2+ Smart Card OS. Therefore, Gemalto highly recommends you use .NET v2+ smart cards and devices for PKCS#11.

Installing PKCS#11 for .NET Smart Cards

Pre-requisites

Smart Card Reader Driver

The smart card reader driver must be installed on the system. Download the latest version from your smart card reader vendor web site. If your smart card reader is a Gemalto smart card reader, you can download its corresponding driver from support.gemalto.com.

Caution: This driver is essential, otherwise PKCS#11 will not work on your computer.

PC/SC

The PC/SC layer must be installed on your system. The versions of Linux supported by the library all contain a PC/SC Lite package that may have been installed on the computer as part of the Linux installation. If it was not, a message appears during the installation of the .NET PKCS#11 library telling you this. In this case, install the PC/SC Lite package using Linux’s installation program. If this does not work, download PC/SC Lite from your operating system vendor web site as follows:

  - For Ubuntu: www.ubuntu.com
  - SUSE: www.novell.com/linux/
  - Debian: www.debian.org
  - RedHat: www.redhat.com

Installing the PKCS#11 for .NET Smart Cards Software

Caution: Before installing the software, make sure that your system has the latest version of the PC/SC Lite and CCID drivers.

To install PKCS#11 for .NET Smart Cards:

1 Navigate to the installation program on the network and download the installation file to your computer. This file is different according to the version of Linux you are using.

You can find this at http://www.gemalto.com/products/dotnet_card/resources/libraries.html?toggler=0

The following table shows the name of the file for each OS and the location where you should download the file:

Table 3 - Installation File Names

OS	**Filename**
Ubuntu	libgemaltodotnetp11_2.1.0-1ubuntu1_i386.deb
RedHat	libgtop11dotnet-2.1.0-rhel5.1.i386.rpm
SUSE	libgtop11dotnet-2.1.0-sled10.1.i586.rpm
Debian	libgemaltodotnetp11_2.1.0-1debian_i386.deb

For all four versions of Linux, the PKCS#11 library is installed as “/usr/lib/pkcs11/libgtop11dotnet.so”.

2 Do one of the following:

- Open a terminal and go to the directory where your installation program is. Start the installation program by typing the following line at the command prompt according to your OS:

  In Ubuntu, type DPKG -i libgemaltodotnetp11_x.x.x-1ubuntu1_i386.deb

  In RedHat, type RPM -i libgtop11dotnet-x.x.x-rhel5.1.i386.rpm

  In SUSE, type RPM -i libgtop11dotnet-x.x.x-sled10.1.i586.rpm

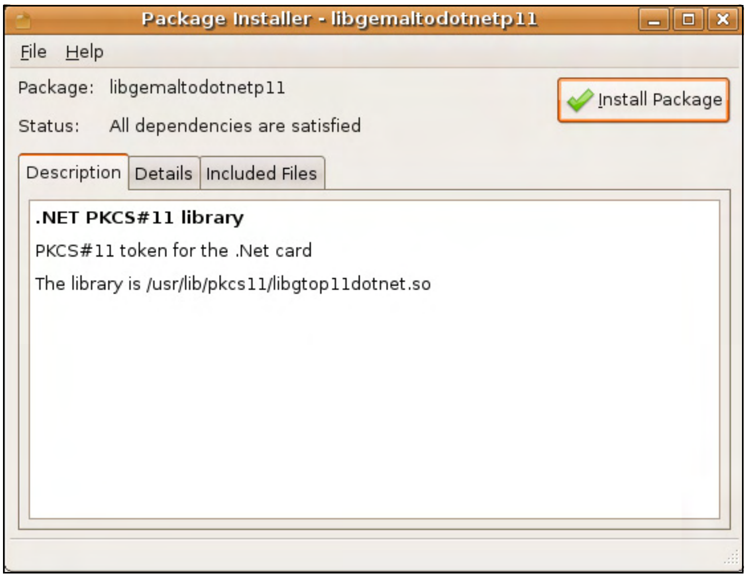
  In Debian, type DPKG -i libgemaltodotnetp11_x.x.x-1debian_i386.deb

  Note: You can do this without going to the installation directory first, but if you do, you must type the full path for the file name.

- Navigate to the appropriate file in “Table 3” and double-click.

3 The package installer window displays as shown in “Figure 6”.

Figure 6 - Package Installer Window



Package: libgemaltodotnetp11

Status: All dependencies are satisfied

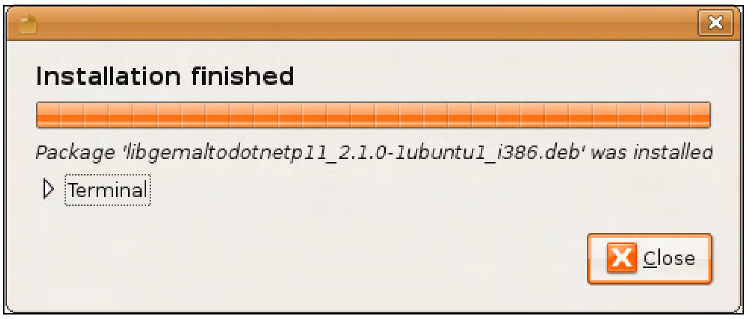
.NET PKCS#11 library

PKCS#11 token for the .Net card

The library is /usr/lib/pkcs11/libgtop11dotnet.so

4 Click **Install Package**. A progress bar displays during the installation. Wait until the **Installation finished** message appears as shown in “Figure 7”.

Figure 7 - Installation Finished Message



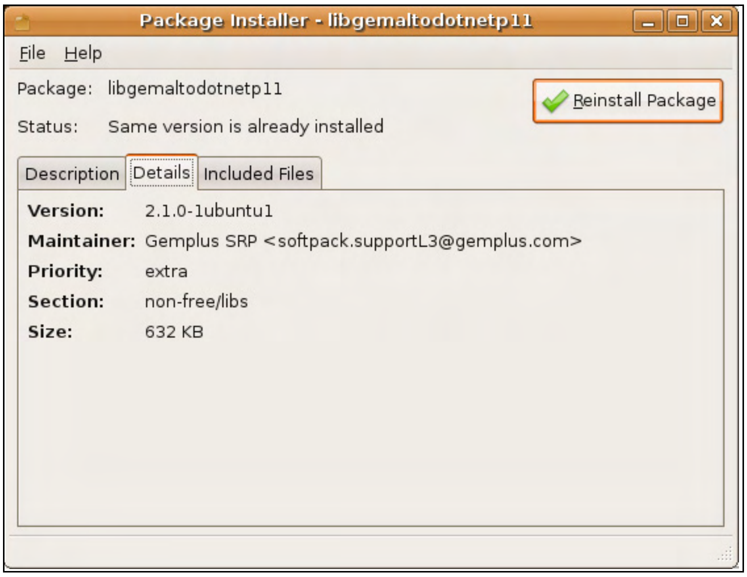
Installation finished

Package 'libgemaltodotnetp11_2.1.0-1ubuntu1_i386.deb' was installed

5 Click Close.

Back in the **Package Installer** window (“Figure 6”) you can click **Details** for more information about the installation package as shown in “Figure 8”.

Figure 8 - Package Installer - Details Tab



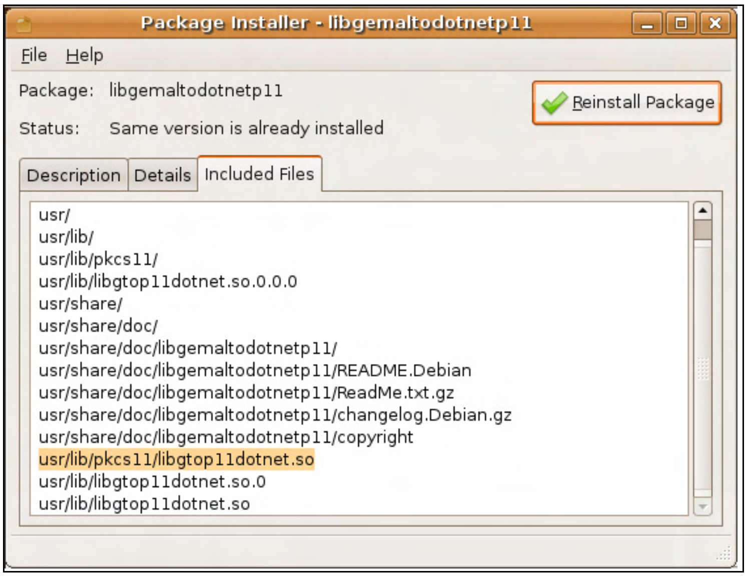
Package: libgemaltodotnetp11

Status: Same version is already installed

Version: 2.1.0-1ubuntu1
Maintainer: Gemplus SRP <softpack.supportL3@gemplus.com>
Priority: extra
Section: non-free/libs
Size: 632 KB

Similarly, the **Included Files** tab, displays all the files and their locations that are installed as part of the package, as shown in “Figure 9”.

Figure 9 - Package Installer - Included Files Tab



Package: libgemaltodotnetp11

Status: Same version is already installed

```
usr/
usr/lib/
usr/lib/pkcs11/
usr/lib/libgtop11dotnet.so.0.0.0
usr/share/
usr/share/doc/
usr/share/doc/libgemaltodotnetp11/
usr/share/doc/libgemaltodotnetp11/README.Debian
usr/share/doc/libgemaltodotnetp11/ReadMe.txt.gz
usr/share/doc/libgemaltodotnetp11/changelog.Debian.gz
usr/share/doc/libgemaltodotnetp11/copyright
usr/lib/pkcs11/libgtop11dotnet.so
usr/lib/libgtop11dotnet.so.0
usr/lib/libgtop11dotnet.so
```

Configuring the PKCS#11 Security Module

Security Modules are software add-ons that provide a variety of cryptographic services, such as secure browsing, and support the use of smart cards/tokens. The PKCS#11 Library needs to be configured as a security module in the Mozilla applications Firefox (browser) and Thunderbird (e-mail), so that they can communicate with the .NET smart card. For details on how to do this, see “Appendix C - Configuring PKCS#11 in Mozilla”.

4

Tasks

This chapter discusses information related to specific tasks that you will most often be required to carry out when using the PKCS#11 for .NET Smart Cards software and where to find the information about them.

These tasks are:

- “How to Get a Certificate” on this page.
- “How to Import a Certificate in the .NET Card” on page 24
- “How to Delete a Certificate from the .NET Card” on page 27
- “How to View the Details of a Certificate in a .NET Card” on page 27
- “How to Use .NET Smart Card for Smart Card Logon” on page 28
- “How to Unblock a User PIN” on page 31
- “How to Change a User PIN” on page 31
- “How to Use E-mail Securely” on page 33
- “How to View Secure Web Sites” on page 38

How to Get a Certificate

A digital certificate contains information about the user and the user’s public key, and is used to authenticate the user’s identity during secure transactions. The certificate identifying the user must be registered with a certificate authority and this information must be available to both parties. To use smart cards/tokens and certificates together, the user must generate a key pair on his card/token and then get a digital certificate corresponding to the public key and store it on the card/token.

You can get a digital certificate from a Certificate Authority (CA). CA’s are trusted organizations that issue and manage digital certificates, such as Verisign.

Tips

When you request a certificate, you will be asked to enter information about yourself such as your name, e-mail address, and the type of certificate you want. The type of information required depends upon what organization is issuing the certificate, and may include the following:

- **Key length value.** The range of values is 1024-2048 in steps of 128 bits.
- **Cryptographic Module** (sometimes referred to as security device). You will need this if requesting a certificate using Mozilla Firefox.

  You must make sure that you specify the name corresponding to the label of your your PKCS#11 .NET smart card, for example, CF.NET P11. If you give a different name, your certificate will be stored on your hard drive instead of your smart card/ token.

How to Import a Certificate in the .NET Card

You can either use Firefox to import a certificate in the .NET card or Gemalto's .NET utilities. For more information about .NET utilities go to:

http://www.netsolutions.gemalto.com/utilities.aspx

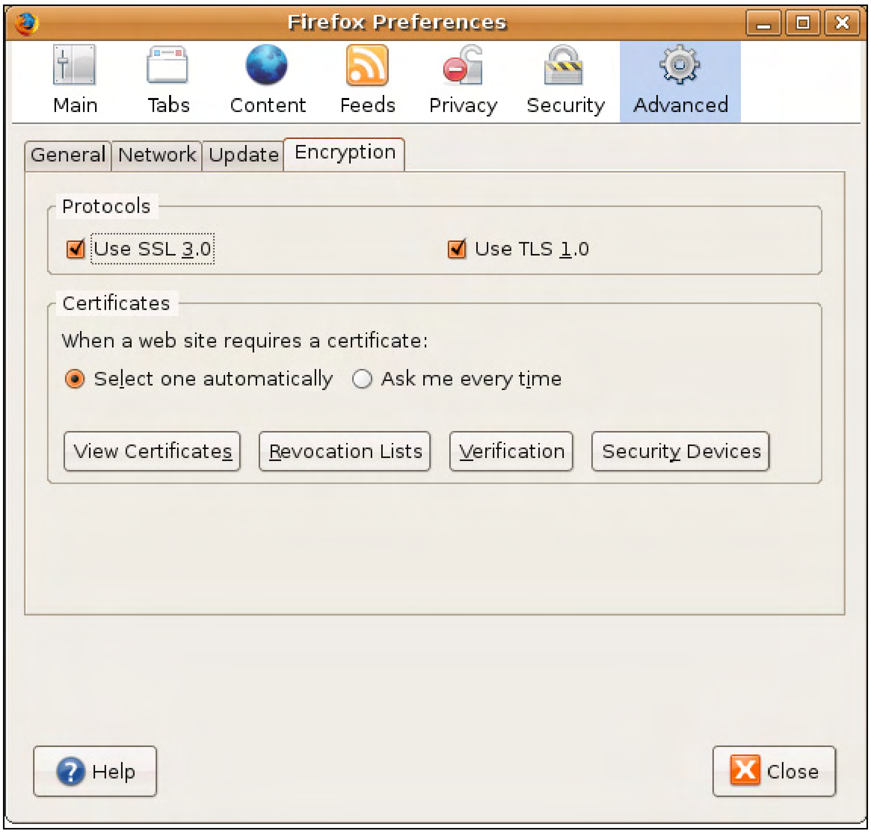
To import certificates in .NET cards using Mozilla Firefox:

1 Make sure your card/token is connected.

2 Open the **Mozilla Firefox** browser and from the **Edit** menu choose **Preferences**.

3 Click the **Advanced** icon, then the **Encryption** tab as shown in “Figure 10”.

Figure 10 - Mozilla Firefox Encryption Options Dialog



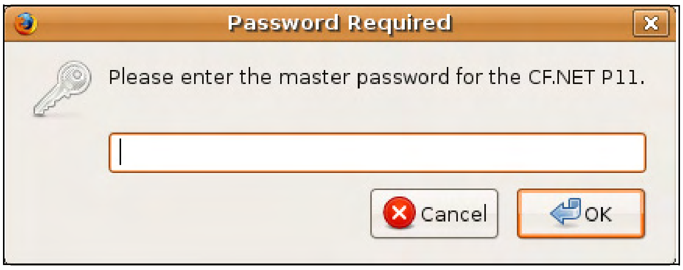
☑ Use SSL 3.0 ☑ Use TLS 1.0

When a web site requires a certificate:

● Select one automatically ○ Ask me every time

4 Click **View Certificates**. You will be prompted for a password as shown in “Figure 11”.

Figure 11 - Password Required

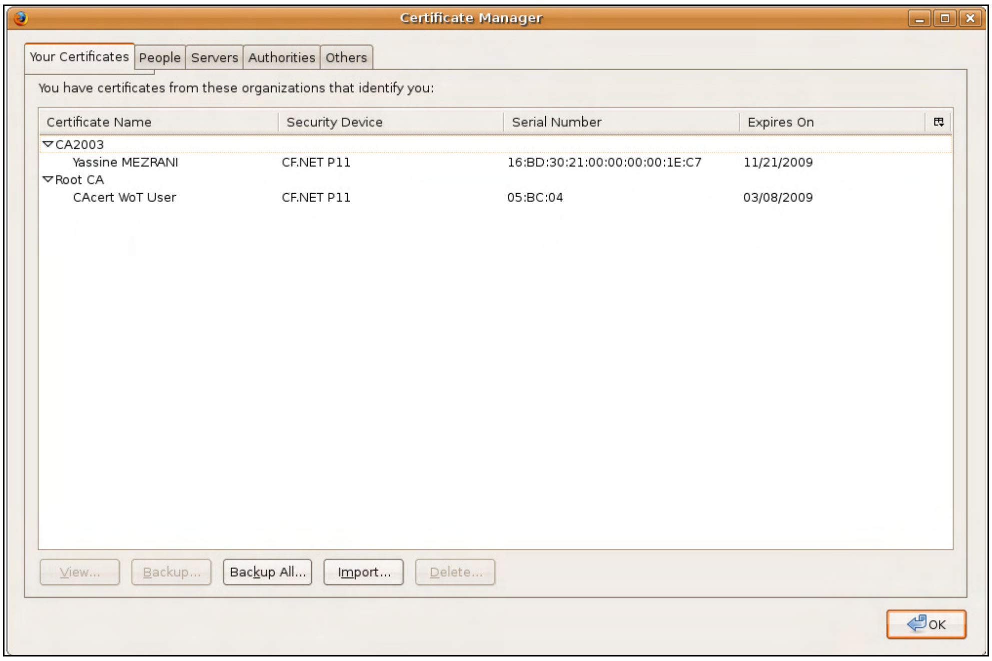


Please enter the master password for the CF.NET P11.

5 Enter the User PIN for your card/token

The **Certificate Manage**r window appears as shown in “Figure 12”.

Figure 12 - Certificate Manager Window



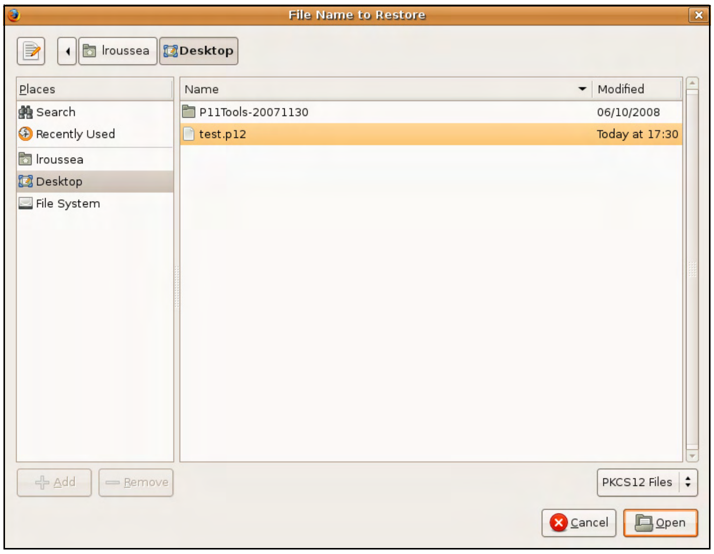
You have certificates from these organizations that identify you:

Certificate Name	Security Device	Serial Number	Expires On
CA2003			
Yassine MEZRANI	CF.NET P11	16:BD:30:21:00:00:00:00:1E:C7	11/21/2009
Root CA			
CAcert WoT User	CF.NET P11	05:BC:04	03/08/2009

The certificates that are currently stored on the card/token appear under **Your Certificates**.

6 Click **Import**. This opens a window called **File Name to Restore** as shown in “Figure 13”.

Figure 13 - File Name to Restore Window

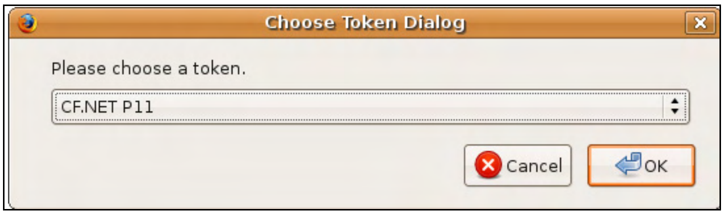


Name	Modified
P11Tools-20071130	06/10/2008
test.p12	Today at 17:30

7 Navigate to the file containing the PKCS#12 certificate (these files end in a .pfx suffix) and click **Open**.

The **Choose Token Dialog** window opens.

Figure 14 - Choose Token Dialog Window



Please choose a token.

CF.NET P11

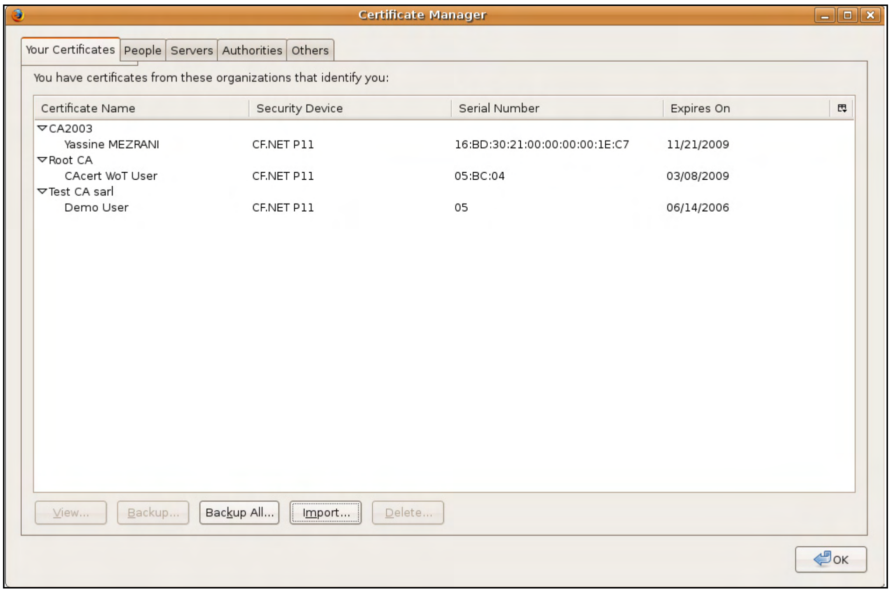
8 Select your .NET card/token from the list and click **OK**.

9 Enter the password for the .NET card/token if prompted.

10 If the certificate is encrypted, you will also be prompted to enter the password that was used with the encryption.

An “Alert” indicates that the certificate and its private keys have been imported. Click **OK** to close the **Alert**. The imported certificate appears in the **Certificate Manager** as shown in “Figure 15”.

Figure 15 - Certificate Manager After Certificate Importation



You have certificates from these organizations that identify you:

Certificate Name	Security Device	Serial Number	Expires On
CA2003			
Yassine MEZRANI	CF.NET P11	16:BD:30:21:00:00:00:00:1E:C7	11/21/2009
Root CA			
CAcert WoT User	CF.NET P11	05:BC:04	03/08/2009
Test CA sarl			
Demo User	CF.NET P11	05	06/14/2006

11 Click **OK** to close the **Certificate Manager**.

How to Delete a Certificate from the .NET Card

You can either use Firefox to delete a certificate from the .NET card or Gemalto’s .NET utilities. For more information about .NET utilities go to:

http://www.netsolutions.gemalto.com/utilities.aspx

To delete certificates in .NET cards using Mozilla Firefox:

1. Follow steps 1 to 5 in “How to Import a Certificate in the .NET Card” on page 24.
2. In the **Certificate Manager**, select the certificate that you want to delete and click **Delete**.
3. You will be asked to confirm the deletion, click **OK**. The certificate is removed from the .NET card and no longer appears in the **Certificate Manager**.

How to View the Details of a Certificate in a .NET Card

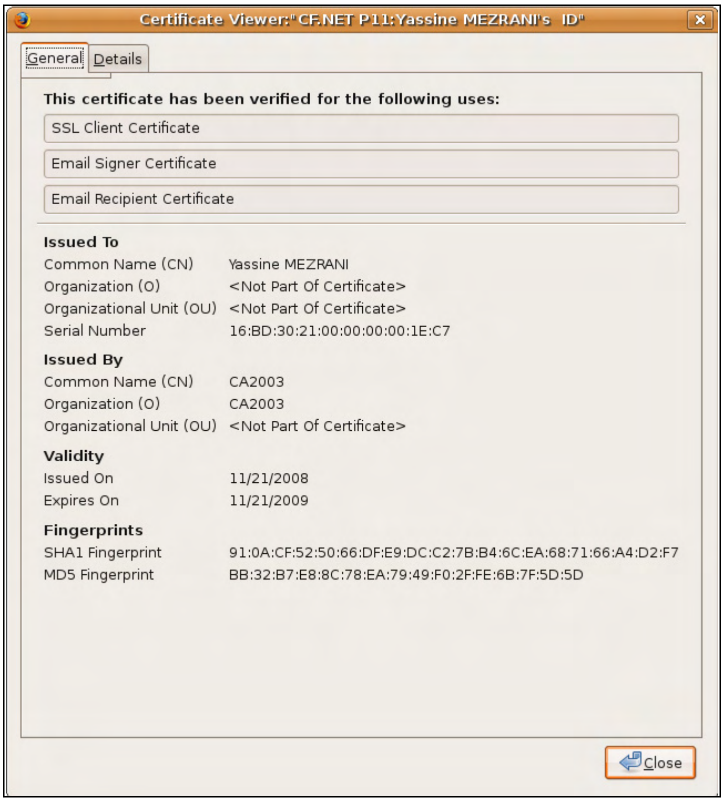
You can either use Firefox to view a certificate in a .NET card or Gemalto’s .NET utilities. For more information about .NET utilities go to:

http://www.netsolutions.gemalto.com/utilities.aspx

To view a certificate in a .NET card using Mozilla Firefox:

1. Follow steps 1 to 5 in “How to Import a Certificate in the .NET Card” on page 24.
2. In the **Certificate Manager**, select the certificate that you want to view and click **View**. The certificate’s details appear as shown in “Figure 16”.

Figure 16 - Certificate Details



Certificate Viewer:"CF.NET P11:Yassine MEZRANI's ID"

This certificate has been verified for the following uses:

SSL Client Certificate

Email Signer Certificate

Email Recipient Certificate

Issued To

Common Name (CN)	Yassine MEZRANI
Organization (O)	<Not Part Of Certificate>
Organizational Unit (OU)	<Not Part Of Certificate>
Serial Number	16:BD:30:21:00:00:00:00:1E:C7

Issued By

Common Name (CN)	CA2003
Organization (O)	CA2003
Organizational Unit (OU)	<Not Part Of Certificate>

Validity

Issued On	11/21/2008
Expires On	11/21/2009

Fingerprints

SHA1 Fingerprint	91:0A:CF:52:50:66:DF:E9:DC:C2:7B:B4:6C:EA:68:71:66:A4:D2:F7
MD5 Fingerprint	BB:32:B7:E8:8C:78:EA:79:49:F0:2F:FE:6B:7F:5D:5D

3 Click **Close** to close the window.

How to Use .NET Smart Card for Smart Card Logon

You can use .NET cards to perform smart card logon on Linux by using a PAM_pkcs#11 module. A PAM_pkcs#11 is a Linux-PAM login module that allows an X.509 certificate based user login. The certificate and its dedicated private key are thereby accessed by means of an appropriate PKCS #11 module. For the verification of the users' certificates, locally stored CA certificates as well as either online or locally accessible CRLs are used. A very flexible, stackable, and configurable Certificate-To-Login mapping scheme is provided to deduce/verify the username to log in.

The OpenSC project wrote a complete tutorial at http://www.opensc-project.org/doc/pam_pkcs11/pam_pkcs11.html, which gives further explanation.

The steps to follow to use this module are:

1 Install the Linux PAM module - you can get this module from the web site of your Linux software:
  - For Ubuntu: www.ubuntu.com
  - SUSE: www.novell.com/linux/
  - Debian: www.debian.org
  - RedHat: www.redhat.com

2 Install the PAM_pkcs#11 module from the site http://www.opensc-project.org/pam_pkcs11/

3 Configure the PAM_pkcs#11 module, as described in the next section.

4 Configure the mapper, as described on page 30.

5 Configure the PAM policies, as described on page 31.

Configuring the PAM_pkcs#11 Module

To do this, edit the PAM_pkcs#11 module’s configuration file:

/etc/pam_pkcs11/pam_pkcs11.conf.

The configuration file describes how to declare a PAM_pkcs11 module for the .NET PKCS#11 library, the CA certificates’ location (“cacerts” subdirectory by default), the certificate verification policy and the associated mappers.

Here is some example code you can include in your pam_pkcs11.conf file - instructions on how to code it are provided as comments lines in the code itself. The important lines are indicated by *****important lines*******

```
#
# Configuration file for pam_pkcs11 module
#
pam_pkcs11 {
  # Allow empty passwords
  nullok = true;

  # Enable debugging support.
  debug = true;

  # Do not prompt the user for the passwords but take them from the
  # PAM_ items instead.
  use_first_pass = false;

  # Do not prompt the user for the passwords unless PAM_(OLD)AUTHTOK
  # is unset.
  try_first_pass = false;

  # Like try_first_pass, but fail if the new PAM_AUTHTOK has not been
  # previously set (intended for stacking password modules only).
  use_authtok = false;


#************Start of important lines**************

  # Filename of the PKCS #11 module. The default value is "default"
  use_pkcs11_module = GemaltoDotNet;

  pkcs11_module GemaltoDotNet {
    module = /usr/lib/pkcs11/libgtop11dotnet.so;
    description = "Gemalto .NET PKCS#11 module";
#************End of important lines**************


# Which slot to use?
    # You can use "slot_num" or "slot_description", but not both, to specify
    # the slot to use.   Using "slot_description" is preferred because the
    # PKCS#11 specification does not guarantee slot ordering. "slot_num" should
    # only be used with those PKCS#11 implementations that guarantee
    # constant slot numbering.
    #
    #  slot_description = "xxxx"
    #      The slot is specified by the slot description, for example,
    #      slot_description = "Sun Crypto Softtoken".  The default value is
    #      "none" which means to use the first slot with an available token.
    #
    #  slot_num = a_number
    #      The slot is specified by the slot number, for example, slot_num = 1.
    #      The default value is zero which means to use the first slot with an
    #      available token.
    #
    slot_description = "none";
# Where are CA certificates stored?
    # You can setup this value to:
    # 1- A directory with openssl hash-links to all certificates
    # 2- A CA file in PEM (.pem) or ASN1 (.cer) format,
    # containing all allowed CA certs
    # The default value is /etc/pam_pkcs11/cacerts.
    ca_dir = /etc/pam_pkcs11/cacerts;

# Path to the directory where the local (offline) CRLs are stored.
    # Same convention as above is applied: you can choose either
    # hash-link directory or CRL file
    # The default value is /etc/pam_pkcs11/crls.
    crl_dir = /etc/pam_pkcs11/crls;

    # Some pcks#11 libraries can handle multithreading. So
    # set it to true to properly call C_Initialize()
    support_threads = false;

    # Sets the Certificate verification policy.
    # "none"        Performs no verification
    # "ca"          Does CA check
    # "crl_online"  Downloads the CRL form the location given by the
    #               CRL distribution point extension of the certificate
    # "crl_offline" Uses the locally stored CRLs
    # "crl_auto"    Is a combination of online and offline; it first
    #               tries to download the CRL from a possibly given CRL
    #               distribution point and if this fails, uses the local
    #               CRLs
    # "signature"   Does also a signature check to ensure that private
    #               and public key matches
    # You can use a combination of ca,crl, and signature flags, or just
    # use "none".
    cert_policy = ca,signature;

    # What kind of token?
    # The value of the token_type parameter will be used in the user prompt
    # messages.   The default value is "Smart card".
    token_type = "Smart card";
  }
```

Since /etc/pam_pkcs11/pam_pkcs11.conf has “debug = true” you should see lots of debug on the console you use to log in. You should also see any error that would occur. If everything works correctly you can change the configuration file and use “debug = false”.

Configure the Mapper

A mapper declares how to authenticate the end-user using the PKCS#11 .NET library indicated by the PAM_pkcs11 module. Check the opensc tutorial to know which mapper is appropriate to your case and how to configure it.

For example, you can choose the “subject” mapper which associates the certificate subject with the user account to allow the smart card logon.

To configure the mapper:

1 Use the “pkcs11_inspect” command line program to get the information you need for the mapper. The “pkcs11_inspect” tool allows you to look at the content of a

certificate, in order to help you in the process of Certificate-to-User mapping configuration. All information about your certificates are displayed. You have to locate the certificate subject needed for the mapper subject.

For example, in our case you are looking for something like:

```
Printing data for mapper subject: /C=FR/O=Gemalto/OU= Corporate Security/
CN=Gemalto Test User
```

Here the "pkcs11_inspect" tool picked the "subject" field of the certificate issued for a Gemalto Test User from the smart card.

2 Edit or create the file "/etc/pam_pkcs11/subject_mapping" to add the above strings followed by "->" and the account login name you want to link to this certificate in the .NET smart card.

For example, in our case the "subject_mapping" file looks like this:

```
# Mapping file for Certificate Subject
# format: Certificate Subject -> login

/C=FR/O=Gemalto/OU= Corporate Security/CN=Gemalto Test User -> TestUser
```

Here the file describes the mapping between the certificate containing the subject of the Gemalto Test User certificate on the smart card and the TestUser user account on the linux machine.

Configuring the PAM Policies

You now have to configure your PAM policies. Depending on the type of Linux OS used, these policies may be either in separate files or grouped together in a single file, "/etc/pam.conf".

If they are in separate files, each file bears the name of the service it applies to. These files are stored in the "/etc/pam.d/" directory. As we are configuring the smart card login feature, we are concerned only with the login service, stored in the file "/etc/pam.d/login". It is up to the PAM-aware program to define its service name and install its PAM configuration file in the pam.d directory.

In the pam.d file (or "login" section of the pam.conf file) add the following line:

```
auth sufficient pam_pkcs#11.so (with no arguments)
```

How to Unblock a User PIN

There are two ways to do this:

- Perform a remote unblock PIN operation using Gemalto's .NET utilities. Go to:

  http://www.netsolutions.gemalto.com/utilities.aspx

  and click **Unblock PIN**.
- Use the Security officer (SO) PIN (sometimes known as the Administrator PIN or the unblock PIN). For information on how to use this PIN and some sample code enabling you to do so, refer to "The Security Officer PIN" on page 12.

How to Change a User PIN

You can change a User PIN in a .NET card in one of the following ways:

- Use Gemalto's .NET utilities. For more information about .NET utilities go to:

  http://www.netsolutions.gemalto.com/utilities.aspx

- Use Mozilla Firefox, as described in the following section:

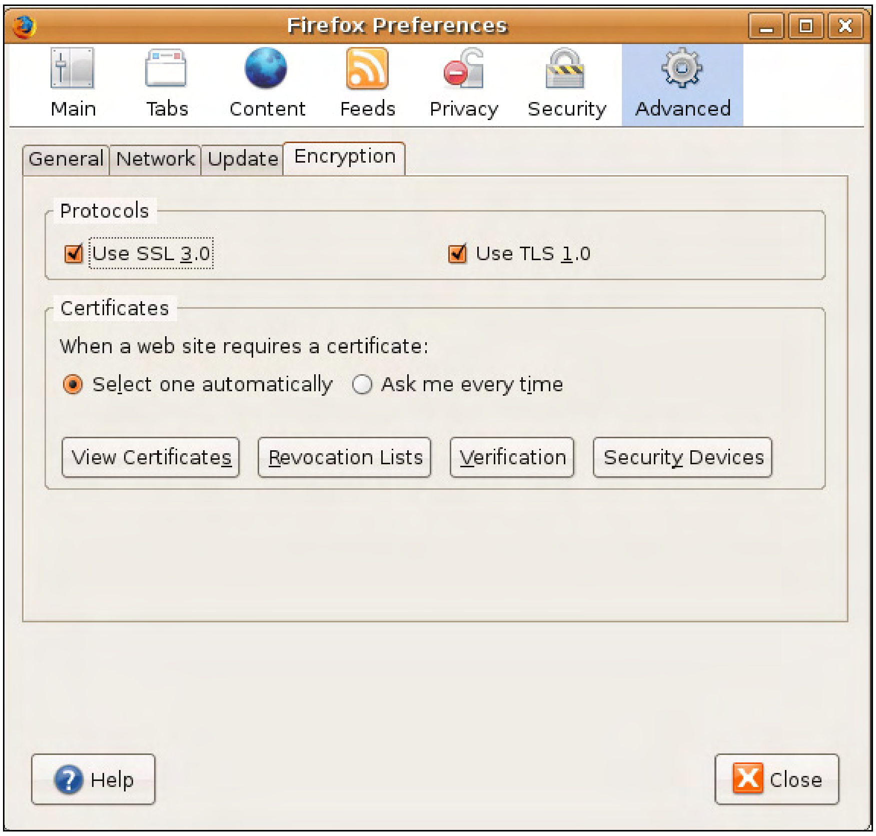
To change a User PIN in a .NET card using Mozilla Firefox:

1 Make sure your card/token is connected.

2 Open the **Mozilla Firefox** browser and from the **Edit** menu choose **Preferences**.

3 Click the **Advanced** icon, then the **Encryption** tab as shown in “Figure 17”.

Figure 17 - Mozilla Firefox Encryption Options Dialog



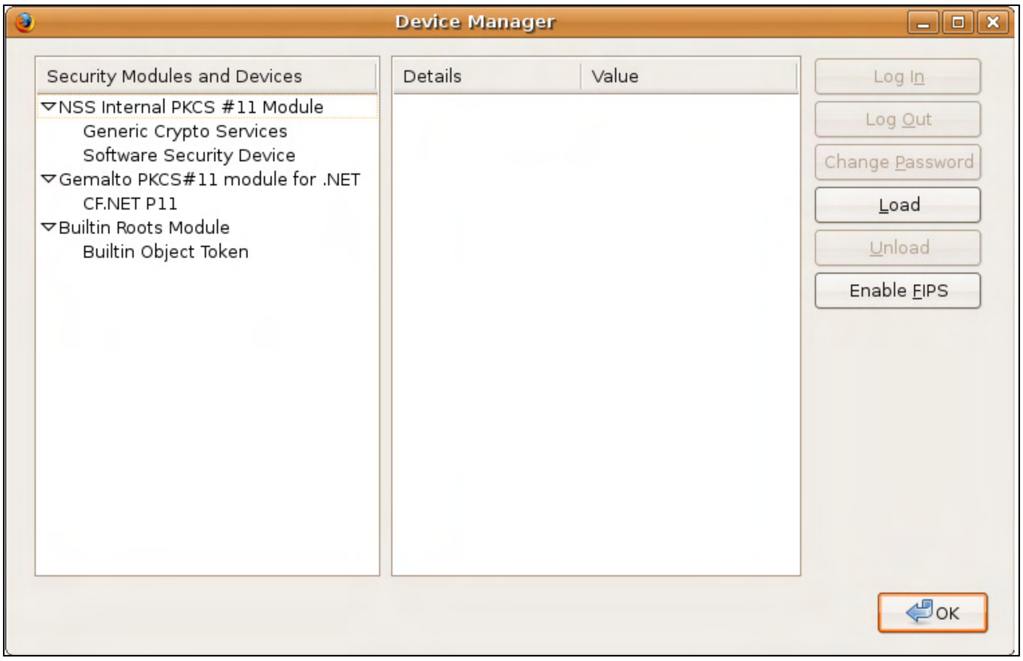
☑ Use SSL 3.0 ☑ Use TLS 1.0

When a web site requires a certificate:

● Select one automatically ○ Ask me every time

4 Click **Security Devices** to display the **Device Manager** window. This displays the modules currently available as shown in “Figure 18”.

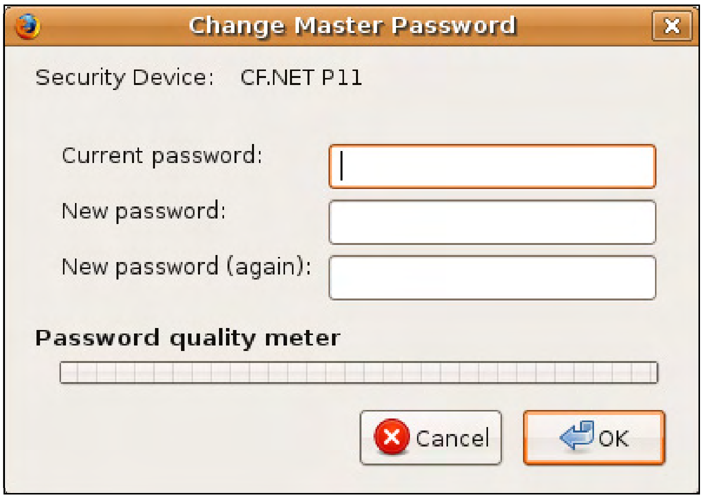
Figure 18 - Device Manager



5 In **Device Manager**, select the card whose PIN you want to change, as shown in “Figure 18”.

6 Click **Change Password**. The window shown in “Figure 19” appears.

Figure 19 - Change Master Password Window



7 In **Current Password**, enter the current PIN value.

8 In **New Password** and **New Password (again)**, enter the new PIN value for the smart card.

9 Click **OK**.

How to Use E-mail Securely

The following sections explain how to send secure e-mail using PKCS#11 for .NET Smart Cards.

About Secure E-mail

With PKCS#11 for .NET Smart Cards, you can improve e-mail security by using the digital certificate on your smart card/token to:

- Sign your e-mail so that the recipient can verify that the message is really from you and has not been altered.
- Encrypt, or “scramble” a message so that only the intended recipient can read it. This eliminates concerns about intercepted messages and e-mail monitoring.
- Sign or encrypt your message using one e-mail program, while your intended recipient can read it with any other S/MIME-enabled e-mail program.
- Receive signed and encrypted e-mail messages.

Setting up Secure E-mail

Depending on your e-mail application you will have to do some or all of the following before you can send secure e-mail:

Configure the application to recognize the PKCS#11 security module

- **Configure security settings**

  Set the security settings for digitally signing and/or encrypting the contents and attachments of outgoing messages.

- **Specify certificates to be used for signing and encryption**

  Choose the digital certificate(s) that you will use to encrypt and digitally sign your e-mails. You can use the same certificate for both operations or two different ones. These certificates are associated with your e-mail account.

- **Send yourself a digitally-signed e-mail**

  When you send a signed e-mail, you sign it with the private key. The recipient receives the corresponding public key with the mail which he or she uses to decipher your mail.

  Before you can send e-mails to anybody else, you need to send a signed message to yourself in order for Thunderbird to store your public key.

  Then you can send your public key to other people, for example by sending them a signed message. Once they have your public key, they can use it to encrypt mails they send to you (which you decipher using your private key).

The following sections describe how to perform the above operations using Mozilla Thunderbird. The dialog boxes shown may differ slightly from your own software, depending on what version you are using.

Working with Mozilla Thunderbird.

The following sections explain how to set up and send secure e-mail with Mozilla’s Thunderbird e-mail program. If you are using the Debian equivalent application, Icedove, the procedure is exactly the same. There are three stages:

1. Configure Thunderbird to recognize the Security Module, described in “Appendix C - Configuring PKCS#11 in Mozilla”.
2. Configure the security settings and specify the certificates to use for signing and encryption, described in the following section.
3. Send a digitally signed e-mail to yourself so that Thunderbird recognizes your public key, described on page 37.

Configuring Settings and Specifying Certificates

You only need to do this the first time you use your card/token to sign or encrypt an e-mail.

Note: Although selecting the certificates is mandatory, this does not mean that you must sign and encrypt e-mails.

---

1 Make sure your smart card/token is connected.

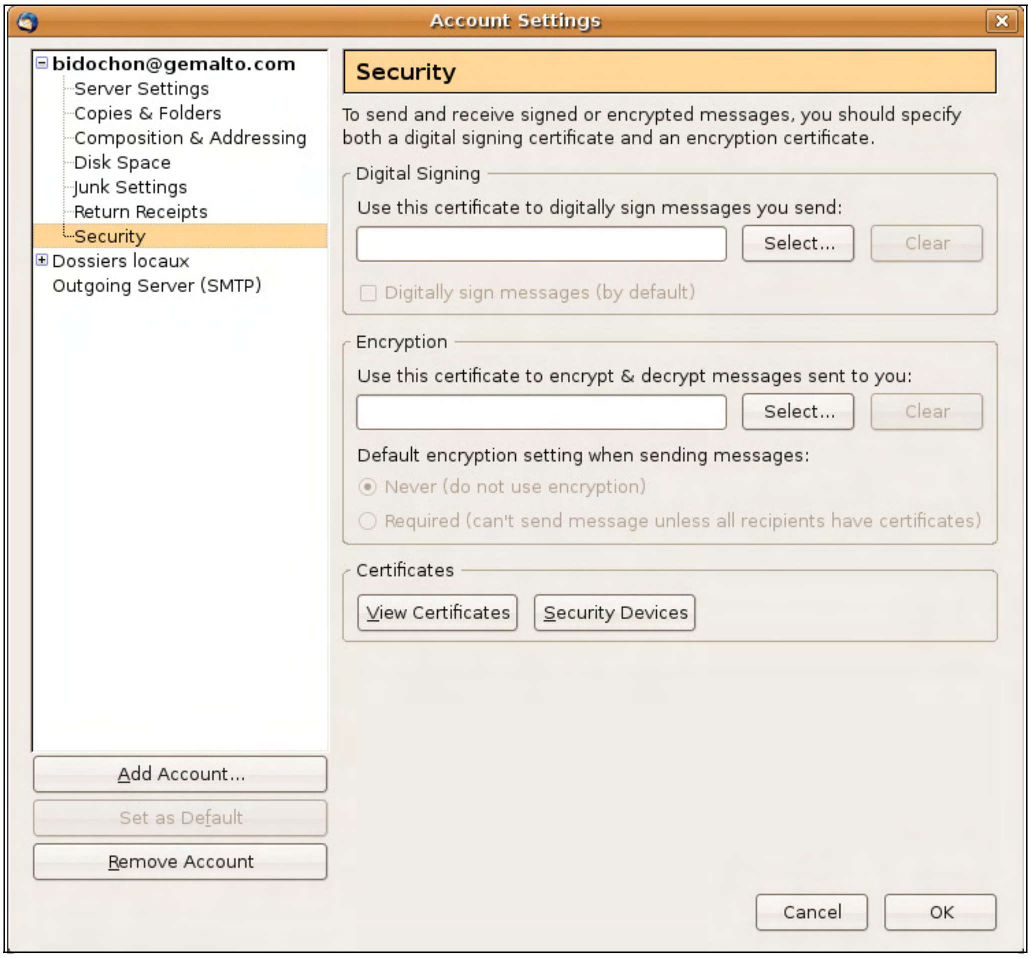
2 Start **Mozilla Thunderbird**.

3 Enter your password if you are prompted for it.

4 From the **Edit** menu, choose **Account Settings**.

5 Select **Security** as shown in “Figure 20”.

Figure 20 - Thunderbird – Security Account Settings



Security

To send and receive signed or encrypted messages, you should specify both a digital signing certificate and an encryption certificate.

Use this certificate to digitally sign messages you send:

☐ Digitally sign messages (by default)

Use this certificate to encrypt & decrypt messages sent to you:

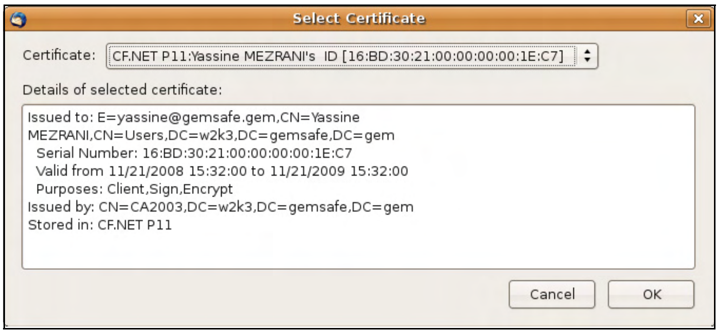
Default encryption setting when sending messages:

● Never (do not use encryption)

○ Required (can't send message unless all recipients have certificates)

6 In **Digital Signing**, click **Select**. The window shown in “Figure 21” appears.

Figure 21 - Thunderbird - Select Certificate



Certificate: CF.NET P11:Yassine MEZRANI's ID [16:BD:30:21:00:00:00:00:1E:C7]

Details of selected certificate:

```
Issued to: E=yassine@gemsafe.gem,CN=Yassine
MEZRANI,CN=Users,DC=w2k3,DC=gemsafe,DC=gem
  Serial Number: 16:BD:30:21:00:00:00:00:1E:C7
  Valid from 11/21/2008 15:32:00 to 11/21/2009 15:32:00
  Purposes: Client,Sign,Encrypt
Issued by: CN=CA2003,DC=w2k3,DC=gemsafe,DC=gem
Stored in: CF.NET P11
```

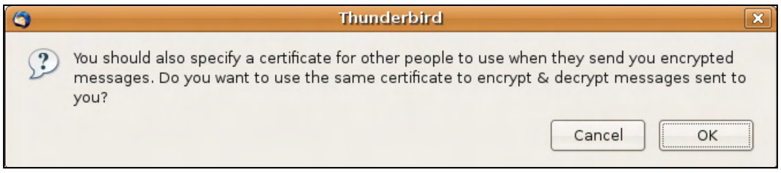
7 Select the certificate you want to use from the list that appears (its details appear in the window).

Note: You may be prompted to enter a "master password". If so, enter the PIN for the card and click **OK**.

---

8 Click **OK**. The following message appears:

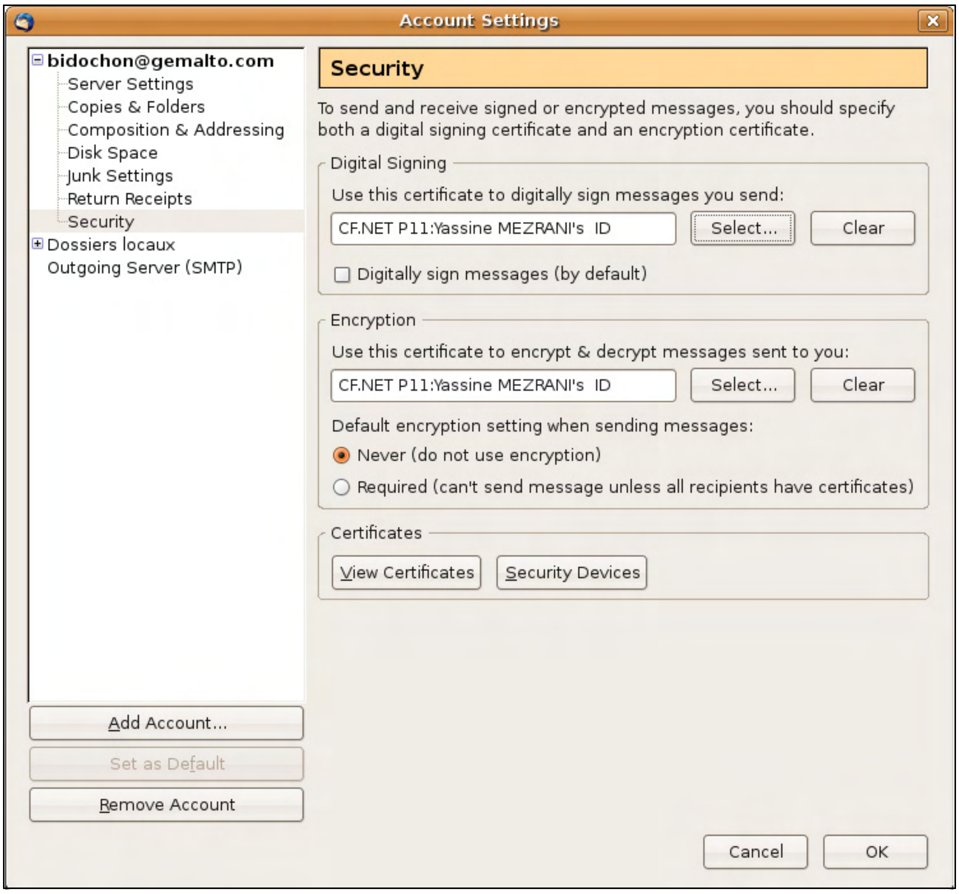
Figure 22 - Thunderbird – "Use Same Certificate" Message



You should also specify a certificate for other people to use when they send you encrypted messages. Do you want to use the same certificate to encrypt & decrypt messages sent to you?

9 If you want to use the same certificate to encrypt and decrypt messages, click **OK**. This selects the certificate for you in the **Encryption** panel as shown in "Figure 23". Otherwise click **Cancel**.

Figure 23 - Thunderbird – Security Account Settings (2)



Security

To send and receive signed or encrypted messages, you should specify both a digital signing certificate and an encryption certificate.

Digital Signing

Use this certificate to digitally sign messages you send: CF.NET P11:Yassine MEZRANI's ID

☐ Digitally sign messages (by default)

Encryption

Use this certificate to encrypt & decrypt messages sent to you: CF.NET P11:Yassine MEZRANI's ID

Default encryption setting when sending messages:

● Never (do not use encryption)

○ Required (can't send message unless all recipients have certificates)

Certificates

10 If you want all of your e-mails to be digitally signed by default, check the box **Digitally sign messages (by default)**.

11 In **Encryption**, if you chose not to use the same certificate as the one used for digital signing, click **Select** and choose the certificate from the list that appears. A message similar to the one in “Figure 22” appears, but this time asking if you want to use the Encryption certificate for digital signing. This is just in case you select your encryption certificate before you select your digital signature certificate.

12 In **Default encryption setting when sending messages**, choose one of the option buttons **Never** or **Required**.

13 Click **OK** to close the **Security Account Settings** window.

Note: If you want to modify the account settings at any point, open the **Account Settings** window from the **Tools** menu by choosing **Account Settings**. This can be done either from the **Compose** window or directly in Thunderbird.

Sending Digitally Signed E-mail with Mozilla Thunderbird

When you send a signed e-mail, you sign it with the private key. The recipient receives the corresponding public key with the mail which he or she uses to decipher your mail.

Before you can send e-mails to anybody else, you need to send a signed message to yourself in order for Thunderbird to store your public key.

Then you can send your public key to other people, for example by sending them a signed message. Once they have your public key, they can use it to encrypt mails they send to you (which you decipher using your private key).

To send a digitally signed e-mail with Mozilla Thunderbird:

1 Click the **Write** icon to open the **Compose** window.

2 In the **Compose** window, click the **Security** icon and choose **Digitally Sign This Message**.

3 Complete the **Compose** window and click **Send**. You may be prompted to enter the “master password” for your security module, which is the User PIN for the smart card/token.

Note: If you need further help in using Thunderbird, consult Thunderbird’s online help (**Help** > **Mozilla Thunderbird Help**).

Sending Encrypted E-mail with Mozilla Thunderbird

Once you have configured your e-mail account in **Mozilla Thunderbird**, you can retrieve a person’s public key when he or she sends a signed message to you. When you send e-mail to that person, you use his or her public key to encrypt the e-mail. This is done automatically by Thunderbird; you just need to specify the recipient(s) of the mail. Since no one except the person who has the private key can decrypt it, the e-mail is secure.

To send an encrypted e-mail with Mozilla Thunderbird:

1 Click the **Write** icon to open the **Compose** window.

2 In the **Compose** window, click the **Security** icon and choose **Encrypt This Message**.

3 Complete the **Compose** window and click **Send**. You may be prompted to enter the “master password” for your security module, which is the User PIN for the smart card/token.

How to View Secure Web Sites

Communicating and conducting business on the Web is quickly becoming the most convenient, effective means of transaction. Therefore, Web sites must be secure to protect the corporation, the individual and the information exchanged.

With your .NET smart card/token, you can browse secure Web sites knowing that your private key and digital certificate are safely stored on your smart card/token instead of your hard drive, where they might be susceptible to unauthorized access.

Note: All secure Web site addresses must begin with https://. Browsers display a lock icon at the bottom of the browser window indicating that the site is secure. A closed lock indicates that you are operating in secure mode. You may need to configure your organization’s network to allow secure browsing.

When you connect to a secure Web site, your certificate must be registered in your browser so that you can authenticate yourself to the Web server. For example, when you bank online, your bank must be sure that you are the correct person to get account information. Your certificate confirms your identity to the online bank.

The following sections explain how to check that your certificates are correctly registered in your browsers when authenticating with secure web sites using different browsers.

Mozilla Firefox

To authenticate yourself using the Mozilla Firefox browser, your certificate must be registered in Firefox. This section describes how to tell Firefox whether it should select the certificate itself, or ask you and also how to check that a certificate is registered.

To tell Firefox how to select a certificate:

1. Follow the procedure for Firefox described in “Appendix C - Configuring PKCS#11 in Mozilla”.
2. In step 4 of that procedure, choose whether you want Firefox to select a certificate for you or whether to ask you each time.

To check certificates are registered in Mozilla Firefox:

1. Follow the procedure for Firefox described in “How to Import a Certificate in the .NET Card” on page 24 until you open the Certificate Manager as shown in “Figure 12” on page 25.
2. Make sure that all the certificates that you want to use to authenticate to secure web sites in Firefox appear under **Your Certificates**.

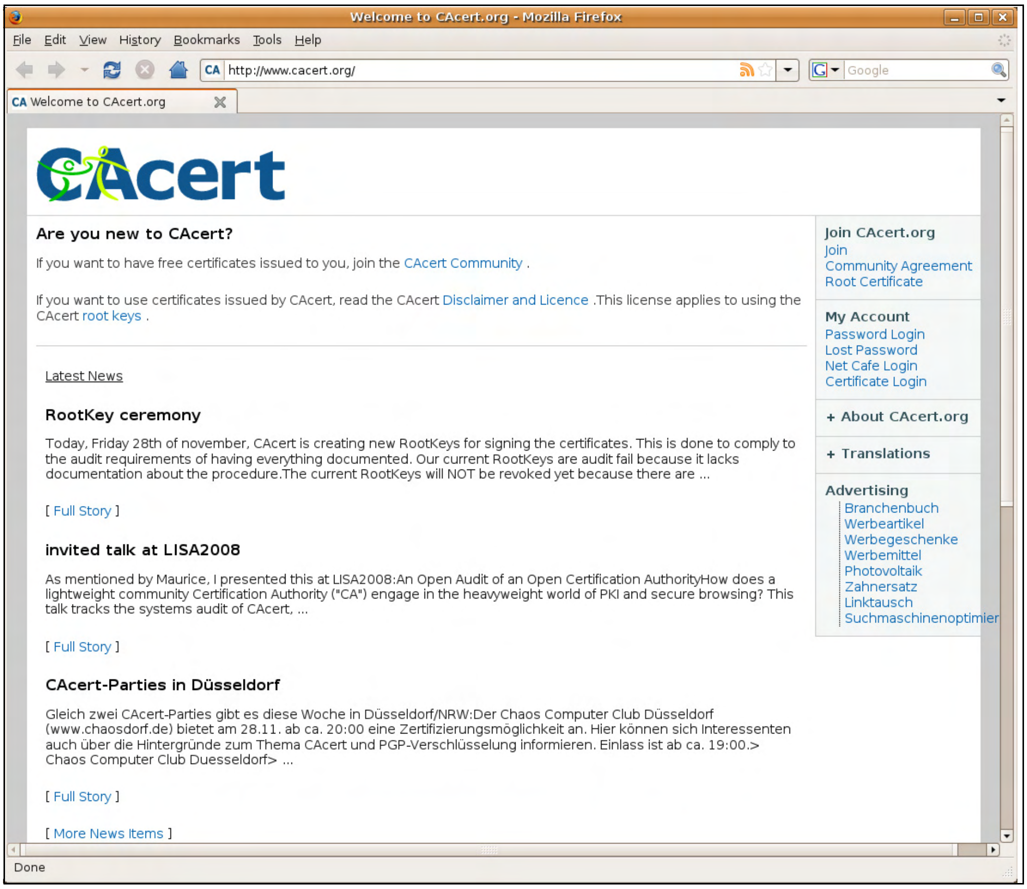
Example of Using the .NET Card to Authenticate to a Web Site

The following procedure provides an example of logging on to a page in the cacert.org web site, a site where you can obtain authorized certificates, that is secured.

To authenticate to a secured web page using Firefox:

1. Make sure the .NET smart card is connected and go to the web site in Firefox, as shown in “Figure 24”.

Figure 24 - Example Web Site Before Authentication



Are you new to CAcert?

If you want to have free certificates issued to you, join the CAcert Community .

If you want to use certificates issued by CAcert, read the CAcert Disclaimer and Licence .This license applies to using the CAcert root keys .

Latest News

RootKey ceremony

Today, Friday 28th of november, CAcert is creating new RootKeys for signing the certificates. This is done to comply to the audit requirements of having everything documented. Our current RootKeys are audit fail because it lacks documentation about the procedure.The current RootKeys will NOT be revoked yet because there are ...

[Full Story]

invited talk at LISA2008

As mentioned by Maurice, I presented this at LISA2008:An Open Audit of an Open Certification AuthorityHow does a lightweight community Certification Authority ("CA") engage in the heavyweight world of PKI and secure browsing? This talk tracks the systems audit of CAcert, ...

[Full Story]

CAcert-Parties in Düsseldorf

Gleich zwei CAcert-Parties gibt es diese Woche in Düsseldorf/NRW:Der Chaos Computer Club Düsseldorf (www.chaosdorf.de) bietet am 28.11. ab ca. 20:00 eine Zertifizierungsmöglichkeit an. Hier können sich Interessenten auch über die Hintergründe zum Thema CAcert und PGP-Verschlüsselung informieren. Einlass ist ab ca. 19:00.> Chaos Computer Club Duesseldorf> ...

[Full Story]

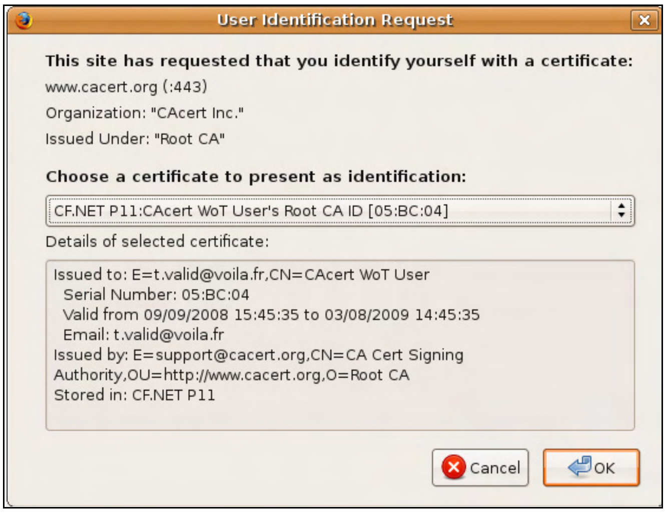
[More News Items]

2 Click the link to the secured page, in this example **Certificate Login** under **My Account** on the right of the page.

3 If prompted for the master password, enter the PIN of the .NET smart card.

4 If Firefox is configured to **Ask Me Each Time** (see step 4 in “Appendix C - Configuring PKCS#11 in Mozilla” for “Firefox” on page 81), you are asked to select a certificate to authenticate yourself as shown in “Figure 25”.

Figure 25 - User Identification Request Window



This site has requested that you identify yourself with a certificate:

www.cacert.org (:443)

Organization: "CAcert Inc."

Issued Under: "Root CA"

Choose a certificate to present as identification:

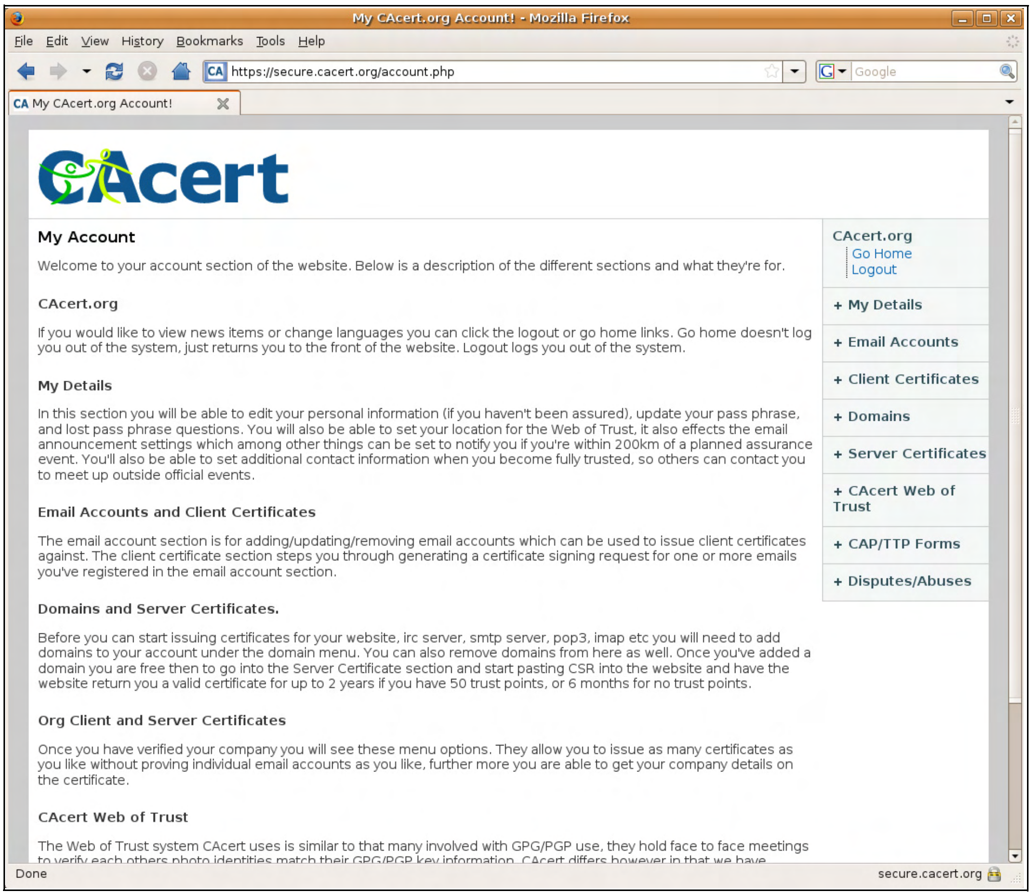
CF.NET P11:CAcert WoT User's Root CA ID [05:BC:04]

Details of selected certificate:

Issued to: E=t.valid@voila.fr,CN=CAcert WoT User
Serial Number: 05:BC:04
Valid from 09/09/2008 15:45:35 to 03/08/2009 14:45:35
Email: t.valid@voila.fr
Issued by: E=support@cacert.org,CN=CA Cert Signing Authority,OU=http://www.cacert.org,O=Root CA
Stored in: CF.NET P11

5 Select the certificate and click **OK**. You are given access to the secured web page as shown in “Figure 26”.

Figure 26 - Secured Web Page After Authentication



My Account

Welcome to your account section of the website. Below is a description of the different sections and what they're for.

CAcert.org

If you would like to view news items or change languages you can click the logout or go home links. Go home doesn't log you out of the system, just returns you to the front of the website. Logout logs you out of the system.

My Details

In this section you will be able to edit your personal information (if you haven't been assured), update your pass phrase, and lost pass phrase questions. You will also be able to set your location for the Web of Trust, it also effects the email announcement settings which among other things can be set to notify you if you're within 200km of a planned assurance event. You'll also be able to set additional contact information when you become fully trusted, so others can contact you to meet up outside official events.

Email Accounts and Client Certificates

The email account section is for adding/updating/removing email accounts which can be used to issue client certificates against. The client certificate section steps you through generating a certificate signing request for one or more emails you've registered in the email account section.

Domains and Server Certificates.

Before you can start issuing certificates for your website, irc server, smtp server, pop3, imap etc you will need to add domains to your account under the domain menu. You can also remove domains from here as well. Once you've added a domain you are free then to go into the Server Certificate section and start pasting CSR into the website and have the website return you a valid certificate for up to 2 years if you have 50 trust points, or 6 months for no trust points.

Org Client and Server Certificates

Once you have verified your company you will see these menu options. They allow you to issue as many certificates as you like without proving individual email accounts as you like, further more you are able to get your company details on the certificate.

CAcert Web of Trust

The Web of Trust system CAcert uses is similar to that many involved with GPG/PGP use, they hold face to face meetings to verify each others photo identities match their GPG/PGP key information. CAcert differs however in that we have

Notice that the URL at the top of the page begins with https, and a padlock icon appears at the bottom right of the page - both indicating that the page is secured.

A

Sample Code

Cryptoki Header Files

In addition to the sample code provided in this appendix, you will need all of the following Cryptoki header files:

- pkcs11.h
- pkcs11f.h
- pkcs11t.h

These can be downloaded from http://www.rsa.com/rsalabs/node.asp?id=2133.

Sample Code Files

This appendix lists several source code files presenting how to use the most common PKCS#11 methods of the library.

The file “main.c” is a generic program executing the linked “SampleFunction” code. To build the program you have to compile the “main.c” file with one of the other files described in this appendix.

main.c

This sample code load the cryptoki and execute the SampleFunction linked with it.

```
#include <stdio.h>
#include <string.h>

#ifdef WIN32

#include <windows.h>
#include <conio.h>
#ifndef _WINDOWS
#define _WINDOWS
#endif

#define LIBRARY_NAME "gtop11dotnet"
#define LIBRARY_EXT ".dll"
#define DLOPEN(lib) LoadLibrary(lib)
#define DLSYM(h, function) GetProcAddress(h, function)
#define DLCLOSE(h) FreeLibrary(h)

#else
#include <stdlib.h>
#include <unistd.h>
```

```
#include <dlfcn.h>

#define LIBRARY_NAME "/usr/lib/pkcs11/libgtop11dotnet"
#ifdef __APPLE__
#define LIBRARY_EXT ".dylib"
#else
#define LIBRARY_EXT ".so"
#endif
#define DLOPEN(lib) dlopen(lib, RTLD_NOW)
#define DLSYM(h, function) dlsym(h, function)
#define DLCLOSE(h) dlclose(h)

#endif

#include "pkcs11.h"
#define CKRLOG(fct, rv) printf("\n%s:%d " fct "() exited with Cryptoki error
0x%08lX: \n", __FILE__, __LINE__, rv)

/*---------------------------------------------------------------------------
                           Static Global Variables
---------------------------------------------------------------------------*/
static CK_SLOT_ID slotID = 0;
static char szDLLName[128] = LIBRARY_NAME LIBRARY_EXT;
static CK_CHAR szPinCode[49] = "0000";/* default PIN code */
#ifndef NO_SESSION
static CK_SESSION_HANDLE hSession;
#endif
static CK_RV rv = CKR_OK;
static CK_FUNCTION_LIST_PTR p = NULL;

void SampleFunction(void);

/****************************************************************************
* void usage(void)
*
* Description : Prints out program information and then terminates the
*               application.
****************************************************************************/

void usage(void);
void usage(void)
{
   printf("(c)2008 Gemalto Development\n\
usage: program -p:<pincode> -s:<slotid> -d:<cryptokidll> -l:\n\n\
All arguments are optional. Defaults are: \n\n\
    <pincode>      %s\n\
    <slotid>       %ld\n\
    <cryptokidll>  %s\n", szPinCode, slotID, szDLLName);

   exit(0);
}

/****************************************************************************
* void main(int argc, char* argv[])
*
* Description : Standard initialization and termination code for all
*               samples.
*
****************************************************************************/
int main(int argc, char *argv[])
{
   CK_RV(*pC_GetFunctionList) (CK_FUNCTION_LIST_PTR_PTR);
```

```
    void *hModule;
    CK_BBOOL IsSlotEnter = FALSE;
    CK_SLOT_ID_PTR pSlotList;
    CK_ULONG count = 0;
    int a;

    /*---- Analyze the command line for parameters (see usage) ----*/
    for (a = 1; a < argc; a++)
    {

       /* Expect arguments of the form -x:<param> */
       if ((strlen(argv[a]) < 3) ||
          (argv[a][0] != '-') || (argv[a][2] != ':'))
          usage();
       switch (argv[a][1])

       {
       case 'p':
          strncpy((char *) szPinCode, &(argv[a][3]), sizeof(szPinCode));
          szPinCode[sizeof(szPinCode) - 1] = '\0';
          printf("Using PIN: %s\n", szPinCode);
          break;
       case 'd':
          strncpy((char *) szDLLName, &(argv[a][3]), sizeof(szDLLName));
          szDLLName[sizeof(szDLLName) - 1] = '\0';
          printf("Using library: %s\n", szDLLName);
          break;
       case 's':
          if (sscanf(&(argv[a][3]), "%lu", &slotID) != 1)
             usage();
          printf("Using slot: %ld\n", slotID);
          IsSlotEnter = TRUE;
          break;
       default:
          usage();
          break;
       }
    }


    /* ---- Load dynamically DLL and retrieve function list pointer -- */
    if ((hModule = DLOPEN(szDLLName)) == 0)
    {
       printf("DLOPEN Error\n");
       exit(0);
    }

 if ((pC_GetFunctionList = (CK_RV (*) (CK_VOID_PTR))DLSYM(hModule,
 "C_GetFunctionList")) == NULL)
    {
       printf("DLSYM Error\n");
       exit(0);
    }

    /* ---- Cryptoki library standard initialization ---- */
    if ((rv = pC_GetFunctionList(&p)) != CKR_OK)
    {
       CKRLOG("C_GetFunctionList", rv);
       exit(0);
    }
    rv = (*p->C_Initialize) (NULL_PTR);
    if (rv != CKR_OK)
```

```
   {
      CKRLOG("C_Initialize", rv);
      exit(0);
   }

   if (!IsSlotEnter)
   {
      unsigned int i;

      /* Get number of slots in system                                      */
      rv = (*p->C_GetSlotList) (FALSE, NULL, &count);
      if (rv != CKR_OK)
      {
         CKRLOG("C_GetSlotList", rv);
         goto end;
      }

      /* no slot found */
      if (0 == count)
      {
         printf("No slot found\n");
         goto end;
      }
      printf("Nb slot found: %ld\n", count);

      /* no slot found */
      if (0 == count)
      {
         printf("No slot found\n");
         goto end;
      }

      pSlotList = calloc(count, sizeof(CK_SLOT_ID));

      pSlotList[0] = 42;
      /* Get First Slot ID, with Token if possible                          */
      rv = (*p->C_GetSlotList) (FALSE, pSlotList, &count);
      if (rv != CKR_OK)
      {
         CKRLOG("C_GetSlotList", rv);
         free(pSlotList);
         goto end;
      }

      printf("Nb slot found: %ld ( ", count);
      for (i=0; i<count; i++)
         printf("%ld ", pSlotList[i]);
      printf(")\n");

      slotID = pSlotList[0];
      printf("using slot: %ld\n", slotID);
      free(pSlotList);
   }

#ifndef NO_SESSION
   /* C_OpenSession */
   rv = (*p->C_OpenSession) (slotID, CKF_SERIAL_SESSION
#ifdef RW_SESSION
      | CKF_RW_SESSION
#endif
      , NULL_PTR, NULL_PTR, &hSession);
   if (rv != CKR_OK)
   {
      CKRLOG("C_OpenSession", rv);
      goto end;
   }

#ifndef NO_LOGIN /* Some samples don't require us to log in */
   /* Login only if a PIN code is presented
    * Needed to get the "Free private memory:" value */
   if (strlen((char *) szPinCode))
   {
      rv = (*p->C_Login) (hSession, CKU_USER, szPinCode,(CK_ULONG)
               strlen((const char *) szPinCode));
      if (rv != CKR_OK)
      {
         CKRLOG("C_Login", rv);
         goto end;
      }
   }
#endif
#endif

   SampleFunction();

#ifndef NO_SESSION
   /* C_CloseSession */
   rv = (*p->C_CloseSession) (hSession);
   if (rv != CKR_OK)
      CKRLOG("C_CloseSession", rv);
#endif

  end:

   /*---- Tidy up ----*/
   (*p->C_Finalize) (NULL_PTR);

   if (hModule != 0)
      DLCLOSE(hModule);

   /*---- Print out error if necessary ----*/
   if (rv != CKR_OK)
      CKRLOG("C_Finalize", rv);

#ifdef WIN32
   printf("Press a key to exit...");
   getchar();
#endif

   return 0;
}
```

getinfo.c

This sample code shows how to retrieve the token information, the slot information and the session information. It is included in other functions such as “tellme.c” on page 76 and “slotevent.c” on page 73.

```
static void getinfo(void)
{
   CK_INFO        info;
   CK_CHAR szId[33], szDescription[33];

   if((rv = (*p->C_GetInfo)(&info)) != CKR_OK)
   {
      CKRLOG("C_GetInfo", rv);
      return;
   }

   memcpy(szId, info.manufacturerID, 32);
   szId[sizeof(szId)-1] = '\0';

   memcpy(szDescription, info.libraryDescription, 32);
   szDescription[sizeof(szDescription)-1] = '\0';

   printf("Library Information:\n\
> Cryptoki Version:    %d.%02d\n\
> Manufacturer Id:     %s\n\
> Flags:               %04lX\n\
> Library Description: %s\n\
> Library Version:     %d.%d\n\n",
      info.cryptokiVersion.major, info.cryptokiVersion.minor,
      szId,
      info.flags,
      szDescription,
      info.libraryVersion.major, info.libraryVersion.minor);
}

static void getslotinfo(CK_SLOT_ID slotid)
{
   CK_SLOT_INFO   sinfo;
   CK_CHAR szId[33], szSlotDescription[65];

   if((rv = (*p->C_GetSlotInfo)(slotID, &sinfo)) != CKR_OK)
   {
      CKRLOG("C_GetSlotInfo", rv);
      return;
   }

   memcpy(szId, sinfo.manufacturerID, 32);
   szId[sizeof(szId)-1] = '\0';

   memcpy(szSlotDescription, sinfo.slotDescription, 64);
   szSlotDescription[sizeof(szSlotDescription)-1] = '\0';

   printf("Slot Information for slot %ld:\n\
> Slot Description:    %s\n\
> Manufacturer Id:     %s\n\
> Flags:               %s %s %s\n\
> Hardware Version:    %d.%d\n\
> Firmware Version:    %d.%d\n\n",
      slotID,
      szSlotDescription,
      szId,
      ((sinfo.flags & 1) ? "CKF_TOKEN_PRESENT" : ""),
      ((sinfo.flags & 2) ? "CKF_REMOVABLE_DEVICE" : ""),
      ((sinfo.flags & 4) ? "CKF_HW_SLOT" : ""),
      sinfo.hardwareVersion.major, sinfo.hardwareVersion.minor,
      sinfo.firmwareVersion.major, sinfo.firmwareVersion.minor);

}

static void gettokeninfo(CK_SLOT_ID slotid)
{
   CK_TOKEN_INFO  tinfo;
   CK_CHAR szId[33], szLabel[33], szModel[17], szSerialNumber[17];

   if((rv = (*p->C_GetTokenInfo)(slotID, &tinfo)) != CKR_OK)
   {
      CKRLOG("C_GetTokenInfo", rv);
      return;
   }

   memcpy(szLabel, tinfo.label, 32);
   szLabel[sizeof(szLabel)-1] = '\0';

   memcpy(szId, tinfo.manufacturerID, 32);
   szId[sizeof(szId)-1] = '\0';

   memcpy(szModel, tinfo.model, 16);
   szModel[sizeof(szModel)-1] = '\0';

   memcpy(szSerialNumber, tinfo.serialNumber, 16);
   szSerialNumber[sizeof(szSerialNumber)-1] = '\0';

   printf("Token Information for slot %ld:\n\
> Label:                %s\n\
> Manufacturer Id:      %s\n\
> Model:                %s\n\
> Serial Number:        %s\n\
> Flags:                %s%s%s%s%s%s%s%s%s%s%s\n\
> Max sessions:         %ld\n\
> Current sessions:     %ld\n\
> Max R/W sessions      %ld\n\
> Current R/W sessions: %ld\n\
> Max Pin Len:          %ld\n\
> Min Pin Len:          %ld\n\
> Total public memory:  %ld\n\
> Free public memory:   %ld\n\
> Total private memory: %ld\n\
> Free private memory:  %ld\n",
      slotID,
      szLabel,
      szId,
      szModel,
      szSerialNumber,
      ((tinfo.flags & 1)   ? "CKF_RNG " : ""),
      ((tinfo.flags & 2)   ? "CKF_WRITE_PROTECTED " : ""),
      ((tinfo.flags & 4)   ? "CKF_LOGIN_REQUIRED " : ""),
      ((tinfo.flags & 8)   ? "CKF_USER_PIN_INITIALIZED " : ""),
      ((tinfo.flags & 16)  ? "CKF_EXCLUSIVE_EXISTS " : ""),
      ((tinfo.flags & 32)  ? "CKF_RESTORE_KEY_NOT_NEEDED " : ""),
      ((tinfo.flags & 64)  ? "CKF_CLOCK_ON_TOKEN " : ""),
      ((tinfo.flags & 128) ? "CKF_SUPPORTS_PARALLEL " : ""),
      ((tinfo.flags & 256) ? "CKF_PROTECTED_AUTHENTICATION_PATH " : ""),
      ((tinfo.flags & 512) ? "CKF_DUAL_CRYPTO_OPERATIONS " : ""),
      tinfo.ulMaxSessionCount,
      tinfo.ulSessionCount,
      tinfo.ulMaxRwSessionCount,
      tinfo.ulRwSessionCount,
      tinfo.ulMaxPinLen,
      tinfo.ulMinPinLen,
      tinfo.ulTotalPublicMemory,
      tinfo.ulFreePublicMemory,
      tinfo.ulTotalPrivateMemory,
      tinfo.ulFreePrivateMemory);
}

#ifndef NO_SESSION
static void getsessioninfo(void)
{
   CK_SESSION_INFO si;

   if((rv = (*p->C_GetSessionInfo)(hSession, &si)) != CKR_OK)
   {
      CKRLOG("C_GetSessionInfo", rv);
      return;
   }

   printf("Session Information\n\
> slotID:        %ld\n\
> state:         %ld\n\
> flags:         %ld\n\
> ulDeviceError: %ld\n\n",
   si.slotID,
   si.state,
   si.flags,
   si.ulDeviceError);
}
#endif
```

deleteall.c

This sample shows how to remove all objects from the token. This can be very useful during development to “reinitialize” the token. Note that the search for the next object to be deleted using the C_FindObjects function has to be reinitialized each time because the calling of the C_DestroyObject function invalidates the previous search.

```
/* include the common code */
#define RW_SESSION
#include "main.c"

/*****************************************************************************
* void SampleFunction(void)
*
******************************************************************************
*/
void SampleFunction(void)
{
   CK_OBJECT_HANDLE hObject;
   CK_ULONG found;
   int i = 0;

   /*---- Get user confirmation (this will, after all, erase the lot)----*/
   printf("All objects in token will be deleted. OK? [y|n]: ");
   if (getchar() != 'y')
      return;
   printf("Deleting...\n");
   do
   {

     /*---- Search for an object ----*/
      if ((rv = (*p->C_FindObjectsInit) (hSession, NULL_PTR, 0)) != CKR_OK)
      {
         CKRLOG("C_FindObjectsInit", rv);
         return;
      }
      if ((rv = (*p->C_FindObjects) (hSession, &hObject, 1, &found))
         != CKR_OK)
      {
         CKRLOG("C_FindObjects", rv);
         return;
      }
      if ((rv = (*p->C_FindObjectsFinal) (hSession)) != CKR_OK)
      {
         CKRLOG("C_FindObjectsFinal", rv);
         return;
      }
      if (found == 1)
      {

       /*---- It's curtains for the object----*/
         if ((rv = (*p->C_DestroyObject) (hSession, hObject)) != CKR_OK)
         {
            CKRLOG("C_DestroyObject", rv);
            return;
         }
         i++;
      }
   }
   while (found == 1);
   printf("\n%d objects deleted.\n", i);
   return;
}
```

dumpit.c

This sample is very useful for debugging. It prints out a listing of all the objects and their attributes in the token. It illustrates how to find objects using the C_FindObject functions and then how to determine their attributes using C_GetAttribute. The user is assumed to have logged in (C_Login), before using this function.

```
/* include the common code */
#include "main.c"
#include <ctype.h>

#ifndef WIN32
#define min(a,b) (((a)<(b))?(a):(b))
#endif
/*****************************************************************************
* void SampleFunction(void)
*****************************************************************************/
void SampleFunction(void)
{
/*  List of possible format types for attributes */
#define FT_ULONG        (1)
#define FT_BYTES        (2)
#define FT_BOOL         (3)
   /*---- Dull list of all possible Cryptoki attributes ----*/
```

```
static const struct
{
   const char *pszName;
   CK_ULONG ulType;
   int nFormat;
} ATypes[] =
{
   /* Name                    Attrbute ID               Format type */
   { "CKA_CLASS", CKA_CLASS, FT_ULONG},
   { "CKA_TOKEN", CKA_TOKEN, FT_BOOL},
   { "CKA_PRIVATE", CKA_PRIVATE, FT_BOOL},
   { "CKA_LABEL", CKA_LABEL, FT_BYTES},
   { "CKA_APPLICATION", CKA_APPLICATION, FT_BYTES},
   { "CKA_VALUE", CKA_VALUE, FT_BYTES},
   { "CKA_CERTIFICATE_TYPE", CKA_CERTIFICATE_TYPE, FT_ULONG},
   { "CKA_ISSUER", CKA_ISSUER, FT_BYTES},
   { "CKA_SERIAL_NUMBER", CKA_SERIAL_NUMBER, FT_BYTES},
   { "CKA_KEY_TYPE", CKA_KEY_TYPE, FT_ULONG},
   { "CKA_SUBJECT", CKA_SUBJECT, FT_BYTES},
   { "CKA_ID", CKA_ID, FT_BYTES},
   { "CKA_SENSITIVE", CKA_SENSITIVE, FT_BOOL},
   { "CKA_ENCRYPT", CKA_ENCRYPT, FT_BOOL},
   { "CKA_DECRYPT", CKA_DECRYPT, FT_BOOL},
   { "CKA_WRAP", CKA_WRAP, FT_BOOL},
   { "CKA_UNWRAP", CKA_UNWRAP, FT_BOOL},
   { "CKA_SIGN", CKA_SIGN, FT_BOOL},
   { "CKA_SIGN_RECOVER", CKA_SIGN_RECOVER, FT_BOOL},
   { "CKA_VERIFY", CKA_VERIFY, FT_BOOL},
   { "CKA_VERIFY_RECOVER", CKA_VERIFY_RECOVER, FT_BOOL},
   { "CKA_DERIVE", CKA_DERIVE, FT_BOOL},
   { "CKA_START_DATE", CKA_START_DATE, FT_BYTES},
   { "CKA_END_DATE", CKA_END_DATE, FT_BYTES},
   { "CKA_MODULUS", CKA_MODULUS, FT_BYTES},
   { "CKA_MODULUS_BITS", CKA_MODULUS_BITS, FT_ULONG},
   { "CKA_PUBLIC_EXPONENT", CKA_PUBLIC_EXPONENT, FT_BYTES},
   { "CKA_PRIVATE_EXPONENT", CKA_PRIVATE_EXPONENT, FT_BYTES},
   { "CKA_PRIME_1", CKA_PRIME_1, FT_BYTES},
   { "CKA_PRIME_2", CKA_PRIME_2, FT_BYTES},
   { "CKA_EXPONENT_1", CKA_EXPONENT_1, FT_BYTES},
   { "CKA_EXPONENT_2", CKA_EXPONENT_2, FT_BYTES},
   { "CKA_COEFFICIENT", CKA_COEFFICIENT, FT_BYTES},
   { "CKA_PRIME", CKA_PRIME, FT_BYTES},
   { "CKA_SUBPRIME", CKA_SUBPRIME, FT_BYTES},
   { "CKA_BASE", CKA_BASE, FT_BYTES},
   { "CKA_VALUE_BITS", CKA_VALUE_BITS, FT_BYTES},
   { "CKA_VALUE_LEN", CKA_VALUE_LEN, FT_ULONG},
   { "CKA_EXTRACTABLE", CKA_EXTRACTABLE, FT_BOOL},
   { "CKA_LOCAL", CKA_LOCAL, FT_BOOL},
   { "CKA_NEVER_EXTRACTABLE", CKA_NEVER_EXTRACTABLE, FT_BOOL},
   { "CKA_ALWAYS_SENSITIVE", CKA_ALWAYS_SENSITIVE, FT_BOOL},
   { "CKA_MODIFIABLE", CKA_MODIFIABLE, FT_BOOL},
   { "CKA_VENDOR_DEFINED", CKA_VENDOR_DEFINED, FT_BYTES},
   { "", 0, 0}
};
CK_OBJECT_HANDLE hObject;
CK_ULONG found = 0, row, j, len;
/* Careful: assume all attrbutes are less than sizeof(szBuffer) */
char szBuffer[4096];
CK_ATTRIBUTE a = { 0, szBuffer, 0 };
CK_BYTE_PTR v;
int i;
int nTotal = 0;
```

```
   printf("Dumping all objects...\n");

   /* Find *all* objects  (NULL_PTR means all objects) */
   if ((rv = (*p->C_FindObjectsInit) (hSession, NULL_PTR, 0)) != CKR_OK)
   {
      CKRLOG("C_FindObjectsInit", rv);
      return;
   }
   do
   {
     /*---- Locate the next object ----*/
      if ((rv = (*p->C_FindObjects) (hSession, &hObject, 1, &found))
         != CKR_OK)
      {
         CKRLOG("C_FindObjects", rv);
         return;
      }

      if (found == 1)
      {
         nTotal++;
         /* Write out a header */
         printf
            ("\n\n\n================== Begin Object %d
===============================\n",
            nTotal);
       /*---- Loop for all possible attributes ----*/
         for (i = 0; (ATypes[i].nFormat != 0); i++)
         {
            /* Load up the attribute which interests us...  */
            a.type = ATypes[i].ulType;
            a.ulValueLen = sizeof(szBuffer);

            rv = (*p->C_GetAttributeValue) (hSession, hObject, &a, 1);

         /*---- Print out the attribute ----*/
            switch (rv)
            {
            case CKR_OK:/* Attribute found */
               printf("%-22s", ATypes[i].pszName);
               switch (ATypes[i].nFormat)
               {
               case FT_ULONG:
                  printf("%lu\n", ((CK_ULONG_PTR) a.pValue)[0]);
                  break;
               case FT_BYTES:
                  /* print out a byte buffer in 16 byte blocks */
                  /* (ugly code but uninteresting) */
                  printf("(%lu bytes)\n", a.ulValueLen);
                  v = (CK_BYTE_PTR) a.pValue;
                  for (row = 0; row < a.ulValueLen; row += 16)
                  {
                     printf(" >  ");
                     len = min(a.ulValueLen - row, 16);
                     for (j = row; j < row + len; j++)
                        printf("%02X ", (int) v[j]);
                     for (j = row + len; j < row + 16; j++)
                        printf("   ");
                     for (j = row; j < row + len; j++)
                        printf("%c", (isprint(v[j]) ? v[j] : '.'));
                     printf("\n");
                  }
                  break;
               case FT_BOOL:
                  printf((((CK_BBOOL *) a.pValue)[0]) ?
                     "TRUE\n" : "FALSE\n");
                  break;
               default:
                  break;
               }
               break;
            case CKR_ATTRIBUTE_SENSITIVE:
               /* Opps! we're not allowed to see it */
               printf("%-22s<sensitive>\n", ATypes[i].pszName);
               break;
            case CKR_ATTRIBUTE_TYPE_INVALID:
               /* This attrbute doesn't exist for this object */
               break;
            default:
               CKRLOG("C_GetAttributeValue", rv);
               /* A real error has occured. */
               return;
            }
         }
         /* Write out a footer */
         printf
            ("================== End Object %d
=================================\n",
            nTotal);
#ifdef WIN32
         printf("Press a key to continue...");
         getch();
#endif
      }
   }
   while (found == 1);
   printf("\nTotal number of object(s) found = %d\n", nTotal);
   return;
}
```

enroll.c

This sample performs all the operations needed to enroll a certificate.

```
/* include the common code */
#define RW_SESSION
#include "main.c"
#include <stdarg.h>

static char szBase64[] =
   "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

#define MAX_KEY_BITS   (1024)
CK_CHAR szKeyId[64] = "Key";
CK_OBJECT_HANDLE hPrivateKey, hPublicKey, hCert;
FILE *fp;
char szFileName[256];
CK_CHAR szLabel[] = "My ID";
char Buffer[4096];
CK_BYTE KeyID[128];
CK_ULONG ulLen;
CK_BBOOL bTrue = TRUE;
CK_OBJECT_CLASS cert_object_class = CKO_CERTIFICATE;
```

```
CK_CERTIFICATE_TYPE certType = CKC_X_509;
CK_ATTRIBUTE CertTemplate[] = {
   {CKA_CLASS, &cert_object_class, sizeof(CK_OBJECT_CLASS)} ,
   {CKA_TOKEN, &bTrue, sizeof(CK_BBOOL)} ,
   {CKA_CERTIFICATE_TYPE, &certType, sizeof(certType)} ,
   {CKA_ID, szKeyId, sizeof(szKeyId) - 1} ,
   {CKA_LABEL, szLabel, sizeof(szLabel) - 1} ,
   {CKA_VALUE, Buffer, sizeof(Buffer)} ,
   {CKA_SUBJECT, (char *)"(no subject)", 12} ,
};
/******************************************************************************
*     BEGIN PKCS#10 FUNCTIONS
*
* These functions and macros are used to format a certificate request. They are
* included for information only.
******************************************************************************/
typedef struct
{
   CK_ULONG ulLen;
   CK_BYTE_PTR pData;
} ADATA, *ADATA_PTR;

#define TAG_INTEGER                 (0x02)
#define TAG_BIT_STRING              (0x03)
#define TAG_OID                     (0x06)
#define TAG_PrintableString         (0x13)
#define TAG_SEQUENCE                (0x30)
#define TAG_SET                     (0x31)
#define O_rsaEncryption          "\x06\x09\x2A\x86\x48\x86\xF7\x0D\x01\x01\x01"
#define O_SHA1withRSAEncryption   "\x06\x09\x2A\x86\x48\x86\xF7\x0D\x01\x01\
x05"
#define O_emailAddress           "\x06\x09\x2A\x86\x48\x86\xF7\x0D\x01\x09\x01"
#define O_commonName              "\x06\x03\x55\x04\x03"
#define O_country                 "\x06\x03\x55\x04\x06"
#define O_organization            "\x06\x03\x55\x04\x0A"
#define O_organizationalUnit      "\x06\x03\x55\x04\x0B"
/* void FreeADATA(ADATA_PTR a) */
#define FreeADATA(x)     {if(x) { if((x)->pData) free((x)->pData); free(x);}}
#define end              (NULL)
/* ADATA_PTR MakeADATA(CK_BYTE_PTR pData, CK_ULONG ulLen) */
#define MakeADATA(x,y)  CreateADATA(x,y,y)
/* ADATA_PTR tlvaz(CK_BYTE Tag, ADATA_PTR a) */
#define tlvaz(x,y)       var_tlv(x, MakeADATA("\x00", 1), y, NULL)
/* ADATA_PTR tlv(CK_BYTE Tag, ADATA_PTR a) */
#define tlv(x,y)         var_tlv(x, y, NULL)
/* ADATA_PTR NULL_DER */
#define NULL_DER         MakeADATA("\x05\x00", 2)
/* ADATA_PTR BIT_STRING_encapsulates(ADATA_PTR a) */
#define BIT_STRING_encapsulates(x) tlvaz(TAG_BIT_STRING, x)
/* ADATA_PTR BIT_STRING(CK_BYTE_PTR pData, CK_ULONG ulLen) */
#define BIT_STRING(x,y)  BIT_STRING_encapsulates(MakeADATA(x, y))
/* ADATA_PTR Printable(CK_CHAR_PTR pszString) */
#define Printable(x)     tlv(TAG_PrintableString, MakeADATA(x, strlen(x)))
/* ADATA_PTR CONTEXT_SPECIFIC(CK_BYTE Tag, ADATA_PTR a) */
#define CONTEXT_SPECIFIC(x, y) tlv((CK_BYTE)(x | 0xA0), y)
/* ADATA_PTR OID(CK_CHAR_PTR x) */
#define OID(x)         MakeADATA(x, strlen(x))

static ADATA_PTR CreateADATA(CK_BYTE_PTR pData,
   CK_ULONG ulAllocLen, CK_ULONG ulCopyLen)
{
   ADATA_PTR a;
   if ((a = calloc(sizeof(ADATA), 1)) == NULL)
      return NULL;
   if (ulAllocLen != 0)
   {
      if ((a->pData = calloc(ulAllocLen, 1)) == NULL)
         return NULL;
      if ((ulCopyLen != 0) && (pData != NULL))
         memcpy(a->pData, pData, ulCopyLen);
   }
   a->ulLen = ulAllocLen;
   return a;
}

static ADATA_PTR ConcatenateADATA(ADATA_PTR a1, ADATA_PTR a2)
{
   ADATA_PTR ptr;

   if (a1 == NULL)
      a1 = CreateADATA(NULL, 0, 0);
   if (a2 == NULL)
      a2 = CreateADATA(NULL, 0, 0);

   if ((ptr = CreateADATA(a1->pData, a1->ulLen + a2->ulLen,
         a1->ulLen)) == NULL)
      return NULL;
   if (a2->ulLen)
      memcpy(&(ptr->pData[a1->ulLen]), a2->pData, a2->ulLen);

   FreeADATA(a1);
   FreeADATA(a2);
   return ptr;
}

static ADATA_PTR ConstructADATA(CK_BYTE Tag, ADATA_PTR pFirst, va_list m)
{

   ADATA_PTR pNext, pCurrent = NULL;
   CK_ULONG i = 0;
   CK_BYTE Buff[8];

   for (pNext = pFirst; pNext != NULL; pNext = va_arg(m, ADATA_PTR))
      pCurrent = ConcatenateADATA(pCurrent, pNext);

   Buff[i++] = (CK_BYTE) Tag;

   if (pCurrent == NULL)
      Buff[i++] = 0x00;
   else if (pCurrent->ulLen < 0x80)
      Buff[i++] = (CK_BYTE) pCurrent->ulLen;
   else if (pCurrent->ulLen < 0x100)
   {
      Buff[i++] = 0x81;
      Buff[i++] = (CK_BYTE) pCurrent->ulLen;
   }
   else if (pCurrent->ulLen < 0x10000)
   {
      Buff[i++] = 0x82;
      Buff[i++] = (CK_BYTE) (pCurrent->ulLen >> 8) & 0xFF;
      Buff[i++] = (CK_BYTE) pCurrent->ulLen & 0xFF;
   }
   return ConcatenateADATA(MakeADATA(Buff, i), pCurrent);
```

```
}

static ADATA_PTR var_tlv(CK_BYTE Tag, ADATA_PTR pFirst, ...)
{
   va_list m;
   va_start(m, pFirst);
   return ConstructADATA(Tag, pFirst, m);
}

static ADATA_PTR _INTEGER(CK_BYTE_PTR pInteger, CK_ULONG ulLen_local)
{
   if (pInteger[0] >= 0x80)
      return tlvaz(TAG_INTEGER, MakeADATA(pInteger, ulLen_local));
   else
      return tlv(TAG_INTEGER, MakeADATA(pInteger, ulLen_local));
}

static ADATA_PTR SET(ADATA_PTR pFirst, ...)
{
   va_list m;
   va_start(m, pFirst);
   return ConstructADATA(TAG_SET, pFirst, m);
}

static ADATA_PTR SEQUENCE(ADATA_PTR pFirst, ...)
{
   va_list m;
   va_start(m, pFirst);
   return ConstructADATA(TAG_SEQUENCE, pFirst, m);
}

/*******************************************************************************
*     END PKCS#10 FUNCTIONS
*******************************************************************************/

/*******************************************************************************
* DeleteAllObjects
*
* As its name implies, this deletes all the objects in the card. See sample
* deleteall.c from which this is pasted for more details
*******************************************************************************/
static CK_BBOOL DeleteAllObjects(void)
{
   CK_OBJECT_HANDLE hObject;
   CK_ULONG found;
   char szBuffer[8];

   /*---- Get user confirmation (this will, after all, erase everything)----*/
   printf("\nThis will delete ALL exisiting certificates or outstanding\n\
certificate requests in the token. Continue? [y|n]: ");
   fgets(szBuffer, sizeof(szBuffer), stdin);
   if (strncmp(szBuffer, "y", 1))
      return FALSE;

   do
   {
     /*---- Search for an object ----*/
      if ((rv = (*p->C_FindObjectsInit) (hSession, NULL_PTR, 0)) != CKR_OK)
      {
         CKRLOG("C_FindObjectsInit", rv);
         return FALSE;
      }
```

```
         if ((rv = (*p->C_FindObjects) (hSession, &hObject, 1,
                  &found)) != CKR_OK)
         {
            CKRLOG("C_FindObjects", rv);
            return FALSE;
         }
         if ((rv = (*p->C_FindObjectsFinal) (hSession)) != CKR_OK)
         {
            CKRLOG("C_FindObjectsFinal", rv);
            return FALSE;
         }

         if (found == 1)
         {
          /*---- It's curtains for the object----*/
            if ((rv = (*p->C_DestroyObject) (hSession, hObject)) != CKR_OK)
            {
               CKRLOG("C_DestroyObject", rv);
               return FALSE;
            }
         }
      }
      while (found == 1);

      return TRUE;
   }

   /*****************************************************************************
   * SignIt
   *
   * Signs some data. This is required because the certificate request needs to be
   * self signed.
   *
   * Ensure that pulSignatureLen is already initialized before calling this
   * function.
   *****************************************************************************/
   static void SignIt(CK_BYTE_PTR pData,
      CK_ULONG ulDataLen,
      CK_BYTE_PTR pSignature,
      CK_ULONG_PTR pulSignatureLen, CK_OBJECT_HANDLE hPrivateKey_local)
   {
      CK_MECHANISM Mechanism = { 0, NULL_PTR, 0 };
      CK_BYTE Hash[64];
      CK_ULONG ulHashLen = sizeof(Hash);

      /*---- Hash the data ----*/
      Mechanism.mechanism = CKM_SHA_1;
      if ((rv = (*p->C_DigestInit) (hSession, &Mechanism)) != CKR_OK)
      {
         CKRLOG("C_DigestInit", rv);
         return;
      }
      if ((rv = (*p->C_Digest) (hSession, pData, ulDataLen, Hash, &ulHashLen))
         != CKR_OK)
      {
         CKRLOG("C_Digest", rv);
         return;
      }

      /*---- Sign the hash ----*/
      Mechanism.mechanism = CKM_RSA_PKCS;
      if ((rv = (*p->C_SignInit) (hSession, &Mechanism, hPrivateKey_local)) !=
   CKR_OK)
      {
         CKRLOG("C_SignInit", rv);
         return;
      }
      if ((rv = (*p->C_Sign) (hSession, Hash, ulHashLen,
               pSignature, pulSignatureLen)) != CKR_OK)
      {
         CKRLOG("C_Sign", rv);
         return;
      }
   }

   /*****************************************************************************
   * RequestCertificate
   *
   * The steps in the certificate request process are:
   *
   * - Generate an RSA key pair
   * - Read the modulus (public key value) from the token
   * - as user for the name to be put in the request
   * - Format the ToBeSigned part of the request which contains the name and
   *   public key
   * - sign ToBeSigned with the private key
   * - append new signature to ToBeSigned
   * - Convert to Base64 and write request to a text file
   *
   *****************************************************************************/
   static void RequestCertificate(void)
   {
      CK_ULONG mod_bits = 1024;

      CK_ATTRIBUTE GenPubTemplate[] = {
         {CKA_MODULUS_BITS, &mod_bits, sizeof(CK_ULONG)} ,
         {CKA_PUBLIC_EXPONENT, (char *)"\x01\x00\x01", 3} ,
         {CKA_TOKEN, &bTrue, sizeof(CK_BBOOL)} ,
         {CKA_ID, szKeyId, 3}
      };
      CK_ATTRIBUTE GenPrivTemplate[] = {
         {CKA_TOKEN, &bTrue, sizeof(CK_BBOOL)} ,
         {CKA_PRIVATE, &bTrue, sizeof(CK_BBOOL)} ,
         {CKA_SENSITIVE, &bTrue, sizeof(CK_BBOOL)} ,
         {CKA_ID, szKeyId, 3}
      };
      CK_BYTE modulus[MAX_KEY_BITS / 8];
      CK_ATTRIBUTE Modulus = { CKA_MODULUS, modulus, sizeof(modulus) };

      CK_BYTE SignBuffer[MAX_KEY_BITS / 8];
      CK_ULONG ulSignLen = sizeof(SignBuffer);

      CK_MECHANISM Mechanism = { CKM_RSA_PKCS_KEY_PAIR_GEN, NULL_PTR, 0 };
      unsigned int i;
      ADATA_PTR pPKCS10req, ToBeSigned;
      typedef struct
      {
         const char *oid;
         char szValue[64];
         const char *text;
      } LTABLE;
      CK_ULONG j, nCertLen;
      CK_BYTE_PTR pCert;
      LTABLE l[] = {
         {O_commonName, "", "name (e.g. Fred Bloggs):"},
         {O_emailAddress, "", "email address (e.g. fred@acme.com):"},
         {O_organizationalUnit, "",
               "organisation unit (e.g. Cleaning Dept.):"},
         {O_organization, "", "organisation (e.g. ACME Inc.):"},
         {O_country, "", "2 letter ISO country code (e.g. US, FR):"}
      };

      /*---- First we generate the key pair ----*/
      if ((rv = (*p->C_GenerateKeyPair) (hSession, &Mechanism,
               GenPubTemplate, 4,
               GenPrivTemplate, 4, &hPublicKey, &hPrivateKey)) != CKR_OK)
      {
         CKRLOG("C_GenerateKeyPair", rv);
         return;
      }
      /* Read modulus */
      if ((rv = (*p->C_GetAttributeValue) (hSession,
               hPrivateKey, &Modulus, 1)) != CKR_OK)
      {
         CKRLOG("C_GetAttributeValue", rv);
         return;
      }

      /*---- Create PKCS#10 Certificate Request ----*/

      /* what is the user's name ? */
      for (i = 0; i < sizeof(l) / sizeof(LTABLE); i++)
      {
         printf("Your %s", l[i].text);
         fgets(l[i].szValue, sizeof(l[i].szValue), stdin);
      }

      /* create the ToBeSigned part of the request */
      ToBeSigned = SEQUENCE(_INTEGER("\x00", 1),// Version
         SEQUENCE(      // SubjectName
            SET(SEQUENCE(OID(l[0].oid), Printable(l[0].szValue), (end)), (end)),
   SET(SEQUENCE(OID(l[1].oid), Printable(l[1].szValue), (end)), (end)),
   SET(SEQUENCE(OID(l[2].oid), Printable(l[2].szValue), (end)), (end)),
   SET(SEQUENCE(OID(l[3].oid), Printable(l[3].szValue), (end)), (end)),
   SET(SEQUENCE(OID(l[4].oid), Printable(l[4].szValue), (end)), (end)), (end)),
   SEQUENCE(// SubjectPublicKeyInfo
            SEQUENCE(OID(O_rsaEncryption), NULL_DER, (end)),
   BIT_STRING_encapsulates(// subjectPublicKey
               SEQUENCE(_INTEGER(Modulus.pValue, Modulus.ulValueLen), _INTEGER("\
   x01\x00\x01", 3), (end))), (end)), CONTEXT_SPECIFIC(0, NULL),/* attributes */
         (end));

      /* sign the request */
      SignIt(ToBeSigned->pData,
         ToBeSigned->ulLen, SignBuffer, &ulSignLen, hPrivateKey);
      if (rv != CKR_OK)
         return;

      /* Append signature */
      pPKCS10req =
         SEQUENCE(ToBeSigned,
         SEQUENCE(OID(O_SHA1withRSAEncryption), NULL_DER, (end)),
         BIT_STRING(SignBuffer, ulSignLen), (end));

      /* write request to file in Base64 format */
      printf("File in which to put certificate request:");
      fgets(szFileName, sizeof(szFileName), stdin);
      if (!strcmp(szFileName, ""))
         return;
      if ((fp = fopen(szFileName, "w")) == NULL)
      {
         printf("Error: Could not open %s\n", szFileName);
         return;
      }

      nCertLen = pPKCS10req->ulLen;
      pCert = pPKCS10req->pData;

      /* Convert to Base64 */
      fprintf(fp, "-----BEGIN CERTIFICATE REQUEST-----");
      for (j = 0; j < (nCertLen - nCertLen % 3); j += 3)
      {
         if (j % 48 == 0)
            fputc('\n', fp);
         fputc(szBase64[0x3F & (pCert[j] >> 2)], fp);
         fputc(szBase64[0x3F & ((pCert[j] << 4) + (pCert[j + 1] >> 4))], fp);
         fputc(szBase64[0x3F & ((pCert[j + 1] << 2) + (pCert[j + 2] >> 6))],
            fp);
         fputc(szBase64[0x3F & (pCert[j + 2])], fp);
      }

      /* Deal with the end conditions */
      if ((nCertLen % 3) == 1)
      {
         if (j % 48 == 0)
            fputc('\n', fp);
         fputc(szBase64[0x3F & (pCert[j] >> 2)], fp);
         fputc(szBase64[0x3F & (pCert[j] << 4)], fp);
         fputc('=', fp);
         fputc('=', fp);
      }
      else if ((nCertLen % 3) == 2)
      {
         if (j % 48 == 0)
            fputc('\n', fp);
         fputc(szBase64[0x3F & (pCert[j] >> 2)], fp);
         fputc(szBase64[(0x3F & (pCert[j] << 4)) + (pCert[j + 1] >> 4)], fp);
         fputc(szBase64[0x3F & (pCert[j + 1] << 2)], fp);
         fputc('=', fp);
      }
      fprintf(fp, "\n-----END CERTIFICATE REQUEST-----\n");

      fclose(fp);
      FreeADATA(pPKCS10req);/* Important! pToBeSigned has already been freed */

      printf("Certificate request complete!\n\
   Send to a CA and restart this program when you receive the certificate.");
   }

   /*******************************************************************************
   * InstallCertificate
   *
   * Following a request, the CA will send us a certificate. We need to install
   * it in our token.
   *
   *******************************************************************************/
   static void InstallCertificate(void)
```

```
{
   int d;
   CK_BYTE modulus[MAX_KEY_BITS / 8];
   CK_ULONG j, k;
   int i = 2;
   CK_ATTRIBUTE Modulus = { CKA_MODULUS, modulus, sizeof(modulus) };

   printf("File which contains the certificate sent to you by CA:");
   fgets(szFileName, sizeof(szFileName), stdin);
   if (!strcmp(szFileName, ""))
      return;
   if ((fp = fopen(szFileName, "r")) == NULL)
   {
      printf("Error: Could not open %s\n", szFileName);
      return;
   }

   /* find block start */
   while (fgets(Buffer, sizeof(Buffer), fp) != NULL)
      if (!memcmp(Buffer, "----", 4))
         break;
   /* Read in the certificate and convert from base64 */
   memset(Buffer, 0, sizeof(Buffer));

   do
   {
      char *p_local;
      if (((d = fgetc(fp)) == EOF) ||/* (unexpected end of file) */
         (ulLen == sizeof(Buffer) - 1))/* (overflow) */
      {
         printf("Error: Badly formatted certificate file.\n");
         return;
      }

      if ((p_local = strchr(szBase64, (char) d)) != NULL)
      {
         if (i != 2)
            Buffer[ulLen - 1] |= (p_local - szBase64) >> (8 - i) % 8;
         if (i)
         {
            Buffer[ulLen] |= (p_local - szBase64) << i;
            ulLen++;
         }
         i = (i + 2) % 8;
      }
   }
   while (strchr("-=", (char) d) == NULL);/* (footer found) */

   ulLen--;

   /* We just check that the certificate contains the modulus somewhere.
   */

   if ((rv = (*p->C_GetAttributeValue) (hSession,
            hPrivateKey, &Modulus, 1)) != CKR_OK)
   {
      CKRLOG("C_GetAttributeValue", rv);
      return;
   }

   for (j = 0, k = 0; (j < ulLen) && (k < Modulus.ulValueLen); j++)
   {
      if (Buffer[j] == ((CK_BYTE_PTR) Modulus.pValue)[k])
         k++;
      else
         k = 0;
   }
   if (k != Modulus.ulValueLen)
   {
      printf
         ("\nError: Certificate does not correspond to previous request");
      return;
   }

   CertTemplate[5].ulValueLen = ulLen;
   if ((rv = (*p->C_CreateObject)
         (hSession, CertTemplate, 7, &hCert)) != CKR_OK)
   {
      CKRLOG("C_CreateObject", rv);
      return;
   }

   printf("\nCertificate successfully installed!");
}

/******************************************************************************
* void SampleFunction(void)
*
*
******************************************************************************/
void SampleFunction(void)
{
#define CERTIFICATE_REQUEST   (1)
#define CERTIFICATE_INSTALL   (2)

   int nChoice;
   char szChoice[32];
   const char *pszState;
   CK_ULONG count;
   CK_OBJECT_CLASS priv_object_class = CKO_PRIVATE_KEY;
   CK_KEY_TYPE key_type = CKK_RSA;
   CK_ATTRIBUTE KeyTemplate[] = {
      {CKA_CLASS, &priv_object_class, sizeof(CK_OBJECT_CLASS)}
      ,
      {CKA_KEY_TYPE, &key_type, sizeof(CK_KEY_TYPE)}
   };
   char szGreeting[] = "\
(c) 2008 Gemalto Development\n\
\n\
Enroll allows you to:\n\
\n\
   1. Request a new certificate\n\
   2. Install a certificate\n\
\n\
Normally you should select 1 to request a certificate and then send\n\
the request to a Certification Authority(CA). The CA will issue you\n\
with a certificate and you should restart this program and select\n\
option 2 to install the new certificate in your token.\n\
\n\
[Current token state: %s]\n\
\n\
Enter your choice: ";
```

```
/*---- Find an RSA Private key ----*/
if ((rv = (*p->C_FindObjectsInit) (hSession, KeyTemplate, 2)) != CKR_OK)
{
   CKRLOG("C_FindObjectsInit", rv);
   return;
}
if ((rv =
      (*p->C_FindObjects) (hSession, &hPrivateKey, 1,
         &count)) != CKR_OK)
{
   CKRLOG("C_FindObjects", rv);
   return;
}
if ((rv = (*p->C_FindObjectsFinal) (hSession)) != CKR_OK)
{
   CKRLOG("C_FindObjectsFinal", rv);
   return;
}
if (count != 1)
   hPrivateKey = NULL_PTR;
/* Update the CK_ID field in the certificate template */
else if ((rv =
      (*p->C_GetAttributeValue) (hSession, hPrivateKey,
         &CertTemplate[3], 1)) != CKR_OK)
{
   CKRLOG("C_GetAttributeValue", rv);
   return;
}

/*---- Find a corresponding certificate ----*/
if ((rv = (*p->C_FindObjectsInit) (hSession, CertTemplate, 4)) != CKR_OK)
{
   CKRLOG("C_FindObjectsInit", rv);
   return;
}
if ((rv = (*p->C_FindObjects) (hSession, &hCert, 1, &count)) != CKR_OK)
{
   CKRLOG("C_FindObjects", rv);
   return;
}
if ((rv = (*p->C_FindObjectsFinal) (hSession)) != CKR_OK)
{
   CKRLOG("C_FindObjectsFinal", rv);
   return;
}
if (count != 1)
   hCert = NULL_PTR;


/*---- Display greeting ----*/
if (hPrivateKey == NULL_PTR)
   pszState = "Empty.";
else if (hCert != NULL_PTR)
   pszState = "Certificate already installed.";
else
   pszState = "Certificate request made.";
printf(szGreeting, pszState);
fgets(szChoice, sizeof(szChoice), stdin);
if (sscanf(szChoice, "%d", &nChoice) != 1)
   return;

/*---- Delete existing objects if necessary ----*/
```

```
   if ((nChoice == CERTIFICATE_REQUEST) && (hPrivateKey != NULL_PTR))
   {
      if (DeleteAllObjects())
      {
         hPrivateKey = NULL_PTR;
         hCert = NULL_PTR;
      }
      else
         return;
   }
   if ((nChoice == CERTIFICATE_INSTALL) && (hPrivateKey == NULL_PTR))
   {
      printf
         ("\nError: Can not install certificate because request has not been
made");
      return;
   }
   if ((nChoice == CERTIFICATE_INSTALL) && (hCert != NULL_PTR))
   {
      printf
         ("\nError: A certificate has already been installed in this token");
      return;
   }

   if (nChoice == CERTIFICATE_REQUEST)
      RequestCertificate();
   else if (nChoice == CERTIFICATE_INSTALL)
      InstallCertificate();
}
```

genkey.c

This sample demonstrates the generation of an RSA key pair. Firstly the bit size of the key is entered by the user. The key generation uses the minimum templates, that is, containing only the mandatory parameters. The Modulus is then printed out. The user is assumed to have previously logged in (C_Login) and the card must have a key container for the requested key size.

```
/* include the common code */
#include "main.c"

/*****************************************************************************
* void SampleFunction(void)
*
*****************************************************************************/
void SampleFunction(void)
{
   char szSize[10];
   CK_BYTE Buffer[512];
   CK_ULONG i, ulSize;
   CK_OBJECT_HANDLE hPubKey, hPrivKey;
   CK_BBOOL bTrue = TRUE;
   CK_MECHANISM Mechanism = { 0, NULL_PTR, 0 };
   CK_ATTRIBUTE PubTemplate[] = {
      { CKA_MODULUS_BITS, &ulSize, sizeof(CK_ULONG)} ,
      { CKA_PUBLIC_EXPONENT, "\x01\x00\x01", 3} ,
      { CKA_TOKEN, &bTrue, sizeof(CK_BBOOL)} };
   CK_ATTRIBUTE PrivTemplate[] = {
      { CKA_TOKEN, &bTrue, sizeof(CK_BBOOL)} ,
      { CKA_PRIVATE, &bTrue, sizeof(CK_BBOOL)} ,
      { CKA_SENSITIVE, &bTrue, sizeof(CK_BBOOL)} };
   CK_ATTRIBUTE GetModulusTemplate[] = {
      { CKA_MODULUS, Buffer, sizeof(Buffer)} };

   /*---- Get the RSA key size ----*/
   printf("\nEnter RSA key bit size: ");
   i = 0;
   fgets(szSize, sizeof(szSize), stdin);
   ulSize = atoi(szSize);
   printf("\n\nGenerating RSA key pair, please wait...\n\n");

   /*---- Set the RSA key generation mechanism ----*/
   Mechanism.mechanism = CKM_RSA_PKCS_KEY_PAIR_GEN;
   Mechanism.pParameter = NULL_PTR;
   Mechanism.ulParameterLen = 0;

   /*---- Generate RSA key pair ----*/
   rv = (*p->C_GenerateKeyPair) (hSession,// Session handle
      &Mechanism,    // RSA Key Gen. mechanism
      PubTemplate,// Template for RSA Public key
      3,                // Attributes in previous template
      PrivTemplate,  // Template for RSA Private key
      3,                // Attributes in previous template
      &hPubKey,      // Handle of Public key, returned
      &hPrivKey      // Handle of Private key, returned
      );
   if (rv != CKR_OK)
      return;

   /*---- Display Modulus Value ----*/
   rv = (*p->C_GetAttributeValue) (hSession,// Session handle
      hPubKey,       // Handle of Public Key
      GetModulusTemplate,// Modulus template
      1                 // Number of attributes
      );
   if (rv != CKR_OK)
      return;

   printf("Modulus value: ");
   for (i = 0; i < GetModulusTemplate[0].ulValueLen; i++)
      printf("%02X", Buffer[i]);
   printf("\n\n");
}
```

loadkey.c

This sample demonstrates the loading of an RSA key pair. The key to load is stored as static variables. The private key part is created and loaded, then the public key part is created and loaded. The user is assumed to have previously logged in (C_Login) and the card must have a key container for the requested key size. The “dumpit” sample (see “dumpit.c” on page 51) can be used to check the key pair is correctly created.

```
/* include the common code */
#define RW_SESSION
#include "main.c"
/******************************************************************************
* void SampleFunction(void)
*
******************************************************************************/
void SampleFunction(void)
{
   // RSA 1024 bit static key
```

```
static unsigned char RSA_MODULUS[] = {
   0xe2, 0xde, 0xaa, 0x0f, 0x43, 0x47, 0x5d, 0xc3, 0x6a, 0xbd, 0xf9,
   0xae, 0xd9, 0x76, 0x94, 0xde, 0x7a, 0xdc, 0xef, 0x01, 0x53, 0xb0,
   0x1d, 0x47, 0x01, 0xcc, 0x3d, 0x62, 0xe7, 0x77, 0xf8, 0x1b, 0xd7,
   0xc8, 0xc4, 0x11, 0x38, 0xa5, 0x5c, 0x69, 0x05, 0xce, 0x36, 0x2c,
   0x34, 0xeb, 0x18, 0x8b, 0x41, 0x74, 0x07, 0x09, 0xc6, 0xd1, 0xb4,
   0xd6, 0xc5, 0xb8, 0x20, 0x66, 0xad, 0xa9, 0x41, 0x4b, 0x10, 0x1f,
   0xe4, 0x0f, 0x37, 0x40, 0xe5, 0x43, 0x7b, 0x6b, 0x84, 0x21, 0x28,
   0xd7, 0x90, 0xe1, 0x6d, 0x5a, 0xa4, 0x02, 0xbe, 0xc6, 0x24, 0x16,
   0x07, 0xec, 0x83, 0x5d, 0xcf, 0x5d, 0x54, 0xe3, 0x69, 0x83, 0xc7,
   0x83, 0x83, 0xbc, 0xa0, 0x4f, 0x67, 0x24, 0x8f, 0x13, 0xb9, 0x24,
   0xb3, 0xa2, 0xfa, 0xf5, 0x2e, 0x5f, 0x85, 0xa3, 0x70, 0xdc, 0xa3,
   0x67, 0xcc, 0xce, 0x44, 0xc1, 0xb1, 0x83
};
static unsigned char RSA_PUBLIC_EXPONENT[] = {
   0x01, 0x00, 0x01
};
static unsigned char RSA_PRIVATE_EXPONENT[] = {
   0x3c, 0xec, 0x2c, 0x60, 0xc2, 0xe1, 0x64, 0x45, 0x78, 0xe1, 0xa1,
   0x2e, 0x1a, 0x09, 0xa4, 0xfa, 0x85, 0xa5, 0xd4, 0xac, 0xd7, 0x8b,
   0x60, 0xa1, 0x53, 0xd3, 0x43, 0xdc, 0xce, 0x69, 0xc1, 0xff, 0xc0,
   0x17, 0x92, 0xc7, 0x49, 0x1d, 0xe6, 0xcd, 0xf1, 0x18, 0x2a, 0x25,
   0xfe, 0xe3, 0xef, 0x08, 0x5e, 0x40, 0x49, 0x2f, 0x8f, 0xeb, 0x7a,
   0x93, 0x7e, 0x2e, 0xee, 0xcc, 0x83, 0xf0, 0x02, 0xaf, 0x23, 0x1d,
   0x2f, 0x02, 0x94, 0x31, 0x4c, 0x1f, 0x63, 0xf8, 0x13, 0xa5, 0x9d,
   0x72, 0xa1, 0xaa, 0xe0, 0x1e, 0xec, 0x77, 0x1e, 0x4c, 0xf4, 0xb8,
   0xc3, 0x5e, 0x30, 0xa8, 0xd3, 0xa8, 0x8f, 0xa3, 0xa7, 0x13, 0x67,
   0xe5, 0x9a, 0x5f, 0x7f, 0xcf, 0xf5, 0x36, 0x6d, 0x5a, 0xa3, 0xef,
   0xde, 0x86, 0x76, 0xb7, 0xd9, 0x78, 0x97, 0x39, 0xe4, 0x2f, 0x71,
   0xc3, 0x76, 0xa6, 0x8a, 0x87, 0xaa, 0xd9
};
static unsigned char RSA_PRIME_1[] = {
   0xf8, 0x29, 0x88, 0x48, 0xc0, 0x44, 0xec, 0x01, 0xd2, 0xcd, 0xb2,
   0xbb, 0x54, 0xfb, 0x8e, 0x28, 0xc7, 0xcd, 0x1a, 0xe1, 0x5b, 0x49,
   0xc7, 0x30, 0x08, 0xcb, 0x6a, 0xe9, 0xaa, 0xd6, 0xb8, 0xa4, 0xf1,
   0x81, 0x37, 0x89, 0x9e, 0xf7, 0xc4, 0x88, 0xcf, 0xed, 0xca, 0xc1,
   0x7c, 0x0d, 0x57, 0x2d, 0xa2, 0xc4, 0x21, 0x49, 0xc4, 0x55, 0xf1,
   0xf2, 0x3a, 0x59, 0x89, 0x40, 0xcc, 0x85, 0x1a, 0x8f
};
static unsigned char RSA_PRIME_2[] = {
   0xea, 0x08, 0xf9, 0xd6, 0xae, 0x51, 0xf3, 0xbf, 0x7b, 0x8f, 0x52,
   0x26, 0xc0, 0xe6, 0xeb, 0x2d, 0x92, 0xa1, 0x8e, 0xaf, 0xbd, 0xee,
   0xf9, 0x03, 0x60, 0xf7, 0xde, 0xf7, 0x6e, 0xcb, 0x22, 0x2e, 0xdd,
   0x61, 0xd7, 0x48, 0x29, 0xca, 0x22, 0xd7, 0x9c, 0x05, 0xed, 0x40,
   0xe5, 0x2e, 0x40, 0x12, 0xe0, 0x8e, 0xf5, 0x8a, 0xee, 0x06, 0x88,
   0x74, 0x01, 0x1f, 0xb6, 0xa0, 0xd0, 0x48, 0x43, 0xcd
};
static unsigned char RSA_EXPONENT_1[] = {
   0x59, 0xfd, 0x0d, 0x8b, 0x7c, 0x15, 0x42, 0x13, 0xc3, 0x32, 0x8e,
   0x2e, 0x7a, 0x39, 0xa2, 0x2a, 0x4d, 0x91, 0xa5, 0xbe, 0xb4, 0x90,
   0xc3, 0x67, 0x3a, 0xb9, 0x30, 0xf2, 0xbc, 0xdd, 0x92, 0x25, 0x55,
   0xe2, 0x0a, 0x96, 0xa0, 0x0b, 0xae, 0xbc, 0x01, 0xd2, 0x73, 0xe4,
   0xc7, 0xac, 0x3a, 0xc1, 0xb4, 0xb5, 0x34, 0x71, 0x00, 0x7e, 0x3d,
   0xf7, 0x1d, 0x8b, 0x9c, 0x46, 0xe0, 0x23, 0x20, 0x27
};
static unsigned char RSA_EXPONENT_2[] = {
   0xd2, 0xb6, 0x6e, 0x2b, 0x6a, 0x1c, 0x03, 0x37, 0xfe, 0x09, 0x96,
   0x4f, 0xaa, 0x12, 0xbe, 0xdf, 0xf3, 0x4b, 0x5e, 0x6a, 0xc0, 0xb9,
   0x6e, 0x9e, 0x2a, 0x2a, 0x42, 0x27, 0xc9, 0x2b, 0x3f, 0x85, 0xae,
   0x71, 0x2f, 0x21, 0x9b, 0xee, 0x90, 0xdc, 0x4b, 0x6f, 0xd2, 0xa3,
   0x41, 0x35, 0x19, 0x1f, 0x65, 0xeb, 0x91, 0x0c, 0x1b, 0x2e, 0xea,
   0xa0, 0x77, 0x7a, 0x94, 0x35, 0xca, 0x11, 0xa6, 0x69
   };
   static unsigned char RSA_COEFFICIENT[] = {
      0x3d, 0x02, 0xde, 0xc3, 0x27, 0x1e, 0xa4, 0x87, 0x10, 0xb8, 0xb2,
      0x83, 0x18, 0x7d, 0x35, 0x59, 0x52, 0x94, 0xb0, 0xc5, 0x67, 0x44,
      0x97, 0x0c, 0xe2, 0x4f, 0x28, 0x7a, 0x86, 0x8d, 0x5d, 0xc1, 0x2e,
      0xf8, 0xab, 0xb5, 0x88, 0x55, 0x89, 0x37, 0xb5, 0x8c, 0xe4, 0x35,
      0x7e, 0xb9, 0xdb, 0x6f, 0x46, 0x0e, 0x07, 0xde, 0x74, 0xb6, 0x7a,
      0x1f, 0xb8, 0x6b, 0x6d, 0xe2, 0x89, 0x83, 0x09, 0x1b
   };
   CK_ULONG ulSize = 1024;
   CK_OBJECT_HANDLE hPubKey, hPrivKey;
   CK_BBOOL bTrue = TRUE;
   CK_KEY_TYPE key_type = CKK_RSA;
   CK_OBJECT_CLASS pub_object_class = CKO_PUBLIC_KEY;
   CK_ATTRIBUTE PubTemplate[] = {
      {CKA_CLASS, &pub_object_class, sizeof(CK_OBJECT_CLASS)} ,
      {CKA_KEY_TYPE, &key_type, sizeof(CK_KEY_TYPE)} ,
      {CKA_TOKEN, &bTrue, sizeof(CK_BBOOL)} ,
      {CKA_MODULUS, RSA_MODULUS, sizeof(RSA_MODULUS)} ,
      {CKA_PUBLIC_EXPONENT, RSA_PUBLIC_EXPONENT, sizeof(RSA_PUBLIC_EXPONENT)}
   };
   CK_OBJECT_CLASS priv_object_class = CKO_PRIVATE_KEY;
   CK_ATTRIBUTE PrivTemplate[] = {
      {CKA_CLASS, &priv_object_class, sizeof(CK_OBJECT_CLASS)} ,
      {CKA_KEY_TYPE, &key_type, sizeof(CK_KEY_TYPE)} ,
      {CKA_TOKEN, &bTrue, sizeof(CK_BBOOL)} ,
      {CKA_PRIVATE, &bTrue, sizeof(CK_BBOOL)} ,
      {CKA_SENSITIVE, &bTrue, sizeof(CK_BBOOL)} ,
      {CKA_MODULUS, RSA_MODULUS, sizeof(RSA_MODULUS)} ,
      {CKA_PUBLIC_EXPONENT, RSA_PUBLIC_EXPONENT, sizeof(RSA_PUBLIC_EXPONENT)} ,
      {CKA_PRIVATE_EXPONENT, RSA_PRIVATE_EXPONENT,
sizeof(RSA_PRIVATE_EXPONENT)} ,
      {CKA_PRIME_1, RSA_PRIME_1, sizeof(RSA_PRIME_1)} ,
      {CKA_PRIME_2, RSA_PRIME_2, sizeof(RSA_PRIME_2)} ,
      {CKA_EXPONENT_1, RSA_EXPONENT_1, sizeof(RSA_EXPONENT_1)} ,
      {CKA_EXPONENT_2, RSA_EXPONENT_2, sizeof(RSA_EXPONENT_2)} ,
      {CKA_COEFFICIENT, RSA_COEFFICIENT, sizeof(RSA_COEFFICIENT)}
   };
/*---- Create and load RSA private part ----*/
   printf("\n\nCreating & Loading RSA private key part, please wait...\n\n");
   rv = (*p->C_CreateObject) (hSession,// Session handle
      PrivTemplate,  // Template for RSA Private key
      13,               // Attributes in previous template
      &hPrivKey      // Handle of Private key, returned
      );
   if (rv != CKR_OK)
      return;

   /*---- Create and load RSA public part ----*/
   printf("\n\nCreating & Loading RSA public key part, please wait...\n\n");
   rv = (*p->C_CreateObject) (hSession,// Session handle
      PubTemplate,// Template for RSA Public key
      5,                // Attributes in previous template
      &hPubKey       // Handle of Public key, returned
      );
   if (rv != CKR_OK)
      return;
}
```

pincode.c

This sample demonstrates the PIN code management functions. In particular, the use of C_SetPin in order to change either the user or the Security Officer’s (SO) PIN and the use of the SO's PIN with the C_InitPin function to unblock the token. A token becomes blocked (unusable) if the user has incorrectly presented the User PIN more than 3 times. If the SO PIN is presented incorrectly 3 times, this will be blocked too and can never be unblocked. PIN codes are normally 4 digits.

```
/* This flag means that the main code will not log in the user at startup */
#define NO_LOGIN
#define RW_SESSION

/* include the common code */
#include "main.c"


#define fromhex(x) (x-((x>='0')&&(x<='9')?'0':((x>='A')&&(x<='F')?'7':'W')))


/******************************************************************************
* void SampleFunction(void)
*
******************************************************************************/
void SampleFunction(void)
{
   char szNewPinCode[49];
   char szConfirmNewPinCode[49];
   CK_USER_TYPE user_type;
   char line[80];
   CK_BBOOL bUnblock;
   char szTempNewPinCode[50];
   char szTempPinCode[50];
   CK_ULONG ulNewPinLen;
   CK_ULONG ulPinLen;
   register size_t i;
   int iCase = 0;

   /*---- Get some pin codes from the console ----*/
   printf("1. Change user pin\n\
2. Change Security Officer pin\n\
3. Unblock user pin.\nChoose option: ");
   fgets(line, sizeof(line), stdin);
   switch( line[0] )
   {
   case '1':
      {
         user_type = CKU_USER;
         bUnblock = FALSE;
         iCase = 1;
      }
      break;

   case '2':
      {
         user_type = CKU_SO;
         bUnblock = FALSE;
         iCase = 2;
      }
      break;

   case '3':
      {
         user_type = CKU_SO;
         bUnblock = TRUE;
         iCase = 3;
      }
      break;

   default:
      {
         printf("Error: Invalid option.\n");
         return;
      }
   }

   printf("\nEnter %s pin code: ", (bUnblock ? "Security Officer" : "old"));
   fgets(szPinCode, sizeof(szPinCode), stdin);
   szPinCode[strlen(szPinCode)-1] = '\0';
   printf("Using %s pin: %s\n", (bUnblock ? "Security Officer" : "old"),
szPinCode);

   printf("\nEnter new %spin code: ", (bUnblock ? "User " : ""));
   fgets(szNewPinCode, sizeof(szNewPinCode), stdin);
   szNewPinCode[strlen(szNewPinCode)-1] = '\0';
   printf("Using new %spin: %s\n", (bUnblock ? "User " : ""), szNewPinCode);

   printf("\nConfirm new %spin code: ", (bUnblock ? "User " : ""));
   fgets(szConfirmNewPinCode, sizeof(szConfirmNewPinCode), stdin);
   szConfirmNewPinCode[strlen(szConfirmNewPinCode)-1] = '\0';
   printf("Using Confirm new %spin: %s\n", (bUnblock ? "User " : ""),
szConfirmNewPinCode);

   printf("\n");
   if (strcmp(szConfirmNewPinCode, szNewPinCode))
   {
      printf("Error: New pin code is not confirmed.\n");
      return;
   }

   ulNewPinLen = (CK_ULONG)strlen(szNewPinCode);
   ulPinLen = (CK_ULONG)strlen(szPinCode);

   // Regarding the SO PIN, we have to consider this is not an ASCII PIN code.
   // This is the administrator key of the .NET smartcard which is a 24 bytes
   // buffer to transform from the incoming 48 characters ASCII buffer.

   // Change PIN user
   if( 1 == iCase )
   {

   }
   // Change PIN adm
   else if( 2 == iCase )
   {
      // Prepare the old PIN SO
      memset( szTempPinCode, 0, sizeof( szTempPinCode ) );
      ulPinLen = ulPinLen / 2;
      for( i = 0 ; i < strlen(szPinCode) ; i += 2 )
      {
         szTempPinCode[ i / 2 ] = ( fromhex( szPinCode[ i ]) << 4 ) + fromhex(
szPinCode[ i + 1 ] );
      }
      memcpy( szPinCode, szTempPinCode, ulPinLen );
      // Prepare the new PIN SO
      memset( szTempPinCode, 0, sizeof( szTempPinCode ) );
      ulNewPinLen = ulNewPinLen / 2;
      for( i = 0 ; i < strlen(szNewPinCode) ; i += 2 )
      {
         szTempNewPinCode[ i / 2 ] = ( fromhex( szNewPinCode[ i ]) << 4 ) +
fromhex( szNewPinCode[ i + 1 ] );
      }
      memcpy( szNewPinCode, szTempNewPinCode, ulNewPinLen );
   }
   // Unbock PIN user
   else if( 3 == iCase )
   {
      // Prepare the PIN SO
      memset( szTempPinCode, 0, sizeof( szTempPinCode ) );
      ulPinLen = ulPinLen / 2;
      for( i = 0 ; i < strlen(szPinCode) ; i += 2 )
      {
         szTempPinCode[ i / 2 ] = ( fromhex( szPinCode[ i ]) << 4 ) + fromhex(
szPinCode[ i + 1 ] );
      }
      memcpy( szPinCode, szTempPinCode, ulPinLen );
   }

   /*---- Pass these pin codes to the token ----*/
   if ((rv = (*p->C_Login) (hSession,
            user_type, szPinCode, ulPinLen)) != CKR_OK)
   {
      CKRLOG("C_Login", rv);
      return;
   }

   if (bUnblock)
   {
      if ((rv = (*p->C_InitPIN) (hSession,
               (CK_CHAR *)szNewPinCode, ulNewPinLen)) != CKR_OK)
      {
         CKRLOG("C_InitPIN", rv);
         return;
      }
   }
   else
   {
      if ((rv = (*p->C_SetPIN) (hSession,
               szPinCode,
               ulPinLen,
               (CK_CHAR *)szNewPinCode, ulNewPinLen)) != CKR_OK)
      {
         CKRLOG("C_SetPIN", rv);
         return;
      }
   }

   if ((rv = (*p->C_Logout) (hSession)) != CKR_OK)
   {
      CKRLOG("C_Logout", rv);
      return;
   }

   printf("Pin code operation sucessful!\n");
}
```

random.c

This sample demonstrates the true random number generation function of the token.

Note: The user must be logged in to use this function.

```
/* include the common code */
#include "main.c"
/******************************************************************************
* void SampleFunction(void)
******************************************************************************/
void SampleFunction(void)
{
   CK_BYTE RndBuff[128];
   unsigned int i;

   if ((rv = (*p->C_GenerateRandom)
         (hSession, RndBuff, sizeof(RndBuff))) != CKR_OK)
      return;

   for (i = 0; i < sizeof(RndBuff); i++)
      printf("%02X%c", RndBuff[i], ((i + 1) % 16 ? ':' : '\n'));

}
```

signit.c

This sample demonstrates how to use the token to create a digital signature. Firstly a private key is located in the token (the first one found), then the contents of a user specified file are fed into a hash function in blocks. The results of this hash operation are then signed using the private key. The signature is then printed out.

Note: This signature can be verified using the corresponding public key by a separate software module.

Before using this function, the user must be logged in (C_Login) and a key pair must already have been created.

```
/* include the common code */
#include "main.c"

/******************************************************************************
* void SampleFunction(void)
*******************************************************************************
*/
void SampleFunction(void)
{
   FILE *fp;
   char szFileName[256];
   CK_BYTE Buffer[512], Hash[64];
   CK_ULONG ulLen, ulHashLen, i, count;
   CK_OBJECT_HANDLE hKey;
   CK_MECHANISM Mechanism = { 0, NULL_PTR, 0 };
   CK_OBJECT_CLASS priv_object_class = CKO_PRIVATE_KEY;
   CK_KEY_TYPE priv_key_type = CKK_RSA;
   CK_ATTRIBUTE Template[] = {
      {CKA_CLASS, &priv_object_class, sizeof(CK_OBJECT_CLASS)}
      ,
      {CKA_KEY_TYPE, &priv_key_type, sizeof(CK_KEY_TYPE)}
   };
```

```
    /*---- Find an RSA Private key ----*/
    if ((rv = (*p->C_FindObjectsInit) (hSession, Template, 2)) != CKR_OK)
       return;
    if ((rv = (*p->C_FindObjects) (hSession, &hKey, 1, &count)) != CKR_OK)
       return;
    if ((rv = (*p->C_FindObjectsFinal) (hSession)) != CKR_OK)
       return;

    if (count != 1)
    {
       printf("Error: No keys found in token.\n");
       return;
    }

    /*---- Hash the contents of a file ----*/
    Mechanism.mechanism = CKM_SHA_1;
    if ((rv = (*p->C_DigestInit) (hSession, &Mechanism)) != CKR_OK)
       return;

    printf("Please enter name of file to be signed:");
    fgets(szFileName, sizeof(szFileName), stdin);
    if ((fp = fopen(szFileName, "rb")) == NULL)
    {
       printf("Error: Could not open %s\n", szFileName);
       return;
    }
    while ((ulLen = fread(Buffer, 1, sizeof(Buffer), fp)) != 0)
    {
       if ((rv = (*p->C_DigestUpdate) (hSession, Buffer, ulLen)) != CKR_OK)
       {
          fclose(fp);
          return;
       }
    }
    fclose(fp);
    ulHashLen = sizeof(Hash);
    if ((rv = (*p->C_DigestFinal) (hSession, Hash, &ulHashLen)) != CKR_OK)
       return;

    /*---- Sign the hash ----*/
    Mechanism.mechanism = CKM_RSA_PKCS;
    ulLen = sizeof(Buffer);
    if ((rv = (*p->C_SignInit) (hSession, &Mechanism, hKey)) != CKR_OK)
       return;
    if ((rv = (*p->C_Sign) (hSession, Hash, ulHashLen, Buffer, &ulLen))
       != CKR_OK)
       return;

    /*---- Print out the signature ----*/
    printf("RSA+SHA1 signature of file %s:\n", szFileName);
    for (i = 0; i < ulLen; i++)
       printf("%02X%c", Buffer[i], ((i + 1) % 16 ? ':' : '\n'));
}
```

slotevent.c

This sample demonstrates how to retrieve information about a “slot event”, that is, the insertion or withdrawal of a card or token. It includes the getinfo.c function (page 66).

```
/* include the common code */
#define NO_SESSION
#include "main.c"
```

```
#include "getinfo.c"

/******************************************************************************
* void SampleFunction(void)
******************************************************************************/
void SampleFunction(void)
{
   CK_SLOT_ID slotid;
   CK_FLAGS flags = 0;
   int i;

   getslotinfo(slotID);
   gettokeninfo(slotID);

   for (i=0; i<10; i++)
   {
      if((rv = (*p->C_WaitForSlotEvent)(flags, &slotid, NULL_PTR)) != CKR_OK)
      {
         CKRLOG("C_WaitForSlotEvent", rv);
         return;
      }

      getslotinfo(slotID);
      gettokeninfo(slotID);
   }
}
```

storeit.c

This sample shows how to store and retrieve your own private (PIN protected) application data in the token. This can be used to store bookmarks, user profiles, passwords and so on. Note that there is no major advantage in encrypting this data first since it can be stored with PIN protection. The data is stored in the CKA_VALUE attribute of a new CKO_DATA object. This data object has the CKA_APPLICATION field filled in, in order to identify your application and then the CKA_LABEL filled in, in order to identify the data's purpose within the context of this application. Obviously if yours is the only application likely to use the token and there is only one type of data, these fields can be left blank to save space in the token. You can create as many data objects in the token if there is enough space, but remember that there is an overhead of a few bytes associated with the existence of each object and each attribute. This means that 200 bytes of application data split up between 5 objects in the token will require more storage space than if all 200 bytes were concatenated together in a single attribute associated with just one object. Note however that memory management within the token is “highly” optimized.

```
/* include the common code */
#define·RW_SESSION
#include "main.c"

/******************************************************************************
* void SampleFunction(void)
******************************************************************************/
void SampleFunction(void)
{

   CK_OBJECT_HANDLE hObject = NULL_PTR;
   CK_BBOOL bTrue = TRUE;
   CK_BBOOL bPinCodeProtected = TRUE;
   CK_OBJECT_CLASS data_class = CKO_DATA;
   CK_ULONG count;
   char szBuffer[256];
   CK_ATTRIBUTE Template[] = {
      {CKA_CLASS, &data_class, sizeof(CK_OBJECT_CLASS)} ,
      {CKA_APPLICATION, "MyApp", sizeof("MyApp") - 1} ,
      {CKA_LABEL, "profile00", sizeof("profile00") - 1} ,
      {CKA_PRIVATE, &bPinCodeProtected, sizeof(CK_BBOOL)} ,
      {CKA_TOKEN, &bTrue, sizeof(CK_BBOOL)} ,
      {CKA_VALUE, szBuffer, sizeof(szBuffer) - 1}
   };

   /* How many items in Template? */
   CK_ULONG ulCount = sizeof(Template) / sizeof(CK_ATTRIBUTE);
   memset(szBuffer, 0, sizeof(szBuffer));

   /*---- Check to see if a data object already exists ----*/
   if ((rv = (*p->C_FindObjectsInit)
         (hSession, Template, ulCount - 1)) != CKR_OK)
      return;

   if ((rv = (*p->C_FindObjects) (hSession, &hObject, 1, &count)) != CKR_OK)
      return;

   if ((rv = (*p->C_FindObjectsFinal) (hSession)) != CKR_OK)
      return;
   /*---- Print out the existing value ----*/
   printf("Current application specific stored data:\n");

   if (count != 1)
      hObject = NULL_PTR;
   else
   {
      if ((rv = (*p->C_GetAttributeValue)
            (hSession, hObject, &(Template[ulCount - 1]), 1)) != CKR_OK)
         return;
      printf(szBuffer);
   }

   /*---- Get new value from user ----*/
   printf("\nEnter new value or ENTER to skip:\n");
   fgets(szBuffer, sizeof(szBuffer), stdin);
   if (strlen(szBuffer) == 0)
      return;
   Template[ulCount - 1].ulValueLen = strlen(szBuffer);

   /*---- Either create or update object with new value ----*/
   if (hObject == NULL_PTR)
   {
      if ((rv = (*p->C_CreateObject)
            (hSession, Template, ulCount, &hObject)) != CKR_OK)
         return;
   }
   else
   {
      if ((rv = (*p->C_SetAttributeValue)
            (hSession, hObject, &(Template[ulCount - 1]), 1)) != CKR_OK)
         return;
   }

   printf("Data successfully stored in token\n");
}
```

tellme.c

This sample code shows how to retrieve the token information, the slot information and the session information.

```
/* include the common code */
# include “main.c”
# include “getinfo.c”
/**************************************************************
* void SampleFunction(void)
***************************************************************/
void SampleFunction(void)
{
   getinfo();
   getsessioninfo();
   getslotinfo(slotID);
   gettokeninfo(slotID);
}
```

cryptoki.h

This is a cryptoki header file.

```
/* cryptoki.h include file for PKCS #11. */
#ifndef ___CRYPTOKI_H_INC___
#define ___CRYPTOKI_H_INC___

  #if defined(_WINDOWS)

    #pragma pack(push, cryptoki, 1)

    /* Specifies that the function is a DLL entry point. */
    #define CK_IMPORT_SPEC __declspec(dllimport)

    /* Define CRYPTOKI_EXPORTS during the build of cryptoki libraries. Do not
define it in applications. */
    #ifdef CRYPTOKI_EXPORTS
      /* Specified that the function is an exported DLL entry point. */
      #define CK_EXPORT_SPEC __declspec(dllexport)
    #else
      #define CK_EXPORT_SPEC CK_IMPORT_SPEC
    #endif

    /* Ensures the calling convention for Win32 builds */
    #define CK_CALL_SPEC __cdecl

    #define CK_PTR *

    #define CK_DEFINE_FUNCTION(returnType, name) \
      returnType CK_EXPORT_SPEC CK_CALL_SPEC name

    #define CK_DECLARE_FUNCTION(returnType, name) \
      returnType CK_EXPORT_SPEC CK_CALL_SPEC name

    #define CK_DECLARE_FUNCTION_POINTER(returnType, name) \
      returnType CK_IMPORT_SPEC (CK_CALL_SPEC CK_PTR name)

    #define CK_CALLBACK_FUNCTION(returnType, name) \
      returnType (CK_CALL_SPEC CK_PTR name)

    #ifndef NULL_PTR
      #define NULL_PTR 0
    #endif

    #include "pkcs11.h"
    #include "pkcs-11v2-20a3.h"

    #pragma pack(pop, cryptoki)

  #else /* not windows */

    #define CK_PTR *

    #define CK_DEFINE_FUNCTION(returnType, name) \
      returnType name

     #define CK_DECLARE_FUNCTION(returnType, name) \
      returnType name

    #define CK_DECLARE_FUNCTION_POINTER(returnType, name) \
      returnType (* name)

    #define CK_CALLBACK_FUNCTION(returnType, name) \
      returnType (* name)

    #define CK_ENTRY

    #ifndef NULL_PTR
      #define NULL_PTR 0
    #endif

    #include "pkcs11.h"
    #include "pkcs-11v2-20a3.h"

  #endif

#endif /* ___CRYPTOKI_H_INC___ */
```

Troubleshooting

The following list covers limitations and minor issues known at the time of release:

Smart Enterprise Guardian (SEG)

The SEG is not supported on Linux and Mac OS X operating systems because the smart card reader is not seen. This is a SEG limitation due to its hardware design.

PC/SC and CCID Issues

Reader Hot-Plug

On Linux and Mac OS X, the reader hot-plug is not detected. This is a limitation of the PC/SC layer.

For example, if you plug a reader while Firefox is running, you have to close and restart the application to see the reader.

Mozilla Firefox and Thunderbird

Simultaneous Smart Cards

When the end-user browses the cryptographic modules on either Firefox or Thunderbird with two smart cards connected and logs on with one of the smart cards, Firefox/Thunderbird considers both smart cards as being logged on.

Adobe Acrobat Reader

Adobe Acrobat Reader does not support certificate importation.

C

Configuring PKCS#11 in Mozilla

This appendix describes how to configure Mozilla applications so they can communicate with the PKCS#11 security module.

Firefox

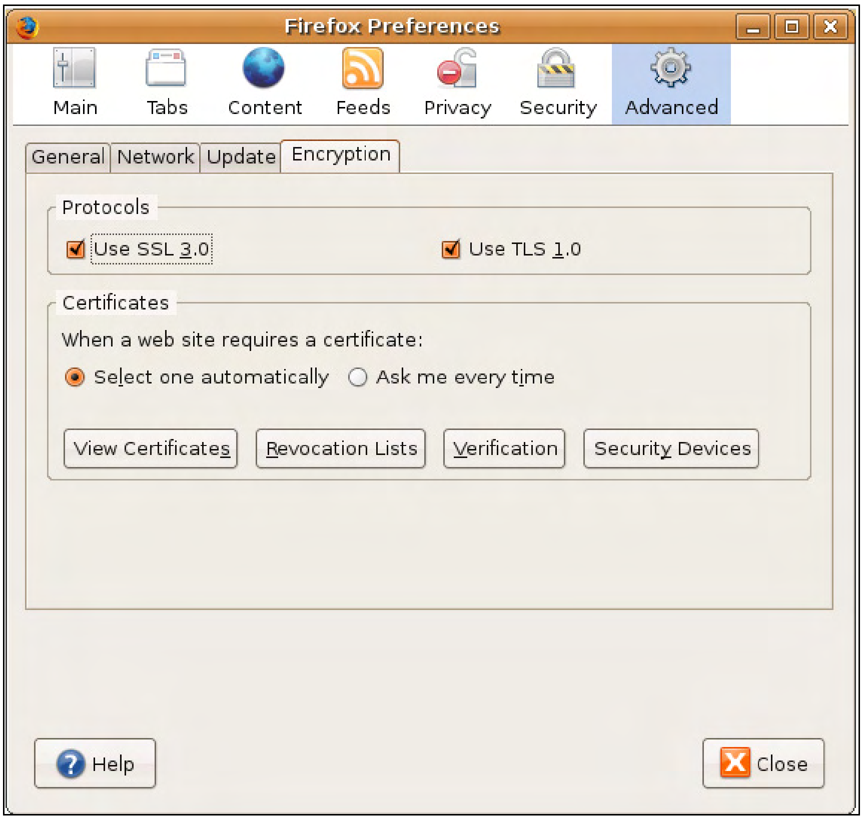
This section describes the necessary configuration for Firefox. You only need to do this once.

If you are using Iceweasel, the Debian equivalent of Firefox, the procedure is exactly the same.

To configure Firefox to recognize the PKCS#11 security module:

1. Make sure your card/token is connected.
2. Open the **Mozilla Firefox** browser and from the **Edit** menu choose **Preferences**.
3. Click the **Advanced** icon, then the **Encryption** tab as shown in “Figure 27”.

Figure 27 - Mozilla Firefox Encryption Options Dialog



Protocols

☑ Use SSL 3.0 ☑ Use TLS 1.0

Certificates

When a web site requires a certificate:

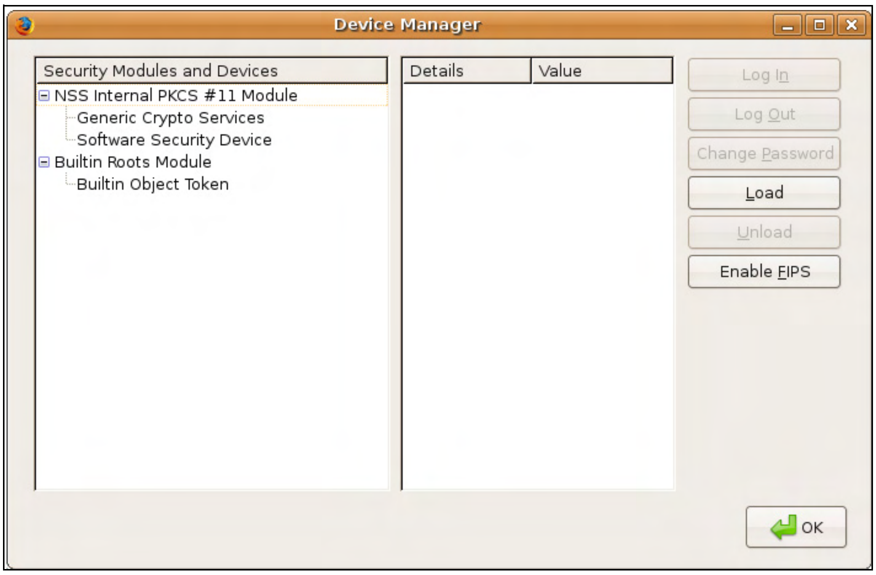
● Select one automatically ○ Ask me every time

4 In **Certificates**, choose one of the options for the action to take when a web site requires a certificate:

- Select one automatically
- Ask me every time

5 Click **Security Devices** to display the **Device Manager** window. This displays the modules currently available as shown in “Figure 28”.

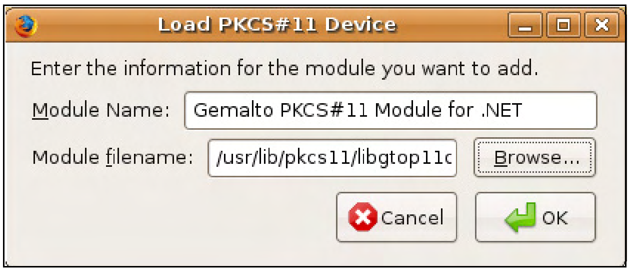
Figure 28 - Device Manager



Security Modules and Devices
- NSS Internal PKCS #11 Module
  - Generic Crypto Services
  - Software Security Device
- Builtin Roots Module
  - Builtin Object Token

6 Click the **Load** button to the right in the dialog. This displays the **Load PKCS#11 Device** window, as shown in “Figure 29”.

Figure 29 - Load PKCS#11 Device Window



Enter the information for the module you want to add.

Module Name: Gemalto PKCS#11 Module for .NET

Module filename: /usr/lib/pkcs11/libgtop11c

7 Enter a **Module Name.**

8 In **Module filename**, enter the full path and filename for the module. By default this is:

/usr/lib/pkcs11/libgtop11dotnet.so

9 Click **OK**. The “Confirm” dialog appears asking if you are sure that you want to install the security module.

10 Click **OK**.

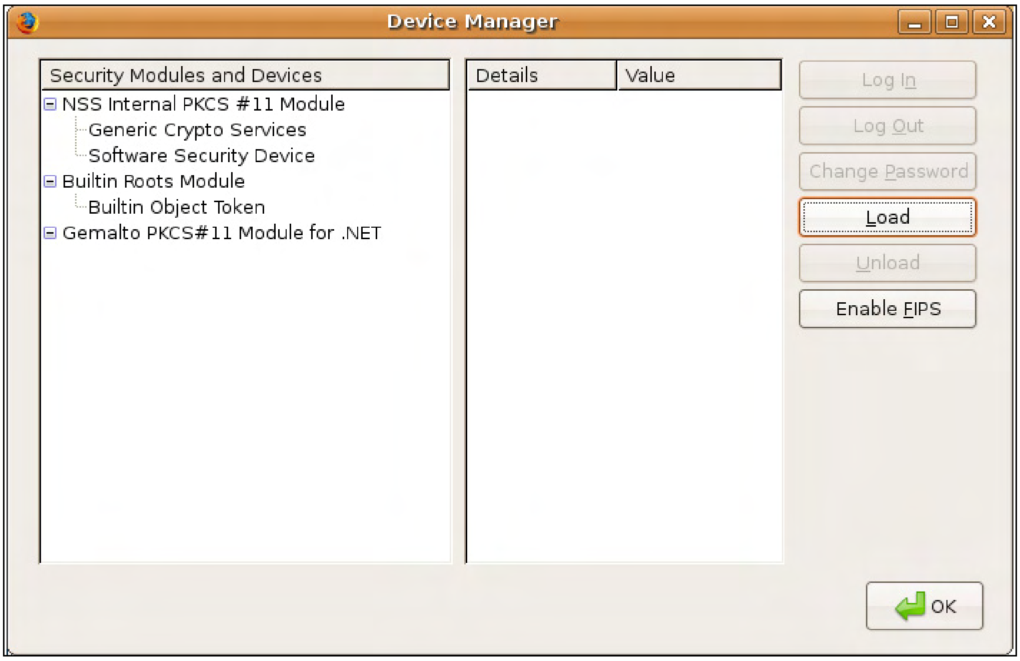
A brief progress dialog appears indicating that the module is being loaded.

When this is completed an “**Alert**” indicates that the module has been installed.

11 Click **OK** to close this **Alert**.

The **Device Manager** indicates the presence of the new module as shown in “Figure 30”:

Figure 30 - Device Manager After Module Configuration



Security Modules and Devices
- NSS Internal PKCS #11 Module
  - Generic Crypto Services
  - Software Security Device
- Builtin Roots Module
  - Builtin Object Token
- Gemalto PKCS#11 Module for .NET

Thunderbird

You only need to make this configuration once in Thunderbird.

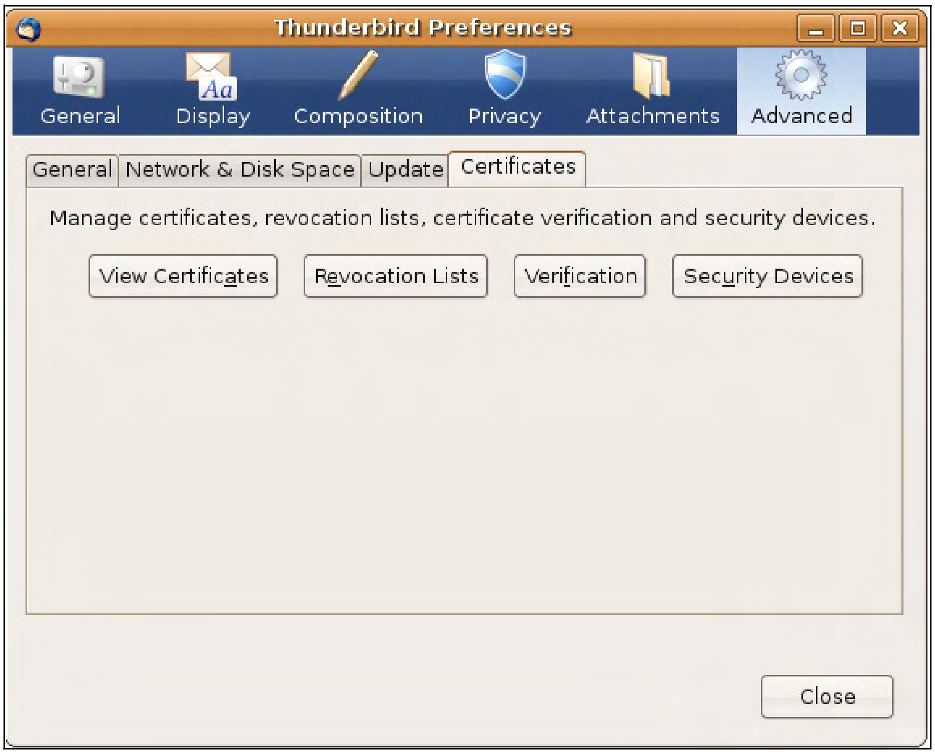
If you are using Icedove, the Debian equivalent of Thunderbird, the procedure is exactly the same.

The procedure to configure Thunderbird is very similar to that of Firefox, but is slightly different.

To configure Thunderbird to recognize the PKCS#11 security module:

1. Make sure your smart card/token is connected.
2. Start **Mozilla Thunderbird**.
3. Enter your password if you are prompted for it and click on **OK**.
4. From the **Edit** menu, choose **Preferences**.
5. In the dialog box that opens, click the **Advanced** icon, then the **Certificates** tab to display the settings as shown in “Figure 31”.

Figure 31 - Thunderbird - Certificates Tab



Manage certificates, revocation lists, certificate verification and security devices.

6. The rest of the procedure is the same as that described for Firefox. Continue from step 5 on page 82.

   This new module will be used with all e-mail you send with Thunderbird.

Abbreviations

API	Application Programming Interface
CA	Certificate Authority
CAPI	Cryptographic Application Programming Interface
CCID	A driver that is needed to communicate with a .NET smart card.
CSN	Card Serial Number
MSB	Most Significant Byte(s)
OS	Operating System
PAM	Pluggable Authentication Module
PC/SC	personal computer/smart card - a specification used in communication between a PC and a smart card.
PIN	Personal Identification Number
PKCS	Public Key Cryptography Standard
PKCS#11	Public Key Cryptography Standard **#11. For further information about this and other PKCS standards, refer to the RSA Laboratories web sit at http://www.rsa.com/rsalabs/**
R/O	Read only (access)
R/W	Read and write (access)
RSA	Rivest, Shamir, Adleman (inventors of public key cryptography standards)
S/MIME	Secure/Multipurpose Internet Mail Extensions
SO	Security Officer
SSL	Secure Sockets Layer A protocol, v.3.0.v, for securing TCP/IP sessions

Glossary

.NET Utilities	A series of utilities developed by Gemalto to provide operations for Gemalto .NET smart cards. They include changing and unblocking a PIN and managing certificates.
Admin PIN	A common name for the SO PIN.
Algorithm	A mathematical formula used to perform computations that can be used for security purposes.
Attribute	A characteristic of a token object.
Certificate	A certificate provides identification for secure transactions. It consists of a public key and other data, all of which have been digitally signed by a CA. It is a condition of access to secure e-mail or to secure Web sites.
Certificate Authority	An entity with the authority and methods to certify the identity of one or more parties in an exchange (an essential function in public key crypto systems).
Cryptography	The science of transforming confidential information to make it unreadable to unauthorized parties.
Cryptoki	The Cryptographic Token Interface defined in the PKCS#11 standard. It is a platform independent API to cryptographic tokens.
Digital Signature	A data string produced using a Public Key Crypto system to prove the identity of the sender and the integrity of the message.
Encryption	A cryptographic procedure whereby a legible message is encrypted and made illegible to all but the holder of the appropriate cryptographic key.
Key	A value that is used with a cryptographic algorithm to encrypt, decrypt, or sign data. Secret key crypto systems use only one secret key. Public key crypto systems use a public key to encrypt data and a private key to decrypt data.
Key Length	The number of bits forming a key. The longer the key, the more secure the encryption. Government regulations limit the length of cryptographic keys.
PAM PKCS#11 module	A Linux-PAM login module that allows a X.509 certificate based user login.
PKCS#11	A software library that implements the Cryptoki
Pluggable Authentication Module (PAM)	A mechanism to integrate multiple low-level authentication schemes into a high-level application programming interface (API), which allows programs that rely on authentication to be written independently of the underlying authentication scheme.
Public Key Crypto system	A cryptographic system that uses two different keys (public and private) for encrypting data. The most well-known public key algorithm is RSA.
Session	A logical connection between an application and a token.

Session object	An object that exists during the time of a session only, it is destroyed when the session is closed.
SO PIN	Security Officer PIN - the PIN used to unblock the card.
SSL	Secure Sockets Layer: A Security protocol used between servers and browsers for secure Web sessions.
SSL Handshake	The SSL handshake, which takes place each time you start a secure Web session, identifies the server. This is automatically performed by your browser.
S/MIME	A Standard offline message format for use in secure e-mail applications.
Token	In a security context, a token is a hardware object like a smart card, but it could also be a pluggable software module designed to interact with a specific hardware module, such as a smart card. Token-based authentication provides enhanced security because success depends on a physical identifier (the smart card) and a personal identification number (PIN).
Token object	An object that exists in the token. It can only be deleted during a read/write session.