



IDGo 500 PKCS#11 Library for Windows

User Guide

All information herein is either public information or is the property of and owned solely by Gemalto NV. and/or its subsidiaries who shall have and keep the sole right to file patent applications or any other kind of intellectual property protection in connection with such information.

Nothing herein shall be construed as implying or granting to you any rights, by license, grant or otherwise, under any intellectual and/or industrial property rights of or concerning any of Gemalto's information.

This document can be used for informational, non-commercial, internal and personal use only provided that:

- The copyright notice below, the confidentiality and proprietary legend and this full warning notice appear in all copies.
- This document shall not be posted on any network computer or broadcast in any media and no modification of any part of this document shall be made.

Use for any other purpose is expressly prohibited and may result in severe civil and criminal liabilities.

The information contained in this document is provided "AS IS" without any warranty of any kind. Unless otherwise expressly agreed in writing, Gemalto makes no warranty as to the value or accuracy of information contained herein.

The document could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Furthermore, Gemalto reserves the right to make any change or improvement in the specifications data, information, and the like described herein, at any time.

Gemalto hereby disclaims all warranties and conditions with regard to the information contained herein, including all implied warranties of merchantability, fitness for a particular purpose, title and non-infringement. In no event shall Gemalto be liable, whether in contract, tort or otherwise, for any indirect, special or consequential damages or any damages whatsoever including but not limited to damages resulting from loss of use, data, profits, revenues, or customers, arising out of or in connection with the use or performance of information contained in this document.

Gemalto does not and shall not warrant that this product will be resistant to all possible attacks and shall not incur, and disclaims, any liability in this respect. Even if each product is compliant with current security standards in force on the date of their design, security mechanisms' resistance necessarily evolves according to the state of the art in security and notably under the emergence of new attacks. Under no circumstances, shall Gemalto be held liable for any third party actions and in particular in case of any successful attack against systems or equipment incorporating Gemalto products. Gemalto disclaims any liability with respect to security for direct, indirect, incidental or consequential damages that result from any use of its products. It is further stressed that independent testing and verification by the person using the product is particularly encouraged, especially in any application in which defective, incorrect or insecure functioning could result in damage to persons or property, denial of service or loss of privacy.

© Copyright 2009-12 Gemalto N.V. All rights reserved. Gemalto and the Gemalto logo are trademarks and service marks of Gemalto N.V. and/or its subsidiaries and are registered in certain countries. All other trademarks and service marks, whether registered or not in specific countries, are the property of their respective owners.

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: DOC1256792A

July 16, 2012

| | | |
|------------------|---|-----------|
| Preface | | vi |
| | Who Should Read This Book | vi |
| | Documentation | vi |
| | Conventions | vi |
| | Typographical Conventions | vi |
| | Additional Resources | vii |
| | For Further Help | vii |
| | If You Find an Error | vii |
| Chapter 1 | Overview | 1 |
| | IDPrime .NET Smart Cards | 1 |
| | Cryptoki | 1 |
| | IDGo 500 PKCS#11 Library | 2 |
| | Supported Platforms and Applications | 3 |
| | Windows | 3 |
| | Linux | 3 |
| | Mac OS | 4 |
| | Applications Tested | 4 |
| Chapter 2 | PKCS#11 Specifics | 5 |
| | Key Sizes Supported | 5 |
| | Number of Simultaneous Reader Connections Supported | 5 |
| | Instant Detection of .NET USB Devices | 5 |
| | File Cache | 6 |
| | Configuration File | 7 |
| | PKCS#11 Methods Supported | 8 |
| | Accessing Objects According to Session Type | 10 |
| | Session Types | 10 |
| | Object Types | 11 |
| | Authentication | 11 |
| | Supported PKCS#11 Objects and Attributes | 11 |
| | Cryptographic Mechanisms Supported | 12 |
| | Cryptographic Algorithms | 12 |
| | Hash Algorithms | 12 |
| | Reading the Card Serial Number | 13 |
| | Product Limitations | 14 |
| | The Security Officer PIN | 14 |
| Chapter 3 | Installation | 18 |
| | System Requirements | 18 |
| | Computer | 18 |
| | Operating Systems | 18 |
| | Peripherals | 19 |
| | Installing the IDGo 500 PKCS#11 Library | 19 |
| | Pre-requisites | 19 |
| | Installing the IDGo 500 PKCS#11 Library | 20 |
| | Configuring the PKCS#11 Security Module | 20 |

| | |
|--|-----------|
| Uninstalling the IDGo 500 PKCS#11 Library | 21 |
| Chapter 4 Tasks | 22 |
| How to Get a Certificate | 22 |
| How to Import a Certificate in the IDPrime .NET Card | 23 |
| How to Delete a Certificate from the IDPrime .NET Card | 27 |
| How to View the Details of a Certificate in an IDPrime .NET Card | 28 |
| How to Unblock a User PIN | 29 |
| How to Change a User PIN | 30 |
| How to Use E-mail Securely | 33 |
| About Secure E-mail | 33 |
| Working with Mozilla Thunderbird. | 33 |
| How to View Secure Web Sites | 37 |
| Mozilla Firefox | 38 |
| Appendix A Sample Code | 42 |
| Cryptoki Header Files | 42 |
| Sample Code Files | 42 |
| main.c | 42 |
| getinfo.c | 47 |
| deleteall.c | 49 |
| dumpit.c | 50 |
| enroll.c | 53 |
| genkey.c | 64 |
| loadkey.c | 65 |
| pincode.c | 68 |
| random.c | 71 |
| signit.c | 71 |
| slovent.c | 72 |
| storeit.c | 73 |
| tellme.c | 75 |
| cryptoki.h | 75 |
| Appendix B Troubleshooting | 77 |
| Conversion from .NET PKCS#11 2.1 to 2.2 | 77 |
| Performance Problems With Memory Management | 77 |
| Mozilla Firefox and Thunderbird | 77 |
| Simultaneous Smart Cards | 77 |
| Fast User Switching | 77 |
| Remote Desktop Connection | 78 |
| CheckPoint VPN Client NGX | 78 |
| Citrix Server | 78 |
| Adobe Acrobat Reader | 78 |
| Appendix C Configuring PKCS#11 in Mozilla | 79 |
| Firefox | 79 |
| Thunderbird | 83 |

| | | |
|--------------------|------------------------------------|-----------|
| Appendix D | The Minidriver Manager Tool | 84 |
| Appendix E | The .NET Utilities Tool | 85 |
| Terminology | | 86 |
| | Abbreviations | 86 |
| | Glossary | 87 |

List of Figures

| | |
|---|----|
| Figure 1 - Smart Card Cryptography Support for Different Platforms and Applications | 3 |
| Figure 2 - Revealing c:\Users\All Users | 7 |
| Figure 3 - Supported PKCS#11 Objects and Attributes | 12 |
| Figure 4 - Install Shield Wizard - Ready To Install Window | 20 |
| Figure 5 - .NET Utilities Portal – Manage Certificates | 23 |
| Figure 6 - .NET Utilities Portal – Import Certificate | 24 |
| Figure 7 - .NET Utilities Portal – Import Certificate – Password Prompt | 24 |
| Figure 8 - .NET Utilities Portal – Import Certificate – Password Prompt | 24 |
| Figure 9 - Mozilla Firefox Encryption Options Dialog | 25 |
| Figure 10 - Password Required | 25 |
| Figure 11 - Certificate Manager Window | 26 |
| Figure 12 - File Name to Restore Window | 26 |
| Figure 13 - Choose Token Dialog Window | 27 |
| Figure 14 - Certificate Manager After Certificate Importation | 27 |
| Figure 15 - Certificate Details | 29 |
| Figure 16 - Mozilla Firefox Encryption Options Dialog | 31 |
| Figure 17 - Device Manager | 32 |
| Figure 18 - Change Master Password Window | 32 |
| Figure 19 - Thunderbird – Security Account Settings | 34 |
| Figure 20 - Thunderbird - Select Certificate | 35 |
| Figure 21 - Thunderbird – “Use Same Certificate” Message | 35 |
| Figure 22 - Thunderbird – Security Account Settings (2) | 36 |
| Figure 23 - Example Web Site Before Authentication | 39 |
| Figure 24 - User Identification Request Window | 40 |
| Figure 25 - Secured Web Page After Authentication | 40 |
| Figure 26 - Mozilla Firefox Encryption Options Dialog | 80 |
| Figure 27 - Device Manager | 81 |
| Figure 28 - Load PKCS#11 Device Window | 81 |
| Figure 29 - Device Manager After Module Configuration | 82 |
| Figure 30 - Thunderbird - Certificates Tab | 83 |
| Figure 31 - .NET Utilities Portal – Security Warning | 85 |
| Figure 32 - .NET Utilities Portal Welcome Window | 85 |

List of Tables

| | |
|---|----|
| Table 1 - PKCS#11 Methods Supported | 8 |
| Table 2 - Access to Objects According to Session Type | 11 |

This document introduces you to the IDGo 500 PKCS#11 Library for IDPrime .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 IDPrime .NET smart cards/tokens and smart card reader technology, as well as computer hardware and software.

It is assumed that the user of the IDGo 500 PKCS#11 library has:

- an understanding of the basic operations in a computer OS.
- administrative privileges for the computer on which the IDGo 500 PKCS#11 Library will be installed.

Documentation

The IDGo 500 PKCS#11 library is delivered with the following documentation:

- *IDGo 500 PKCS#11 Library for Windows User Guide* (this document) - IDGo 500 PKCS11_Lib_.NET__Windows_User_Guide.pdf.
- A *Release Notes* file. This contains any relevant information about the installation and the complete version history.

Conventions

The following conventions are used in this document:

Typographical Conventions

The IDGo 500 PKCS#11 library 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 IDGo 500 PKCS#11 library running on Windows XP.

Additional Resources

For further information or more detailed use of the IDGo 500 PKCS#11 library, additional resources and documentation are available on the following web site:

www.gemalto.com/products/dotnet_card

For Further Help

You can find information on how to contact your Gemalto representative by clicking **Contact Us** at the Gemalto web site, www.gemalto.com.

If You Find an Error

Gemalto makes every effort to prevent errors in its documentation. However, if you discover any errors or inaccuracies in this document, please inform your Gemalto representative. Please quote the document reference number found at the bottom of the legal notice on the inside front cover.

Overview

IDPrime .NET Smart Cards

Gemalto's IDPrime .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. IDPrime .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.

IDPrime .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.

IDPrime .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 IDGo 500 PKCS#11 library is Gemalto's implementation of Cryptoki, that grants cryptographic applications access to the IDPrime .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 IDPrime .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 8.

Note: The IDGo 500 PKCS#11 library is fully compliant with *PKCS#11 v2.10* and partially with *PKCS#11 v2.20*.

IDGo 500 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 IDPrime .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 IDPrime .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 IDGo 500 PKCS#11 library. Digital certificates are stored on smart cards according to the CSP architecture.

The IDGo 500 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 IDGo 500 PKCS#11 library, please refer to "Supported Platforms and Applications" on page 3.

With this IDGo 500 PKCS#11 library you can use the digital certificates stored on IDPrime .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.

The IDGo 500 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 the IDGo 500 PKCS#11 library to:

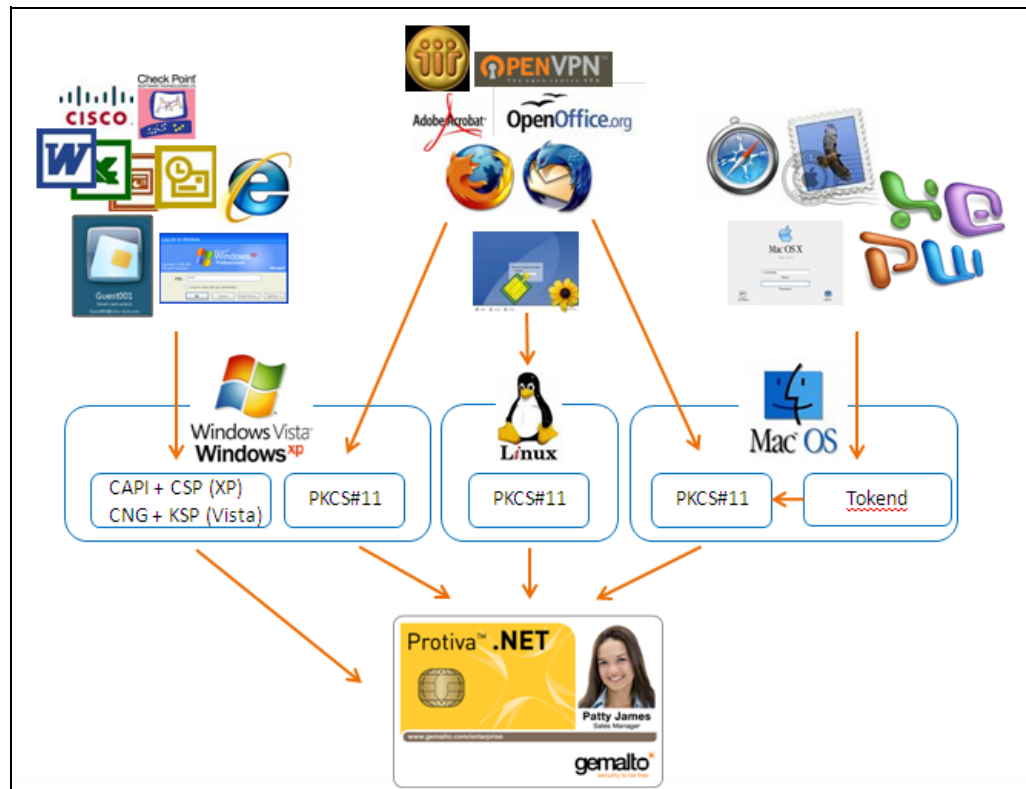
- Load and generate RSA Keys
- Create RSA digital signatures
- Manage PIN codes
- Store certificates
- Store miscellaneous data specific to an application, for example, user profiles and bookmarks
- Generate secure random numbers.

Supported Platforms and Applications

The IDGo 500 PKCS#11 library allows IDPrime .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, Adobe Acrobat and others.

Gemalto's IDGo 500 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 OS

On Mac OS, there is no Microsoft CAPI/CSP. All native Apple applications (Mac Logon, Safari, mail client and so on) use a Mac OS 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 Mac OS 10.6 (Snow Leopard) and 10.7 (Lion), Gemalto's IDGo 500 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 the IDGo 500 PKCS#11 library, 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 IDGo 500 PKCS#11 library and IDPrime .NET cards.

Key Sizes Supported

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

The .NET V2+ and IDPrime .NET smart cards support 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 IDGo 500 PKCS#11 library can manage up to five smart card reader connections. Beyond this, a CKR_HOST_MEMORY error occurs when you call the `C_GetSlotList` function.

Instant Detection of .NET USB Devices

The IDGo 500 PKCS#11 library can detect the insertion and withdrawal of .NET USB Devices. These devices contain a smart card. When this type of device is inserted, the IDGo 500 PKCS#11 library automatically creates a PKCS#11 slot dedicated to this type of reader.

The library has five permanent static reader slots. When an application calls the **`C_GetSlotList`** function, the IDGo 500 PKCS#11 library returns these five empty reader slots by default. When a .NET USB device is inserted, the IDGo 500 PKCS#11 library indicates that an event has occurred for the slot concerned. The application that calls the **`C_WaitForSlotEvent`** function (regardless of whether it is in blocking mode or not) is able to receive this event. The application can call the **`C_GetSlotList`** function at any time in order to know which slots are available.

This new feature enables an application such as Firefox to detect the insertion or withdrawal of .NET USB devices while it is running.

File Cache

In order to improve the performance of the IDGo 500 PKCS#11 library, it is now possible to activate a permanent file cache on the hard disk of the computer where the IDGo 500 PKCS#11 library is installed. The user can choose whether to activate the cache or not by means of the IDGo 500 PKCS#11 library configuration file. By default, the cache is activated.

When the file cache is activated, all the card's read and write operations are stored in the permanent cache. When possible, the IDGo 500 PKCS#11 library then uses the data stored in the cache rather than the card itself.

Initially the cache is empty. When the IDGo 500 PKCS#11 library needs information to perform its current operation, it first looks in the cache. If the information is not there, the IDGo 500 PKCS#11 library reads the information from the card and stores it in the permanent cache. The permanent cache file is a simple binary file whose name is the card's serial number. As long as the card's contents do not change, the file cache does not change.

The IDGo 500 PKCS#11 library monitors the card to see if it is modified (for example the addition or deletion of a card certificate). In such a case, the permanent file cache is destroyed and recreated to contain the new contents of the card.

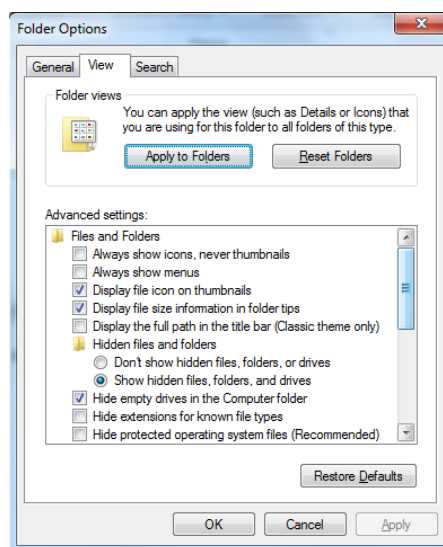
Note: Only public information is stored in the file cache. PINs and private keys are never stored there.

By default, the file cache is put in a fixed place, in order to be sure that the IDGo 500 PKCS#11 library can find it. In Windows, this is in the directory "C:\Documents and Settings\All Users\Application Data\Gemalto\DotNet PKCS11" for Windows XP and Server 2003 and in "C:\Users\All Users\Gemalto\DotNet PKCS11" for Windows Vista/Server 2008/Seven/Server 2008 R2.

Revealing the All Users Folder

If C:\Users\All Users is not visible, modify the **Advanced Settings** in Windows as follows:

- 1 Open Windows Explorer.
- 2 In **Organize**, choose **Folder and Search Options**.
- 3 In **Folder Options**, click the **View** tab.
- 4 In **Advanced Settings**, for **Hidden files and folders**, choose **Show hidden files, folders and drives** and clear the box **Hide protected operating system files** as shown in the following figure:

Figure 2 - Revealing c:\Users\All Users

5 Click **OK** to close the window.

Configuration File

The IDGo 500 PKCS#11 library reads a configuration file when started. This file contains the following information:

- Activation (or not) of the log file. The log file is located in the same directory as the configuration file.
- Activation (or not) of the cache file described in the previous section.

The configuration file is called "Gemalto.NET.PKCS11.ini" and is in the following format.

```
; =====
; === Gemalto .NET PKCS#11 configuration file ===
; =====
; === Log Section
[Log]
; Enable the log
; Value: 0 to disable the log or 1 to enable the log
Enable = 0
; === Cache Section
[Cache]
; Enable the disk cache
; Value: 0 to disable the cache or 1 to enable the cache
Enable = 1
```

If the configuration file is not present, the following default values are used; cache file but no log file.

By default, the configuration file is put in a fixed place, in order to be sure that the IDGo 500 PKCS#11 library can find it. In Windows, this is in the directory "C:\Documents and Settings\All Users\Application Data\Gemalto\DotNet PKCS11" for Windows XP and

Server 2003 and in “C:\Users\All Users\Gemalto\DotNet PKCS11” for Windows Vista/Server 2008/Seven/Server 2008 R2. If C:\Users\All Users is not visible, follow the instructions in “Revealing the All Users Folder” on page 6.

PKCS#11 Methods Supported

Not all the PKCS#11 interface methods specified in the PKCS#11 v2.20 are implemented in the IDGo 500 PKCS#11 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 |
| 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 connection 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 |

Table 1 - PKCS#11 Methods Supported (continued)

| Category | Function | Description |
|---|---------------------|--|
| 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 |
| 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 |

Table 1 - PKCS#11 Methods Supported (continued)

| Category | Function | Description |
|--|-----------------------|---|
| 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 |
| Parallel function management functions | C_GetFunctionStatus | NOT SUPPORTED |
| | C_CancelFunction | NOT SUPPORTED |
| Callback function | | NOT SUPPORTED |
| Proprietary Gemalto functions | C_GetCardProperty | Calls the GetCardProperty in the .NET minidriver and uses the same parameters. For a description of this function, please refer to the <i>IDPrime .NET Smart Card Integration Guide</i> . |
| | C_SetCardProperty | Calls the SetCardProperty in the .NET minidriver and uses the same parameters. For a description of this function, please refer to the <i>IDPrime .NET Smart Card Integration Guide</i> . |

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.

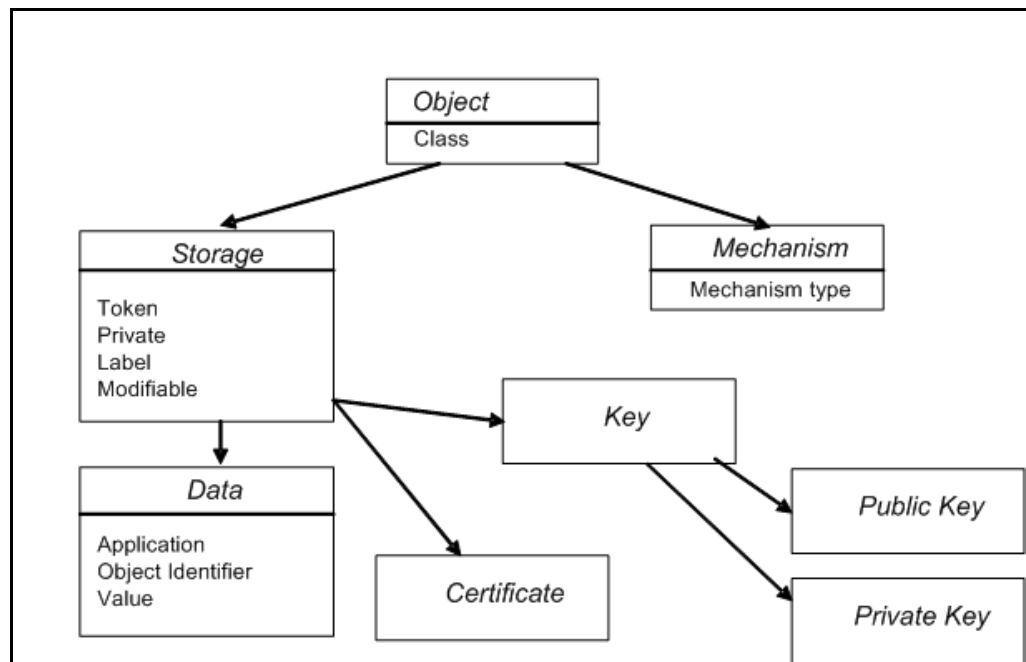
The following table summarizes the kind of access each type of session has to each type of object.

Table 2 - Access to Objects According to Session Type

| Type of object | Type of session | | | | |
|------------------------|-----------------|------------|----------|----------|--------|
| | 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 3 - Supported PKCS#11 Objects and Attributes

The IDGo 500 PKCS#11 library does not support **Hardware Feature** objects and **Domain Parameters** objects. These objects were introduced in the 2.20 version of the RSA standard. The IDGo 500 PKCS#11 library only deals with the object hierarchy specified in the 2.10 version.

The IDGo 500 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 IDGo 500 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 IDGo 500 PKCS#11 library 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 IDGo 500 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 IDPrime .NET card serial number (CSN) can be read in one of the following ways:

- A direct command to the card (**GetCardProperty**) which can return either the CSN or GUID values. For more details, please refer to the *IDPrime .NET Smart Card Integration Guide*.
- By using Gemalto's Minidriver Manager tool or .NET utilities web tool that send a command directly to the card and return its CSN value.
- Via the IDGo 500 PKCS#11 library, by using the **C_GetTokenInfo** function. For more details, see "IDGo 500 PKCS#11 Library".

The IDGo 500 PKCS#11 library interprets the CSN differently to the first two methods.

Direct Command, Minidriver Manager and .NET Utilities Tools

These three methods both read the CSN directly from the card and return it as a 24-digit (or 12-byte) number as defined in the Microsoft Minidriver specification. This number is the unique chip serial number.

The card can also return a GUID property (or card ID file content) that includes the CSN. This GUID is 16-bytes and is built as follows:

0x2E 4E 45 54 (4-byte fixed value) || CSN (12 bytes)

Note: Please refer to "Appendix D - The Minidriver Manager Tool" before using the Minidriver Manager. Please refer to "Appendix E - The .NET Utilities Tool" for information on how to access the .NET Utilities tool.

To read the CSN and GUID with the Minidriver Manager:

- 1 Start the Minidriver Manager and insert the IDPrime .NET card in the reader.
- 2 Click **Connect**.
- 3 In the **Select Smart Card** dialog, select the correct reader and click **OK**.

The CSN and GUID display in the right panel.

To read the CSN with the .NET Utilities Tool:

- 1 Access the .NET Utilities tool and insert the IDPrime .NET card in the reader as described in "Appendix E - The .NET Utilities Tool".
- 2 In the .NET Utilities Portal, click **Device Information** in the left panel.

The CSN displays as **Serial Number** in the right panel under **Device Attributes**.

IDGo 500 PKCS#11 Library

The IDGo 500 PKCS#11 library performs an MD5 hash of the CSN provided by the minidriver giving a result of 16-bytes. As the **C_GetTokenInfo** 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 IDGo 500 PKCS#11 library requests the 12-byte Card Serial Number (CSN) from the card ID file:
0x57 0x01 0x13 0x51 0x26 0xC7 0xD6 0x10 0x29 0x27 0xFF 0xFF
- 2 IDGo 500 PKCS#11 library 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 IDGo 500 PKCS#11 library 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 IDGo 500 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 IDGo 500 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.

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 IDGo 500 PKCS#11 library manages the challenge-response exchange using this key to perform all cryptographic operations requiring the SO PIN.

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

Caution: The security of the IDPrime .NET card is based upon the SO PIN. It is therefore very important to change the value from the default one.

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 );

    // Retrieve 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 IDGo 500 PKCS#11 library, 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 IDGo 500 PKCS#11 library. 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

The IDGo 500 PKCS#11 library comes in two versions, one for 64-bit operating systems and one for 32-bit operating systems (OS).

For a list of the operating systems supported by the IDGo 500 PKCS#11 library, please refer to the *Release Notes*.

Peripherals

The IDGo 500 PKCS#11 library 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:

www.gemalto.com/readers/index.html

You can download the drivers for Gemalto's readers at:

<http://support.gemalto.com/?id=184#292>

Smart Cards

The following types of IDPrime .NET smart card are supported:

- .NET V2
- .NET V2+
- IDPrime .NET (.NET V3 operating system)
- All devices containing any of the above 3 smart cards such as:
 - IDBridge tokens
 - IDPrime .NET tokens

Note: Although .NET v2.0 and v2+ smart cards are supported, the IDGo 500 PKCS#11 library has been designed to take advantage of the latest enhancements introduced in the .NET v3 Smart Card OS. Therefore, Gemalto highly recommends you use IDPrime .NET smart cards and IDBridge devices for the IDGo 500 PKCS#11 library.

Installing the IDGo 500 PKCS#11 Library

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 the IDGo 500 PKCS#11 library will not work on your computer.

PC/SC

The PC/SC layer must be installed on your system. This is included with all versions of Windows from XP to Windows 7 inclusive.

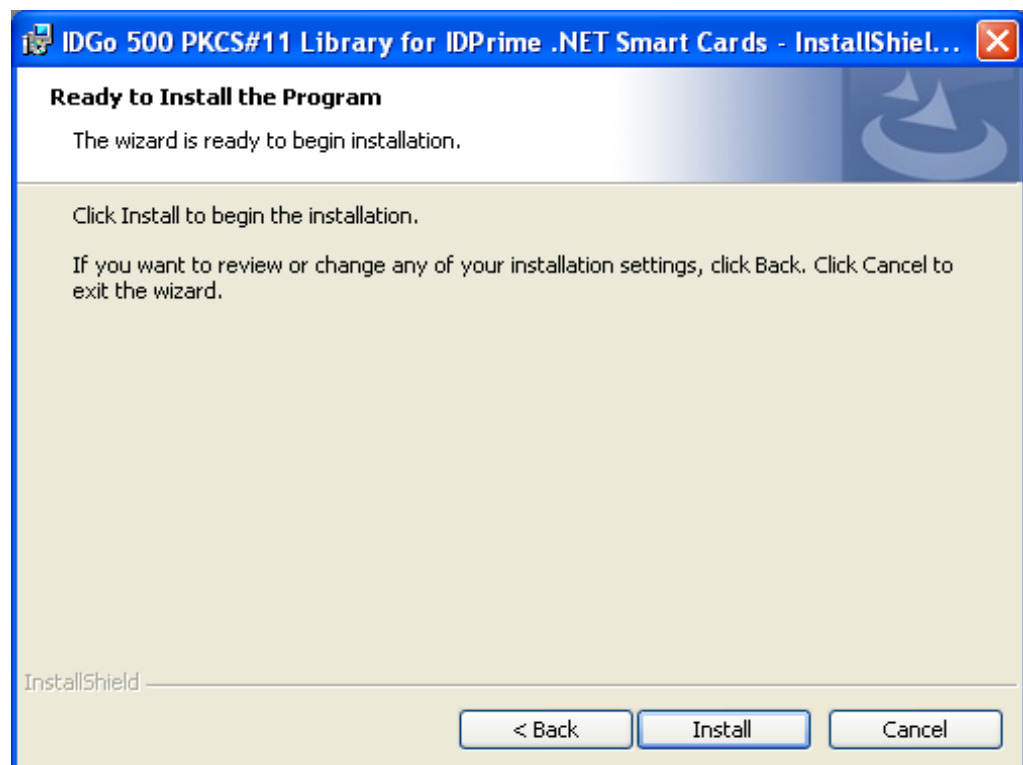
Installing the IDGo 500 PKCS#11 Library

Caution: Before installing the software, make sure you disconnect all devices (smart cards/tokens).

To install the IDGo 500 PKCS#11 library:

- 1 Locate the IDGo500PKCS11Libraries.msi file. It should be in the same location as where you found this reference manual. If for some reason, you cannot find it, you can download it from:
http://www.gemalto.com/products/dotnet_card/resources/libraries.html?toggl=0
- 2 Double-click the IDGo500PKCS11Libraries.msi file. This starts the InstallShield Wizard.
- 3 Wait until the **Welcome** window appears and click **Next**.
- 4 In the **License Agreement** window, accept the terms and click **Next**.
 The **Ready to Install** window appears as shown in “Figure 4”.

Figure 4 - Install Shield Wizard - Ready To Install Window



- 5 Click **Install**. A progress window displays while the installation takes place.
- 6 Wait until the final **InstallShield Wizard Completed** appears, then click **Finish**.

Note: Two .dll files are installed to your PC; "gtop11dotnet.dll" for 32-bit versions of Windows and "gtop11dotnet64.dll" for 64-bit versions.

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 IDGo 500 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 IDPrime .NET smart card. For details on how to do this, see “Appendix C - Configuring PKCS#11 in Mozilla”.

Uninstalling the IDGo 500 PKCS#11 Library

You must have administrator rights to the computer on which you are installing the library.

To remove the IDGo 500 PKCS#11 Library in Windows XP and Server 2003:

- 1 Open the **Control Panel** (**Start > Settings > Control Panel**).
- 2 Double-click **Add or Remove Programs**.
- 3 Locate **IDGo 500 PKCS#11 Library for IDPrime .NET Smart Cards**. (To remove versions that precede the renaming to IDGo 500, locate **Gemalto .NET PKCS#11**).
- 4 Click **Remove**. A message box displays asking “Are you sure you want to remove IDGo 500 PKCS#11 Library for IDPrime .NET Smart Cards from your computer.
- 5 Click **Yes** to confirm the removal. A progress bar appears during the removal.
At the end of the removal, the progress bar closes, removal is complete and the IDGo 500 PKCS#11 library is removed from your computer.

To remove the IDGo 500 PKCS#11 Library in Windows Vista, 7, Server 2008 & Server 2008 R2:

- 1 Open the **Control Panel** (**Start > Control Panel**).
- 2 Double-click **Programs and Features** (if you are using the Control Panel view under Vista/Server 2008 or the Category view under 7/Server 2008 R2, then under **Programs**, click **Uninstall a program** instead).
- 3 Select **IDGo 500 PKCS#11 Library for IDPrime .NET Smart Cards**, (to remove versions that precede the renaming to IDGo 500, select **Gemalto .NET PKCS#11**) and click **Uninstall** (the **Uninstall** button appears when you select **IDGo 500 PKCS#11 Library for IDPrime .NET Smart Cards**).
- 4 A message box displays asking “Are you sure you want to uninstall IDGo 500 PKCS#11 Library for IDPrime .NET Smart Cards?”.
- 5 Click **Yes** to confirm the removal.
- 6 If User Account Control is activated, the warning “An unidentified program wants access to your computer” appears. Choose **Allow**.
- 7 Again, if User Account Control is activated, it is possible that a message may appear to tell you to close certain applications. If so, choose the **Automatically close applications** option and click **OK**.
- 8 A progress bar appears during the removal.
At the end of the removal, the progress bar closes, removal is complete and the IDGo 500 PKCS#11 library is removed from your computer.

Tasks

This chapter discusses information related to specific tasks that you will most often be required to carry out when using the IDGo 500 PKCS#11 Library 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 IDPrime .NET Card” on page 23
- “How to Delete a Certificate from the IDPrime .NET Card” on page 27
- “How to View the Details of a Certificate in an IDPrime .NET Card” on page 28
- “How to Unblock a User PIN” on page 29
- “How to Change a User PIN” on page 30
- “How to Use E-mail Securely” on page 33
- “How to View Secure Web Sites” on page 37

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 IDPrime .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 IDPrime .NET Card

You can either use Firefox to import a certificate in the IDPrime .NET card or one of Gemalto's tools such as the Minidriver Manager tool, the .NET Utilities tool, DAS or VSEC:CMS or any other Card Management System compatible with .NET cards.

Note: Please refer to "Appendix D - The Minidriver Manager Tool" before using the Minidriver Manager. Please refer to "Appendix E - The .NET Utilities Tool" for information on how to access the .NET Utilities tool.

For more information about DAS go to:

<http://www.gemalto.com/products/das/>

For more information about VSEC:CMS go to:

<http://www.gemalto.com/products/versatile/>

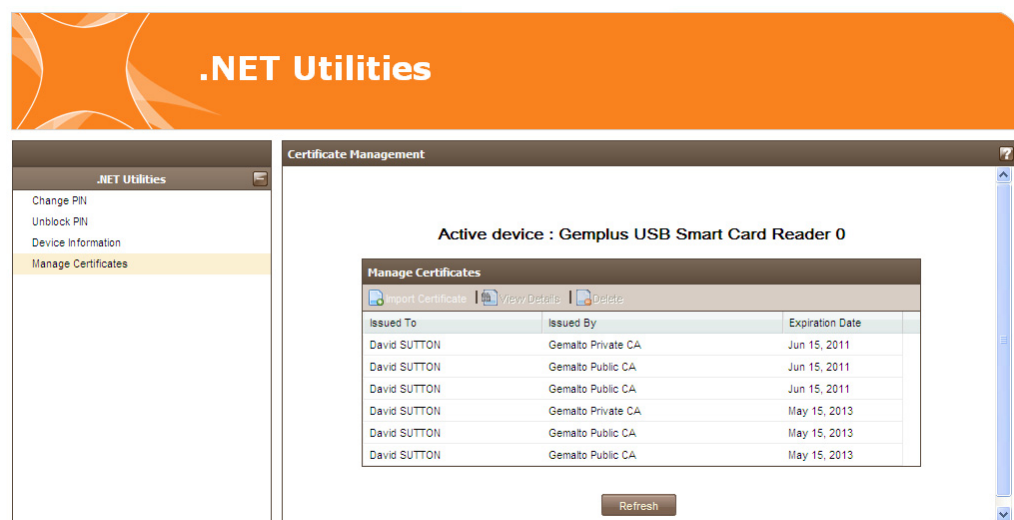
To import certificates in IDPrime .NET cards using the Minidriver Manager:

- 1 Start the Minidriver Manager and insert the IDPrime .NET card in the reader.
- 2 Click **Connect**.
- 3 In the **Select Smart Card** dialog, select the correct reader and click **OK**.
- 4 In the left panel, select a container, right-click and choose **Load Container**.

To import certificates in IDPrime .NET cards using the .NET Utilities tool:

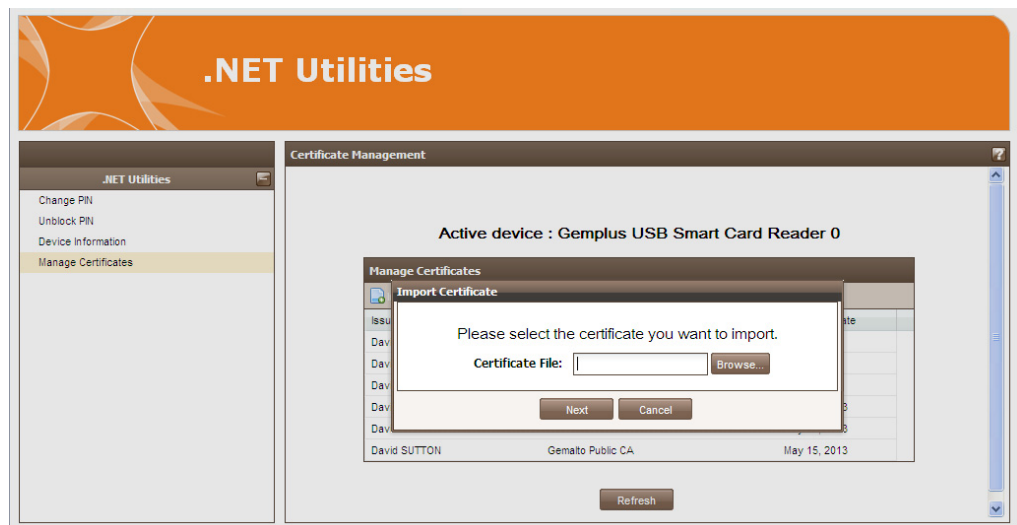
- 1 Access the .NET Utilities tool and insert the IDPrime .NET card in the reader as described in "Appendix E - The .NET Utilities Tool".
- 2 In the .NET Utilities Portal, click **Manage Certificates** in the left panel. The right panel displays the certificates currently present in the card, as shown:

Figure 5 - .NET Utilities Portal – Manage Certificates



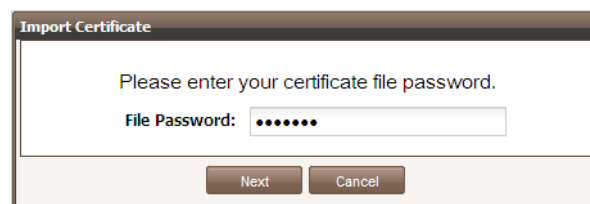
- 3 Click **Import Certificate**. The Import Certificate dialog appears:

Figure 6 - .NET Utilities Portal – Import Certificate



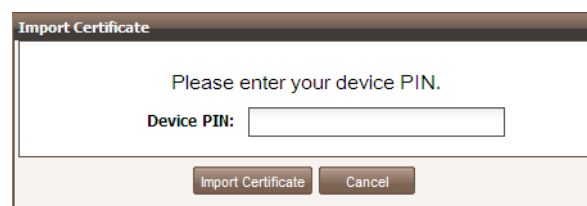
- 4 Browse to the certificate file on your computer and click **Next**. Certificate files must be .pfx or .p12. You are prompted to enter the password for the certificate as follows:

Figure 7 - .NET Utilities Portal – Import Certificate – Password Prompt



- 5 Enter the password and click **Next**. You are prompted to enter the PIN for the IDPrime .NET smart card as follows.

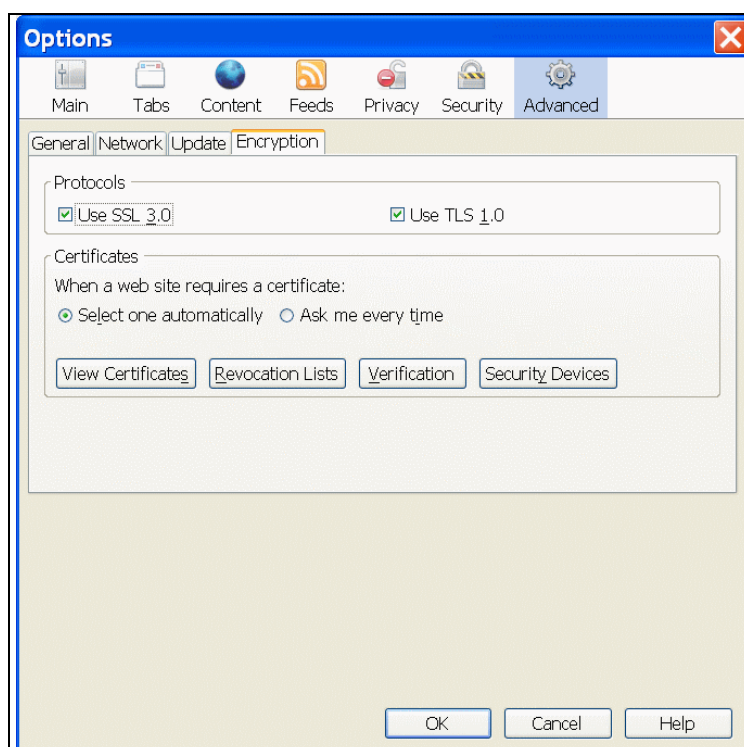
Figure 8 - .NET Utilities Portal – Import Certificate – Password Prompt



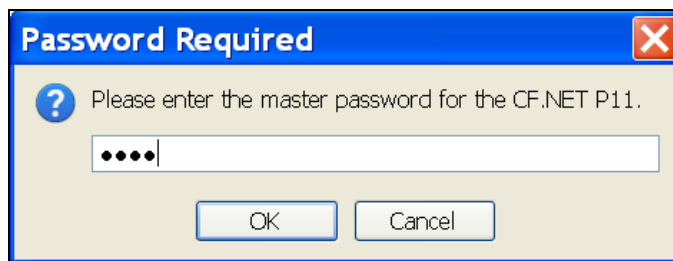
- 6 Enter the PIN and click **Import Certificate**.

To import certificates in IDPrime .NET cards using Mozilla Firefox:

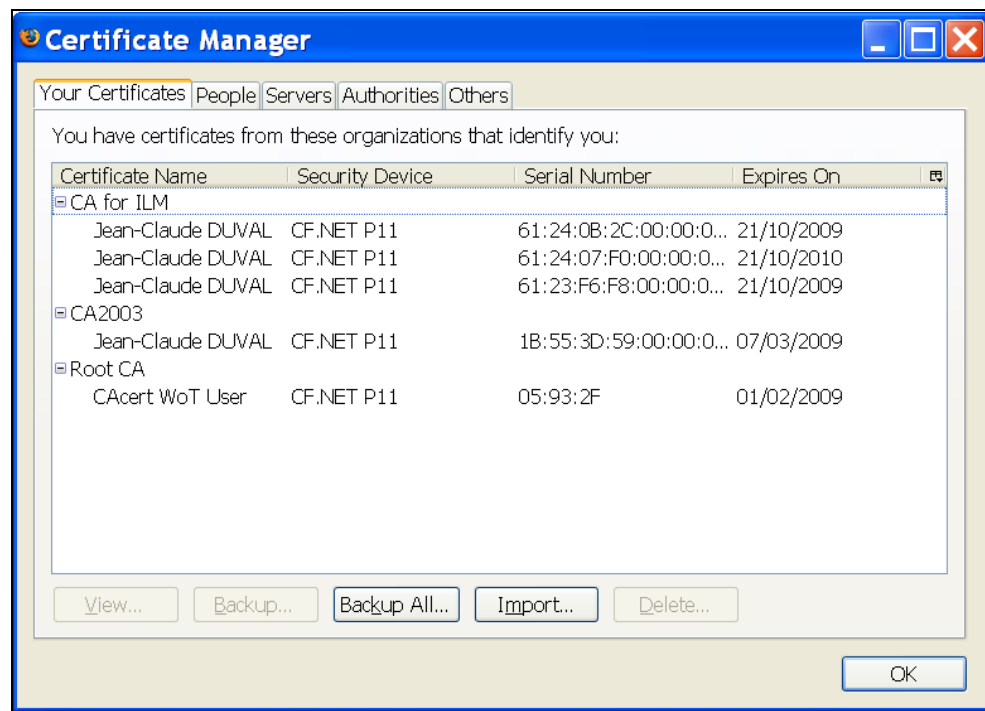
- 1 Make sure your card/token is connected.
- 2 Open the **Mozilla Firefox** browser and from the **Tools** menu choose **Options**.
- 3 Click the **Advanced** icon, then the **Encryption** tab as shown in "Figure 9".

Figure 9 - Mozilla Firefox Encryption Options Dialog

- 4 Click **View Certificates**. You will be prompted for a password as shown in “Figure 10”.

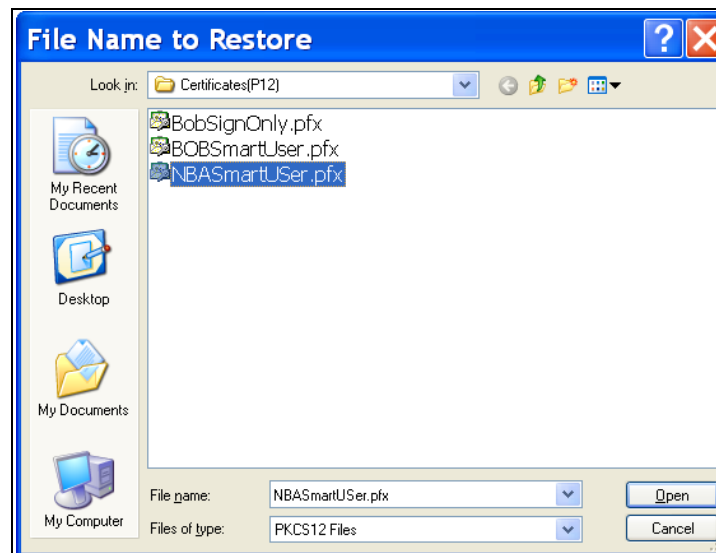
Figure 10 - Password Required

- 5 Enter the User PIN for your card/token
The **Certificate Manager** window appears as shown in “Figure 11”.

Figure 11 - Certificate Manager Window

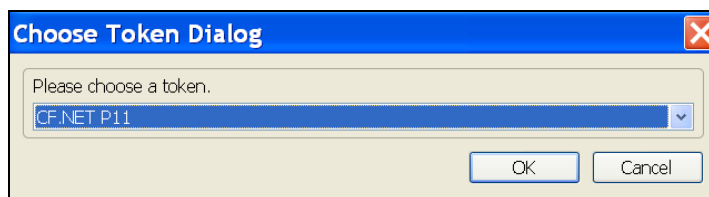
The certificates that are currently stored on the card/token appear under **Your Certificates**.

- 6 Click **Import**. This opens a window called **File Name to Restore** as shown in "Figure 12".

Figure 12 - File Name to Restore Window

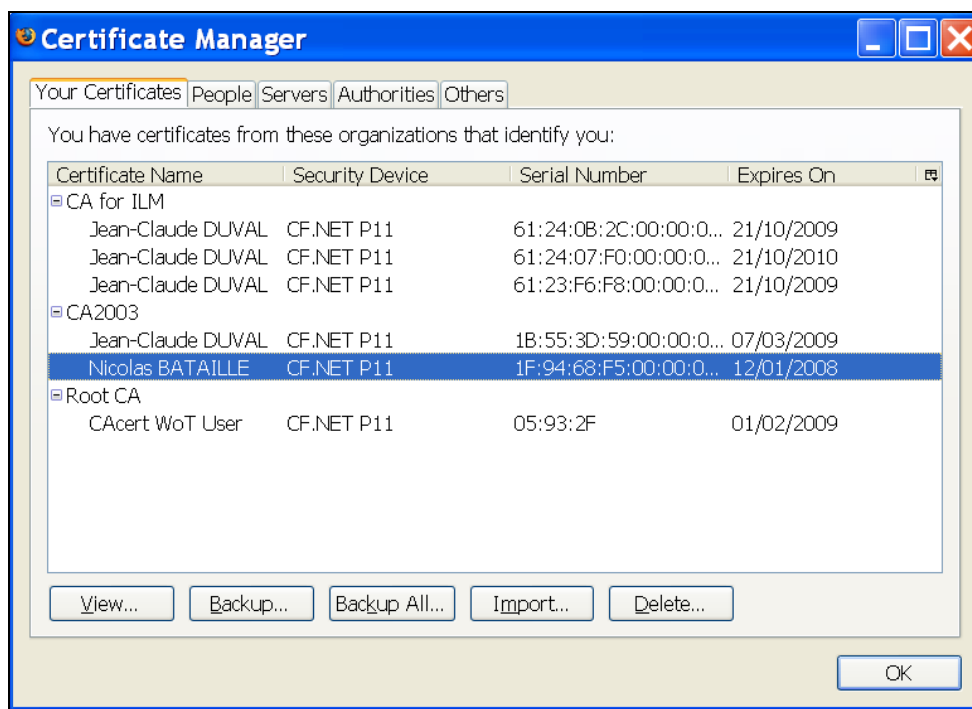
- 7 Navigate to the file containing the PKCS#12 certificate (these files end in a .pfx suffix) and click **Open**.

The **Choose Token Dialog** window opens.

Figure 13 - Choose Token Dialog Window

- 8 Select your IDPrime .NET card/token from the list and click **OK**.
- 9 Enter the password for the IDPrime .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 14".

Figure 14 - Certificate Manager After Certificate Importation

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

How to Delete a Certificate from the IDPrime .NET Card

You can either use Firefox to delete a certificate from the IDPrime .NET card or one of Gemalto's tools, such as the Minidriver Manager or .NET Utilities tool.

Note: Please refer to "Appendix D - The Minidriver Manager Tool" before using the Minidriver Manager. Please refer to "Appendix E - The .NET Utilities Tool" for information on how to access the .NET Utilities tool.

To delete certificates in IDPrime .NET cards using the Minidriver Manager:

- 1 Start the Minidriver Manager and insert the IDPrime .NET card in the reader.
- 2 Click **Connect**.
- 3 In the **Select Smart Card** dialog, select the correct reader and click **OK**.
- 4 In the left panel, select a container, right-click and choose **Delete Container**.

To delete certificates in IDPrime .NET cards using the .NET Utilities tool:

- 1 Access the .NET Utilities tool and insert the IDPrime .NET card in the reader as described in “Appendix E - The .NET Utilities Tool”.
- 2 In the .NET Utilities Portal, click **Manage Certificates** in the left panel. The right panel displays the certificates currently present in the card as shown in “Figure 5 - .NET Utilities Portal – Manage Certificates” on page 23
- 3 Select the certificate you want to delete in the list and click **Delete**.

To delete certificates in IDPrime .NET cards using Mozilla Firefox:

- 1 Follow steps 1 to 5 in “How to Import a Certificate in the IDPrime .NET Card” on page 23.
- 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 IDPrime .NET card and no longer appears in the **Certificate Manager**.

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

You can either use Firefox to view a certificate in a IDPrime .NET card or one of Gemalto’s tools, such as the Minidriver Manager or .NET Utilities tool

Note: Please refer to “Appendix D - The Minidriver Manager Tool” before using the Minidriver Manager. Please refer to “Appendix E - The .NET Utilities Tool” for information on how to access the .NET Utilities tool.

To view a certificate in a IDPrime .NET card using the Minidriver Manager:

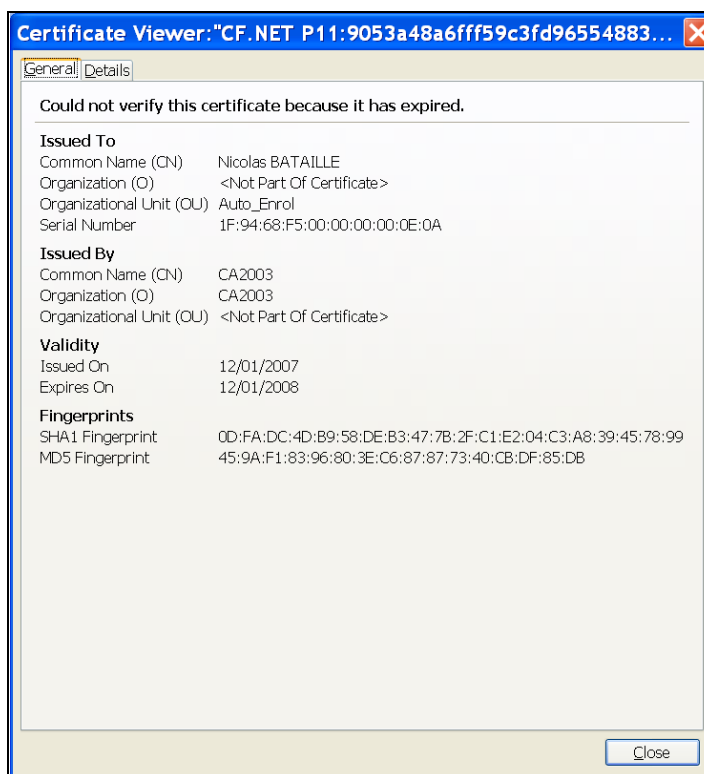
- 1 Start the Minidriver Manager and insert the IDPrime .NET card in the reader.
- 2 Click **Connect**.
- 3 In the **Select Smart Card** dialog, select the correct reader and click **OK**.
- 4 In the left panel, select a container. Its contents display in the right panel.

To view a certificate in IDPrime .NET cards using the .NET Utilities tool:

- 1 Access the .NET Utilities tool and insert the IDPrime .NET card in the reader as described in “Appendix E - The .NET Utilities Tool”.
- 2 In the .NET Utilities Portal, click **Manage Certificates** in the left panel. The right panel displays the certificates currently present in the card as shown in “Figure 5 - .NET Utilities Portal – Manage Certificates” on page 23.
- 3 Double click the certificate that you want to view. A **Certificate Viewer** window opens, displaying details about the certificate.

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

- 1 Follow steps 1 to 5 in “How to Import a Certificate in the IDPrime .NET Card” on page 23.
- 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 15”.

Figure 15 - Certificate Details

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

How to Unblock a User PIN

There are several ways to do this:

- Use Gemalto’s Minidriver Manager tool.

Note: Please refer to “Appendix D - The Minidriver Manager Tool” before using the Minidriver Manager.

- Use Gemalto’s .NET Utilities tool.

Note: Please refer to “Appendix E - The .NET Utilities Tool” for information on how to access the .NET Utilities tool.

- 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 14.
- Use a card management system (CMS) such as Gemalto’s DAS or VSEC:CMS or a third party CMS.

To unblock a User PIN in IDPrime .NET cards using the Minidriver Manager:

- 1 Start the Minidriver Manager and insert the IDPrime .NET card in the reader.
- 2 Click **Connect**.
- 3 In the **Select Smart Card** dialog, select the correct reader and click **OK**.
- 4 Click **Unblock PIN**.

To unblock a User PIN in IDPrime .NET cards using the .NET Utilities tool:

- 1 Access the .NET Utilities tool and insert the IDPrime .NET card in the reader as described in “Appendix E - The .NET Utilities Tool”.
- 2 In the .NET Utilities Portal, click **Unblock PIN** in the left panel. The right panel displays some information about the card.
- 3 Enter the new value for the PIN in **New PIN** and **Confirm New PIN** and click **Unblock**.

Note: The **Reset** button clears the **New PIN** and **Confirm New PIN** fields. The **New Challenge** button generates a new challenge, which may be used to remotely unblock the PIN. You can ignore this feature.

How to Change a User PIN

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

- Use Gemalto’s Minidriver Manager tool.

Note: Please refer to “Appendix D - The Minidriver Manager Tool” before using the Minidriver Manager. Please refer to “Appendix E - The .NET Utilities Tool” for information on how to access the .NET Utilities tool.

- Use Gemalto’s .NET Utilities tool.

Note: Please refer to “Appendix E - The .NET Utilities Tool” for information on how to access the .NET Utilities tool.

- Use a card management system (CMS) such as Gemalto’s DAS or VSEC:CMS or a third party CMS.
- Use Mozilla Firefox, as described in the following section:

To change a User PIN in IDPrime .NET cards using the Minidriver Manager:

- 1 Start the Minidriver Manager and insert the IDPrime .NET card in the reader.
- 2 Click **Connect**.
- 3 In the **Select Smart Card** dialog, select the correct reader and click **OK**.
- 4 Click **Change PIN**.

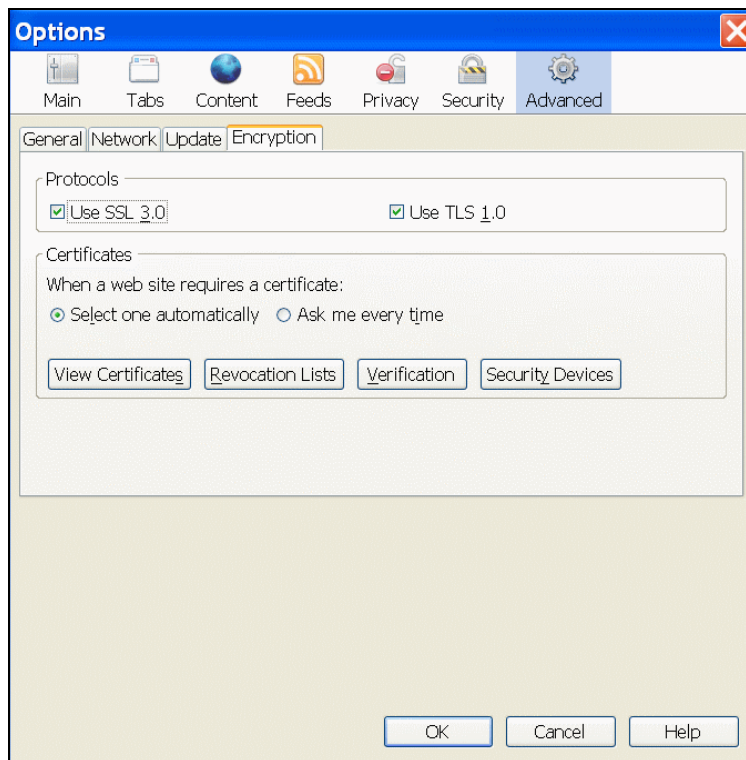
To change a User PIN in IDPrime .NET cards using the .NET Utilities tool:

- 1 Access the .NET Utilities tool and insert the IDPrime .NET card in the reader as described in “Appendix E - The .NET Utilities Tool”.
- 2 In the .NET Utilities Portal, click **Change PIN** in the left panel.
- 3 In the right panel, enter the current value of the PIN and **Current PIN** and the new value for the PIN in **New PIN** and **Confirm New PIN**.
- 4 Click **Change PIN**.

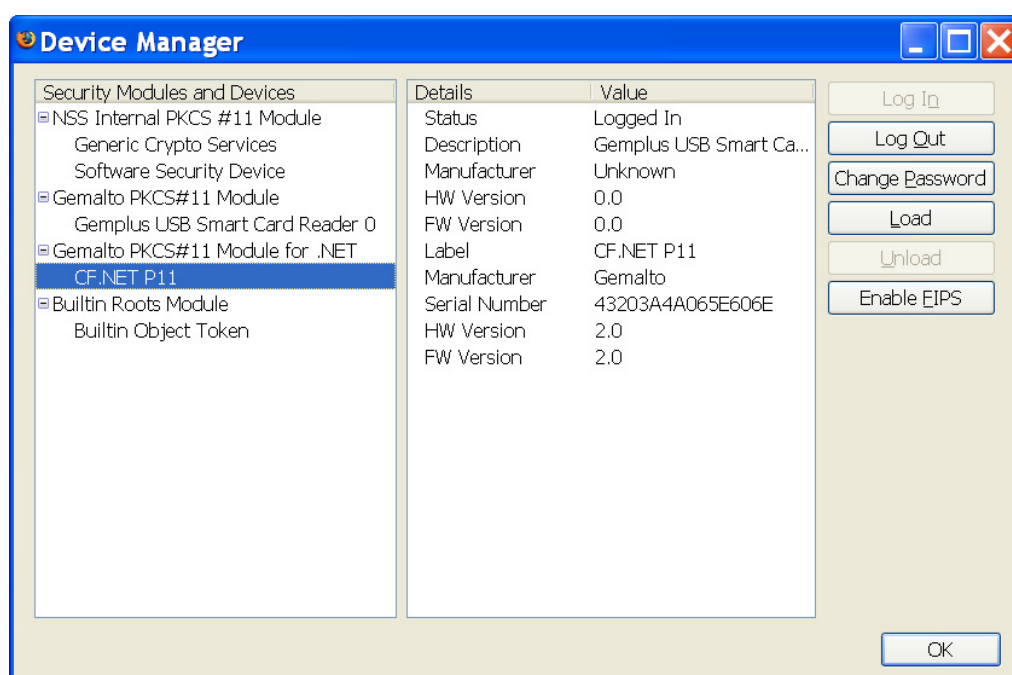
To change a User PIN in a IDPrime .NET card using Mozilla Firefox:

- 1 Make sure your card/token is connected.
- 2 Open the **Mozilla Firefox** browser and from the **Tools** menu choose **Options**.
- 3 Click the **Advanced** icon, then the **Encryption** tab as shown in “Figure 16”.

Figure 16 - Mozilla Firefox Encryption Options Dialog



- 4 Click **Security Devices** to display the **Device Manager** window. This displays the modules currently available as shown in “Figure 17”.

Figure 17 - Device Manager

- 5 In **Device Manager**, select the card whose PIN you want to change, as shown in "Figure 17".
- 6 Click **Change Password**. The window shown in "Figure 18" appears.

Figure 18 - 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 the IDGo 500 PKCS#11 Library.

About Secure E-mail

With the IDGo 500 PKCS#11 Library, 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. 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 36.

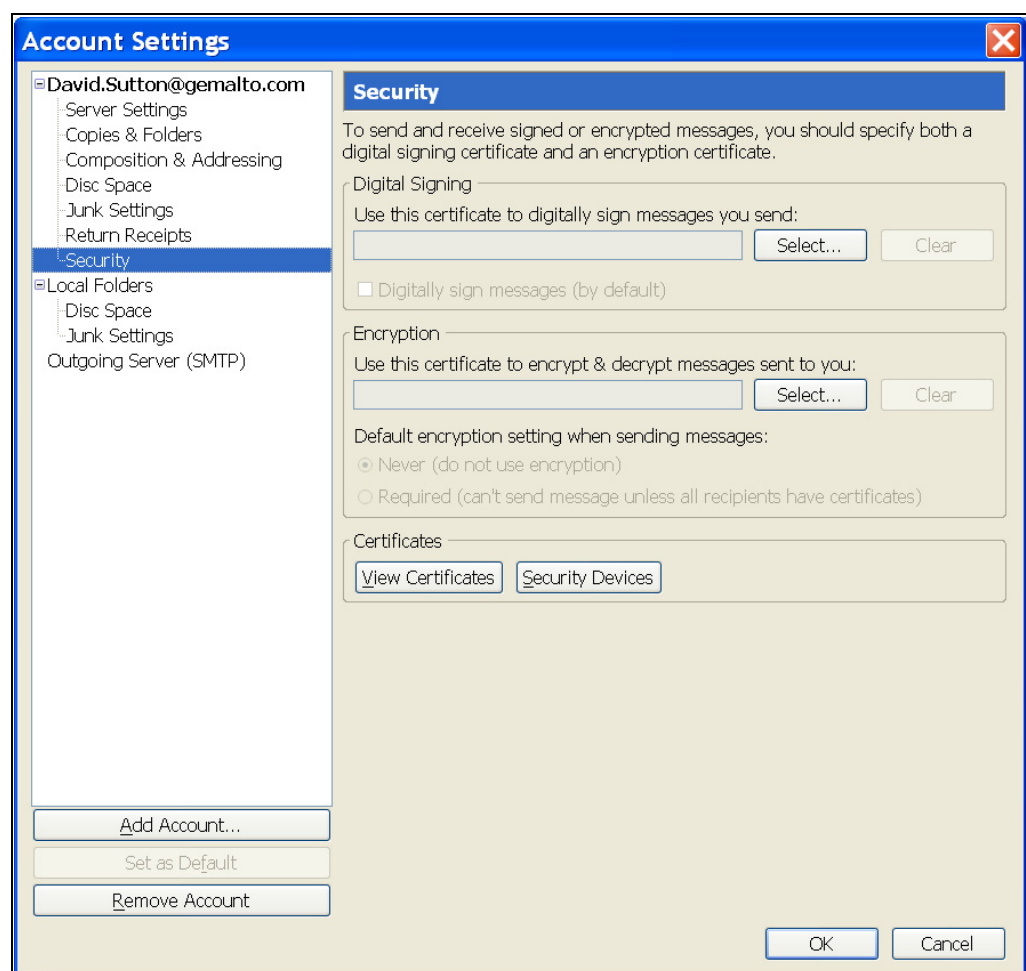
Configuring Settings and Specifying Certificates

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

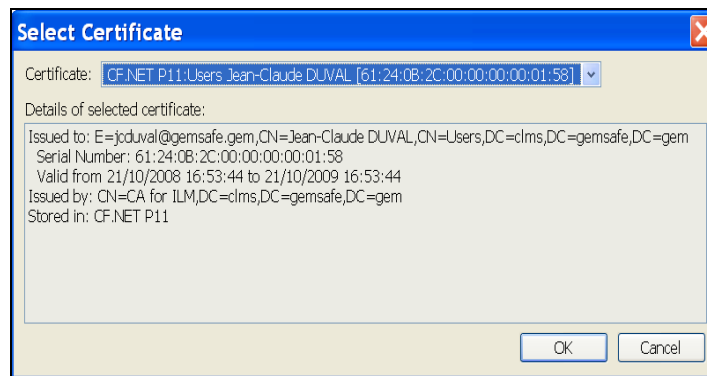
Note: Although selecting the certificates is mandatory, this does not mean that you must sign and encrypt e-mails.

- 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 **Tools** menu, choose **Account Settings**.
- 5 Select **Security** as shown in “Figure 19”.

Figure 19 - Thunderbird – Security Account Settings



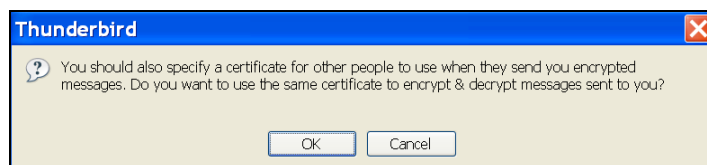
- 6 In **Digital Signing**, click **Select**. The window shown in “Figure 20” appears.

Figure 20 - Thunderbird - Select Certificate

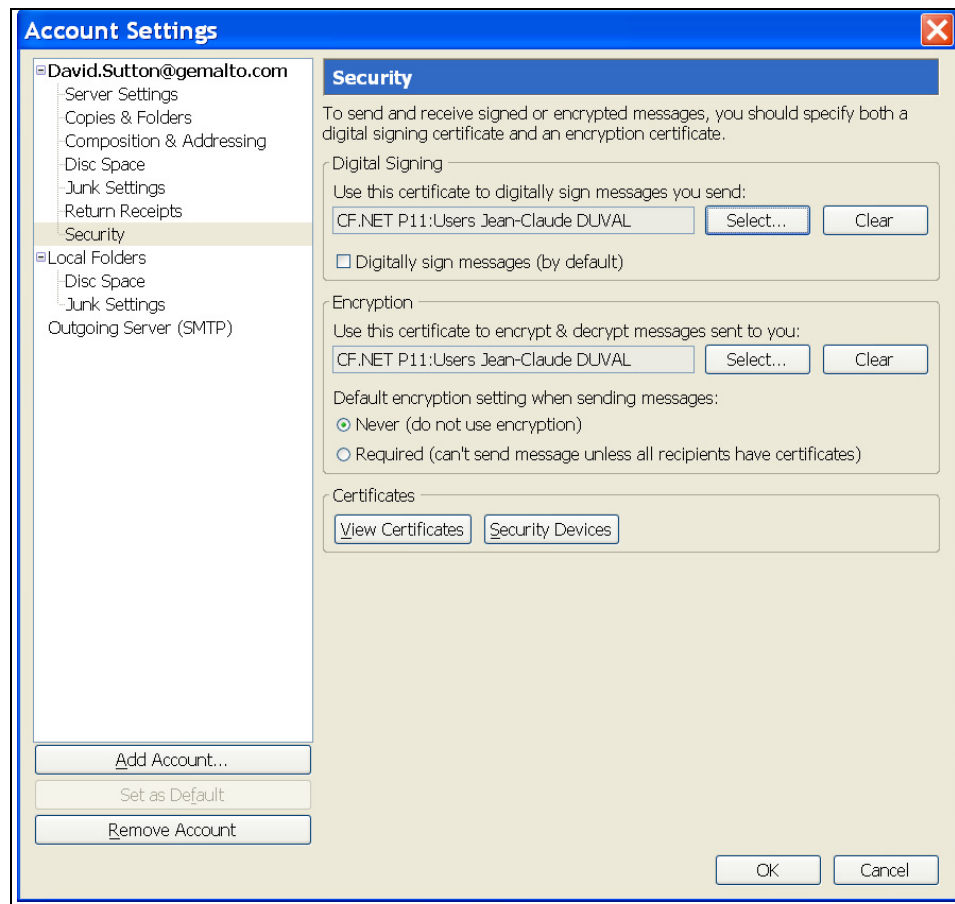
- 7 Select the certificate you want to use from the list that appears (its details appear in the window).

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 21 - Thunderbird – “Use Same Certificate” Message

- 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 22”. Otherwise click **Cancel**.

Figure 22 - Thunderbird – Security Account Settings (2)

- 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 21” 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 IDPrime .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 Firefox as an example.

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:

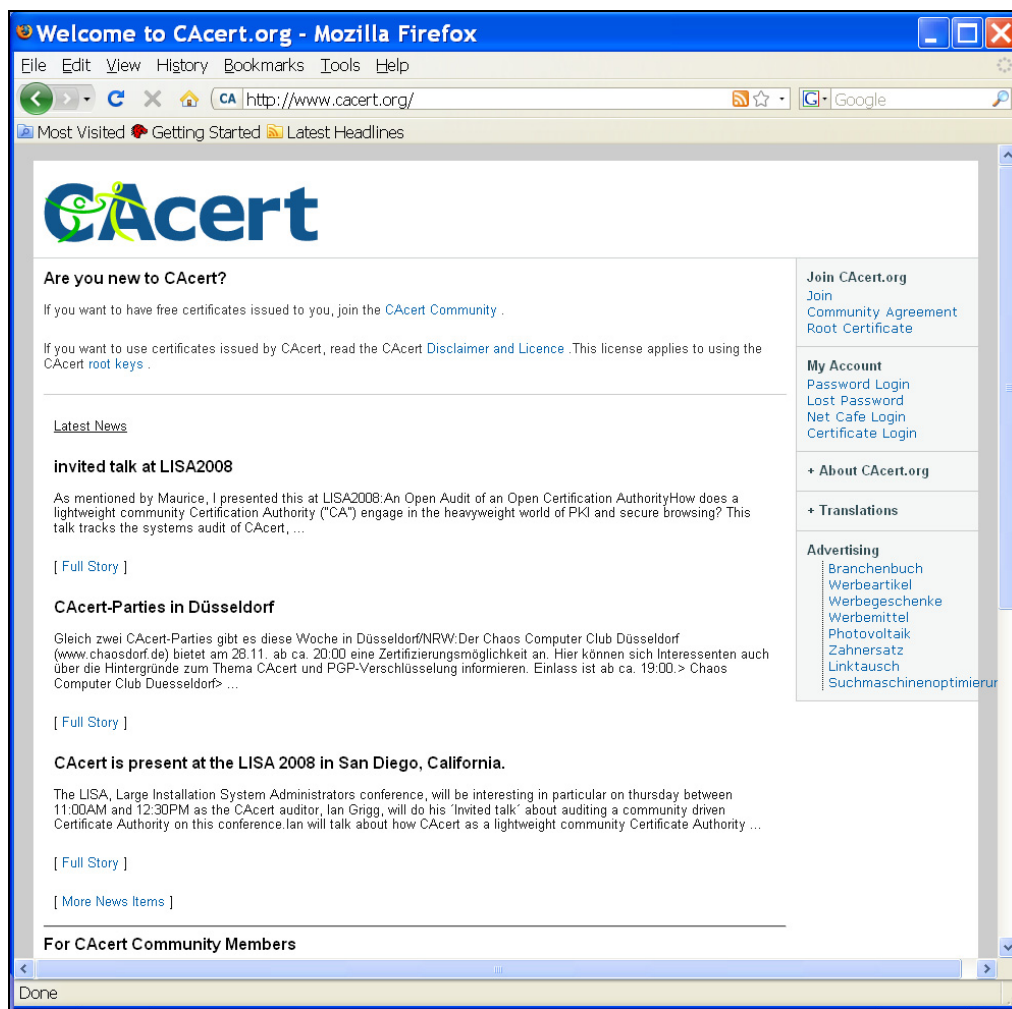
- 1 Follow the procedure for Firefox described in “How to Import a Certificate in the IDPrime .NET Card” on page 23 until you open the Certificate Manager as shown in “Figure 11” on page 26.
- 2 Make sure that all the certificates that you want to use to authenticate to secure web sites in Firefox appear under **Your Certificates**.

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

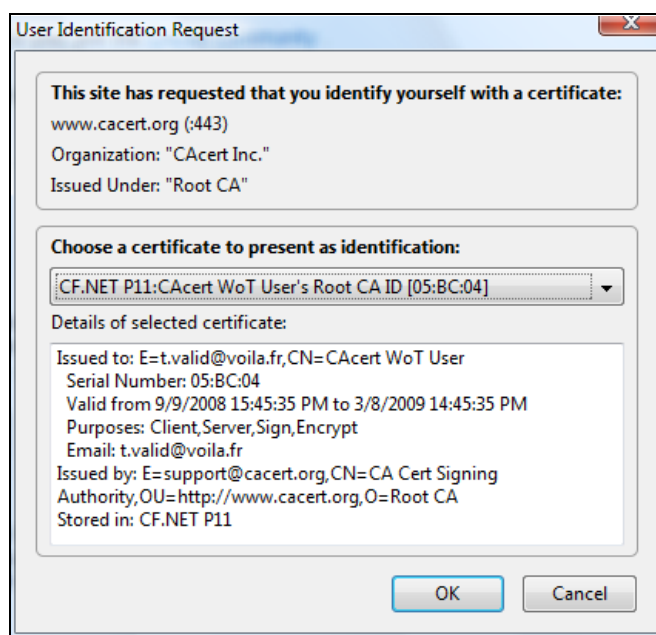
The following procedure provides an example of logging on to a page in the cacert.org web site, a site where you can obtain authorized certificates, that is secured.

To authenticate to a secured web page using Firefox:

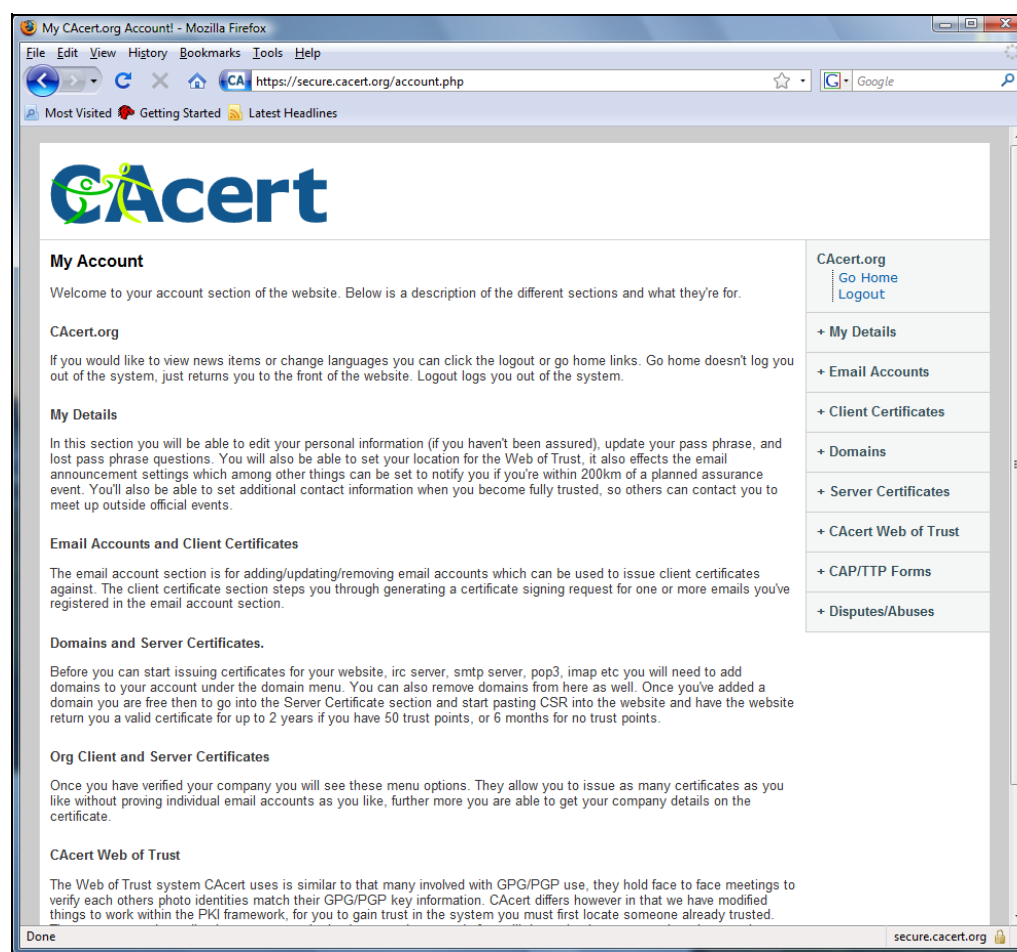
- 1 Make sure the IDPrime .NET smart card is connected and go to the web site in Firefox, as shown in “Figure 23”.

Figure 23 - Example Web Site Before Authentication

- 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 IDPrime .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 79), you are asked to select a certificate to authenticate yourself as shown in "Figure 24".

Figure 24 - User Identification Request Window

- 5 Select the certificate and click **OK**. You are given access to the secured web page as shown in "Figure 25".

Figure 25 - Secured Web Page After Authentication

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.

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\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 */
#ifdef 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);
}

#ifdef 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;
    }

#ifdef 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();

#ifdef 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 75 and “slothevent.c” on page 72.

```
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\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);
    }

#ifdef 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>

#ifdef 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                Attribute 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 attributes 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 attribute doesn't exist for this object */
    break;
default:
    CKRLOG("C_GetAttributeValue", rv);
    /* A real error has occurred. */
    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\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++)

```



```
/*----- 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 50) 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, 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 );
    }
    // Unblock 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 65).

```

/* 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__

#ifdef _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
#endif
```

```
#endif

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

#pragma pack(pop, cryptoki)

#else /* not windows */

#define CK_PTR *

#define CK_DECLARE_FUNCTION(returnType, name) \
    returnType name

#define CK_DECLARE_FUNCTION_POINTER(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:

Conversion from .NET PKCS#11 2.1 to 2.2

The way in which Root certificates are imported in version 2.1 causes a problem when you upgrade to version 2.2. The card's minidriver file system manages this incorrectly, which means that the card is not interpreted correctly by the host minidriver, even though it may appear to be fine as far as the card is concerned.

This means that when a card that was personalized under version 2.1 of .NET PKCS#11 is read for the first time by version 2.2, the IDGo 500 PKCS#11 library corrects the card's file system automatically. Unfortunately this correction means that the garbage collector has to create new files and delete old ones which is resource intensive and can take a long time (around 20 seconds for each certificate that needs to be moved).

Note: This automatic correction of the file system is only performed once - the first time the card is read by IDPrime .NET PKCS#11 2.2.

Performance Problems With Memory Management

The IDPrime .NET card can be slow when performing operations that require using the card's memory such as, for example, loading a large number of certificates or keys, or large PKCS#11 data objects. This is mainly operations that involve the garbage collector.

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.

Fast User Switching

If the end-user switches from one account to another while Firefox/Thunderbird is running, the PC/SC context is broken.

Firefox/ Thunderbird must be restarted after the switch to communicate with the smart card.

Remote Desktop Connection

The end-user must log off from any active remote desktop connection before accessing it from the host.

CheckPoint VPN Client NGX

On a Windows 64-bit OS, CheckPoint is not able to enroll the end-user on the smart card.

Citrix Server

Applications using the IDGo 500 PKCS#11 library (such as Firefox) remain in memory after the end-user closes the application. Check the latest hotfix on the Citrix web site.

Adobe Acrobat Reader

Adobe Acrobat Reader does not support certificate importation.

Configuring PKCS#11 in Mozilla

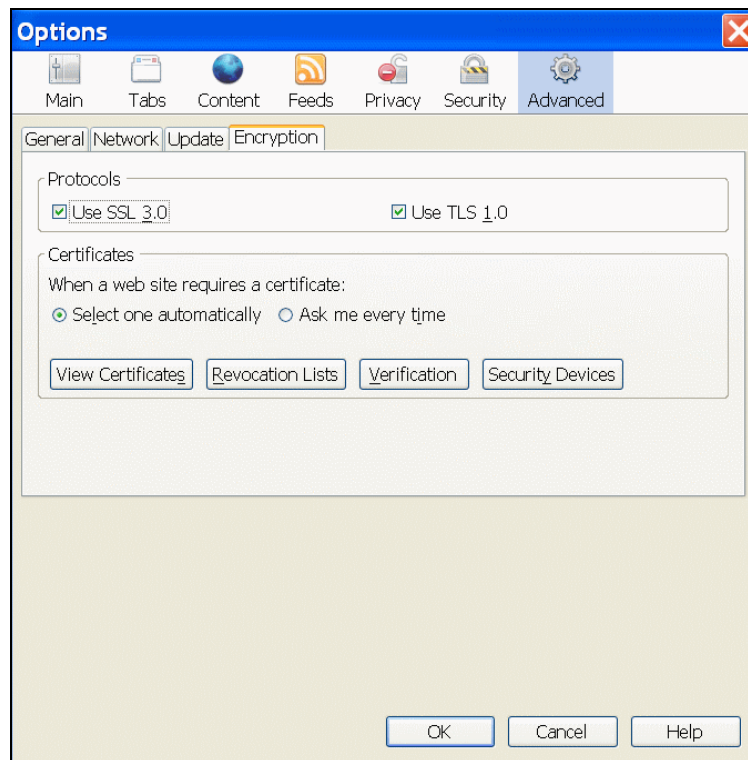
This appendix describes how to configure Mozilla applications so they can communicate with the PKCS#11 security module.

Firefox

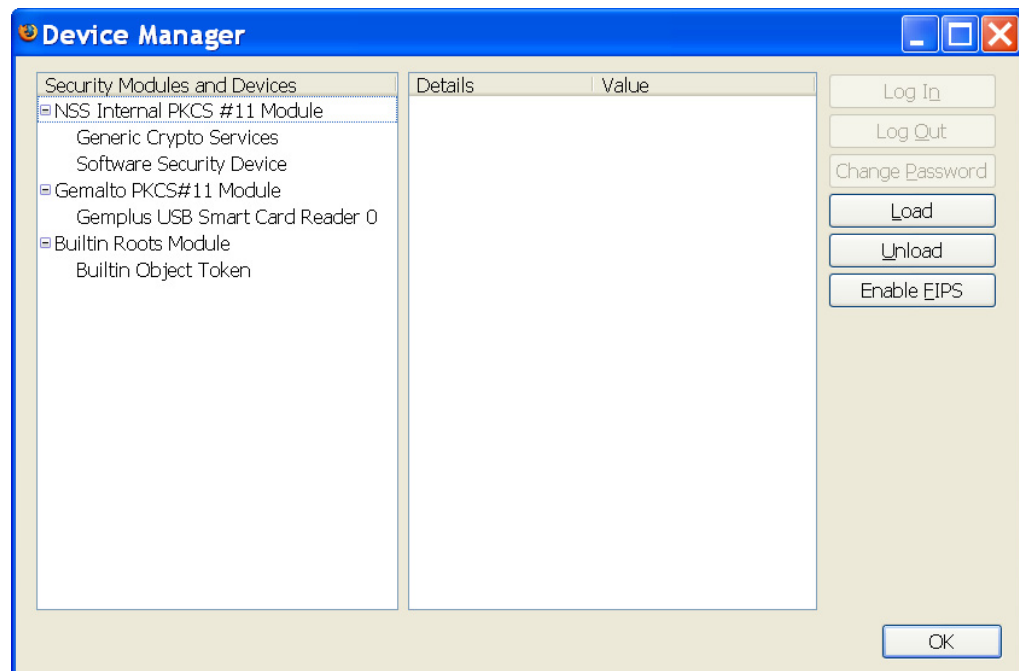
This section describes the necessary configuration for Firefox. You only need to do this once.

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

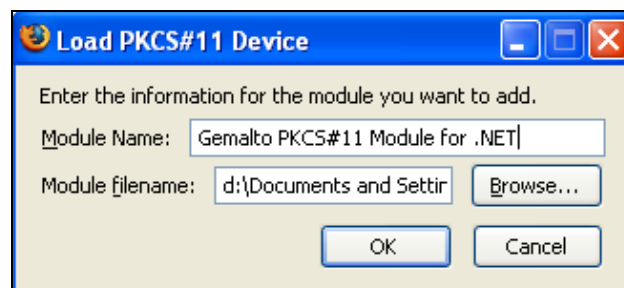
- 1 Make sure your card/token is connected.
- 2 Open the **Mozilla Firefox** browser and from the **Tools** menu choose **Options**.
- 3 Click the **Advanced** icon, then the **Encryption** tab as shown in “Figure 26”.

Figure 26 - Mozilla Firefox Encryption Options Dialog

- 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 27”.

Figure 27 - Device Manager

- 6 Click the **Load** button to the right in the dialog. This displays the **Load PKCS#11 Device** window, as shown in “Figure 28”.

Figure 28 - Load PKCS#11 Device Window

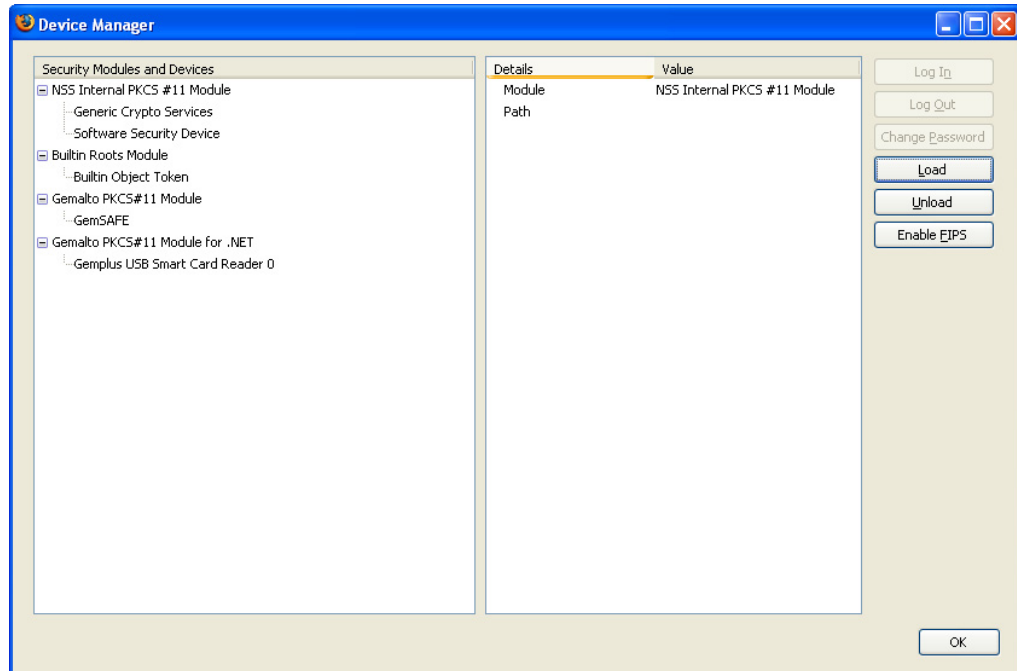
- 7 Enter a **Module Name**.
- 8 In **Module filename**, use the **Browse** button to select the .dll file as follows:
- C:\Program Files\Gemalto\DotNet PKCS11\gtop11dotnet.dll - for 32-bit versions of Windows
 - C:\Program Files\Gemalto\DotNet PKCS11\gtop11dotnet64.dll - for 64-bit versions of Windows

Note: If you are running a 64-bit version of Windows, the location is “Program Files (x86)” instead of “Program Files”.

- 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 29”:

Figure 29 - Device Manager After Module Configuration



Note: The example shown in “Figure 29” shows “smart card reader” because no card is inserted in the reader. If a card is inserted at the time you are loading the module, then the name of the card appears instead of the reader.

Thunderbird

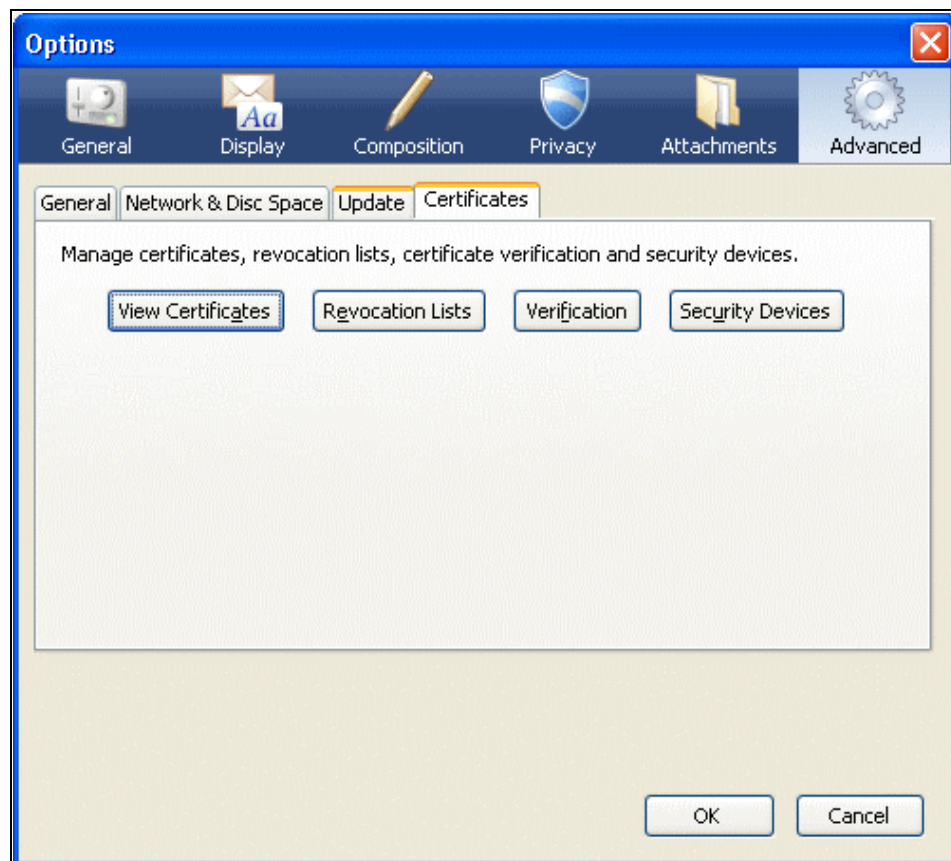
You only need to make this configuration once in Thunderbird.

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 **Tools** menu, choose **Options**.
- 5 In the dialog box that opens, click the **Advanced** icon, then the **Certificates** tab to display the settings as shown in "Figure 30".

Figure 30 - Thunderbird - Certificates Tab



- 6 The rest of the procedure is the same as that described for Firefox. Continue from step 5 on page 80.

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

The Minidriver Manager Tool

The Minidriver Manager tool is an R&D tool developed by Gemalto. It has been placed on the http://www.gemalto.com/products/dotnet_card/ web site to provide some additional useful help to customers. As such it is not officially supported by Gemalto.

In order to use the Minidriver Manager tool, the minidriver dll must be installed on your computer.

For Windows XP, Vista, Server 2003 and Server 2008, you will need to install the minidriver dll manually.

For Windows 7 and Windows Server 2008 R2, the dll is installed automatically by the Windows “plug and play” feature when you insert the IDPrime .NET card. However if your administrator has blocked this function on your computer, it will not work and you will need to install the minidriver dll manually.

For instructions on how to install the minidriver dll manually, please refer to the *IDPrime .NET Smart Cards in a Windows Environment Administration and User Guide*.

You can download the Minidriver Manager tool from:

http://www.gemalto.com/products/dotnet_card/resources/development.html?toggler=0

The .NET Utilities Tool

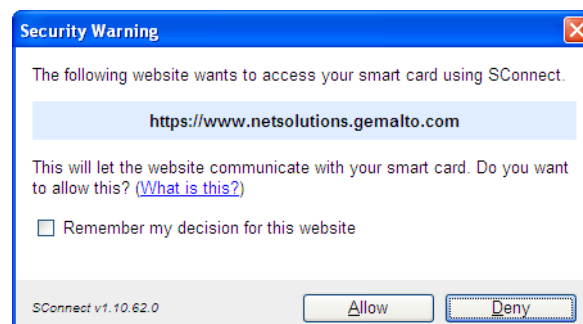
The .NET Utilities tool is an R&D tool developed by Gemalto. It has been placed on the <https://www.netsolutions.gemalto.com/netutils/Default.aspx> web site to provide some additional useful help to customers. As such it is not officially supported by Gemalto.

One advantage of using the .NET utilities tool is that it does not require any files to be installed on your computer.

To access the .NET Utilities tool:

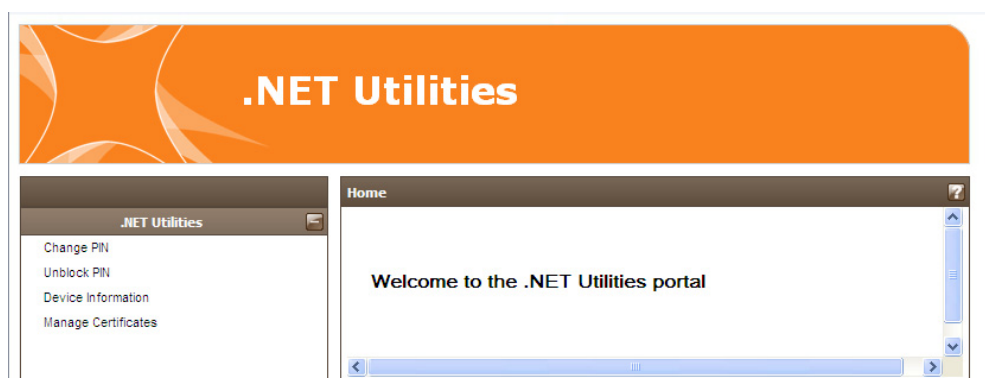
- 1 Click <https://www.netsolutions.gemalto.com/netutils/Default.aspx>. This displays the following security warning:

Figure 31 - .NET Utilities Portal – Security Warning



- 2 Click **Allow**.

Figure 32 - .NET Utilities Portal Welcome Window



- 3 Click the operation you want to perform in the panel on the left of the portal, such as Change PIN.
- 4 Insert your IDPrime .NET card when prompted and follow the instructions.

Abbreviations

| | |
|-----------------|---|
| API | Application Programming Interface |
| CA | Certificate Authority |
| CAPI | Cryptographic Application Programming Interface |
| CCID | A driver that is needed to communicate with a IDPrime .NET smart card. |
| CSP | Cryptographic Service Provider |
| CSN | Card Serial Number |
| DAS | Device Administration Service |
| MSB | Most Significant Byte(s) |
| OS | Operating System |
| 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 |
| vSEC:CMS | Versatile Security Card Management System |

Glossary

| | |
|--|--|
| 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. |
| Base (CSP) | Microsoft's default software library that implements the Cryptographic Application Programming Interface (CAPI). |
| 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. |
| Device Administration Service (DAS) | A Gemalto web-hosted service used to manage smart card devices. |
| 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. |
| Versatile Security Card Management System (vSEC:CMS) | A card management system developed by Gemalto's partner Versatile security and embedded in some of Gemalto's smart card devices. |