Autenticación en la Red: ACerO y JCCM:* Java Card Certificate Management

Ignacio Díaz Asenjo, Arturo García Ares, María Celeste Campo Vázquez, Andrés Marín López, Carlos Delgado Kloos, Carlos García Rubio, Peter T. Breuer

Universidad Carlos III de Madrid.
Avd. Universidad 30 28911 Leganés

e-mail:eticket@it.uc3m.es

Abstract

Public-key cryptosystem are the best option to reinforce systems security, because they solve the interchange of keys problem. Certification Autorities and digital certificate management use these cryptosystems and offer a trust model to build applications on top of. This is the last step to provide secure systems, integrating these concepts in the procedures of the organizations. This integration brings with itself technical and legal problems, which are broached jointly in public keys infrastructures.

In this article we move at the level of Certification Authorities and certificate management. We want to show the two more important results of our work in this area: ACerO, in the server side, and JCCM, in the client side. Both developments will be soon available in the Web. Our motivation is to offer a Web based Certification Authority related to the use of smartcards javacards for certificate storage. All has been developed for Linux and Netscape. In the server side we emphasized the use of servlets and libraries of native code; in the client side, JCCM offers a flexible certificate management and independence with respect to the smartcard manufacturer. JCCM is available for their use with Netscape for secure browsing, ciphering and signing of e-mails.

1. Introducción

Desde que Alicia hizo pública su clave, enviarle información privada o comprobar su autoría se facilitaron enormemente. El problema de la distribución de claves se facilitó enormemente gracias a los algoritmos de cifrado asimétricos. Las iniciativas de desarrollo de software criptográfico basado en este nuevo paradigma, como openssl (`http://openssl.org`), ponen a disposición de los usuarios finales estas nuevas capacidades.

El siguiente paso en la cadena es la construcción de autoridades de certificación y las facilidades de gestión de certificados digitales. También hay software para este fin, de hecho el propio openssl lo hace. El problema es que para que sea realmente utilizable es necesario portarlo a su uso en el web, y para esto hemos desarrollado ACerO. Similar a otras iniciativas como la de la Universidad de Murcia (con el proyecto UMGina), o la OpenCA. En la sección 3 exponemos las ventajas de nuestro enfoque con la OpenCA.

La segunda parte de este artículo se sitúa en la parte de cliente. El uso de aplicaciones criptográficas empieza a popularizarse a partir de la inclusión de capacidades criptográficas en los navegadores. Hoy en día la mayoría de los usuarios del web han accedido alguna vez a páginas *seguras* (utilizando el protocolo [HTTPS]), y poco a poco se empiezan a enviar correos electrónicos firmados o cifrados. La piedra angular de estas nuevas capacidades radica en la utilización de certificados digitales ([X.509]) que establecen un mecanismo para comprobar la identidad de un usuario o de un sitio, para dar fe de que un usuario ha firmado un mensaje y garantizar la integridad del mismo, o bien para proteger una clave de sesión mediante la cual cifrar un mensaje. Estos certificados encapsulan distintas claves, entre ellas la más importante es la clave pública. El punto más débil del sistema está en el almacenamiento de claves (en concreto de la clave privada). Las tarjetas inteligentes son una alternativa muy segura para el almacenamiento y gestión de dichas claves. Las tarjetas actuales son dispositivos inviolables y además tienen una considerable capacidad de cómputo y almacenamiento.

Las aplicaciones que utilizamos, por ejemplo nuestro navegador favorito, incluye las capacidades necesarias para gestionar estos certificados digitales. Por ejemplo, el navegador de Netscape, tiene su propia biblioteca, la *Netscape Security Li-*

*Este trabajo ha sido desarrollado dentro del proyecto E-TICKET CYCYT Nº2FD1997-1269-C02-01(TEL)

brary (NSL) para estos fines.

El estándar [PKCS#11] define un interfaz que proporciona a las aplicaciones que desean utilizar servicios criptográficos una visión lógica común de los dispositivos capaces de ofrecer dichos servicios: dispositivos con capacidad de almacenamiento, proceso, y específicamente, capaces de proteger de manera efectiva los elementos sensibles (claves de cifrado) almacenados. Las aplicaciones que utilizamos suelen utilizar este API (por ejemplo, la NSL de Netscape sigue este estándar).

La alternativa más segura es utilizar módulos que implementen PKCS #11 para las aplicaciones que lo admitan, y utilizar tarjetas inteligentes para compartir la gestión y para almacenar de forma segura claves y certificados. Sin embargo, los desarrollos que se han realizado hasta la actualidad de módulos PKCS #11 ([SMARTSIGN], [GPKPKCS#11], [SLBCBPKCS#11]) con tarjetas inteligentes emplean tarjetas no programables o bien no utilizan la facilidad de programación de las tarjetas que sí son programables. Estas implementaciones se limitan a adaptar la funcionalidad criptográfica predefinida en las tarjetas a la visión lógica que debe ofrecer un módulo PKCS #11, recortando la semántica definida por el estándar para adaptarla a la que ofrece el fabricante en cada tarjeta.

Nuestro sistema, Java Card Certificate Management (JCCM), emplea una tarjeta inteligente programable mediante tecnología Java Card como dispositivo criptográfico, trasladando parte de la implementación del estándar PKCS #11 a la tarjeta, resultando en una mayor flexibilidad, e independencia del dispositivo.

Firmar mensajes de correo electrónico mediante algoritmos de clave pública requiere un complejo sistema en el que participan (1) una autoridad de certificación y mecanismos de gestión y distribución de claves en la organización, que en nuestro caso será ACerO, (2) agentes de usuario habilitados para generar y verificar firmas digitales (Netscape), y (3) dispositivos capaces de ejecutar algoritmos criptográficos y de almacenar datos sensibles de manera segura que llamaremos *tokens criptográficos* o simplemente *tokens*. JCCM abarca las partes (2) y (3), dos piezas complementarias que se materializan en:

- Una biblioteca de enlace dinámico (DLL) que sirve de interfaz entre la tarjeta inteligente y la biblioteca de seguridad (NSL) de Netscape –el agente de usuario.
- Un *cardlet*[1] que implementa las funciones requeridas.

La estructura de este artículo es la siguiente: la sección 2 explica el estándar PKCS #11 y la arquitectura de seguridad de Netscape, punto integrador del sistema. La sección 3 describe la parte de servidor de nuestro sistema: ACerO, las tecnologías que utiliza, estándares soportados, arquitectura y funcionamiento. La sección 4 describe la parte de cliente: JCCM, Java Card la arquitectura y una comparativa del rendimiento de las dos tarjetas en que hemos implementado JCCM.

2. Netscape y PKCS #11

PKCS #11 es uno más del creciente grupo de estándares PKCS, acrónimo de Public-Key Cryptography Standards (estándares de criptografía de clave pública) y publicados por RSA Labs. Estos estándares abarcan aspectos del sistema que acabamos de esbozar tales como:

- Definición de los algoritmos criptográficos (tanto de clave pública como de clave simétrica) y algoritmos de apoyo.
- Procedimientos para aplicar los algoritmos criptográficos a flujos de bytes y para producir firmas digitales.
- Definición de tipos de datos (certificados, claves, ...) y sus correspondientes sintaxis de transferencia o formatos de representación binaria.
- Procedimiento para efectuar transacciones HTTP seguras
- Definición de formatos para transmisión de correo electrónico cifrado e inclusión de firma digital.
- Interfaz de programación para aplicaciones (API) que permite hacer uso de los algoritmos, procedimientos y tipos de datos.

La biblioteca descrita por el estándar PKCS #11 recibe el nombre de Cryptoki, abreviatura de "cryptographic token interface". Cryptoki aisla a la aplicación de los detalles del dispositivo criptográfico. En terminología de Cryptoki, estos dispositivos se llaman *tokens*. Estos tokens pueden ser de carácter portátil (p.e. una tarjeta inteligente) o fijos (p.e. una tarjeta PCI insertada en la placa madre del ordenador). Los tokens portátiles se insertan en su correspondiente *slot*, nombre asignado por Cryptoki para la abstracción del terminal lector. Un token almacena *objetos* y es capaz de ejecutar *primitivas criptográficas* empleando esos objetos, posiblemente tras un paso inicial de autenticación del usuario/aplicación al token mediante la presentación de un PIN. Toda comunicación con el token se efectúa dentro del contexto de una *sesión*, que representa un canal de comunicación establecido con un token presente en un slot.

Netscape incorpora una arquitectura de seguridad que permite tráfico HTTP seguro y gestionar correo cifrado y firmado. Esta arquitectura de seguridad, conocida como "Netscape Security Library" (NSL), hace uso de las primitivas presentes

[1] Programa residente en la tarjeta inteligente (o token) escrito acorde con la especificación Java Card

en PKCS #11 para ofrecer la funcionalidad de alto nivel a la que nos referimos. Netscape se distribuye con un módulo PKCS #11 interno que constituye una implementación bastante completa del estándar: incluye tanto los mecanismos basados en RSA como algunos mecanismos de clave simétrica. La versión y la distribución (con o sin restricciones de exportación) de Netscape imponen los mecanismos que podemos utilizar y las limitaciones de los mismos en cuanto a longitud máxima de claves. Este módulo PKCS #11 interno ofrece una visión lógica del propio ordenador como dispositivo criptográfico: utiliza el sistema de archivos como almacenamiento persistente para almacenar los objetos de Cryptoki y la capacidad de proceso de la CPU para efectuar las operaciones de cifrado. Gracias a este módulo interno Netscape es plenamente capaz de ofrecer las capacidades mencionadas, pero adolece de un problema: la utilización del sistema de archivos del ordenador para almacenar certificados y claves rompe la premisa inicial de almacenamiento seguro de datos sensibles. La solución adoptada por Netscape es permitir la incorporación de módulos PKCS #11 externos que sirvan de interfaz a hardware criptográfico especializado, como es el caso de nuestro módulo JCCM.

Tanto las claves como los certificados generados por ACerO serán recogidos en última instancia por la NSL de Netscape dando la posibilidad de esta forma de almacenarse según elija el usuario o bien en el sistema de archivos tradicional o bien en una tarjeta inteligente gracias a la presencia dee este nuevo módulo.

3. ACerO

A lo largo del artículo se hace referencia a una serie de objetos personales e intrasferibles como son claves privadas y certificados, que deseamos almacenar dentro de tarjetas inteligentes. De esta forma conseguimos un medio cómodo de transportar nuestra identidad digital y en cierta forma un entorno protegido frente a intrusos que quieran apoderarse de esta información para hacerse pasar por nosotros.

Lo que se persigue en todo momento es una forma de disponer de una especie de "DNI digital"que nos permita identificarnos ante un determinado sistema preparado para tales fines.

Resulta evidente pues, la necesidad de disponer sistema capaz de generar y administrar esos certificados digitales, y que sea además un entidad en la que podamos confiar, tal y como confiamos en el Ministerio de Interior como emisor de nuestros DNIs tradicionales, para tener la certeza de que ese certificado corresponde realmente a la persona adecuada y no a un intruso.

OpenCA[2] de Massimiliano Pala se puede considerar como uno de los primeros y más conocidos esfuerzos de llevar a cabo la implementación de una infraestructura de clave pública (PKI) abierta y de libre distribución. En gran medida ha sido fuente de inspiración de ACerO, es por ello que la filosofía principal en la que se basa este producto es muy parecida: Interfaz Web y OpenSSL para realizar las funciones criptográficas

OpenCA está realizado en su totalidad por medio de scripts y CGIs en Perl, y utiliza OpenSSL realizando llamadas directamente a un ejecutable *openssl* resultante de la instalación de dicho paquete. Es una implementación pensada para un tráfico de peticiones muy bajo debido a que este tipo de diseño carga demasiado el servidor Web donde reside. Por cada una de las operaciones en el servidor se abrirán dos procesos, uno propio del CGI y otro como resultado de llamar a un ejecutable, *openssl*, de nuestro sistema desde un CGI.

ACerO mejora en gran medida esta enorme carga utilizando soluciones tecnológicamente más avanzadas. Los servlets de Java sustituyen a los CGIs en Perl, y las llamadas al ejecutable *openssl* por invocaciones a métodos nativos. Este cambio desde la base es uno de los principales motivos por los que se prefirió comenzar un nuevo diseño en lugar de contribuir a mejorar y ajustar a nuestras necesidades la OpenCA.

Otras ventajas que ofrece ACerO frente a OpenCA es su mayor escalabilidad. OpenCA da la sensación de estar pensado únicamente para contemplar una única RA ya que no realiza una repartición de los certificados dependiendo de las RAs de las que proviene la petición, sino que envía todos los certificados a todas. ACerO suple esta deficiencia enviando a cada una de las RAs los certificados que ellas solicitaron.

ACerO es una implementación de libre distribución de un Sistema de Gestión de Certificados cuyos pilares han sido concebidos en un principio para funcionar sobre plataforma Linux, pero utiliza tecnología que facilita su posterior portado a otras arquitecturas como puede ser Windows. Esto significa que en la actualidad la parte correspondiente a la Autoridad de Certificación debe instalarse en dicha plataforma.

Su diseño totalmente orientado hacia el navegador Netscape desemboca en una utilización de cierta funcionalidad proporcionada por su módulo interno y la total compatibilidad con el módulo PKCS#11 descrito con anterioridad.

Se trata de una serie de aplicaciones Web que se apoyan en servlets de Java para proporcionar la funcionalidad que el sistema ofrece a través del navegador. Para realizar los operaciones criptográficas que se llevan a cabo se utilizan llamadas a métodos nativos que se basan en el código de implementación de OpenSSL.

3.1. Estándares de PKI admitidos

La cuestión principal de tener un PKI es poder generar certificados y llevarlos donde sea necesa-

[2] `http://www.openca.org`

rio.

Existen estándares de seguridad abiertos para los componentes de un PKI que han sido desarrollados con la finalidad de promover la interoperabilidad entre sistemas. Entre las organizaciones dedicadas a esta tarea se encuentran el W3C (World-Wide Consortium), IEFT (Internet Engineering Task Force) e ITU (International Telecommunication Union).

ACerO se ajusta a los siguientes estándares:

- X.509 versión 3 encargado de definir el formato y el contenido de los certificados digitales
- CRL versión 1 encargado de establecer el formato y el contenido de las listas de revocación de certificados
- Familia PKCS encargado de definir el formato y el comportamiento del intercambio y distribución de claves públicas.

De manera que como resultado obtenemos un producto compatible con múltiples herramientas existentes en la actualidad.

3.2. Arquitectura Interna

Internamente el sistema se apoya en una serie de componentes bastante conocidos y que podemos obtener gratuitamente de la red: Apache 1.3.x, Tomcat 3.2, Netscape 4.x o sup, Java2, y OpenSSL 0.9.6.

ACerO utiliza *servlets* de Java, proporcionando una sustitución potente y eficiente de los CGI (*Common Gateway Interface*). Sus ventajas son:

- La **portabilidad** debido a que los servlets de Java son independientes de plataformas y protocolos.
- El **rendimiento** ya que tienen un ciclo de vida más eficiente que los programas CGI o FastCGI. A diferencia de los programas CGI, los servlets no crean un nuevo proceso en cada petición entrante. En su lugar, utilizan diferentes hilos de ejecución (*threads*) en el servidor.
- La **seguridad** porque al ser una derivación del lenguaje Java, heredan todos sus mecanismos de seguridad además de algunas características propias. La *sand-box* del servlet proporciona un entorno controlado en el que el servlet puede funcionar y utiliza un controlador de seguridad para vigilar la actividad del servlet y prevenir operaciones no autorizadas.

Por medio Netscape además de la funcionalidad intrínseca de navegación e interfaz de las aplicaciones se proporcionará la generación de claves y firma de mensajes que pasamos a la CA.

Se han desarrollado una serie de librerías en C, actualmente dependientes de arquitecturas tipo *Linux ix86 (.so),* en las que se implementan con la ayuda de OpenSSL operaciones de: generación de un par de claves RSA para la CA, autofirmado de la petición de certificado de la CA, comprobación e interactuación con el fichero de configuración de la CA, verificación de un firma PKCS#7, firma de una solicitud de certificado, revocación de un certificado, generación de la lista de revocación, exportación de certificados en formato PKCS#12, o mostrar información de un certificado X509.

3.3. Arquitectura Lógica

La arquitectura del sistema de certificación se divide en tres partes, y conceptualmente está diseñada para operar en varias máquinas distintas.

- Usuario (*Operaciones de Usuario)* : En esta parte del sistema se sitúa la Persona o Servicio que solicita y recoge un certificado.
- RA (*Autoridad de Registro*) : La idea de la RA responde a una estrategia para aislar al usuario del núcleo del sistema. La CA delega en la RA las labores de comunicación con el usuario, recopilación de sus datos e identificación del mismo. En cada RA existirá la figura del Operador de la RA. Su misión será la de notario digital, comprobando que los datos que aparecen en la petición son correctos y corresponden realmente con el ente que realizó dicha petición.
- CA (*Autoridad de Certificación*) : La CA es el corazón del sistema, y como tal debe aislarse lo máximo posible ya que si su integridad fuese violada desmoronaría todo el entramado de certificados. En esta parte se situará la figura del Operador de la CA o persona que supervisa la información que la RA validó y firma digitalmente los certificados confirmando de esta forma la relación existente entre la clave pública contenida y la identidad del propietario.

3.4. Funcionamiento

De una manera muy general podremos resumir un escenario típico de funcionamiento de la siguiente manera:

1. Un usuario desea obtener un certificado de nuestra Autoridad de Certificación. Primero deberá enviar sus datos y su clave pública a una Autoridad de Registro dependiente de la Autoridad de Certificación. El par de claves en este caso es generado por el navegador, siendo ésta la forma más segura de realizar esta operación, ya que la clave privada se guarda en todo momento.

2. Los datos del solicitante y su clave pública se añaden a la lista de espera de certificados pendientes de la RA, a la espera de ser aprobados por un *operador*.

3. En la RA únicamente puede operar un usuario de confianza de la CA. Él se encargará de comprobar que los datos recibidos en la petición corresponden a quien dicen ser, y firmará esa información para enviársela a la CA. La firma se realiza por medio de un certificado del operador disponible en el navegador o en su tarjeta inteligente, utilizando el módulo criptográfico del Netscape para de esta forma obtener una salida firmada en formato PKCS#7

4. Cuando la RA desee mandar los datos a la CA, se realizará un envío de información por medio de un paso de exportación de la RA y otro de importación por parte de la CA.

5. Al recibir la CA una serie de solicitudes de certificados aprobados por una o varios RAs, deberá comprobar por cada una de ellas la validez de la firma plasmada por el operador de la RA correspondiente. En el caso de que la verificación sea correcta, la CA firmará la solicitud del certificado devolviendo un certificado con un formato basado en el estándar X.509.

6. La CA enviará a la RA, por medio de un proceso de importación y exportación de datos, el/los certificado/s aprobado/s (*emitidos*), y se notificará al usuario correspondiente que su certificado ya está listo para ser recogido.

7. Una vez que el usuario ha recibido la notificación podrá obtener su certificado personal, puede por medio del número de serie indicado en la notificación, recoger su certificado aprobado por la CA. El certificado se incluirá en el dispositivo donde se creó la correspondiente clave privada, o bien en la base de datos del Netscape (por defecto) o bien en la tarjeta inteligente en el caso de utilizar JCCM.

4. JCCM: Tarjetas inteligentes y Java Card

Una tarjeta inteligente contiene un pequeño ordenador que consta de un microprocesador, memoria volátil (RAM) y memoria persistente (EEPROM). Las capacidades típicas de estos elementos son: microprocesador de 8 bits a 4MHz, RAM de 512 bytes, 32Kb de EEPROM y 16Kb de ROM. Los contactos presentes en la superficie de la tarjeta aportan la tensión de alimentación, la señal de reloj y un canal de comunicación serie (a 9600bps[3]). Estos contactos son gobernados por el terminal lector en el que esté insertada la tarjeta, por lo tanto el microprocesador sólo estará activo cuando la tarjeta esté insertada en un terminal. Extraer la tarjeta del terminal tiene el mismo efecto que apagar un ordenador: se interrumpe la actividad de la CPU y se pierde el contenido de la RAM, pero el almacenamiento persistente permanece; en un ordenador este almacenamiento persistente está constituido por dispositivos magnéticos (discos duros), mientras que en una tarjeta inteligente es la memoria EEPROM. Otra característica importante de este tipo de memoria es su relativamente corta esperanza de vida, unos 100000 ciclos de escritura, y su elevado tiempo de acceso para ciclos de escritura, del orden de 10ms.

La tarjeta inteligente es un dispositivo pasivo: recibe un comando proveniente del ordenador a través del terminal en el que esté insertada, lo procesa, genera una respuesta que es devuelta al ordenador y espera el siguiente comando. Estos comandos se denominan APDUs (Application Protocol Data Unit) y su estructura está definida en [ISO/IEC 7816-4]. Existen dos tipos de APDU, las que codifican un comando enviado por el ordenador y las que codifican la respuesta generada por la tarjeta.

Existen tarjetas no programables, que aceptan un juego de comandos predefinido, y tarjetas programables capaces de incorporar dinámicamente a su memoria persistente programas provenientes del exterior y enriquecer así el conjunto de comandos que son capaces de ejecutar. Una de las tecnologías de tarjetas programables es Java Card. Una Java Card es una tarjeta inteligente capaz de incorporar y ejecutar programas escritos en un subconjunto del lenguaje de programación Java. La principal ventaja que aporta esta tecnología es la independencia de la plataforma, que nos permite desarrollar aplicaciones Java Card que se pueden ejecutar sobre cualquier tarjeta acorde a la especificación, independientemente del fabricante de la misma.

Debido a las restricciones de memoria y procesamiento de las tarjetas inteligentes el lenguaje Java Card es una versión reducida del lenguaje Java [JCADG 2.1]: sólo existen los tipos primitivos `boolean`, `byte`, `short`, carece de la clase `Thread` y `String` y no tiene recolector de basura, por lo tanto todo objeto que es instanciado en memoria persistente permanece en ella hasta que se elimine el cardlet de la tarjeta, siendo ésta una de las limitaciones que más a condicionado nuestro desarrollo.

4.1. Arquitectura de JCCM

Los desarrollos que se han realizado hasta la actualidad de módulos PKCS #11 con tarjetas inteligentes emplean tarjetas no programables o bien no utilizan la facilidad de programación de

[3] El estándar ISO7816-3 permite una velocidad de hasta 115Kbps

las tarjetas que sí son programables. Estas implementaciones se limitan a adaptar la funcionalidad criptográfica predefinida en las tarjetas a la visión lógica que debe ofrecer un módulo PKCS #11. La semántica definida por el estándar se recorta para adaptarla a la que ofrece la tarjeta para la que se realiza la implementación, que además queda ligada a una tarjeta de un determinado fabricante.

La característica distintiva de nuestro desarrollo es la implementación por parte del cardlet presente en la tarjeta de parte de la funcionalidad definida en el estándar PKCS #11. Los objetos se almacenan en la tarjeta con los mismos atributos definidos en el estándar, y es el propio cardlet quien procesa las operaciones de gestión de objetos de Cryptoki, soportando así el almacenamiento de cualquier tipo de objeto e implementando la semántica completa de búsqueda, copia, creación y borrado.

En los siguientes apartados se explican las diferentes partes de que consta la aplicación Java Card desarrollada en este proyecto.

4.1.1. Gestión de memoria dinámica en tarjetas Java Card

La memoria en una tarjeta inteligente es un recurso limitado y es una de las restricciones más importantes a tener en cuenta cuando se realizan aplicaciones para este tipo de sistemas. Para almacenar datos de forma persistente en una tarjeta inteligente, es necesario que residan en memoria EEPROM. El estándar ISO 7816-4 definió para ello una estructura de sistema de ficheros similar a la que tenemos en un ordenador y estandarizó APDUs para poder gestionarla. Las primeras versiones de Java Card, hasta la 2.0, imitaban este modelo de gestión de la memoria persistente: existían clases que replicaban la funcionalidad de las APDUs de acceso a ficheros definidas por ISO y que trataban de imitar el modelo de *streams* de Java. A partir de la versión 2.1 se abandonó este modelo de acceso a memoria persistente para sustituirlo por el método nativo en Java de instanciación de objetos: para disponer de un bloque de bytes en el almacenamiento persistente se instancia un objeto de la clase deseada; el acceso estructurado a la EEPROM, a través de los miembros del objeto instanciado, es así más directo que el efectuado a través de streams. Cuando en Java Card se habla de almacenamiento persistente no se bromea; los objetos instanciados no serán destruidos ni tampoco será liberado el espacio que ocupen por ningún recolector de basura, pues el estándar Java Card no contempla esta facilidad. Por lo tanto, todo objeto instanciado permanecerá durante la vida del cardlet: hasta que éste sea eliminado de la tarjeta.

Las funciones de Cryptoki permiten la creación dinámica de objetos y nosotros deseábamos que nuestra implementación no limitase esa flexibilidad. Podíamos haber diseñado una implementación capaz de almacenar un número predeterminado de objetos Cryptoki, p.e. un certificado X.509 y una clave privada RSA, pero eso limitaba la funcionalidad de nuestra implementación y por lo tanto las aplicaciones potenciales que podrían beneficiarse de ella. Topamos con el problema de desarrollar un cardlet capaz de crear, destruir y modificar dinámicamente un número indeterminado de objetos en un esquema de asignación de memoria que nunca libera los bloques asignados. La solución a este problema consiste en desarrollar un módulo de asignación de memoria dinámica que ofrezca funcionalidad similar a las tradicionales `malloc()` y `free()` de la biblioteca estándar del lenguaje C. La asignación de bloques se hace sobre un array de bytes Java cuyo tamaño se especifica en tiempo de compilación y que es reservado en memoria persistente en el instante de instalación del cardlet en la tarjeta.

Se ha adoptado un sencillo esquema para gestionar los bloques libres y ocupados: todo bloque tiene una cabecera de dos bytes utilizado para la gestión de la memoria y el resto disponible para almacenar datos. La cabecera puede considerarse como una palabra de dos bytes, primero el byte de mayor peso, en la que el bit más a la izquierda es un indicador de si el bloque está libre (0) u ocupado (1) y los 15 bits restantes indican el tamaño del bloque en bytes excluyendo la cabecera. Por tanto, el tamaño máximo de bloque y de memoria dinámica que podemos manejar con este esquema es de 2^{15} bytes (32Kb), suficiente para abarcar toda la memoria persistente disponible en tarjetas inteligentes de tecnología actual. Otra implicación es que las direcciones son de dos bytes. El tamaño del array empleado en la implementación actual es de 4Kb, suficiente para albergar dos certificados propios, es decir, para los que se almacena también una clave privada asociada, (cada pareja certificado,clave privada ocupa algo menos de 2Kb en nuestra estructura de almacenamiento persistente) y soportar además búsquedas, que requieren para su ejecución disponer de almacenamiento temporal.

En cualquier instante el array estará compuesto por una serie de bloques contiguos libres y ocupados mezclados, dependiendo de la secuencia de asignaciones y liberaciones previa. Inicialmente tenemos un bloque libre que abarca el array completo. Cada vez que se solicita un bloque se recorren los bloques en secuencia desde el primero (que siempre está en la dirección 0, índice `[0]` en el array) buscando un bloque libre de tamaño suficiente. Cuando ésto ocurre, si el bloque encontrado es de mayor tamaño que el bloque solicitado y suficientemente grande como para poder hacer un nuevo bloque de la parte sobrante, se divide en dos: un fragmento ocupado de tamaño exacto para albergar el tamaño solicitado y el fragmento restante que se marca como libre. Durante este recorrido de búsqueda se fusionan todos los bloques libres contiguos que se vayan encontrando. Si alcanzamos el final del array sin encontrar un bloque

suficientemente grande se devuelve la dirección 0, que es una dirección no válida pues por definición apunta a la cabecera del primer bloque. Las clases que solicitan memoria reciben la dirección del campo de datos del bloque, no de la cabecera.

4.1.2. Almacenamiento y gestión de objetos de Cryptoki

En PKCS #11 los objetos están definidos como un array de atributos, donde cada atributo es una estructura de tamaño fijo con tres campos: un tipo de atributo, un puntero al valor y la longitud del valor. Las estructuras empleadas en el cardlet siguen un esquema parecido que describimos a continuación. La estructura de los objetos, es de longitud variable y consta de los siguientes campos:

- Enlace al siguiente objeto, 2 bytes. Todos los objetos PKCS #11 que se crean en la tarjeta se mantienen en una lista enlazada.
- Número de atributos del objeto, 1 byte. Este campo puede deducirse en función del tamaño del bloque en memoria dinámica donde se aloja la estructura y el tamaño de la parte fija, pero se ha optado por incluir el número de atributos en la estructura del objeto porque el ahorro en consumo de memoria que hubiese supuesto su no inclusión es despreciable: típicamente la tarjeta almacenará únicamente dos objetos, un certificado y una clave privada asociada por lo que el ahorro total en este caso sería de 2 bytes.
- Un número variable de estructuras de atributos, tantas como número de atributos tenga el objeto.

La estructura de un atributo consta de los siguientes campos:

- Tipo de atributo, 4 bytes. Es el valor binario definido en el estándar.
- Puntero al valor del atributo, 2 bytes. El campo valor, al ser de tamaño inherentemente variable, se almacena en su propio bloque de memoria dinámica. No almacenamos la longitud del valor en la estructura del atributo para ahorrar memoria. La longitud de este campo se obtiene de la cabecera del bloque de memoria dinámica en el que se aloja.

El almacenamiento de estas dos estructuras se hace sobre la memoria dinámica descrita en el apartado anterior. Existe una clase Java para ayudar en el manejo de cada una de estas estructuras, son la clase `ObjPatr` para los objetos y la clase `AttrPatr` para los atributos. Estas clases no pueden instanciarse debido al problema de no liberación de memoria mencionado en el apartado 4.1.1, por lo que únicamente poseen métodos y miembros estáticos. Se utilizan para mapear porciones del array de memoria dinámica sobre la estructura correspondiente y para acceder cómodamente a los distintos campos de cada estructura: poseen métodos para establecer/obtener el valor de cada campo y para liberar toda la memoria dinámica asociada a cada una de estas estructuras. Estas dos clases extienden la clase `Patr`, que aporta los métodos básicos para acceder a campos de tipo multibyte. Antes de emplear una de estas clases para acceder a una estructura alojada en memoria dinámica es necesario establecer primero la dirección inicial de la estructura mediante una llamada a `setAddr(short addr)`, lo que establece la dirección de referencia empleada por todos los métodos que acceden a miembros de una estructura (`get...()`, `set...()`).

4.1.3. Implementación del cardlet en tarjetas comerciales

Con el objetivo de demostrar la independencia de dispositivo de JCCM, en este proyecto se han empleado dos kits de desarrollo Java Card para entorno Linux pertenecientes a diferentes fabricantes:

- **GemXpresso RAD 211is de Gemplus**, [GemXpresso RAD 211 UG], estas tarjetas sólo implementan algoritmos de clave simétrica, por lo que el cardlet que se desarrolló en este caso, se limita al almacenamiento de claves y certificados, que se transfieren al ordenador (incluida la clave privada) para realizar las operaciones criptográficas soportadas por nuestra implementación.
- **Cyberflex for Linux Starter's Kit 2.1**, [Cyberflex SDK], estas tarjetas implementan RSA por lo que el cardlet desarrollado realiza operaciones criptográficas. La principal limitación con la que nos enfrentamos es que las tarjetas son Java Card 2.0 y en esta especificación todavía no se incluía un paquete de seguridad (`javacard.security` en Java Card 2.1). Así, para el acceso desde Java Card a las capacidades criptográficas, es necesario utilizar la extensión proporcionada por Schlumberger `javacardx.crypto`, por lo que la solución desarrollada sólo es válida para este tipo de tarjetas.

En la tabla 1 se muestra una comparativa entre las prestaciones proporcionadas por cada una de las tarjetas empleadas en nuestro desarrollo. Se observa que las tarjetas GemXpresso tienen unas velocidades de almacenamiento y lectura mucho mayores que las Cyberflex, tanto en escritura como en lectura. Estas diferencias creemos que son debidas a la mejor implementación de la máquina virtual Java de las tarjetas de Gemplus respecto a las de Schlumberger.

Tarjeta	Tamaño del Cardlet (Bytes)	Almacenamiento (ms)			Lectura (ms)
		Clave privada	Clave pública	Certificado	Certificado
GemXpresso	6437	20312	12998	16228	8934
Cyberflex	3992	38122	28180	34036	16155

Cuadro 1: Comparativa entre tarjetas

5. Conclusiones

En este artículo hemos presentado la base de la PKI que hemos desarrollado, con la parte de cliente representada por JCCM, y la de servidor por ACerO.

JCCM es una alternativa flexible para gestionar certificados de cualquier tipo en tarjetas inteligentes, frente a otros trabajos que emplean tarjetas no programables o bien no utilizan la facilidad de programación. Nuestro enfoque no se limita a adaptar la funcionalidad criptográfica predefinida en las tarjetas a la visión lógica que debe ofrecer un módulo PKCS #11, ni recorta la semántica definida por el estándar para adaptarla a la que ofrece el fabricante en cada tarjeta. En la actualidad JCCM soporta dos tipos de tarjetas inteligentes: las GemXpresso RAD 211, y las Cyberflex Access de Schlumberger, y está disponible para su uso con Netscape para navegar de forma segura, cifrar y firmar correos.

ACerO representa nuestra apuesta por seguir utilizando un software tan probado como es openssl, a través del uso de servlets para exportarlo al mundo web de forma robusta y escalable.

JCCM y ACerO son partes de un mismo sistema, en el que faltan definir el soporte legal para poder denominarla PKI. En cualquier caso dejamos esta tarea a nuestros futuros usuarios, y abordamos los trabajos futuros en la línea de la certificación de atributos, mediante la integración en el sistema de módulos de autenticación PAM y listas de control de acceso.

Referencias

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

[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.

[FAQ Schlumberger] Schlumberger, http://cyberflex.com/Support/support.html

[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.

[CORE00] "Core Servlets and JavaServer Pages". Hall, Marty. Prentice Hall 2000.

[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).

[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