JCCM*: Flexible Certificates for smartcards with Java Card

Mª Celeste Campo, Andrés Marín, Arturo García, Ignacio Díaz, Peter T. Breuer, Carlos Delgado, Carlos García

Universidad Carlos III de Madrid
Avd. Universidad 30 28911, Leganés
`eticket@it.uc3m.es`

Abstract. Smartcards and PKCS #11 are an appealing solution for combined storage and certificate management at the enduser level. Many applications use PKCS #11 primitives for security reasons: a popular browser, like Netscape Navigator contain a PKCS #11 cryptographic module that plays a critical role in secure web surfing and e-mail signing and encryption. Nevertheless, most market-ready solutions ([SMARTSIGN], [GPKPKCS#11], [SLBCBPKCS#11]) use non-programmable cards or else do not exploit the card's programmable capabilities. Instead they utilize cryptographic functions built into the card. This results in applications having the card manufacturer's semantics instead of PKCS #11 semantics.
In this article we present our work: Java Card Certificate Management (JCCM). **JCCM moves PKCS #11 middleware into the card** itself. This results in **greater flexibility and less implementation dependence** for applications. We have developed JCCM for two cards: the **GemXpresso RAD 211is** and the **Cyberflex for Linux Starter's Kit 2.1**. We have also developed the corresponding dynamic library for Netscape enabling our endusers to use JCCM in their daily.

1 Introduction

The number of users with Internet access keeps growing, and so do their security needs. The increasing number of sites offering sensitive information (like bank accounts) via HTTPS [HTTPS] protocol is an example. International e-mail, the "intelligence is in the net"-approach in network computing and e-business demand stronger security. Browsers do implement secure web surfing and digital signed and encrypted e-mail using digital certificates ([X.509]), but they lack corresponding secure certificate storage mechanisms.

Smartcards are tamper-proof devices, where tamper-resistance is a term with practical connotation that takes into account the cost/benefit relation of the attacks. Given unlimited funds we could break the security of a card. In [USENIX 99] some sophisticated chemical and physical attacks are described,

* This work has been partially supported by the project E-TICKET CYCYT N°2FD1997-1269-C02-01(TEL)

together with effective and low-cost countermeasures. The article concludes that they see no really effective short-term protection against carefully planned invasive tampering (involving focused ion-beam tools). “Zeroization” mechanisms for erasing secrets when tampering is detected require a continuous power supply that the credit-card form factor does not allow. The attacker can thus safely disable the zeroization mechanism before powering up the processor.

2 The PKCS #11 Standard

RSA Laboratories has developed, in cooperation with representatives of industry, academia and government, a family of standards called Public-Key Cryptography Standards, or PKCS for short. These standards cover RSA encryption, Diffie-Hellman key exchange, password-based encryption, an extended-certificate syntax, cryptographic message syntax, private key syntax, and certification request syntax, as well as selected attributes.

Table 1. A significative set of Cryptoki functions

General purpose funtions	**Slot and token management functions**	**Cryptographic functions**
		C_EncryptInit
C_Initialize	C_GetSlotList	C_Encrypt
C_Finalize	C_GetSlotInfo	C_DecryptInit
C_GetInfo	C_GetTokenInfo	C_Decrypt
Objects management functions	C_GetMechanismList	C_SignInit
	C_SetPIN	C_Sign
C_CreateObject	**Session management functions**	C_VerifyInit
C_CopyObject		C_Verify
C_DestroyObject	C_OpenSession	C_DigestInit
C_GetAttributeValue	C_CloseSession	C_Digest
C_SetAttributeValue	C_CloseAllSessions	C_GenerateKey
C_FindObjectsInit	C_GetSessionInfo	C_GenerateKeyPair
C_FindObjects	C_Login	C_WrapKey
C_FindObjectsFinal	C_Logout	C_UnwrapKey

The PKCS #11 specifies an application programming interface (API) for cryptographic services, called Cryptoki, short for “cryptographic token interface”. Cryptoki isolates an application from details of the cryptographic device, which is called “token’. Portable cryptographic tokens, such as smartcards, are inserted in “slots”, which correspond to a physical reader or other device interface. A token stores objects and can perform cryptographic functions on it. Cryptoki defines three classes of object: data, certificates, and keys. A **data object** is defined by an application. A **certificate object** stores a certificate. A **key object** stores a cryptographic key and it further specialices in concrete types for the various cryptographic algorithms, such as RSA public and private key. Cryptographic operations are performed in the context of sessions, which represent and established communication path with a token present in a slot.

The Cryptoki API consists of a number of functions, encompassing slot and token management and object management, as well as cryptographic functions. Table 1 shows a significative set of these functions.

3 Netscape and PKCS #11

Netscape incorporates a security architecture that allows for web surfing and signed and encrypted e-mail. This security architecture, well-known as the "Netscape Security Library" (NSL), makes use of PKCS #11 cryptographic modules in order to offer the appropriate high level functionality. Netscape contains an internal module PKCS #11 that constitutes a fairly complete implementation of the standard: it includes mechanisms based on RSA as well as some symmetrical key mechanisms. The internal module offers a logical vision of the computer as a cryptographic device: it uses the file system for persistent storage of Cryptoki objects and the CPU for cryptographic processing. Thanks to this module, Netscape is able to offer the capabilities mentioned above, but it has a problem: the use of the file system to store certificates and keys breaks the premise of safe storage of sensitive data. The solution adopted by Netscape is password protecting data stored on disk and to allow the incorporation of external PKCS #11 modules, which then serve as specialized interfaces to cryptographic hardware, as in the case of our JCCM module. Another peculiarity of the NSL is that it allows the simultaneous presence of several PKCS #11 modules: our module can coexist with the internal module, and therefore it is not necessary to implement the mechanisms already supported; mechanisms oriented towards data encryption are not implemented (symmetric key) because the data transfer rate between the computer and the card makes it unsuitable for the encrypting arbitrary volumes of data.

A trace of all the calls to our library needed to sign a mail from Netscape is shown below. The trace has been simplified and includes only the exit trace line for each Cryptoki functions called.

– *Lines 1 to 10*: Library boot and initial data exchange. Lines 1 to 5, they are called when starting Netscape before we initiate any operation. Line 6, the token was present in the reader. Line 7, that obtains data about the token and its capabilities (lines 8 to 9). Finally, line 10, a session begins with the token.

```
 1 Mar  2 12:06:20: C_GetFunctionList Returns (CKR_OK) 0x0
 2 Mar  2 12:06:20: C_Initialize Returns (CKR_OK) 0x0
 3 Mar  2 12:06:20: C_GetInfo Returns (CKR_OK) 0x0
 4 Mar  2 12:06:20: C_GetSlotList Returns (CKR_OK) 0x0
 5 Mar  2 12:06:20: C_GetSlotList Returns (CKR_OK) 0x0
 6 Mar  2 12:06:21: C_GetSlotInfo Returns (CKR_OK) 0x0
 7 Mar  2 12:06:21: C_GetTokenInfo Returns (CKR_OK) 0x0
 8 Mar  2 12:06:21: C_GetMechanismList Returns (CKR_OK) 0x0
 9 Mar  2 12:06:21: C_GetMechanismList Returns (CKR_OK) 0x0
10 Mar  2 12:06:21: C_OpenSession Returns (CKR_OK) 0x0
```

– *Lines 11 to 14* : 32 seconds are spent from line 10 to 11; they correspond to the time spent in writing the e-mail. These lines correspond to the request for the PIN and the corresponding call to `C_Login()`. The elapsed time between lines 13 and 14 correspond to the manual introduction of the PIN.

```
11 Mar  2 12:06:53: C_GetSlotInfo Returns (CKR_OK) 0x0
12 Mar  2 12:06:53: C_GetSessionInfo Returns (CKR_OK) 0x0
13 Mar  2 12:06:53: C_GetSessionInfo Returns (CKR_OK) 0x0
14 Mar  2 12:06:57: C_Login Returns (CKR_OK) 0x0
```

– *Lines 15 to 66*: The rest of the trace corresponds to the calls made by Netscape in order to generate a digital signature for a e-mail. The two last lines correspond to the signature. The time used for this operation (lines 65 to 66) is only 2s, whereas the total time for the signature (lines 15 to 66) is 26s. Those extra 24s are used in certificate searches and data transfer from the token.

```
15 Mar  2 12:06:59: C_FindObjectsInit Returns (CKR_OK) 0x0
16 Mar  2 12:06:59: C_FindObjects Returns (CKR_OK) 0x0
17 Mar  2 12:06:59: C_FindObjectsFinal Returns (CKR_OK) 0x0
18 Mar  2 12:07:00: C_FindObjectsInit Returns (CKR_OK) 0x0
19 Mar  2 12:07:00: C_FindObjects Returns (CKR_OK) 0x0
20 Mar  2 12:07:00: C_FindObjectsFinal Returns (CKR_OK) 0x0
21 Mar  2 12:07:07: C_GetAttributeValue Returns (CKR_OK) 0x0
22 Mar  2 12:07:07: C_GetAttributeValue Returns (CKR_OK) 0x0
23 Mar  2 12:07:08: C_GetSessionInfo Returns (CKR_OK) 0x0
24 Mar  2 12:07:08: C_GetAttributeValue Returns (CKR_OK) 0x0
25 Mar  2 12:07:08: C_GetAttributeValue Returns (CKR_OK) 0x0
26 Mar  2 12:07:12: C_FindObjectsInit Returns (CKR_OK) 0x0
26 Mar  2 12:07:12: C_FindObjects Returns (CKR_OK) 0x0
27 Mar  2 12:07:12: C_FindObjectsFinal Returns (CKR_OK) 0x0
28 Mar  2 12:07:12: C_GetMechanismList Returns (CKR_OK) 0x0
29 Mar  2 12:07:12: C_GetMechanismList Returns (CKR_OK) 0x0
30 Mar  2 12:07:16: C_FindObjectsInit Returns (CKR_OK) 0x0
31 Mar  2 12:07:16: C_FindObjects Returns (CKR_OK) 0x0
32 Mar  2 12:07:16: C_FindObjectsFinal Returns (CKR_OK) 0x0
33 Mar  2 12:07:17: C_GetAttributeValue Returns (CKR_OK) 0x0
34 Mar  2 12:07:17: C_GetAttributeValue Returns (CKR_OK) 0x0
35 Mar  2 12:07:17: C_GetSessionInfo Returns (CKR_OK) 0x0
36 Mar  2 12:07:17: C_GetAttributeValue Returns (CKR_OK) 0x0
37 Mar  2 12:07:17: C_GetAttributeValue Returns (CKR_OK) 0x0
38 Mar  2 12:07:18: C_FindObjectsInit Returns (CKR_OK) 0x0
39 Mar  2 12:07:18: C_FindObjects Returns (CKR_OK) 0x0
40 Mar  2 12:07:18: C_FindObjectsFinal Returns (CKR_OK) 0x0
41 Mar  2 12:07:18: C_GetSlotInfo Returns (CKR_OK) 0x0
42 Mar  2 12:07:18: C_GetSessionInfo Returns (CKR_OK) 0x0
43 Mar  2 12:07:18: C_GetSessionInfo Returns (CKR_OK) 0x0
44 Mar  2 12:07:18: C_FindObjectsInit Returns (CKR_OK) 0x0
45 Mar  2 12:07:18: C_FindObjects Returns (CKR_OK) 0x0
46 Mar  2 12:07:18: C_FindObjectsFinal Returns (CKR_OK) 0x0
47 Mar  2 12:07:18: C_GetAttributeValue Returns (CKR_OK) 0x0
48 Mar  2 12:07:19: C_GetAttributeValue Returns (CKR_OK) 0x0
49 Mar  2 12:07:19: C_GetSessionInfo Returns (CKR_OK) 0x0
50 Mar  2 12:07:19: C_GetAttributeValue Returns (CKR_OK) 0x0
51 Mar  2 12:07:19: C_GetAttributeValue Returns (CKR_OK) 0x0
52 Mar  2 12:07:19: C_FindObjectsInit Returns (CKR_OK) 0x0
53 Mar  2 12:07:19: C_FindObjects Returns (CKR_OK) 0x0
```

```
Mar  2 12:07:19: C_FindObjectsFinal Returns (CKR_OK) 0x0
Mar  2 12:07:19: C_GetSessionInfo Returns (CKR_OK) 0x0
Mar  2 12:07:20: C_GetAttributeValue Returns (CKR_OK) 0x0
Mar  2 12:07:20: C_GetAttributeValue Returns (CKR_OK) 0x0
Mar  2 12:07:20: C_FindObjectsInit Returns (CKR_OK) 0x0
Mar  2 12:07:20: C_FindObjects Returns (CKR_OK) 0x0
Mar  2 12:07:20: C_FindObjectsFinal Returns (CKR_OK) 0x0
Mar  2 12:07:22: C_GetAttributeValue Returns (CKR_OK) 0x0
Mar  2 12:07:23: C_GetAttributeValue Returns (CKR_OK) 0x0
Mar  2 12:07:23: C_GetAttributeValue Returns (CKR_OK) 0x0
Mar  2 12:07:23: C_GetSessionInfo Returns (CKR_OK) 0x0
Mar  2 12:07:23: C_SignInit Returns (CKR_OK) 0x0
Mar  2 12:07:25: C_Sign Returns (CKR_OK) 0x0
```

It is necessary to point out that this penalty is only incurread the first time that the certificate and the associated private key is used; the generation of a digital signature for the next and subsequent messages takes only 6s.

4 Smartcards and Java Card

Smartcards, besides being practical tamper-proof devices have ever-increasing computation and storage capabilities. They can be integrated in a natural way with the users' applications, for instance a favourite browser, through the use of [PKCS#11].

Smartcards are present in a number of solutions on the market. Most of these solutions use the standard [PKCS#11] for certificate management, i.e., they provide the users with dynamic libraries that can be accessed by applications in order to handle security. The bad news is that these solutions ([SMARTSIGN], [GPKPKCS#11], [SLBCBPKCS#11]) tend to offer the applications a subset of PKCS #11 semantics, reduced to that provided by the smartcard manufacturer.

Java Card is a reduced version of Java. In particular the virtual machine is very restricted. There is no garbage collector, and every object is instantiated in persistent memory until the end of the life-cycle of the "cardlet" (Java application running in a card). With respect to language, Java Card restricts the available packages and datatypes. The programmer has to deal with a simplified `Object` class, no `String` class, and only 16 bit integers.

We have implemented our system in Java Card because this technology has several unique benefits:

- Platform independence: this allows us to run our cardlet Cryptoki on different vendors' cards.
- Uses the Java language: which enables high programmer productivity and all the advantages of object-oriented programming.
- Multi-application capable: multiple applications can run on a single card.
- Compatible with Existing Smart Card Standards, such as ISO7816.

5 JCCM: Java Card Certificate Management

We have designed and implemented a system named Java Card Certificate Management (JCCM). JCCM moves part of PKCS #11 code inside a Java Card. The JCCM *cardlet* is responsible for object management in conformance to Cryptoki, implementing the corresponding Cryptoki functions. This is one of the key issues in the JCCM design: the cardlet handles management layer in Cryptoki objects, implementing the full management functionality defined in Cryptoki, that is: creation, lookup, copy and deletion of objects, and cryptographic functions. The set of cardlet Cryptoki APDU's (Application Data Unit) is in Table 2.

Table 2. Set of cardlet Cryptoki APDU's

General purpose	**Session management**
Ident	DownLoadObj_Init
Object management	DownLoadObj_Attr
Login	DownLoadObj_Create
Logout	DownLoadObj_Copy
Cryptographic	DownLoadObj_SetAttr
Sign	DownLoadObj_Find
	GetAttr
	DeleteObj

Another key issue in our design is the dynamic memory management in smartcards. EEPROM memory is the place where persistent objects live. It is a very limited resource and must be handled carefully. Cryptoki object management functions need to allocate and free memory, and we have implemented a simple memory management layer for this reason. This layer defines a spool of memory blocks that are to be marked used or free. The first two bytes of a memory block contain a free/used bit and a 15-bit block length (2^{15} Bytes=32KB), sufficient to include full available EEPROM in the current Java Cards. These implementation details are hidden from the calling functions. Memory management also merges small blocks into larger ones to avoid fragmentation.

PKCS #11 represents objects as arrays of attributes. Attributes are fixed-size structures formed by three fields: attribute type, value pointer and the length of the value. A JCCM cardlet uses a similar scheme. An object structure is formed by three fields:

- The "next object" pointer, 2 bytes. All the objects that are created in the card are maintained in a linked list.
- Number of attributes of the object, 1 byte. This field can be deduced based on the size of the block in dynamic memory, but it has been chosen to include it: typically the card will store solely two objects, a certificate and a associated private key.

– A variable number of structures of attributes, so many as number of attributes has the object.

An attribute object has the following fields:

– Type of attribute, 4 bytes. It is the binary value defined in the standard.
– An attribute pointer, 2 bytes. The value is stored in its own block of dynamic memory. The length of this field is obtained from the head of the block of dynamic memory in which it is stored.

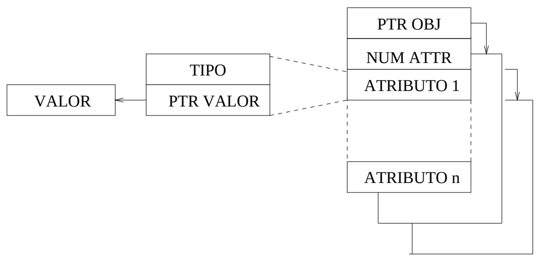

Fig. 1. Objects structure

A Java class encapsulates each of these structures; the class `ObjPatr` for the objects and the class `AttrPatr` for the attributes. These classes cannot be instantiated due to the lack of garbage collection mentioned in the Section 4, they have only static methods and members. They are used to map the array of dynamic memory to the corresponding object structure: they also have methods to set/get the value of each field and to release the associated dynamic memory. These two classes extend the class `Patr`, that supplies basic access to fields of type multibyte. Before using one of these classes to access a structure stored in dynamic memory it is necessary to establish the address of the structure by means of a call of `setAddr(short addr)`, which sets the reference address used by all the methods that access the members of the structure `(get...(),set...())`.

6 JCCM Implementation

We have developed the corresponding dynamic library for Netscape that enables our final users to use JCCM in their security operations. The implementation has been done in Linux, using PC/SC Muscle and Netscape. We are porting now to Netscape for Microsoft Windows.

To demonstrate device independence, we have ported JCCM to cards from two different manufacturers: Gemplus **GemXpresso RAD 211is** and Schlumberger **Cyberflex for Linux Starter's Kit 2.1**. The differences we found are twofold: the cryptographic capabilities are not standard in Java Card 2.0 (there were no Java Card 2.1 kits on the market when we begun to develop JCCM), so we are forced to use proprietary hooks, and the way to load software differs between the cards.

- **GemXpresso RAD 211is** [GemXpresso RAD 211 UG] uses Visa Open Platform for cardlet uploading. DES and 3DES are available, but not RSA. We need to transfer the private key to the computer to perform digital signing with RSA.
- **Cyberflex for Linux Starter's Kit 2.1**, [Cyberflex SDK] uses a proprietary application (based on TCL/TK) for software loading. RSA via Schlumberger's extension `javacardx.crypto`.

With respect to the cardlet itself, there are some differences; for example, the maximum size of the responses. The source code contains some compilation directives which adapt the code to the cards and it has to be precompiled to obtain java code that is then optimized for the card in question. A comparison of both cards is in Table 3. The size of the cardlet in the GemXpresso is much larger than in the Cyberflex, but in both cards there is room enough for storing up to 4 certificates (each certificate takes 1KB). GemXpresso is significantly faster storing and retrieving certificates (almost twice as fast as Cyberflex), perhaps because of better efficiency of the virtual machine implementation.

Table 3. Cardlet comparison

Card	**Size of Cardlet**	**Storage (ms)**			**Retrieval (ms)**
	(Bytes)	**Private key**	**Public key**	**Certificate**	**Certificate**
GemXpresso	6437	20312	12998	16228	8934
Cyberflex	3992	38122	28180	34036	16155

References

[ISO/IEC 7816-4] "ISO/IEC 7816-4: Integrated circuit(s) cards with contacts. Part 4: Interindustry commands for interchange", ISO/IEC, 1995.

[ISO/IEC 7816-3] "ISO/IEC 7816-3: Integrated circuit(s) cards with contacts. Part 3: Electronic signals and transmission protocols", ISO/IEC, 1997.

[JCADG 2.1] "Java Card Applet Developer's Guide. Java Card Version 2.0", SUN Microsystems, Agosto de 1998.

[JCADG 2.0] 'Java Card Applet Developer's Guide. Java Card Version 2.1", SUN Microsystems, Agosto de 1999.

[GemXpresso RAD 211 UG] "GemXpresso RAD 211 User Guide Version 1.0", Gemplus, Octubre 1999

[GemXpresso RAD 211 CRM] "GemXpresso RAD 211 Card Reference Manual Version 1.0", Gemplus, Octubre 1999

[Cyberflex PG] "Cyberflex Access Developer's Series. Programmer's Guide", Schlumberger, Septiembre 1999.

[Cyberflex SDK] "Cyberflex Access Software Developer's Kit 2 - Release Notes", Schlumberger, Noviembre 1999.

[HTTPS] "HTTP Over TLS", Rescorla, E., IETF RFC 2818, Mayo 2000.

[X.509] "Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP". R. Housley, P. Hoffman. IETF RFC 2585, Mayo 1999.

[USENIX 99] "Design Principles for Tamper-Resistant Smartcard Processors" by Oliver Kömmerling, Markus Kuhn, Workshop on Smartcard Technology Proceedings, Chicago, Illinois, USA, Mayo 10-11, 1999

[SC SDK] 'Smart Card Developer's Kit", Scott B. Guthery, Timothy M. Jurgensen. Macmillan Technical Publishg. 1998.ISBN 1-57870-027-2.

[SC APP. DEV. JAVA] 'Smart Card. Application Developement Using Java", Uwe Hansmann, Martin S. Nicklous, Thomas Schack y Frank Seliger, Springer, 2000. ISBN 3-540-65829-7.

[PKCS#11] "PKCS #11 v2.10: Cryptographic Token Interface Standard", RSA Laboratories Inc., Diciembre 1999 (003-903052-210-000-000).

[PKCS#1] "PKCS #1 v2.1: RSA Cryptography Standard", RSA Laboratories Inc.

[PKCS#5] "PKCS #5 v2.0: Password-Based Cryptography Standard", RSA Laboratories Inc.

[PKCS#8] "PKCS #8 v1.2: Private-Key Information Syntax Standard", RSA Laboratories Inc.

[STALL99] 'Cryptography and Network Security: Principles and Practices", Stallings, W., 2ed, Prentice-Hall Inc., 1999

[SMARTSIGN] "Smart Sign", Tommaso Cucinotta,
http://sourceforge.net/projects/smartsign

[GPKPKCS#11] "GemSAFE Products", Gemplus,
http://www.gemplus.com/products/software/gemsafe/index.html

[SLBCBPKCS#11] "Cyberflex Access SDK", Schlumberger,
http://www.cyberflex.com/Products