Universidad de las Ciencias Informáticas

Título: Aplicación para tarjetas inteligentes en una Infraestructura de Clave Pública

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas.

Autor(es): Kenly Rodríguez Ruíz
Yoan Cruz González

Tutor(es):MSc. Adonis Cesar Legón Campo
Ing. Dayron Almeida Sotolongo

Ciudad de la Habana

Junio 2012

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores del presente trabajo titulado: Aplicación para tarjetas inteligentes en una Infraestructura de Clave Pública y autorizamos a la Universidad de las Ciencias Informáticas y al Centro de Identificación y Seguridad Digital a usarlo en su beneficio.

Para que así conste firmamos la presente a los _____ días del mes de_____ del año _____

_____ _____

Kenly Rodríguez Ruíz Yoan Cruz González

_____

Adonis César Legón Campo

DATOS DE CONTACTO

A continuación se detalla una breve descripción de los datos de los tutores.

Nombre y Apellidos: Adonis Cesar Legón Campo.

Correo: alegon@uci.cu.

Situación laboral: Jefe del departamento de Tarjetas Inteligentes del Centro de Identificación y Seguridad Digital.

Años de graduado: 5

Especialidad de graduación: Ingeniería en Ciencias Informáticas.

Institución a la que pertenece: Universidad de las Ciencias Informáticas (UCI).

Nombre y Apellidos: Dayron Almeida Sotolongo.

Correo: dalmeida@uci.cu.

Situación laboral: Profesor del departamento de Tarjetas Inteligentes del Centro de Identificación y Seguridad Digital.

Años de graduado: 3

Especialidad de graduación: Ingeniería en Ciencias Informáticas.

Institución a la que pertenece: Universidad de las Ciencias Informáticas (UCI).

AGRADECIMIENTOS

Kenly

Agradecer infinitamente a mi madre, por apoyarme y guiarme siempre, por tu amor, por tu dedicación, por tu compresión. A mi tía Mami, mis primos Cuco y Misleidys, ustedes también son parte de este triunfo. A mi novio Leandro por su amor compresión y apoyo durante toda mi carrera. A Yisel por su amistad, su ayuda y su apoyo. A los tutores Adonis y Dayron por atendernos y dedicarnos su tiempo. A mi compañero de tesis. A todos mis compañeros de aula durante los cinco años, sin mencionar nombres para que no se quede nadie fuera.

Yoan

Agradezco a mis padres que siempre me han apoyado en todas mis decisiones y me han servido de ejemplo y guía para alcanzar mis metas. A mi hermana que siempre me ayudó en las tareas que requerían de un toque femenino, como lavar la ropa, cocinar, etc. A mi familia en sentido general porque siempre me apoyaron de una forma u otra. A todos mis compañeros de cuarto de estos 5 años, que resultaron ser personas agradables y buenos compañeros demostrando en más de una ocasión que podía contar con ellos para lo que fuera. A mis tutores que me ayudaron a comprender y desarrollar este trabajo lo cual no fue una tarea fácil. A las personas que conocí en el laboratorio de tarjetas inteligentes tanto los que están hoy como los que ya se graduaron. En general a todas las personas que conozco aquí en la UCI.

DEDICATORIA

Kenly

A la mejor madre del mundo, quien en todo momento me ha apoyado, quien ha estado pendiente de mis caprichos siempre dando lo mejor de sí, a quien le debo la vida y estar graduándome hoy como ingeniera, gracias por existir.

A mi abuelo que aunque no pudo estar físicamente en muchos momentos importantes de mi vida siempre lo llevo presente en mi pensamiento y corazón.

Yoan

A mis padres que en estos 5 años siempre apoyaron mis decisiones y me ayudaron en todo momento, este trabajo es tanto de ellos como mío, a mi hermana que estuvo estos 5 años conmigo y a mi familia en general.

RESUMEN

Los progresos alcanzados en microelectrónica en 1970 permitieron almacenar y procesar información en un chip con solo unos pocos milímetros de tamaño, dando lugar a un nuevo dispositivo, la tarjeta inteligente, que posteriormente se convertiría en una importante herramienta de seguridad, al permitir en su interior la ejecución de operaciones criptográficas. Éstas son utilizadas para disímiles funciones, siendo una de ellas, incrementar la seguridad de una Infraestructura de Clave Pública (*PKI* por sus siglas en inglés), una combinación de *hardware*, *software*, políticas y procedimientos de seguridad. El grado de seguridad de una *PKI* se incrementa considerablemente con el empleo de las tarjetas inteligentes, puesto que permite la creación y almacenamiento de llaves y certificados digitales en un entorno seguro dentro de la tarjeta.

El Centro de Identificación y Seguridad Digital (*CISED*) de la Universidad de las Ciencias Informáticas (*UCI*), a partir de la experiencia adquirida en el área de las tarjetas inteligentes, ha identificado la necesidad del desarrollo de una aplicación para tarjetas inteligentes en una Infraestructura de Clave Pública que cumpla con los principales estándares asociados a las mismas y con la capacidad de realizar operaciones criptográficas y de gestión de certificados digitales. Este documento recoge los resultados de la investigación realizada, describiéndose las principales características de los sistemas analizados, la arquitectura y el diseño de la solución a desarrollar. Además se describen las herramientas, tecnologías utilizadas y los artefactos generados en el proceso de desarrollo.

Palabras claves: certificados digitales, Infraestructura de Clave Pública, tarjeta inteligente.

ÍNDICE DE CONTENIDO

Índice de tablas

Índice de figuras

Introducción

El constante desarrollo mundial sumerge a las sociedades actuales en un amplio mar de información. Incrementando en gran medida los volúmenes de datos que se procesan, transmiten y almacenan en cualquier esfera de la sociedad. La necesidad de gestionar esta información trajo consigo que los métodos conocidos anteriormente para transmitirla y almacenarla, como libros, pergaminos, etc. evolucionaran hacia la era digital para alcanzar mayor capacidad y durabilidad, pero trayendo consigo sus propios problemas de seguridad.

La búsqueda de mecanismos para asegurar la información tuvo sus frutos con el descubrimiento de la criptografía, esta se basa en transformar un grupo de datos de entrada en otro sin significado aparente, solo entendible para la entidad deseada, a través de algoritmos matemáticos.

Existen dos tipos de criptografía, simétrica y asimétrica, y ambas proporcionan autenticidad, confidencialidad e integridad a la información. La criptografía asimétrica es la base de la Infraestructura de Clave Pública, una combinación de *hardware*, *software*, políticas y procedimientos de seguridad que permiten la ejecución de operaciones criptográficas como el cifrado y la firma digital, esta última provee los mecanismos para evitar problemas como el fraude y el robo de identidad. La firma digital también otorga a los documentos electrónicos las propiedades de autenticidad, no repudio e integridad. (Lucena López, 1999)

Los progresos alcanzados en microelectrónica en 1970 permitieron integrar almacenamiento de datos y lógica de procesado en un chip con solo unos pocos milímetros de tamaño, dando lugar a un nuevo dispositivo, la tarjeta inteligente, que posteriormente se convertiría en una importante herramienta de seguridad, al permitir en su interior la ejecución de operaciones criptográficas. Ésta fue patentada en este período y aunque existen discusiones sobre quien fue su inventor original, se reconoce a Juergen Dethloff de Alemania, Arimura de Japón y Roland Moreno de Francia como sus creadores. (Effing, 2002)

Las tarjetas inteligentes proporcionan seguridad y exactitud en la verificación de identidad. Cuando ellas se combinan con otros sistemas tecnológicos de identificación tales como: identificación biométrica y certificados digitales, pueden elevar la seguridad de los sistemas y proteger la privacidad de la información. Desde su creación, las tarjetas inteligentes han aumentado sus capacidades técnicas gracias al desarrollo de su tecnología, lo que ha propiciado un aumento en las aplicaciones que pueden tener las mismas. (Smart Card Operating Systems: Past, Present, 2003)

El grado de seguridad de una Infraestructura de Clave Pública se incrementa considerablemente con el empleo de las tarjetas inteligentes, puesto que permite la creación y almacenamiento de llaves y certificados digitales en un entorno seguro dentro de la tarjeta. Las tarjetas inteligentes, que forman parte de una Infraestructura de Clave Pública, son ideales para los clientes que requieren los más altos estándares de seguridad, como las autoridades públicas y las corporaciones que implementan servicios con acceso en línea, operadores de red e instituciones financieras que cuentan con sistemas de comercio electrónico (e-commerce) y permiten transacciones seguras utilizando firma digital. Es por ello que las tarjetas inteligentes son un vínculo vital en la cadena de confianza de la Infraestructura de Clave Pública. (Alliance, Smart Card, 2006)

El desarrollo de aplicaciones (applets) para *PKI* en tarjetas inteligentes, está mayormente distribuido por los países europeos. Por ser una tecnología relativamente nueva, no es de extrañar que la mayoría de las aplicaciones que se desarrollan sean privativas, impidiendo que los países menos desarrollados aprovechen sus ventajas. Existe una aplicación libre para *PKI* en tarjetas inteligentes desarrollada por Wojciech Mostowski, con el nombre *Java Card PKI Applet* que cumple con el estandar *PKCS#15*, pero no soporta autenticación mutua por *GlobaPlatform*, solo contiene un par de llaves *RSA* para firma y una llave secreta para realizar autenticación mutua *ISO/IEC-7816*.

En Cuba, aunque existen instituciones que trabajan en el desarrollo de aplicaciones para tarjetas inteligentes, se puede decir que la experiencia en este sentido es insuficiente. Específicamente en el campo de las aplicaciones para Infraestructura de Clave Pública existen desarrollos nacionales, pero no en tarjetas inteligentes, o al menos no se encontraron en la bibliografía analizada aplicaciones de este tipo. Actualmente las empresas que desarrollan aplicaciones para tarjetas inteligentes en Infraestructura de Clave Pública, hacen que sus productos cumplan con los principales estándares internacionales que se aplican en este tipo de soluciones, con el objetivo de lograr una mayor interoperabilidad y al mismo tiempo satisfacer las necesidades del cliente.

La Universidad de las Ciencias Informáticas (*UCI*) cuenta con varios centros de producción, uno de ellos es el Centro de Identificación y Seguridad Digital (*CISED*), el mismo está compuesto por varios departamentos especializados en diferentes temas. El departamento de Tarjetas Inteligentes se dedica a desarrollar aplicaciones referentes a esta tecnología, con el fin de obtener productos y servicios basados en tarjetas inteligentes.

Partiendo de lo anteriormente expuesto surge como **situación problémica**: la solución libre existente para tarjetas inteligentes en una *PKI* no permite realizar autenticación mutua por *GlobalPlatform* y no permite definir varios objetos de seguridad (llaves, *PINs*, entornos de seguridad) para la protección de los archivos y operaciones criptográficas, por tanto, es necesario para el *CISED*, contar con una aplicación en tarjetas inteligentes, que cumpla con los principales estándares internacionales y que, permita el uso de estos dispositivos en operaciones de firma digital y autenticación utilizando certificados digitales.

Luego de haber planteado la situación anterior, estamos en presencia del siguiente **problema de investigación:** ¿Cómo mejorar el uso de tarjetas inteligentes para realizar operaciones criptográficas de firma digital y autenticación utilizando certificados digitales?

Se tiene como **objeto de estudio** de la presente investigación: las tarjetas inteligentes como dispositivo de seguridad.

Para dar solución al problema existente se ha definido como **objetivo genera**l: Desarrollar una aplicación utilizando tecnología *Java Card*, que permita al *CISED* el uso de tarjetas inteligentes en operaciones criptográficas de firma digital y autenticación utilizando certificados digitales.

El **campo de acción** se centra en las tarjetas inteligentes en una Infraestructura de Clave Pública.

Tareas de Investigación. (Ver anexo 1)

- Descripción de las soluciones existentes sobre el uso de tarjetas inteligentes en una Infraestructura de Clave Pública.
- Caracterización de la tecnología *Java Card*.
- Análisis de los principales aspectos de la norma *ISO/IEC 7816* relacionados con la comunicación con tarjetas inteligentes, la estructura para almacenamiento de información y los modelos de seguridad.
- Análisis del estándar *PKCS#15* relacionado con la estructura de la información para un dispositivo criptográfico.
- Análisis de los principales algoritmos de criptografía simétricos y asimétricos, protocolos de autenticación, algoritmos de firma digital y certificados digitales.
- Análisis y diseño de la aplicación *Java Card* para tarjetas inteligentes, cumpliendo los estándares analizados.
- Implementación de la aplicación *Java Card* a partir del diseño realizado.
- Realización de las pruebas de calidad a la aplicación desarrollada.

Es necesario llevar a cabo una investigación profunda del objeto de estudio en cuestión, mientras no se conozca del tema, o el conocimiento no sea el suficiente para alcanzar el objetivo propuesto. Para ello es favorable utilizar como **estrategia de investigación** la exploratoria.

Los métodos científicos de investigación se clasifican en **métodos teóricos** y **métodos empíricos.**

Los **métodos teóricos** permiten estudiar particularidades del objeto de investigación, además de proporcionar la construcción de modelos. Por estas razones los métodos teóricos que se evidencian en la presente investigación son: Analítico-Sintético y Análisis Histórico-Lógico.

- **Analítico–Sintético:** Mediante este método se realizará un estudio de diferentes soluciones existentes en la actualidad y los principales estándares relacionados con las tarjetas inteligentes, de manera tal que podamos integrar las mismas con una Infraestructura de Clave Pública. Además se consultará toda la bibliografía necesaria para dar cumplimiento a las tareas de la investigación.
- **Histórico-Lógico:** Posibilita el análisis histórico del proceso de realizar operaciones criptográficas y de gestión de certificados digitales, en una Infraestructura de Clave Pública. Este método se emplea durante todo el proceso de diseño y mejoramiento de funcionalidades de la aplicación de *Java Card*.

Los **métodos empíricos** permiten, como parte del procedimiento trazado para la investigación, determinar el método de recolección de datos y tipo de instrumento que será utilizado, es por ello que el método empírico que se evidencia en el presente trabajo de diploma es: Observación.

- **Observación:** Este método, precisamente por ser un procedimiento sencillo de realizar, se utilizó para obtener información primaria en el ámbito de la Infraestructura de clave pública en tarjetas inteligentes y las operaciones criptográficas de firma digital y autenticación utilizando certificados digitales.

El fundamento de la presente investigación está dado por la importancia que posee el desarrollo de este tipo de aplicaciones, la cual ha constituido un impacto social, ya que al utilizar una tarjeta inteligente como elemento de seguridad en una Infraestructura de clave pública, aumentaría la seguridad de ésta, al permitir que la llave privada utilizada en las operaciones de firma digital se genere dentro de la tarjeta, por lo que no estaría expuesta a un medio inseguro. La investigación permitirá al *CISED*, desarrollar productos y servicios que

empleen las tarjetas inteligentes como dispositivo criptográfico, para aumentar la seguridad de una *PKI*, cumpliendo así, con una de las líneas de investigación del departamento de tarjetas inteligentes.

La investigación está estructurada en tres capítulos:

Capítulo 1 Fundamentación Teórica: Este capítulo contiene una base teórica para concebir el problema planteado. Se describen los conceptos fundamentales para el dominio del problema, se muestra el estudio de aplicaciones existentes en el mundo, además de hacer referencia a estándares, tecnologías, herramientas y metodologías actuales que se usaron en el presente trabajo de diploma.

Capítulo 2 Análisis y Diseño de la aplicación para tarjetas inteligentes en una Infraestructura de Clave Pública: Se presentan las fases de Exploración y Planificación definidas por la metodología *XP* para dar solución al problema científico. Se identifican las historias de usuarios y los requerimientos no funcionales, se realiza el plan de iteraciones y plan de entregas.

Capítulo 3 Implementación y diseño de pruebas de la aplicación para tarjetas inteligentes en una Infraestructura de Clave Pública: Se da cumplimiento a los planes trazados a través de las fases: Iteraciones a primera liberación y Producción, se codifica la solución diseñada y se le realizan pruebas a la aplicación.

Capítulo 1: Fundamentación Teórica

1.1 Introducción

El objetivo principal de la elaboración del presente capítulo es comprender las bases teóricas de la situación problémica planteada. Se hace además un estudio de los diferentes estándares internacionales asociados para la implementación y puesta en práctica de la aplicación (*applet*), así como de los principales conceptos para el dominio del problema. Se aborda también sobre las diferentes aplicaciones existentes en la actualidad relacionadas con las tarjetas inteligentes integradas a una Infraestructura de Clave Pública, que permitan operaciones de firma digital y autenticación, utilizando certificados digitales. Igualmente se realiza el estado del arte referente a las herramientas y metodologías con las cuales se desarrollará la aplicación.

1.2 Criptografía

Según el Diccionario de la Real Academia, la palabra Criptografía proviene del griego *cripto*, que significa oculto, y *grafía*, que significa escritura, y su definición es: "Arte de escribir con clave secreta o de un modo enigmático". Obviamente la Criptografía hace años que dejó de ser un arte para convertirse en una técnica, o más bien un conglomerado de técnicas, que tratan sobre la protección —ocultamiento frente a observadores no autorizados— de la información. (Lucena López, 1999)

Conviene hacer notar que la palabra Criptografía sólo hace referencia al uso de códigos, por lo que no engloba a las técnicas que se usan para romper dichos códigos, conocidas en su conjunto como Criptoanálisis.

La criptografía se representa a través de un modelo al cual se le llama **criptosistema**. Un criptosistema es una quíntupla M, C, K, E, D donde:

- M representa el conjunto de todos los mensajes sin cifrar.
- C representa el conjunto de todos los posibles mensajes cifrados, o **criptogramas**.
- K representa el conjunto de llaves que se pueden emplear en el criptosistema.
- E es el conjunto de transformaciones de cifrado o familia de funciones que se aplica a cada elemento de M para obtener un elemento de C. Existe una transformación diferente E_k para cada valor posible de la llave k.
- D es el conjunto de transformaciones de descifrado, análogo a E.

Todo criptosistema ha de cumplir la siguiente condición: $D_k\ E_k(m\ ) = m$. (Lucena López, 1999)

Existen dos tipos fundamentales de criptosistemas:

- **Criptosistemas simétricos o de clave privada**. Son aquellos que emplean la misma clave k tanto para cifrar como para descifrar. Presentan el inconveniente de que para ser empleados en comunicaciones, la clave k debe estar tanto en el emisor como en el receptor, lo cual nos lleva a preguntarnos cómo transmitir la clave de forma segura.
- **Criptosistemas asimétricos o de llave pública**, que emplean una doble clave (kp, kP). kp se conoce como clave privada y kP se conoce como clave pública. Una de ellas sirve para la transformación E de cifrado y la otra para la transformación D de descifrado. En muchos casos son intercambiables, esto es, si empleamos una para cifrar la otra sirve para descifrar y viceversa. Ofrecen un espectro superior de posibilidades, pudiendo emplearse para establecer comunicaciones seguras por canales inseguros —puesto que únicamente viaja por el canal la clave pública, que sólo sirve para cifrar—, o para llevar a cabo autentificaciones. (Lucena López, 1999)

1.3 Firma Digital

El término firma digital se emplea para denominar un esquema matemático que es utilizado para garantizar la autenticidad de documentos electrónicos o mensajes transmitidos electrónicamente, a partir de la verificación de esta firma se puede determinar si el documento o mensaje ha sido alterado. Una firma digital tiene la propiedad de ser producida solo por un individuo, pero puede ser verificada por cualquiera que posea la llave de verificación que es de dominio público.

El término firma digital usualmente sólo se utiliza relacionado con algoritmos criptográficos asimétricos debido a que la separación de las llaves pública y privada permite una mayor cantidad de aplicaciones para la firma digital. Sin embargo las firmas basadas en algoritmos simétricos son utilizadas a menudo en la práctica, aunque estas solamente permiten comprobar la autenticidad de documentos si la llave utilizada para generar la firma es conocida. (Effing, 2002)

Una firma digital es una secuencia de bits que se añade a una pieza de información cualquiera, y que permite garantizar su autenticidad de forma independiente del proceso de transmisión, tantas veces como se desee. Presenta una analogía directa con la firma manuscrita, y para que sea equiparable a esta última debe cumplir las siguientes propiedades (Lucena López, 1999):

- Va ligada indisolublemente al mensaje. Una firma digital válida para un documento no puede ser válida para otro distinto. (Lucena López, 1999)

- Sólo puede ser generada por su legítimo titular. Al igual que cada persona tiene una forma diferente de escribir, y que la escritura de dos personas diferentes puede ser distinguida mediante análisis grafológicos, una firma digital sólo puede ser construida por la persona o personas a quienes legalmente corresponde.(Lucena López, 1999)
- Es públicamente verificable. Cualquiera puede comprobar su autenticidad en cualquier momento, de forma sencilla.(Lucena López, 1999)

1.4 Certificado digital

Para verificar una firma digital se necesita la llave pública apropiada, esta llave no puede ser enviada sin protección, pues no habría forma de validar su autenticidad, por lo que es necesario que sea firmada por una entidad confiable, cuya autenticidad sea verificable, esta entidad es denominada autoridad de certificación. Se denomina certificado digital a la combinación de la llave pública que ha sido firmada por la autoridad de certificación, la firma digital de esa llave pública y algunos parámetros adicionales. (Effing, 2002)

1.5 Tecnología de Infraestructura de Clave Pública

Permitir la autenticación de usuarios frente a otros usuarios y usar la información de los certificados de identidad para cifrar y descifrar mensajes, firmar digitalmente cualquier información, y garantizar el no repudio de un envío son de las principales responsabilidades y funciones de la tecnología de Infraestructura de Clave Pública.

Las operaciones criptográficas de clave pública, constituyen procesos, en los que son usados algoritmos de cifrado conocidos y accesibles para todos. Es por ello que la seguridad que puede aportar esta tecnología está fuertemente ligada en la privacidad de la llave privada y las Políticas de seguridad aplicadas.

1.5.1 Componentes de una Infraestructura de Clave Pública

Infraestructura de Clave Pública es la forma común de referirse a un sistema complejo necesario para la gestión de certificados digitales y aplicaciones de la Firma digital. Una *PKI* tiene varios componentes como son: usuarios, entidades certificadoras (*CA*), certificados, y repositorios de certificados. Se encarga de proveer una forma de estructurar estos componentes y define estándares con este propósito. Una forma particular de *PKI* es la que se muestra en la figura 1. En la imagen la *CA* raíz denominada *Root* certifica a las *CAs* del segundo nivel denominadas autoridades regionales (*RA* por sus siglas en inglés), que pueden representar las *CAs* de algún país o una región y estas a su vez certifican a las *CAs* que generan los certificados *X.509* de organizaciones e individuos. Cuando la autoridad raíz

autoriza una *RA*, genera un certificado *X.509* con la llave pública de la *RA* incluida y firmado por ella, que establece que ha aprobado esa *RA*. Similar al proceso anterior ocurre en el caso de las demás *CAs*. Para verificar la legitimidad de un certificado solo se debe seguir la cadena de confianza (verificar cada certificado con la llave pública almacenada en el certificado que emitió la *CA* superior), hasta encontrar una *CA* en la cual se confíe. (Tanenbaum, 2003)

La siguiente imagen muestra una *PKI* estructurada jerárquicamente.

(a) Estructura de la *PKI*.

(b) Cadena de certificados.

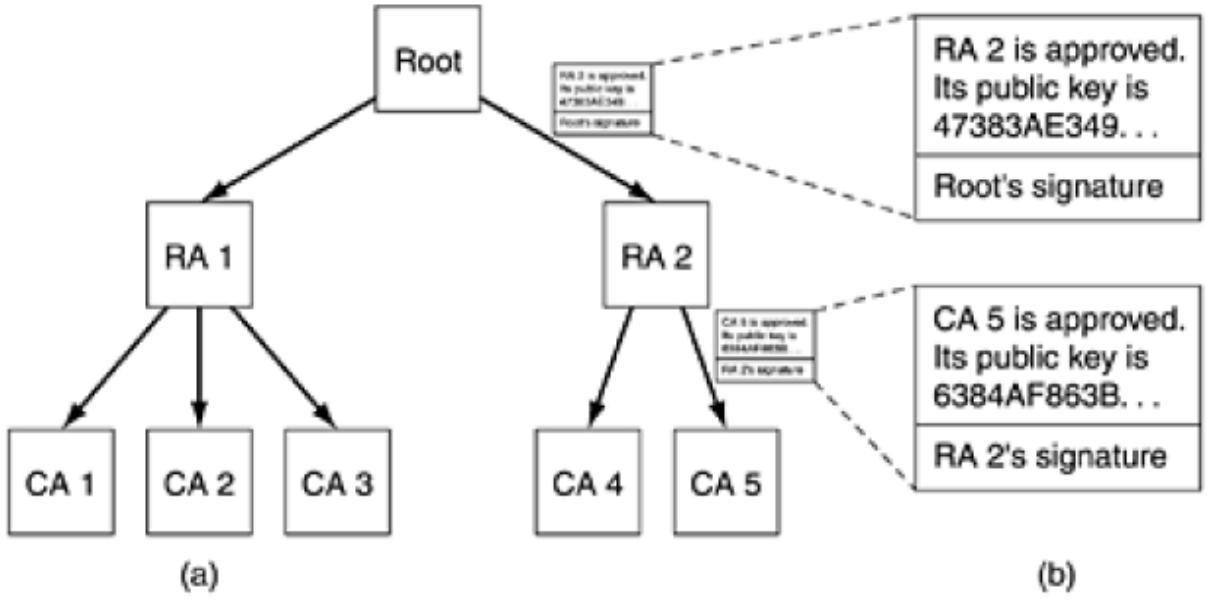


Figura 1: Ejemplo de estructura de PKI (Tanenbaum, 2003)

1.6 Tarjetas Inteligentes

Las tarjetas inteligentes fueron concebidas y patentadas por alemanes, japoneses y franceses en 1970. Estos dispositivos son tarjetas de plástico que contienen un circuito integrado, son semejantes en tamaño y otros estándares físicos a las tarjetas de crédito. Dicho circuito puede ser de solo memoria o contener un microprocesador con un sistema operativo que le permite un grupo de tareas tales como:

1. Almacenar información.
2. Encriptar información.
3. Leer y escribir datos, como una computadora. (Effing, 2002)

Las tarjetas inteligentes con su seguridad hacen que los datos que se consideran personales solo sean accesibles por los usuarios apropiados. Las tarjetas inteligentes aseguran la portabilidad, seguridad y confiabilidad en los datos.(Effing, 2002)

Son las tarjetas inteligentes basadas en estándares las utilizadas para autenticar la identidad de una persona o entidad. Es importante destacar el nivel de seguridad que poseen para el almacenamiento de la información. La tarjeta inteligente puede ser multi-aplicación, o sea puede contener información de una o varias aplicaciones al mismo tiempo, esto puede ocurrir debido a la característica que poseen, ya que tienen una jerarquía de almacenamiento de la información. Cada una de las aplicaciones contenidas en la tarjeta se protege independientemente. (Effing, 2002)

Con el surgimiento de las tarjetas inteligentes se han desarrollado cuantiosas aplicaciones, que hasta hace unos años eran consideradas prácticamente imposibles. En la actualidad se pueden apreciar en diferentes ámbitos, siendo los más comunes:

- Telefonía Móvil Digital (*GSM*): donde la identificación del titular del número telefónico, la herramienta de cifrado y la base de datos del propietario constituyen las funciones de la tarjeta inteligente.
- Banca: su desarrollo está dado a través de los Monederos Electrónicos, permitiéndole al usuario cargarlo con una determinada cantidad de dinero y realizar compras teniendo en cuenta el saldo disponible en la tarjeta. (Effing, 2002)

Las tarjetas inteligentes son clasificadas de acuerdo a sus capacidades, estructura del sistema operativo, formato e interfaz de comunicación. Esta última está compuesta por otras dos clasificaciones: tarjetas de contacto y tarjetas sin contacto. Las tarjetas de contacto disponen de unos contactos metálicos visibles y debidamente estandarizados (parte 2 de la Organización Internacional de Normalización (*ISO* por sus siglas en inglés) y la Comisión Electrónica Internacional (*IEC* por sus siglas en inglés) 7816), no siendo así en las tarjetas sin contacto. Las tarjetas de contacto, por su parte, deben ser insertadas en una ranura de un lector para llevar a cabo operaciones en las mismas. Luego el lector alimenta eléctricamente a la tarjeta y transmite los datos oportunos para operar con ella conforme al estándar. Por su parte las tarjetas sin contacto utilizan la tecnología de identificación por radiofrecuencia (*RFID* por sus siglas en inglés) para leer y escribir desde el chip incrustado en la tarjeta.

1.7 Applet

Es un componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo un navegador web. El *applet* debe ejecutarse en un contenedor, que lo proporciona un programa anfitrión. (Oracle)

Los *applets*, según el contexto del trabajo, son aplicaciones implementadas utilizando la tecnología *Java Card* y son ejecutadas dentro de las tarjetas inteligentes. El *applet* para tarjetas inteligentes en una *PKI*, que gestiona la firma digital y la autenticación utilizando certificados digitales, se ejecuta en el contexto del *Java Card Runtime–Environment* (*JCRE*). Este *applet* es el encargado de ejecutar las operaciones que maneja, mediante los comandos *APDU*[1] (Ver tabla 1) que le son enviados, retornando los resultados mediante *APDU* de respuestas.

1.8 Tecnología en tarjetas inteligentes

1.8.1 Java Card

Esta tecnología permite en tarjetas inteligentes y similares dispositivos incrustados, ejecutar aplicaciones Java. Ofrece la posibilidad y capacidad al usuario de implementar soluciones aplicadas en distintas esferas de la sociedad y la economía, con elevados niveles de seguridad, ejemplos claros los podemos encontrar en las aplicaciones que permiten la firma digital, la autenticación de usuarios, tarjetas de monedero electrónico y además en las tarjetas módulo de identificación del suscriptor (*SIM* por sus siglas en inglés) utilizadas en la telefonía celular.

Una tarjeta inteligente posee una estructura lógica además de su estructura física, la figura 2 muestra las diferentes capas por las cuales está conformada. El microprocesador es el encargado de realizar todas las operaciones, mientras el sistema operativo se encarga de traducir las acciones de las capas superiores en instrucciones que el microprocesador pueda entender. La máquina virtual de *Java Card* permite la ejecución de aplicaciones *Java Card* dentro de la tarjeta, mientras que las interfaces de programación de aplicación (*APIs* por sus siglas en inglés) de *Java Card* poseen funcionalidades utilizadas por los *applets.*

[1] Unidad de Datos del Protocolo de Aplicación.

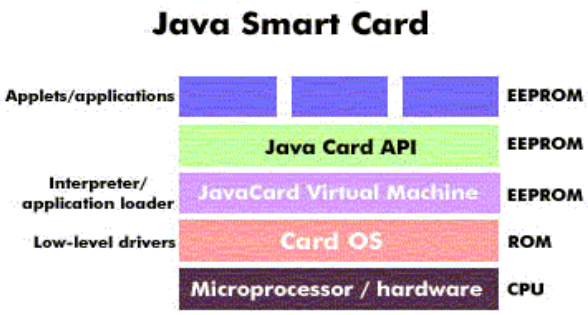


Figura 2: Estructura del *software* de la tarjeta. (Perovich, et al., 2001)

1.8.1.1 Aplicaciones de *Java Card*

Desde el surgimiento de las tarjetas inteligentes el número de aplicaciones para *Java Card* ha ido aumentando considerablemente y en diversas ramas, ejemplos típicos se mencionan a continuación:

- Monedero electrónico: esta aplicación es utilizada como dinero, donde luego de haber fijado un monto inicial, se puede realizar consultas, operaciones de débito o crédito. Normalmente para evitar fraudes u otros delitos esta tarjeta lleva asociado algún sistema de seguridad.
- Transacciones Seguras: Las tarjetas inteligentes son menos vulnerables que las tarjetas magnéticas debido a su nivel de seguridad, pues es normal que incluyan un *API* de Criptografía fuerte.
- Identificación Digital / Firma Digital: Validar la identidad del portador de la tarjeta, o certificar el origen de ciertos datos es el principal objetivo de esta aplicación. Comúnmente se basa en las primitivas criptográficas del *API* y/o las que están implementadas en *hardware*.
- Sistemas de Prepago: El propietario de la tarjeta la carga con una cierta cantidad de servicios y según él vaya haciendo uso del servicio, entonces la cantidad de servicios se irá decrementando.
- Tarjeta de salud: Este sistema es implementado en algunos hospitales para la identificación y almacenamiento de los principales datos de la historia clínica del paciente en tarjetas inteligentes, con el objetivo de proporcionar una mejor atención. En estos momentos la capacidad de almacenamiento es muy pequeña, pero día a día esto puede incrementar, llegando a poder almacenar toda la historia clínica.
- Control de Acceso y de Asistencia: Existen en la actualidad varios sistemas de control de acceso y asistencia implementados en base a tarjetas inteligentes o a iButtons.

Existen más aplicaciones de *Java Card*, aunque, algunas son variantes de las que se mencionan anteriormente, también hay otras que satisfacen necesidades específicas de una empresa.

1.8.1.2 *Java Card* Applet

Los *applets* son las aplicaciones que corren dentro de una tarjeta *Java Card*. Dichas aplicaciones interactúan en todo momento con el *JCRE* utilizando los servicios que éste brinda, e implementan la interfaz definida en la clase abstracta *javacard.framework.Applet.*

Una vez registrada en el *JCRE* (con el método register ()), entonces el *applet* está listo para ser ejecutado.

La clase *javacard.framework.Applet* define cuatro métodos públicos que son utilizados por el *JCRE* para hacer funcionar las aplicaciones.

- Método install(byte(), short, byte)

El *JCRE* antes de crear la instancia del *applet* en la tarjeta, invoca a este método, cuya implementación habitual es llamar al constructor de la clase (generalmente este constructor es privado), crear todos los objetos necesarios para ejecutar el *applet* y por último registrar el *applet* con el método registrer().

- Método select()

Como resultado de la recepción a un *SELECT APDU* es solicitado este método por el *JCRE*. El *APDU* contiene el identificador de aplicación (*AID* por sus siglas en inglés) del *applet* a seleccionar.

El *AID* es una secuencia de entre 5 y 16 bytes, donde la *ISO* asigna los 5 primeros bytes (*RID*) y los proveedores de aplicaciones definen los siguientes 11 bytes (*PIX*), esto hace que la aplicación sea identificada de forma única, de acuerdo a la *ISO/IEC 7816*, y es la propia *ISO* quien le otorga los *AIDs*.

- Método process(APDU)

Cuando llega un *APDU* el *JCRE* invoca este método del *applet* seleccionado, pasándole como parámetro el *COMMAND APDU* recibido. Dentro de este método, el *applet* identifica el comando asociado al *APDU* y los parámetros, si los hay, y los procesa de acuerdo al protocolo que se haya definido para la interacción entre el *applet* y la aplicación terminal.

- Método deselect()

Para avisar al *applet* que se encuentra en ese momento seleccionado que va a dejar de estarlo, es que el *JCRE* invoca a este método.

La tecnología *Java Card* tiene como principal objetivo llevar los beneficios del desarrollo del *software* orientado a objetos al mundo de las tarjetas inteligentes.

1.8.2 Soluciones existentes para *PKI* en tarjetas inteligentes

Durante el estudio de las soluciones existentes relacionadas con las tarjetas inteligentes en una Infraestructura de Clave Pública, se identificaron un conjunto de productos. A continuación se muestran algunos de ellos:

- Classic TPC: Tarjeta con aplicación *PKI*, privativa y compatible con el estándar de criptografía de clave pública (*PKCS* por sus siglas en inglés) *PKCS#15* sobre la tecnología *Java Card* y se integra al middleware Classic Client. El proveedor de este producto es Gemalto. (Gemalto., 2011)
- .Net Card. Solución privativa que incluye el *middleware* con *PKCS11* (Windows, Linux y MAC) y *CSP*, y las tarjetas *PKI* y One Time Password (*OTP*) con tecnología .Net. Se integra a servicios de administración de identidades (Microsoft) y de autenticación. Su proveedor es Gemalto. (Gemalto., 2011)
- Private Card. Tarjeta con aplicación *PKI*, privativa que integra verificación biométrica con el *Match On Card* de Precise Biometrics. Su proveedor es *ARX*. (ARX, 2012)
- *Java Card* PKI *applet*. Solución libre desarrollada por Wojciech Mostowski que soporta canal seguro *GlobalPlatform*, tamaño de llave de 1024, dos *PINs* y tres llaves.

1.8.3 Estándares relacionados con tarjetas inteligentes

1.8.3.1 Estándar ISO/IEC- 7816

El estándar internacional *ISO/IEC – 7816* está estrechamente relacionado con las tarjetas de identificación específicamente con las tarjetas inteligentes, gestionado de conjunto por la Organización Internacional de Normalización (*ISO*) y la Comisión Electrónica Internacional (*IEC*). Esta norma es una extensión de dos anteriores *ISO/IEC 7810* e *ISO/IEC 7811*, donde se definen características físicas con que deben cumplir las tarjetas de identificación. Cualquier tarjeta básica cumple con lo establecido en la *ISO/IEC 7816*. El objetivo que persiguen estas normas es lograr la interoperabilidad entre los diferentes fabricantes y los lectores de tarjetas, en lo que respecta a características físicas, comunicación de datos y seguridad.

Descripción de las partes del estándar (ISO/IEC 7816).

- ✓ **7816-1: Características físicas.**

En el estándar *ISO/IEC 7816* parte 1 están definidos los tamaños físicos para tarjetas inteligentes que se presentan a continuación:

- **ID 000**: el establecido para las tarjetas *SIM* usadas para teléfonos móviles *GSM*. Por lo general tienen este formato también las tarjetas *SAM* (*Security Access Module*) utilizadas para la autenticación criptográfica mutua de tarjeta y terminal.
- **ID 00**: tamaño intermedio poco usado comercialmente.
- **ID 1**: es para las tarjetas de crédito y el más frecuente.

✓ **7816-4: Organización, seguridad y comandos para el intercambio de información.**

ISO/IEC 7816-4:2005 es independiente de la tecnología de la interfaz física. Se aplica a muchas tarjetas dígase: de contactos, de proximidad, radiofrecuencia, entre otras. Este estándar especifica (ISO, 2005):

- Contenido de los pares comando-respuesta que se utilizan en la comunicación con la tarjeta.
- El significado de los objetos de datos almacenados en la tarjeta.
- Estructura de aplicaciones y datos.
- Métodos de acceso a los ficheros y a los datos.
- Métodos de mensajería segura.
- Arquitectura de seguridad, definiendo los permisos para tener acceso a los ficheros de la tarjeta.

Este estándar define una estructura de aplicaciones y datos, donde dos categorías son permitidas, los ficheros elementales (*EF* por sus siglas en inglés) y los ficheros dedicados (*DF* por sus siglas en inglés), los primeros pueden contener datos y no pueden ser raíz de otros ficheros, mientras que los *DFs* pueden contener varios *EFs*, además de aplicaciones (*DF*) y objetos de datos, un ejemplo de esta estructura se muestra a continuación. (ISO, 2005)

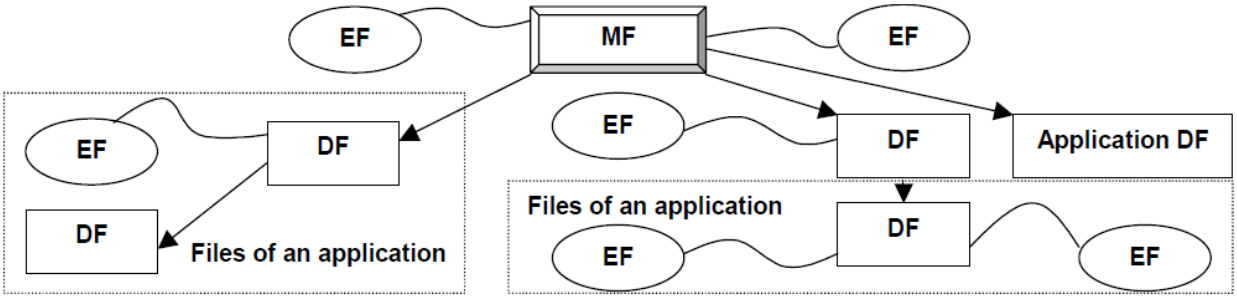


Figura 3: Estructura de ficheros ISO/IEC 7816-4(ISO, 2005)

La estructura de los comandos *APDU* y respuestas definidos por *ISO/IEC 7816-4* es la siguiente:

Estructura del Comando:

Campo	Descripción	Cantidad de bytes
Cabecera del comando	Byte de la clase denominado CLA.	1
	Byte de instrucción denominado INS.	1
	Parámetros denominados P1 y P2.	2
Lc	Lc=Nc, presente solo si Nc> 0 (Nc = longitud de los datos).	0, 1 o 3
Campo de datos	Cadena de Nc bytes si Nc>0.	Nc
Le	Le=Ne, presente solo si Ne>0 (Ne = longitud de los datos esperados)	0, 1, 2 o 3

Tabla 1: Estructura de los comandos APDU

Estructura de la respuesta:

Datos de la respuesta	Cadena de Nr bytes si Nr>0 (Nr = longitud de la respuesta).	Nr<= Ne
Palabras de estado	Denotan el estado del procesamiento del comando en la tarjeta.	2

Tabla 2: Estructura de la respuesta APDU

- ✓ **7816-5: Sistema de numeración y procedimientos de registro.**

ISO/IEC 7816-5 define la forma de uso de un identificador de aplicación para determinar la presencia y recuperar una aplicación en la tarjeta.

- ✓ **7816-8: Comandos inter-industriales relacionados con seguridad.**

Desde su resumen, especifica los comandos complementarios de las tarjetas que pueden utilizarse para operaciones criptográficas.

- ✓ **7816-9: Comandos adicionales inter-industriales y atributos de seguridad.**

Especifica los comandos de las tarjetas inteligentes para gestionar ficheros. Dichos comandos comprenden todo el ciclo de vida de la tarjeta y determinados comandos pueden ser utilizados antes de que la tarjeta haya sido expedida a su titular o después de que ésta haya caducado.

- ✓ **7816-15: Aplicación de información criptográfica.**

Especifica una aplicación que contiene información sobre las funcionalidades criptográficas de una tarjeta. Por otra parte, *ISO/IEC 7816-15:2004* define una sintáxis común (en ASN.1) y el

formato de codificación de la información y los mecanismos para compartir esta información en el momento necesario.

1.8.3.2 ISO\IEC 14443

El estándar *ISO\IEC 14443* describe los métodos y parámetros de operación de las tarjetas sin contacto o de radiofrecuencia.

Este estándar consta de 4 partes:

1- **Características físicas**

Esta sección del estándar define las propiedades mecánicas de las tarjetas inteligentes, dimensiones, resistencia a la exposición a los rayos ultra violeta y rayos X, resistencia a la torsión y flexión dinámicas, características del campo electromagnético y resistencia a las descargas electrostáticas.

2- **Interfaces de radio frecuencia**

En esta parte del estándar se especifican las características de la radiofrecuencia así como la interfaz de la señal de las tarjetas sin contacto para el tipo A y B:

- ✓ Frecuencia de portadora (13,56 MHz ± 7 KHz).
- ✓ Rangos de operatividad del campo magnético.
- ✓ Características de las tarjetas sin contacto Tipo A y B.

3- **Inicialización y colisiones**

En esta parte del estándar se describe el inicio de la comunicación y como resolver una posible colisión cuando más de una tarjeta se presenta en al campo electromagnético del lector, así como la selección de la aplicación de la tarjeta a procesar.

4- **Protocolos de Transmisión**

Una vez establecida la comunicación entre el lector y la tarjeta ya se puede comenzar a enviar los datos que se quieren procesar o escribir en la misma. Esta parte del estándar describe la estructura de los datos y el protocolo necesario para llevar a cabo esta tarea y además explica cómo gestionar los errores de transmisión para que los datos puedan ser transmitidos sin errores. (Finkenzeller, 2010)

1.8.3.3 Estándar *GlobalPlatform*

Este estándar lidera mundialmente el desarrollo en temas de infraestructura de tarjetas inteligentes. Sus descripciones técnicas probadas para las tarjetas, dispositivos y sistemas son consideradas como las normas de la industria para lograr implementaciones interoperables, flexibles y sostenibles por tarjetas que soportan multi-aplicación y multi-actor e implementaciones multi-modelo de negocio. (GlobalPlatform, 2003)

GlobalPlatform es un estándar para la administración de la estructura de las tarjetas inteligentes, comprende la instalación y borrado de las aplicaciones y la administración de otras tareas de las tarjetas. El emisor de la tarjeta tiene el control total sobre el contenido de ésta, pero puede permitir a otras instituciones administrar sus propias aplicaciones, esto se logra aplicando protocolos criptográficos, permitiendo a cada institución tener su propia área segura en la tarjeta. (GlobalPlatform, 2003)

El siguiente diagrama muestra la arquitectura definida por *GlobalPlatform*:

Runtime Environment: comprende la máquina virtual de *Java Card* (*JCVM*) junto a clases y servicios definidos en la *Java Card Application Programming Interface (API)*.

Security Domain: Es un área protegida en la tarjeta inteligente, a estos dominios de seguridad se le asignan aplicaciones que pueden utilizar los servicios criptográficos que el dominio ofrece.

Card Manager: Es el componente central de una tarjeta con arquitectura *GlobalPlatform*. Todos los servicios son ejecutados por él y ofrece interfaces para utilizar los servicios, internamente a través de las *APIs* de *GlobalPlatform* y externamente a través de los comandos *APDU*.

GlobalPlatform API: Permite el acceso a las funciones del *CardManager* y autenticar el terminal con la tarjeta, utilizando un canal seguro.

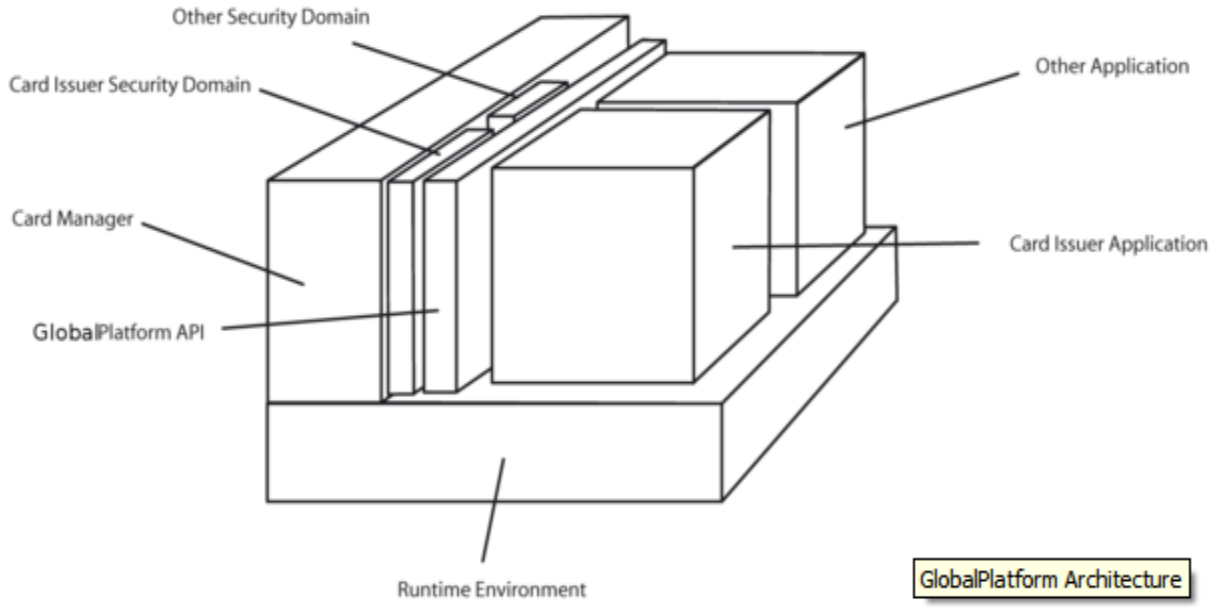


Figura 4: Arquitectura de las tarjetas según *GlobalPlatform*. (GlobalPlatform, 2003)

1.8.3.4 PKCS

Estándar que se refiere a un grupo importante de normas de criptografía de clave pública, previstos y luego publicados por los laboratorios de *RSA* en California, en cooperación con

desarrolladores del resto del mundo, con la finalidad de apresurar el despliegue de la criptografía de clave pública.

A continuación un resumen del estándar utilizado en el desarrollo de la solución propuesta.

- **PKCS #15**

  Estándar de formato de información de dispositivo criptográfico. Define un estándar que admite a los usuarios de dispositivo criptográficos identificarse con aplicaciones independientemente de la implementación de *PKCS#11* que posean.

  Este estándar comprende dos tipos de dispositivos, las tarjetas con circuito integrado y los *tokens* implementados completamente en software, para las tarjetas con circuito integrado desde enero del 2004 existe un estándar basado en el estándar *PKCS#15*, denominado *ISO/IEC 7816-15*. (RSA Laboratories, 1999)

  En la solución propuesta este estándar se utiliza para conformar la estructura de ficheros que permitirá al middleware almacenar objetos *PKCS#11* (llaves, certificados, etc). El sistema de ficheros definido por el estándar *PKCS#15* requiere que la tarjeta soporte los tipos de ficheros definidos en la norma *ISO/IEC 7816-4*.

  El siguiente diagrama muestra un ejemplo de la estructura de ficheros definida por este estándar, los ficheros que se muestran se explicarán durante el desarrollo de la solución:

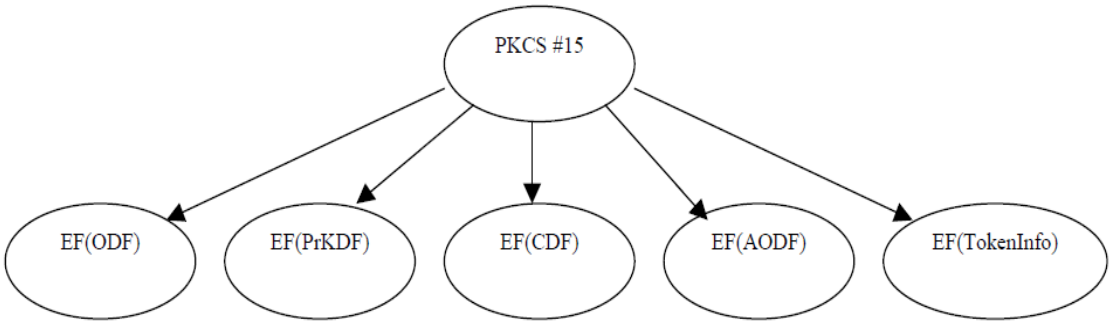


Figura 5: Ejemplo de estructura de ficheros PKCS#15

1.8.3.5 UIT-T X.509

Un certificado es esencialmente una clave pública y un identificador, firmados digitalmente por una autoridad de certificación, y su utilidad es demostrar que una clave pública pertenece a un usuario concreto. El formato de certificados *X.509* es el más común y extendido en la actualidad. El estándar *X.509* sólo define la sintáxis de los certificados, por lo que no está atado a ningún algoritmo en particular y contempla los siguientes campos:

- Versión.

- Número de serie.
- Identificador del algoritmo empleado para la firma digital.
- Nombre del certificador.
- Período de validez.
- Nombre del sujeto.
- Clave pública del sujeto.
- Identificador único del certificador.
- Identificador único del sujeto.
- Extensiones.
- Firma digital de todo lo anterior generada por el certificador.

Estos certificados se estructuran de forma jerárquica, de manera tal que podemos verificar la autenticidad de un certificado comprobando la firma de la autoridad que lo emitió, que a su vez tendrá otro certificado expedido por otra autoridad de rango superior. De esta forma se va subiendo en la jerarquía hasta llegar al nivel más alto, que deberá estar ocupado por un certificador que goce de la confianza de toda la comunidad. Normalmente las claves públicas de los certificadores de mayor nivel se suelen publicar, para que cualquiera pueda verificar si el certificado fue realmente emitido por ellos. (Effing, 2002)

El mecanismo que debe emplearse para conseguir un certificado *X.509* es enviar la clave pública a la autoridad de certificación, después de haberse identificado positivamente frente a ella. Existen autoridades de certificación que, frente a una solicitud, generan un par llaves pública-privada y lo envían al usuario. Es importante destacar que en este caso, si bien se tiene un certificado válido, el certificador correspondiente podrá descifrar todos los mensajes. (Lucena, Mayo de 2003)

1.9 Metodologías de desarrollo

Para el desarrollo de un *software* existen una enorme cantidad de actividades y etapas, una de las etapas fundamentales es la elección de una metodología de *software* adecuada, el impacto de elegir la metodología para un equipo en un determinado proyecto es trascendental para el éxito del producto. La selección depende en gran medida de dos factores, los tipos de proyectos y la forma de trabajo que exista en la empresa.

La bibliografía detalla dos grandes enfoques: metodologías tradicionales y metodologías ágiles. Las tradicionales recalcan el uso profundo de documentación durante todo el ciclo de vida del proyecto, la cual es recomendada para proyectos con grandes equipos de trabajo. Las ágiles

recalcan una vital importancia en la capacidad de respuesta a los cambios y mantener una buena relación con el cliente.

1.9.1 Metodologías Tradicionales

Las metodologías tradicionales ofrecen cierta resistencia a cambios que sean necesarios durante el ciclo de vida del proyecto. En este tipo de metodología el proceso es muy controlado, con mucha documentación, plantillas, normas, etcétera. Se realiza un contrato prefijado y el cliente puede interactuar con el equipo de desarrollo del proyecto. Se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada, además la arquitectura de software es esencial y se expresa mediante modelos.

1.9.1.1 Rational Unified Process (RUP)

Esta metodología constituye un proceso formal, cuyo objetivo fundamental se centra en la producción de *software* con alto nivel de calidad. Satisfacer los requerimientos de los usuarios finales también es premisa para *RUP*, todos estos requisitos deben cumplirse sin violar el cronograma ni el presupuesto propuesto.

RUP es un proceso iterativo e incremental, centrado en la arquitectura y guiado por los casos de uso, utiliza *UML* como lenguaje de notación. Incluye artefactos y roles. (Ivar Jacobson)

1.9.2 *Metodologías Ágiles*

Las metodologías ágiles están preparadas para enfrentar cambios durante el ciclo de vida del proyecto, el proceso es menos controlado. El contrato, en caso de existir, es bastante flexible. El cliente es parte del equipo de desarrollo del proyecto. Se hace menos énfasis en la arquitectura del *software* y existen pocos roles y artefactos.

1.9.2.1 Scrum

Está indicado para proyectos en entornos complejos, donde los requisitos sean cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales. El *software* se desarrolla mediante iteraciones conocidas como sprints, el resultado de estas iteraciones se le muestra al cliente. Se realizan además diferentes reuniones, por ejemplo, la reunión diaria de 15 minutos. (proyectos Ágiles)

1.9.2.2 Extreme Programming (XP)

Es la más sobresaliente dentro de las metodologías ágiles. Está diseñada para que el cliente reciba el *software* que necesita en el tiempo que lo necesita. Su principal objetivo es satisfacer las necesidades de los usuarios y es por ello que se considera un éxito. El trabajo es desarrollado en equipo, preocupándose en todo momento del aprendizaje de los

desarrolladores y estableciendo un buen clima de trabajo. *Extreme Programming* le confiere las facultades a los desarrolladores para que puedan responder con confianza a las necesidades cambiantes de los clientes. Es la metodología más apropiada para entornos volátiles. Este tipo de programación es la adecuada para los proyectos con requisitos imprecisos, muy cambiantes y con un riesgo técnico excesivo. (Joskowicz, 2008)

1.9.2.2.1 Fundamentación de XP como metodología a utilizar

Los aspectos esenciales que incidieron para la elección de esta metodología son:

- Necesidad de obtener un producto a corto plazo.
- Cantidad de miembros del equipo de desarrollo (dos personas).
- Presencia del cliente en el grupo de desarrollo.
- Asumir una metodología robusta implica una cantidad excesiva de roles y gran volumen de información generada durante todo el ciclo de vida del proyecto. Es por ello que se hace difícil el desarrollo de esta metodología por un equipo pequeño.

1.10 UML (Unified Modeling Language)

De los lenguajes de modelado de sistemas de software el más conocido y utilizado actualmente es el Lenguaje Unificado de Modelado. Es un lenguaje de modelado visual que se usa para visualizar, especificar, construir y documentar un sistema de software. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. Está concebido para ser utilizado en herramientas interactivas de modelado visual que tengan generadores de código así como generadores de informes. La especificación de *UML* no define un proceso estándar pero está pensado para ser útil en un proceso de desarrollo iterativo. (Ivar Jacobson, 2000)

1.11 Altova UModel

Herramienta utilizada para crear e interpretar diseños de *software* mediante la potencia del estándar *UML 2.1*. Dibuja el diseño de la aplicación y puede generar código para Java o C# a partir de diagramas, además permite realizar ingeniería inversa de programas existentes a diagramas *UML* claros y precisos para abarcar rápidamente su arquitectura. Utilizando *UModel* se puede corregir el código generado o los modelos y completar la ronda produciendo automáticamente diagramas nuevos o regenerando el código. Se utilizó esta herramienta de modelado por todas las características anteriormente expuestas y por ser la herramienta definida por el departamento de tarjetas inteligentes.

1.12 Gemalto Developer Suite

Developer Suite es un *IDE* (Entorno de Desarrollo Integrado), que proporciona un conjunto de herramientas para crear y depurar *applets Java Card*. Este además facilita el desarrollo de soluciones inalámbricas para los desarrolladores de Java. Brinda un ambiente favorable para el diseño y la implementación de *applets* y posibilita simular las funcionalidades de los *applets* antes de ser instalados en las tarjetas inteligentes. (Gemalto, 2010)

Developer Suite a diferencia del *NETBeans* nos permite desarrollar aplicaciones para la versión de *Java Card* que poseen las tarjetas utilizadas en el departamento de tarjetas inteligentes del *CISED*, *Java Card 2.2.1*, además tiene integrado el *JCardManager* para la simulación de las funcionalidades de los *applet*.

1.13 JCardManager

JCardManager es una solución de *Gemalto* para permitir a cualquier desarrollador de *Java Card* administrar y probar nuevas aplicaciones con *Java Card* y *GlobalPlatform*. Sus herramientas ofrecen la mejor asistencia en el proceso de desarrollo de la aplicación e incluyen todos los elementos necesarios para ejecutar y probar una aplicación. (Gemalto)

1.14 NetBeans

Es un proyecto de código abierto fundado por Sun Microsystems especialmente diseñado para el desarrollo de aplicaciones en Java, pero acepta otros lenguajes de programación. Consta de una gran base de usuarios y una comunidad en constante crecimiento, lo que le ha permitido, al igual que muchos otros sistemas libres, el progreso paulatino de sus prestaciones y la eliminación de *Bugs* que pudiesen existir. NetBeans permite el desarrollo de aplicaciones *Java Card* integrado al *JCDK* con un simulador para probar los *applet*.

Ventajas: La plataforma *NetBeans* permite que las aplicaciones se desarrollen a partir de un conjunto de módulos o componentes de *software*. Un módulo contiene clases de java escritas para interactuar con las *APIs* de *NetBeans* y un archivo especial que lo identifica como módulo. *NetBeans IDE* es fácil de instalar y de uso instantáneo y se ejecuta en varias plataformas incluyendo Windows, Linux, Mac OS X y Solaris. Además del soporte completo para todas las plataformas Java (Java SE, Java EE, Java ME, y JavaFX), *NetBeans IDE 6.5* es además la herramienta ideal para el desarrollo de *software* con PHP, Ajax y Javascript y C/C++. (SUN)

1.15 Eclipse

Eclipse fue desarrollado originalmente por *IBM*. Es ahora desarrollado por la Fundación Eclipse, una organización independiente que fomenta una comunidad de código abierto y un

conjunto de productos complementarios, capacidades y servicios. Es un entorno de desarrollo integrado, de Código abierto y Multiplataforma. Mayoritariamente se utiliza para desarrollar lo que se conoce como "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores, es una potente y completa plataforma de Programación, desarrollo y compilación de elementos tan variados como sitios web, programas en C++ o aplicaciones Java. Además contiene una atractiva interfaz que lo hace fácil y agradable de usar. Eclipse permite el desarrollo de aplicaciones *Java Card* integrado al *JCDK* con un simulador para probar los *applet*. (Eclipse)

1.16 Propuesta y selección de herramientas

Dada la necesidad de implementar una aplicación (*applet)* para tarjetas inteligentes en una *PKI*, se determina que el *applet* se va a realizar utilizando la tecnología *Java Card*, la cual permitirá ejecutar aplicaciones dentro de las tarjetas. El objetivo fundamental de la solución es desarrollar y probar un *applet* que permita realizar las operaciones de firma digital y autenticación, utilizando certificados digitales. Para la implementación de la aplicación se empleará la herramienta *Developer Suite*, de la cual se posee licencia para su uso y es la herramienta definida en el departamento de tarjetas inteligentes para el desarrollo de *applets* con tecnología *Java Card*, además de poseer integrado el simulador *JCardManager*, una herramienta con una interfaz intuitiva que ayudará a agilizar el trabajo permitiendo probar las funcionalidades del *applet* antes de cargarlo en una tarjeta. Para interpretar las necesidades del sistema y diseñar la solución propuesta se utilizará la metodología ágil *XP* por ajustarse a las condiciones del proyecto y el equipo. Se modelará la aplicación utilizando la herramienta *Altova UModel*, definida por el departamento de tarjetas inteligentes, cumpliendo de esta forma con los requisitos que se hacen necesarios para la implementación de la solución: Aplicación para tarjetas inteligentes en una Infraestructura de Clave Pública.

1.17 Conclusiones

- El estudio de las soluciones existentes sobre el uso de tarjetas inteligentes en una Infraestructura de Clave Pública permitió identificar los principales elementos de seguridad y los estándares asociados a este tipo de tecnologías, facilitando su compresión.
- Partiendo del estudio realizado se pudo identificar la metodología y las herramientas más completas y ágiles para el desarrollo de la aplicación que se quiere implementar.

Capítulo 2: Análisis y Diseño del *applet* para tarjetas inteligentes en una Infraestructura de Clave Pública.

2.1 Introducción

En el presente capítulo se modelará la solución propuesta utilizando la metodología de desarrollo de *software XP*. La misma se centra específicamente en la satisfacción del cliente. Debido a las características del *software* que se produce en el Departamento de Tarjetas Inteligentes perteneciente al Centro de Identificación y Seguridad Digital de la Universidad de las Ciencias Informáticas, el mismo ha asumido el uso de metodologías ágiles, dentro de las que se incluye *XP*, para guiar los procesos de desarrollo que se llevan a cabo en esta línea.

El objetivo que se persigue con la elaboración de este capítulo es mostrar la evolución de la solución durante las fases iniciales de Exploración y Planificación, donde se plantean a grandes rasgos las historias de usuario, al mismo tiempo que el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas, se exploran las posibilidades de la arquitectura del sistema; además se establece la prioridad de cada historia de usuario y los programadores realizan una estimación del esfuerzo necesario de cada una de ellas.

2.2 Características de XP

XP es una metodología de desarrollo ligera basada en una serie de valores, principios y una docena de prácticas que propician un aumento en la productividad a la hora de generar *software.*(Wells., 2011)

En el ciclo de vida de un proyecto los cambios siempre van a aparecer y a veces el equipo de desarrollo no está preparado para enfrentarlos. *XP* desarrolla los siguientes valores para garantizar el éxito de un proyecto de desarrollo de *software:*

- Comunicación
- Coraje
- Simplicidad
- Retroalimentación

La mayoría de las siguientes prácticas no son nuevas, sino que han sido reconocidas por la industria como las mejores prácticas durante años.

- ✓ Planificación Incremental.
- ✓ Pruebas.
- ✓ Programación en parejas.
- ✓ Refactorización.

- ✓ Diseño simple.
- ✓ Propiedad colectiva del código.
- ✓ Integración continua.
- ✓ Cliente en el equipo.
- ✓ Entregas pequeñas.
- ✓ Semanas de 40 horas.
- ✓ Estándares de codificación.
- ✓ Uso de Metáforas.

XP define cuatro variables para proyectos de *software:* coste, tiempo, calidad y ámbito. Además de estas cuatro variables, Beck el creador esta metodología, propone que sólo tres puedan ser establecidas por las fuerzas externas (jefes de proyecto y clientes), mientras que el valor de la cuarta variable debe ser establecido por los programadores en función de las otras tres.

XP como metodología de desarrollo también cuenta con diferentes artefactos, entre los que se encuentran:

- Historias del usuario.
- Tareas de ingeniería.
- Pruebas de aceptación.
- Pruebas unitarias y de integración.
- Plan de entrega.
- Código.

2.2.1 Fases de XP

El ciclo de vida de *XP* se enfatiza en el carácter interactivo e incremental del desarrollo, donde una iteración es un período de una a cuatro semanas, en el cual el cliente selecciona las funcionalidades que desea que se implementen en dicha iteración. Las mismas son relativamente cortas ya que se piensa que entre más rápido se desarrolle y se le entregue al cliente, mayor retroalimentación habrá, lo que representa una mejor calidad del producto a largo plazo. Las fases que intervienen en esta metodología son las siguientes (Joskowicz, 2008):

- Exploración

  En esta fase, los clientes le plantearon al equipo de desarrollo las Historias de Usuario (*HU*) de interés para la primera entrega del producto, se confeccionó la metáfora del sistema ayudando al equipo a entender las relaciones entre los principales componentes del sistema. Al mismo tiempo el equipo de desarrollo se familiarizó con las herramientas,

tecnologías y prácticas que se utilizaron en el proyecto. Se probó la tecnología y se exploraron las posibilidades de la arquitectura del sistema, construyendo un prototipo. (Joskowicz, 2008)

- Planeamiento

  En esta fase el cliente establece la prioridad de cada historia de usuario. Se tomaron acuerdos sobre el contenido de la primera entrega y se determinó un cronograma en conjunto con el cliente. Los artefactos que se generaron fueron el Plan de iteraciones donde los elementos a tener en cuenta durante su elaboración son: historias de usuario no abordadas, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior y el *Plan de entrega,* compuesto por iteraciones de no más de tres semanas. (Joskowicz, 2008)

- Iteraciones

  Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. Todo el trabajo de la iteración se expresó en tareas de programación, cada una de ellas fue asignada a un programador como responsable. Se diseñaron las tarjetas *CRC* (Clase, Responsabilidad, Colaboración). Una tarjeta *CRC* representa un objeto. El nombre de la clase se colocó a modo de título en la tarjeta, las responsabilidades se colocan a la izquierda y las clases que se implican en cada responsabilidad a la derecha, en la misma línea que su requerimiento correspondiente.

  Para cada una de estas iteraciones se confeccionaron las Unidades de prueba y se aprobaron las mismas antes de empezar a codificar la solución, estas ayudaron a los programadores a tener una mejor visión del comportamiento del programa. (Joskowicz, 2008)

- Producción

  La fase de producción necesitó pruebas adicionales y revisiones de rendimiento antes de que el sistema se trasladara al entorno del cliente. Al mismo tiempo, se tomaron decisiones sobre la inclusión de nuevas características a la versión actual, ya que realizar cambios durante esta fase implica costos, entonces se tendría que analizar la factibilidad de estos cambios. (Joskowicz, 2008)

2.3 Modelo de Dominio

En la actualidad en el CISED los procesos relacionados con la gestión de información criptográfica y operaciones de firma digital en las tarjetas inteligentes no están bien identificados, pero con el estudio de las tecnologías y del estado del arte del tema, se

identificaron un conjunto de conceptos que definen correctamente cada eslabón de la solución, determinándose entonces la creación de un modelo de dominio, que evidencia claramente cómo funciona el entorno en el cual está enmarcado el problema.

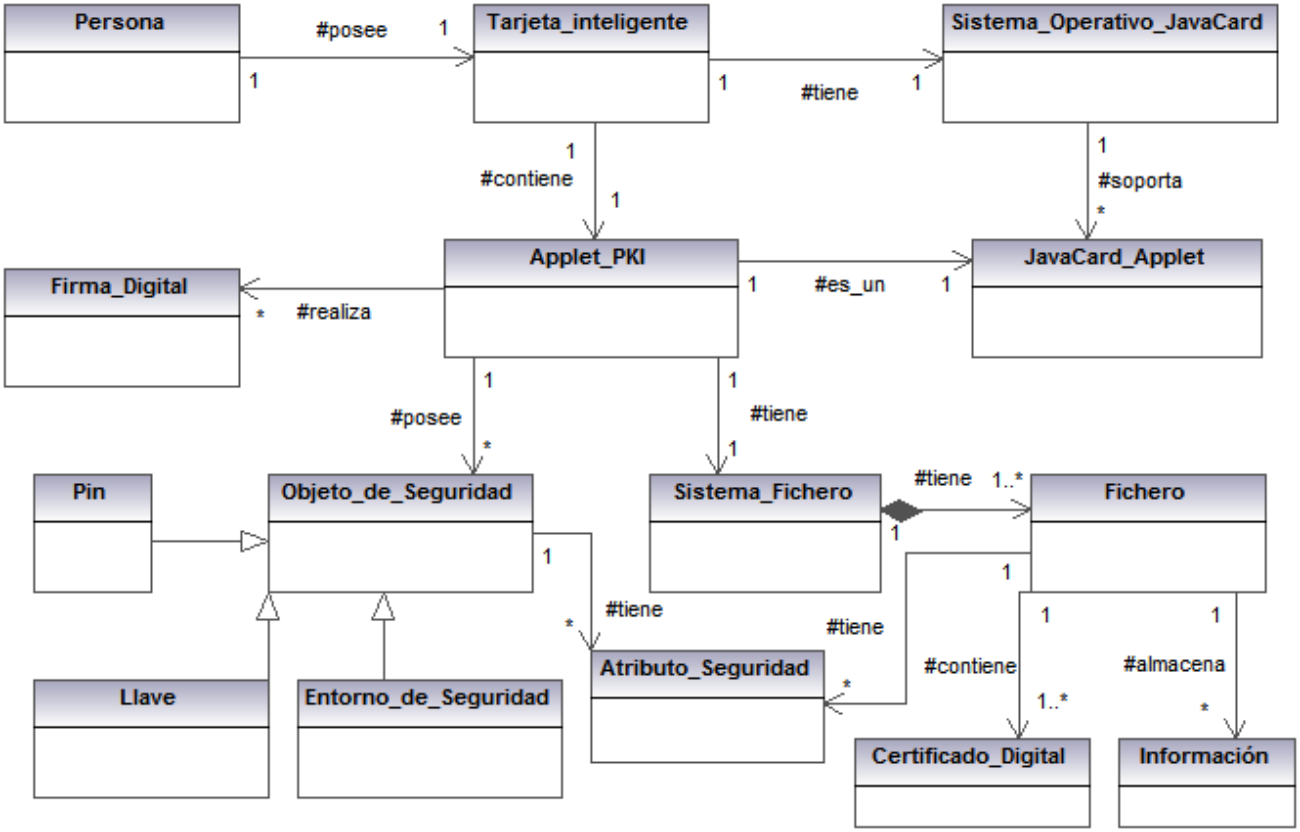


Figura 6: Modelo de dominio

2.3.1 Glosario de términos del modelo de dominio

A continuación los conceptos tratados en el modelo de dominio serán definidos dentro del contexto del problema a resolver.

Persona: Posee y utiliza la tarjeta inteligente.

Tarjeta_Inteligente: Es un dispositivo de plástico similar en tamaño y otras características físicas a las tarjetas de crédito, presentan un circuito integrado, el mismo puede ser de solo memoria o contener un microprocesador (*CPU*) con un sistema operativo.

Sistema_Operativo_JavaCard: La tecnología *Java Card* combina parte del lenguaje de programación Java con un entorno de ejecución optimizado para tarjetas inteligentes y similares. El objetivo de la tecnología *Java Card* es llevar los beneficios del desarrollo de *software* en Java al mundo de las tarjetas inteligentes.

JavaCard_Applet: Los *applets* son las aplicaciones que corren embebidas en una tarjeta *Java Card*. Dichas aplicaciones interactúan en todo momento con el *JCRE* utilizando los servicios que éste brinda.

Applet_PKI: Aplicación *Java Card* para tarjetas inteligentes, que cumple con los principales estándares internacionales asociados, y con la capacidad de realizar operaciones criptográficas y de gestión de certificados digitales, en una Infraestructura de Clave Pública.

Firma_Digital: Esquema matemático que es utilizado para garantizar la autenticidad de documentos electrónicos o mensajes transmitidos electrónicamente.

Llave: Es una secuencia de caracteres, que puede contener letras, dígitos y símbolos, y que es convertida en un número, utilizada por los algoritmos de cifrado para codificar y decodificar mensajes.

Sistema_Fichero: Sistema de Archivos Elementales (*EF*) y Archivos Dedicados (*DF*) definido en el estándar *ISO/IEC 7816-4*. Incluye además el procesamiento de los principales comandos definidos por este estándar, para la gestión de archivos y objetos de seguridad como llaves secretas, llaves privadas y públicas, *PINs* y Entornos de Seguridad, y el soporte para mecanismos de autenticación mutua y de canal seguro con el *middleware* en el terminal.

Fichero: Los Ficheros pueden ser elementales (*EF*) o dedicados (*DF*), los primeros almacenan información sobre los objetos de seguridad del *applet* y los segundos pueden guardar otros ficheros u objetos del sistema.

Certificado_Digital: Es la combinación de la llave pública que ha sido firmada por la autoridad de certificación, la firma digital de esa llave pública y algunos parámetros adicionales. El formato del certificado digital está regido por el estándar *UIT-T X.509*.

Atributo_Seguridad: Definen para cualquier objeto del sistema, que acciones están permitidas sobre él y bajo qué condiciones.

Información: Datos de los objetos de seguridad que se almacenan en el *Applet_PKI*.

2.4 Historias de Usuario

Para especificar los requisitos del *software* desde el punto de vista del cliente en *XP* son utilizadas las historias de usuario. Estas tienen el mismo propósito que los casos de uso, aunque no sean iguales. Son escritas por los propios clientes, tal y como ven ellos las necesidades del sistema, por lo que serán descripciones cortas y en lenguaje de usuario, sin vocabulario técnico. A continuación se presentan las historias de usuario identificadas para desarrollar la solución:

Historia de Usuario	
Número: HU_1	**Nombre de Historia de Usuario**: Crear fichero
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Desarrollador	**Iteración asignada**: 1
Prioridad en negocio: Alta	**Puntos estimados:** 1.0
Riesgo en desarrollo: Alto	**Puntos reales:** 1.1
Descripción: El *applet* debe crear los ficheros relacionados con la estructura PKCS#15 u otros ficheros elementales o dedicados que el usuario desee.	
Observaciones: El *applet* responderá con un error en caso que: • No exista espacio disponible para crear el fichero. • Exista un fichero con el mismo identificador en el *DF* seleccionado.	

Tabla 3: HU Crear fichero

Historia de Usuario	
Número: HU_2	**Nombre de Historia de Usuario**: Crear entorno de seguridad
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Desarrollador	**Iteración asignada**: 1
Prioridad en negocio: Alta	**Puntos estimados:** 0.9
Riesgo en desarrollo: Medio	**Puntos reales:** 1.0
Descripción: El *applet* debe permitir adicionar entornos de seguridad a los cuales estarán asociados los objetos de seguridad.	
Observaciones: El *applet* responderá con un error en caso que: • El entorno de seguridad ya exista. • El *applet* no se encuentre en la fase de Personalización.	

Tabla 4: HU Crear entorno de seguridad

El resto de las historias de usuario se encuentran en los anexos 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12.

2.5 Requerimientos no funcionales

Un requisito no funcional es una característica requerida del sistema, del proceso de desarrollo, del servicio prestado o de cualquier otro aspecto del desarrollo, que señala una restricción del mismo. Estos constituyen todas las exigencias de cualidades que se imponen al proyecto, ya sean exigencias de usar un cierto lenguaje de programación o una plataforma tecnológica específica.

- ✓ **Requisitos de *hardware***

- Tarjeta Inteligente (*Smart Card*).
- Mínimo 26 kb de memoria libre en *EEPROM*.
- Chip con procesador criptográfico**.**

✓ **Requisitos de *software***

Estos requisitos fueron definidos por el departamento de Tarjetas Inteligentes del *CISED*, puesto que es la tecnología posee el mismo para el desarrollo de aplicaciones para tarjetas inteligentes.

- Sistema operativo *Java Card 2.2.1*
- *Open Platform* 2.0.1

✓ **Usabilidad**

- Podrá hacer uso del sistema todo aquel que posea una tarjeta con los requisitos descritos anteriormente, posea el middleware y la documentación básica del *applet.*

✓ **Rendimiento**

- El *Applet* debe ser capaz de realizar sus operaciones de manera eficiente, garantizando su funcionalidad en un corto intervalo de tiempo.

✓ **Soporte**

- Manual de usuarios. Con la descripción de los comandos *APDU*, los estados internos del *applet* y la estructura de los archivos para la información criptográfica.
- Sistema de ayuda.

✓ **Seguridad**

- Si se ingresa el *PIN* de forma incorrecta un número veces definido en la etapa de creación del *PIN* el *applet* queda en estado bloqueado hasta que se presente el *PUK* requerido para desbloquearlo.
- Los usuarios deberán estar autenticados a través de un *PIN,* antes de realizar cualquier operación.
- Solo se tendrá acceso a un objeto del *applet*, por ejemplo una llave, si se han cumplido todas las condiciones de seguridad.

✓ **Disponibilidad**

- El sistema podrá ser usado en cualquier momento por los usuarios autorizados, mientras el *applet* no esté en estado bloqueado.

2.6 Metáfora

En una Infraestructura de Clave Pública el usuario final posee certificados de confianza (autoridades certificadoras), certificados personales y llaves privadas asociadas a estos últimos. Con un dispositivo criptográfico se pueden generar claves públicas y privadas para luego solicitar a una autoridad certificadora, en la que se confíe, la firma de un certificado digital y garantizar la autenticidad de la información que se envía, pero el nivel de seguridad que proporciona este procedimiento depende de la seguridad del dispositivo criptográfico, del medio por el que viaja la llave en caso de que requiera ser transportada y de la seguridad del lugar donde se almacena la llave privada generada. La aplicación garantiza la seguridad de las llaves privadas, proporcionando un ambiente seguro dentro de la tarjeta para la generación de estas. La solución propuesta cumple con varios estándares para garantizar la protección de su información. Siguiendo los estándares *PKCS#15* e *ISO/IEC 7816-4* se conformó una estructura de ficheros que permite la interoperabilidad del *applet* con otros dispositivos criptográficos. Para la protección de los ficheros y objetos almacenados en la tarjeta se pueden crear diferentes objetos de seguridad como son Entornos de Seguridad y *PIN*. El *applet* permite que se realice la firma digital de cualquier tipo de información únicamente dentro de la tarjeta, puesto que como medida de seguridad no se proveen mecanismos de lectura para las llaves privadas ni siquiera presentando un *PIN*, solo las llaves públicas y los certificados digitales pueden ser leídos de la tarjeta para realizar una autenticación o solicitar la firma de un certificado digital. En sentido general *la Aplicación para tarjetas inteligentes en una PKI (uSafeApp)* puede ser utilizada por cualquier cliente que requiera incrementar la seguridad de una *PKI*.

2.7 Arquitectura

La arquitectura definida es una mezcla de los estilos arquitectónicos n-capas y componentes. Está compuesta por varias capas: Aplicación, Gestión de archivos y comandos y *Java Card SDK.*

- La capa Aplicación se encarga de realizar el proceso de autenticación mutua de *GlobalPlatform* y la entrega de los comandos al intérprete de comandos, esta capa contiene la aplicación ***uSafeApp***.
- La capa de Gestión de archivos y comandos se encarga de procesar los comandos *ISO/IEC 7816* enviados a la tarjeta y el manejo de los archivos almacenados en la misma. Esta capa contiene el paquete Sistema de Ficheros *ISO/IEC 7816* encargado

de procesar los comandos y el Paquete *PKCS#15* encargado de crear la estructura de ficheros del *applet* utilizando el Sistema de Ficheros *ISO/IEC 7816*.

- La capa *Java Card* SDK contiene las APIs de *Java Card* 2.2.1 y Open Platform 2.0.1 las cuales poseen funcionalidades necesarias para agilizar el desarrollo de la aplicación.

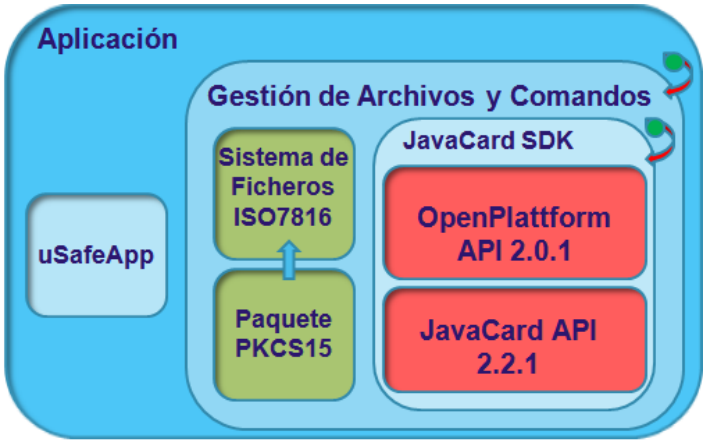


Figura 7: Arquitectura del *applet* uSafeApp

2.8 Plan de entregas

Para elaborar el plan estimado de entrega se utilizaron las historias de usuario. Este plan de entrega se usó para realizar los planes de iteración para cada iteración. En este preciso momento serán tomadas las decisiones técnicas y comerciales por parte de las personas que se desempeñen en estos roles. Con cada historia de usuario previamente evaluada en tiempo de desarrollo ideal, el cliente las agrupó en orden de importancia. De esta forma se pudo trazar el plan de entregas en función de estos dos parámetros: tiempo de desarrollo ideal y grado de importancia para el cliente. Este artefacto se elabora con la intención de fijar qué período de tiempo puede tardar la implementación de cada una de las historias, definiéndose las fechas en que serán liberadas las versiones funcionales del producto. (Ver anexo 2)

2.9 Estructura de ficheros del Applet

El applet uSafeApp posee una estructura de ficheros que cumple con el estándar PKCS#15, la misma es la que se muestra a continuación.

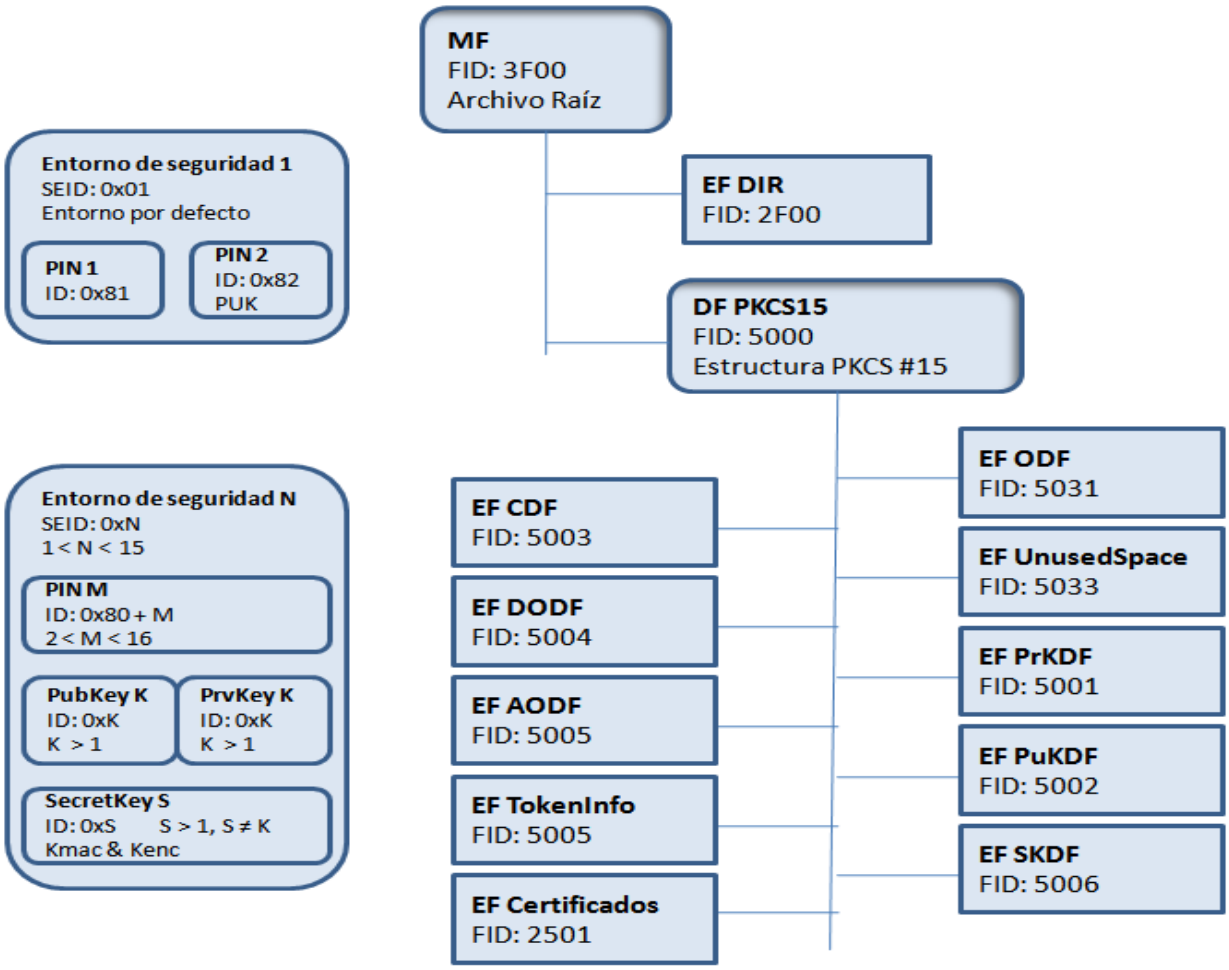


Figura 8: Estructura de ficheros del *applet*

2.9.1 Descripción de los ficheros y objetos de seguridad

MF: Archivo raíz de la estructura de ficheros del *applet*.

EF DIR: Fichero elemental que contiene referencias a las aplicaciones PKCS #15 que existen en la tarjeta.

DF PKCS15: Fichero dedicado que contiene los ficheros elementales definidos por el estándar PKCS #15.

EF ODF: Fichero elemental que contiene punteros hacia los demás ficheros elementales que contiene la estructura PKCS #15.

EF UnusedSpace: Fichero elemental que contiene referencias al espacio sin utilizar de los ficheros que han sido creados.

EF PrKDF: Fichero elemental que contiene atributos generales de las llaves privadas que se almacenan en la tarjeta como son descripción, propósito, identificador, la referencia al valor de la llave, entre otros.

EF PuKDF: Fichero elemental que contiene atributos generales de las llaves públicas que se almacenan en la tarjeta como son descripción, identificador, la referencia al valor de la llave, entre otros.

EF SKDF: Fichero elemental que contiene atributos generales de las llaves secretas que se almacenan en la tarjeta como son descripción, identificador, la referencia al valor de la llave, entre otros.

EF CDF: Fichero elemental que contiene atributos generales de los certificados que se almacenan en la tarjeta como son descripción, identificador, la referencia al valor del certificado, entre otros.

EF DODF: Fichero elemental que contiene atributos generales de los objetos de datos que se almacenan en la tarjeta, diferentes de llaves, certificados y *PIN*.

EF AODF: Fichero elemental que contiene atributos generales de los objetos de autenticación.

EF TokenInfo: Fichero elemental que contiene información sobre la tarjeta como son el número de serie de la tarjeta, los tipos de ficheros soportados, entre otros.

EF Certificados: Fichero elemental que contiene los certificados almacenados en la tarjeta.

Entorno de Seguridad 1: Objeto de datos que se crea por defecto junto con la estructura de ficheros del *Applet*, el mismo tiene asociado lo *PIN* 1 y el *PIN* 2 o *PUK* que es utilizado para desbloquear el *PIN* 1.El *Applet* uSafeApp permite crear un número máximo de 14 entornos de seguridad donde cada uno puede tener varias plantillas de referencia de control (*CRT* por sus siglas en inglés) las cuales poseen objetos de datos de referencia de control (*CRDOs* por sus siglas en inglés) que referencian los objetos que se deben o pueden verificar en caso de una operación sobre un objeto protegido por un entorno de seguridad.

***PIN*:** Objeto de datos utilizado para la autenticación del usuario con la tarjeta.

SecretKey: Objeto de datos que contiene llaves simétricas utilizadas para la autenticación mutua entre el terminal y la tarjeta.

PubKey: Objeto de datos que contiene una llave pública RSA utilizada en operaciones criptográficas como son: la verificación de firma digital, el cifrado de datos y autenticación.

PrbKey: Objeto de datos que contiene una llave privada RSA utilizada en operaciones criptográficas como son: la firma digital, el cifrado de datos y autenticación.

2.10 Diagrama de clases del *Applet*

El diagrama de clases del *applet* uSafeApp se dividió en dos fragmentos y se muestra en las figuras 12 y 13 (Ver Anexos 11 y 12), en el mismo se puede observar en la clase ISO7816FileSystem el uso del patrón Instancia Única (*Singleton*, en inglés), éste se utiliza con el fin de restringir la creación de objetos de esta clase a una sola instancia.

2.10.1 Descripción de las clases

Nombre: ISO7816File	
Tipo de clase: Entidad	
Atributo	**Tipo**
_Parent	ISO7816File
_FCI	Byte[]
_ISO7816FileCollection	List<ISO7816File>
_Data	Byte[]
_ISO7816FileType	boolean
Para cada responsabilidad:	
Nombre:	ISO7816File(ISO7816File parent, byte[] fciData , short lengthFCI, boolean type)
Descripción:	Construye un objeto de la clase ISO7816File
Nombre:	Init(ISO7816File parent,byte[] fciData,boolean type)
Descripción:	Inicializa los atributos de la clase dependiendo del tipo de fichero invocado por la variable type (DF si es verdadero y EF si es falso).
Nombre:	getISO7816FileType()
Descripción:	Devuelve el tipo de fichero(EF o DF)
Nombre:	getParent()
Descripción:	Obtiene el fichero padre del objeto
Nombre:	setParent(ISO7816File parent)
Descripción:	Establece el Fichero padre del objeto
Nombre:	getFCI()
Descripción:	Obtiene la información de control del fichero
Nombre:	setFCI(byte[] _fci)
Descripción:	Establece la información de control del fichero
Nombre:	AddISO7816(ISO7816File file)
Descripción:	Agrega un fichero a la lista _ISO7816FileCollection en el caso de los DF
Nombre:	getISO7816File(byte [] bufferData, short offsetFileID, short lenFileID)
Descripción:	Obtiene de la lista _ISO7816FileCollection el fichero cuyo identificador se pasa por parámetros devolviendo null en caso de no encontrarlo
Nombre:	getData()
Descripción:	Obtiene los datos del fichero elemental almacenados en el atributo _Data
Nombre:	setData(byte[] data, short offsetData, short lengthData)
Descripción:	Establece los datos del fichero elemental almacenándolos en el arreglo _Data

Tabla 5: Descripción de la clase ISO7816File

Nombre: uSafeApp	
Tipo de clase: controladora	
Atributo	**Tipo**
_CLA_INITIALIZE_UPDATE	byte
INS_INITIALIZE_UPDATE	byte
CLA_EXTERNAL_AUTHENTICATE	byte
INS_EXTERNAL_AUTHENTICATE	byte

secureChannel	byte
channelOpened	boolean
authenticationDone	boolean
Enciphered	boolean
SW_INCORRECT_DATA_PARAMETERS	short
cantRSAkeys	short
Cantsymetrickeys	short
Cantcertif	short
_PKCS15F_ID	Byte[]
_FileSystem	ISO7816FileSystem
_APDUInterpreter	ISO7816APDUInterpreter
Para cada responsabilidad:	
Nombre:	uSafeApp(byte[] buffer, short offset, byte length)
Descripción:	Crea una instancia del *applet*
Nombre:	install(byte[] bArray, short bOffset, byte bLength)
Descripción:	Instala el *applet* llamando al constructor de la clase para crear la instancia del *applet*
Nombre:	select()
Descripción:	Llamado por el JCRE cuando se selecciona el applet actual.
Nombre:	deselect()
Descripción:	Se llama cuando se deselecciona el applet
Nombre:	process(APDU apdu)
Descripción:	Se encarga de procesar los comandos APDU enviados a la tarjeta entregándoselos al intérprete de comandos
Nombre:	verifyKeyContainerCreate(APDU apdu)
Descripción:	Verifica si los comandos PUT Data enviados a la tarjeta son para crear algún tipo de llave
Nombre:	init_update(APDU apdu)
Descripción:	Inicia una autenticación mutua de GlobalPlatform
Nombre:	external_authenticate(APDU apdu)
Descripción:	Autentica al terminal con la tarjeta
Nombre:	reset_security()
Descripción:	Cierra un canal seguro GlobalPlatform si existe
Nombre:	PKCS15init(short cantcertif)
Descripción:	Inicializa la estructura de ficheros PKCS#15 del *applet*

Tabla 6: Descripción de la clase uSafeApp

El resto de las descripciones de las clases se encuentran en los anexos 18, 19, 20, 21, 22, 23, 24, 25.

2.11 Conclusiones

- Este capítulo permitió determinar las bases del por qué la elección de un modelo de dominio, el cual agrupa todos los conceptos asociados a la solución, así como las relaciones entre estos conceptos, que son determinados luego del estudio de las herramientas, tecnologías y elementos tangibles asociados al dominio de la solución.
- De la definición de las historias de usuario que caracterizaran la solución y los requerimientos no funcionales que brindarán las cualidades que se deben tener en cuenta para desarrollar la solución adecuada, se obtuvo una visión para definir la arquitectura, la estructura de ficheros según el estándar PKCS15 y el diagrama de clases con el que contará el sistema a desarrollar.

Capítulo 3: Implementación y diseño de pruebas

3.1 Introducción

En el presente capítulo se mostrará una representación de cómo fue implementado el *applet* uSafeApp, enfocándose en el diagrama de estados de la solución desarrollada. También se encuentra un grupo de comandos con sus estructuras que fueron utilizados para el desarrollo de la aplicación. Además de que se validará y verificará el cumplimiento de los requerimientos estipulados, mediante la aplicación de métodos de pruebas que garanticen la calidad del sistema.

3.2 Diagrama de estados del *applet*

El siguiente diagrama muestra los estados por los que pasa el *applet* durante su ciclo de vida, los estados marcados como persistentes (*Persistent State*) son los estados a los cuales regresa el *applet* en caso de que la tarjeta sea retirada del lector o el *applet* deje de estar seleccionado, regresando siempre al último estado persistente alcanzado.

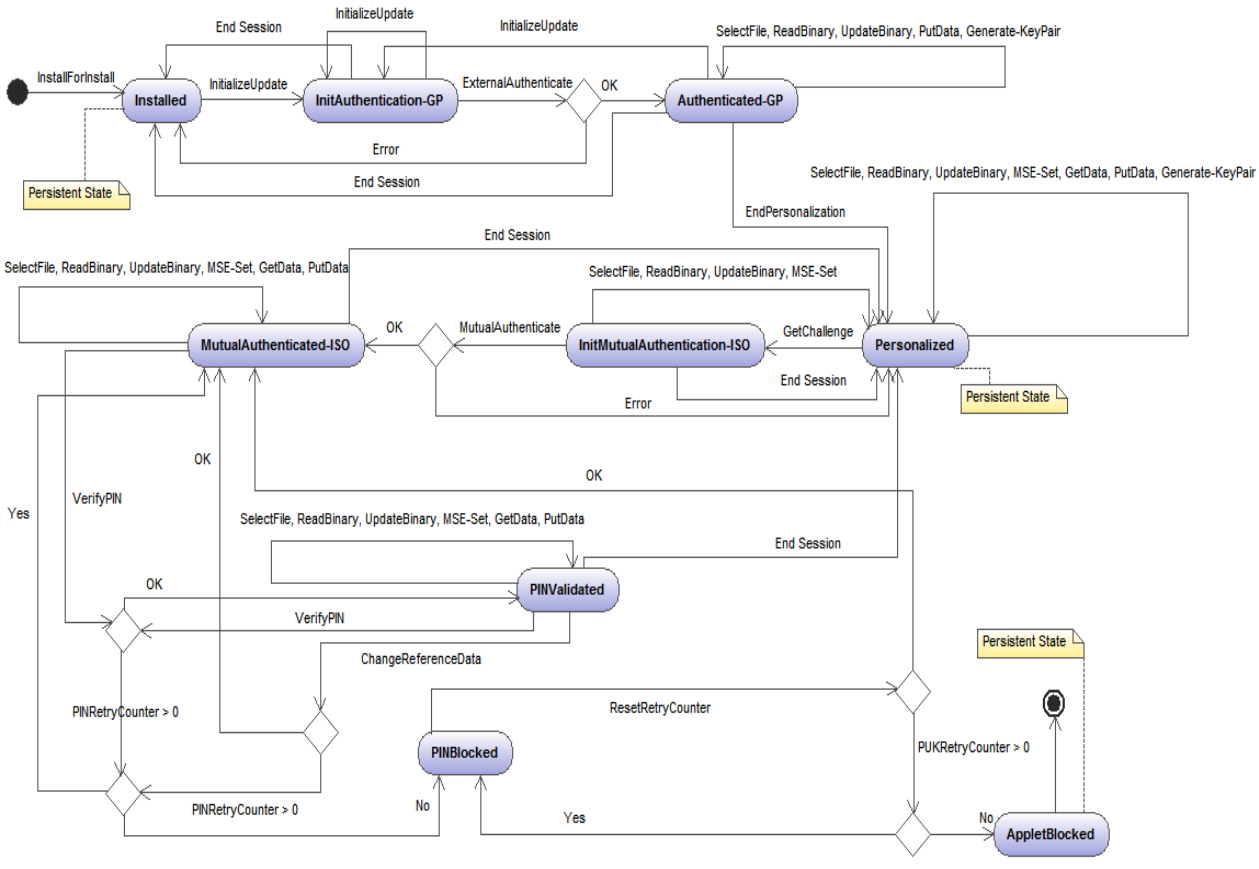


Figura 9: Diagrama de estados del applet

3.3 Comandos aceptados por el *applet*

El *applet* uSafeApp soporta los comandos que se muestran a continuación, la mayoría de estos comandos están definidos en el estándar ISO/IEC 7816-4, los mismos fueron divididos en dos etapas, la etapa de personalización y la etapa de aplicación, siendo algunos comandos válidos para ambas etapas. La etapa de personalización es en la que se guardan las llaves, *PINs* y entornos de seguridad que serán utilizados en las operaciones definidas por la tarjeta, esta etapa termina al ser enviado el comando END PERSONALIZATION. La etapa de aplicación comienza cuando el *applet* ha sido personalizado.

Comando	**Descripción**	**Etapa**	
		Personalización	**Aplicación**
CHANGE REFERENCE DATA	Comando que se utiliza para reemplazar el valor del PIN por un nuevo valor.		(X)
CREATE FILE	Comando que se utiliza para crear un EF o un DF.	(X)	(X)
END PERSONALIZATION	Comando es utilizado para finalizar el proceso de personalización y almacena información de este proceso.	(X)	
Generate PUBLIC KEY Pair	Comando para inicializar o actualizar un par de llaves existentes.	(X)	
GET CHALLENGE	Comando que genera un número aleatorio para ser usado por los comandos MUTUAL AUTHENTICATE, este comando inicia el proceso de autenticación mutua.		(X)
GET DATA	Comando que recupera algunos conjuntos de datos tales como: los datos de personalización del *applet* y los datos de un entorno de seguridad y las llaves públicas.		(X)
MUTUAL AUTHENTICATE	Comando definido por el estándar ISO/IEC 7816-4, que permite a la tarjeta y el terminal autenticarse el uno con el		(X)

	otro, verificando la presencia de las dos llaves secretas K.ENC y K.MAC, este comando es válido solo en la fase de aplicación, en la fase de personalización existe el comando EXTERNAL AUTHENTICATE para el mismo propósito.		
EXTERNAL AUTHENTICATE	Comando definido por el estándar *GlobalPlatform*, que permite a la tarjeta y el terminal autenticarse el uno con el otro.	(X)	
MSE- Set	Comando que actualiza un CRT en el Entorno de Seguridad actual.	(X)	(X)
PUT DATA	Comando que se utiliza para actualizar el valor de objetos de datos durante la etapa de aplicación.	(X)	(X)
READ BINARY	Comando que se utiliza para leer la información contenida en un EF.	(X)	(X)
RESET RETRY COUNTER	Comando que se utiliza para desbloquear un *PIN.*		(X)
SELECT FILE	Comando que se utiliza para seleccionar un archivo especificado ya sea un DF o un EF.	(X)	(X)
UPDATE BINARY	Comando que se utiliza para escribir información dentro de un EF.	(X)	(X)
VERIFY	Comando que se utiliza para autenticar un usuario, realizando la verificación del *PIN* introducido.		(X)
PSO-HASH	Comando que se utiliza para enviar un hash a la tarjeta para ser firmado posteriormente.		(X)
PSO-Compute Digital	Comando que se utiliza para firmar un		(X)

Signature	hash que se ha enviado previamente a la tarjeta.		

Tabla 7: Lista de comandos del *applet*

3.3.1 Etapa de Personalización

3.3.1.1 COMANDO END PERSONALIZATION

Este comando es utilizado al finalizar el proceso de personalización. Actualiza el estado del ciclo de vida del *applet* a Personalizado.

El formato del comando es el siguiente:

CLA	INS	P1	P2	Lc	Data
CLA	E4h	00h	00h	0Ah	Datos de personalización

Tabla 8: Estructura del comando END PERSONALIZATION

Donde

CLA: 00h Transmite normalmente

0Ch Transmite con mensajería segura

Data: contiene los datos de personalización que son almacenados en un formato TLV como se muestra:

Etiqueta	Longitud	Valor	Descripción
DF32h	Eh	'AAAAMMDDhhmmss'	Fecha de personalización del *applet* codificada en ASCI.

Tabla 9 Ejemplo de data: END PERSONALIZATION

Respuesta:

Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado sin error.
69h	85h	Condiciones de uso no satisfechas.

Tabla 10: Posibles valores para SW1 y SW2

3.3.1.2 COMANDO READ BINARY

Este comando lee una parte o todo de un EF.

Si el EF que se quiere leer contiene datos protegidos por un atributo de seguridad, entonces este atributo debe ser introducido para poder ejecutar el comando.

El EF debe haber sido seleccionado previamente usando el comando SELECT FILE. Los datos son leídos directamente desde el EF actual. En este caso el offset es especificado sobre P1 y P2.

El formato del comando es el siguiente:

CLA	INS	P1	P2	Le
CLA	B0h	P1	P2	Le

Tabla 11: Descripción del comando READ BINARY

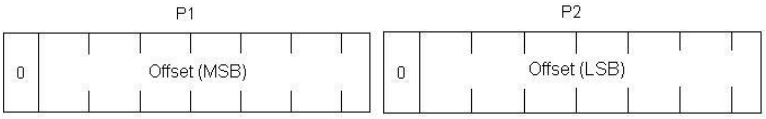
Donde:

CLA: 00h Transmite normalmente

04h Transmite con mensajería segura

P1, P2: son los bytes de los parámetros

Para leer del EF que se encuentra seleccionado, P1 y P2 especifican el *offset,* en bytes, desde el comienzo del archivo al primer byte a ser leído. P1 y P2 tienen el siguiente formato:



Le: indica el número de bytes a leer. Si está vacío el campo o es cero, entonces el comando lee hasta el fin del archivo, un máximo de 255 bytes cuando no se usa mensajería segura.

Respuesta:

En el campo Data vienen los datos leídos desde la tarjeta, seguidos por los códigos de estado SW1 y SW2. Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado sin error.
67h	00h	Longitud incorrecta de Lc (si usa mensajería segura).
69h	82h	Condición de seguridad no satisfecha, por ejemplo: • Los atributos de seguridad no han sido satisfechos. • Error durante la mensajería segura.
69h	86h	Comando no permitido (No EF actual)
6Ah	80h	Incorrectos Lpv, Lcg o Le cuando usa mensajería segura.
6Ah	81h	Función no soportada: SM en modo ENC/MAC especificado por la respuesta.
6Ah	86h	Incorrectos P1 y P2 (se chequea que el offset es dentro del EF).

Tabla 12 Ejemplo de data: READ BINARY

3.3.1.3 COMANDO UPDATE BINARY

Este comando actualiza todo o una parte de un EF. El EF debe haber sido seleccionado previamente usando el comando SELECT FILE. Los datos son leídos directamente desde el EF actual. En el offset es especificado sobre P1 y P2.

El comando actualiza el número de bytes especificados en Lc desde el *offset*.

Si el EF que se quiere leer contiene datos protegidos por un atributo de seguridad, entonces este atributo debe ser introducido para poder ejecutar el comando.

El formato del comando es el siguiente:

CLA	INS	P1	P2	Lc	Data
CLA	D6h	P1	P2	Lc	Data

Tabla 13: Descripción del comando UPDATE BINARY

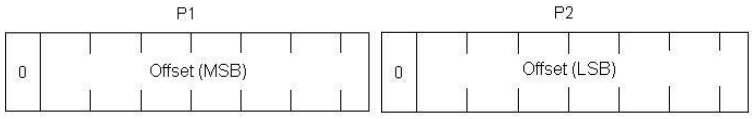
Donde:

CLA: 00h Transmite normalmente

04h Transmite con mensajería segura

P1, P2: son los bytes de los parámetros

Para actualizar el EF que se encuentra seleccionado, P1 y P2 especifican el desplazamiento (offset*),* en bytes, desde el comienzo del archivo al primer byte a ser leído. P1 y P2 tienen el siguiente formato:



Lc: es la longitud del campo *Data* hasta un máximo de 248 bytes.

Data: contiene el valor de los datos que van a ser actualizados en la tarjeta.

Respuesta:

En el campo Data vienen los datos leídos desde la tarjeta, seguidos por los códigos de estado SW1 y SW2. Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado sin error.
62h	82h	Fin del archivo alcanzado (offset +Le es más grande que el fin del archivo).
67h	00h	Longitud incorrecta de Lc (si usa mensajería segura).
69h	82h	Condición de seguridad no satisfecha, por ejemplo: • Los atributos de seguridad no han sido satisfechos. • Error durante la mensajería segura.

69h	86h	Comando no permitido (No EF actual).
6Ah	80h	Incorrectos Lpv, Lcg o Le cuando usa mensajería segura.
6Ah	86h	Incorrectos P1 y P2 (se chequea que el offset es dentro del EF).

Tabla 14 Ejemplo de data: UPDATE BINARY

El resto de los comandos de la etapa de personalización se encuentran en los anexos 26, 27 y 28.

3.3.2 Etapa de Aplicación

3.3.2.1 COMANDO CHANGE REFERENCE DATA

Este comando reemplaza el valor del *PIN* de referencia por un nuevo valor. El valor actual del *PIN* debe ser presentado como parte de este comando.

Si el comando CHANGE REFERENCE DATA tiene éxito, ocurren las siguientes acciones:

- El contador de intentos del *PIN* de referencia es actualizado con el número de intentos
- La bandera de validación del *PIN* de referencia es actualizada a falso (porque el nuevo valor no ha sido aún verificado).
- El valor del *PIN* de referencia es actualizado con el nuevo valor enviado en el comando.
- La bandera de cambio del *PIN* de referencia es actualizada a verdadero.
- El contador de uso del *PIN* de referencia no cambia.

Por otra parte, si el comando CHANGE REFERENCE DATA falla, ocurren las siguientes acciones:

- El contador de intentos del *PIN* de referencia es decrementado en 1.
- La bandera de validación del *PIN* de referencia es actualizado a falso.

El formato del comando es el siguiente:

CLA	INS	P1	P2	Lc	Data
CLA	24h	00h	P2	10h-20h	*PIN actual* || Nuevo *PIN.*

Tabla 15: Descripción del comando CHANGE REFERENCE DATA

Donde:

CLA: 00h Transmite normalmente.

0Ch Transmite con mensajería segura.

P2: es la referencia al *PIN* de referencia (81h-8Fh).

Data: contiene lo siguiente: *PIN* actual 8-16 bytes.

Valor del nuevo *PIN* 8-16 bytes.

Respuesta:

La tarjeta retorna los datos solicitados por el parámetro de control de referencia P2, seguido por los códigos de estado SW1 y SW2. Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado sin error.
63h	Cxh	*PIN* de referencia no verificado. X intentos restantes para cambiar el *PIN.*
67h	00h	Longitud incorrecta de Lc, por ejemplo: • Lc debe ser menos de 248 bytes cuando usa mensajería segura. • El nuevo valor del *PIN* no es igual al valor existente.
69h	83h	Método de autenticación bloqueado (*PIN* de cambio bloqueado).
69h	84h	Datos de referencia invalidados (Límite de intentos o Contador de uso del *PIN* de referencia o el *PIN* de cambio ha alcanzado cero.
6Ah	86h	Incorrectos parámetros P1 y P2.
6Ah	88h	Datos referenciados no encontrados.
69h	86h	El comando solo puede ser usado en fase de aplicación.

Tabla 16: Posibles valores para SW1 y SW2

3.3.2.2 COMANDO UPDATE BINARY

Este comando es Igual que en la etapa de Personalización (página 70).

3.3.2.3 COMANDO READ BINARY

Este comando es Igual que en la etapa de Personalización (página 66).

3.3.2.4 COMANDORESET RETRY COUNTER

Este comando desbloquea un *PIN*. El *PIN* es bloqueado cuando el número de intentos incorrectos consecutivos permitidos es excedido. Se puede opcionalmente especificar un valor nuevo para el *PIN* de referencia. Los *PINs* están protegidos por un atributo de seguridad (la referencia del *PIN* de desbloqueo en los atributos del *PIN* almacenados en el propio objeto de datos *PIN*). Esta referencia del *PIN* de desbloqueo indica la referencia al *PIN* de desbloqueo. Si es puesto a cero, el *PIN* de referencia nunca puede ser desbloqueado.

Si el comando RESET RETRY COUNTER tiene éxito, ocurren las siguientes acciones:

- El contador de intentos del *PIN* de referencia es actualizado con el límite de intentos.
- La bandera de validación del *PIN* de referencia es actualizada a falso (porque el *PIN* aún no se ha verificado).

- El contador de desbloqueo del *PIN* es decrementado en 1, a menos que el valor sea A5h (no existe límite para el número de veces en que el *PIN* de referencia puede ser desbloqueado)
- El contador de intentos del *PIN* de desbloqueo es actualizado con el límite de intentos del *PIN* de desbloqueo.
- La bandera de validación del *PIN* de desbloqueo es actualizada a falso.
- El contador de uso del *PIN* de desbloqueo es decrementado en 1, a menos que el valor sea FFh (no existe límite para el número de veces en que el *PIN* de desbloqueo puede ser usado) o 00h (ya ha sido usado el máximo de veces permitido).
- Si un nuevo valor para el *PIN* de referencia es especificado, el valor del *PIN* de referencia es actualizado con el valor enviado en el comando.
- Si un nuevo valor para el *PIN* de referencia es especificado, la bandera de cambio del *PIN* de referencia es actualizada a falso.

Por otra parte, si el comando RESET RETRY COUNTER falla, ocurren las siguientes acciones:

- El contador de intentos del *PIN* de desbloqueo es decrementado en 1.
- La bandera de validación del *PIN* de desbloqueo es actualizada a falso.

El contador de intentos y el contador de uso del *PIN* de ambos *PIN*, el de referencia y el de desbloqueo no debe ser cero.

El formato del comando es el siguiente:

CLA	INS	P1	P2	Lc	Data
CLA	2Ch	Modo	P2	Lc	*PIN* de desbloqueo||Nuevo *PIN* (opcional).

Tabla 17: Descripción del comando RESET RETRY COUNTER

Donde:

CLA: 00h Transmite normalmente
0Ch Transmite con mensajería segura

Modo: 00h Resetear el contador de intentos y reemplazar el valor del *PIN* de referencia
01h Resetear el contador de intentos solamente

P2: es la referencia al *PIN* de referencia (81h – 8Fh)

Lc: 10h-20h si existe un valor para reemplazar el *PIN* de referencia (P1=00h)
08h-10h si no existe un valor para reemplazar el *PIN* de referencia (P1=01h)

Data: si P1 = 00h *PIN* de desbloqueo o nuevo *PIN* (ambos *PIN* son codificados en 16 bytes)
si P1 = 01h solamente el *PIN* de desbloqueo (8-16 bytes)

Respuesta:

Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado sin error. No retornados datos FCI.
63h	Cxh	*PIN* de desbloqueo no verificado. X intentos restantes para desbloquear el *PIN.*
67h	00h	Longitud incorrecta de Lc, pues debe ser menos de 248 bytes cuando usa mensajería segura.
69h	82h	Condiciones de seguridad no satisfechas (error durante la mensajería segura o atributos de seguridad para el *PIN* de referencia es nunca provisto).
69h	83h	Método de autenticación bloqueado (el *PIN* de desbloqueo que se necesita para desbloquear el *PIN* de referencia se encuentra bloqueado él mismo).
69h	84h	Datos de referencia invalidados (Contador de desbloqueo del *PIN* para el *PIN* de referencia o contador de intentos del *PIN* o contador de uso del *PIN* para el *PIN* de desbloqueo ha alcanzado cero).
6Ah	86h	Incorrectos parámetros P1 y P2.
6Ah	88h	Datos referenciados no encontrados.

Tabla 18: Posibles valores para SW1 y SW2

El resto de los comandos de la etapa de aplicación se encuentran en los anexos 29, 30, 31, 32, 33, 34, 35 y 36.

3.4 Fase de producción

Para el desarrollo de esta fase el equipo de trabajo se centró en los pedidos del cliente, refinando el diseño y el código continuamente. Para perfeccionar el código fue necesario realizarle al mismo un grupo de pruebas automatizadas que aseguraron la ejecución correcta del sistema en todo el período de desarrollo.

Usualmente cuando se desarrolla, se le realizan a la aplicación pruebas para verificar que el código esté correcto. Dichas pruebas tienen que ser efectuadas varias veces y se ven afectadas por los cambios que se introducen conforme se va desarrollando.

Como forma de evaluar la calidad de la aplicación, se llevó a cabo un proceso de pruebas que validó la implementación de las funcionalidades definidas. Para ello se realizaron pruebas de caja blanca y pruebas de caja negra.

3.4.1 Pruebas de Caja blanca o Estructurales

La puesta en práctica de este método requiere del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones del diseño o el código. Se basa en la comprobación de los caminos lógicos del *software* dado un código específico.

3.4.1.1 Pasos para el diseño de pruebas utilizando el camino básico.

• Obtener el grafo de flujo, a partir del diseño o del código del módulo

• Obtener la complejidad ciclomática del grafo de flujo

• Definir el conjunto básico de caminos independientes

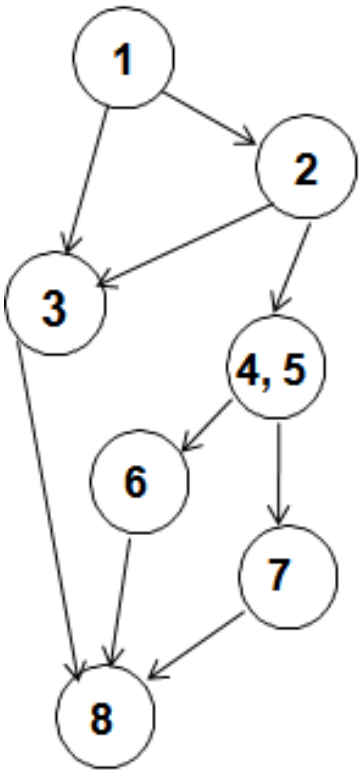
• Determinar los casos de prueba que permitan la ejecución de cada uno delos caminos anteriores.

• Ejecutar cada caso de prueba y comprobar que los resultados son los esperados.

3.4.1.1.1 Casos de prueba

De acuerdo al siguiente segmento de código correspondiente a la historia de usuario "*Establecer Canal Seguro*", se realiza la prueba de caja blanca.

```
Private short processGetChallenge(APDU apdu, short le)
 {
if (!hasAutenticationkeys || SessionManager.MutualAuthentStates[0]==(byte)0x01)(1)(2)
{
        ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);(3)
}
byte[] buffer = apdu.getBuffer ();(4)
le = apdu.setOutgoing ();(4)
if (le != 8)(5)
{
ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);(6)
}
randomData.generateData(_session.getICCrandom(), (short) 0, le);(7)
Util.arrayCopy(_session.getICCrandom(), (short) 0, buffer, bufferOffset, le);(7)
SessionManager.MutualAuthentStates [0]=(byte)0x02;(7)
return le;(7)
}(8)
```



Complejidad ciclomática.

V (G): *Número de regiones del grafo.*

V (G) = A – N + 2

V (G) = P + 1

A: *Número* de aristas del grafo.

N: *Número de n*odos.

P: *Número* de nodos predicados.

A=9

N=7

P=3

V (G) = 4

Caminos:

1, 3, 8

1, 2, 3, 8

1, 2, 4, 5, 6, 8

1, 2, 4, 5, 7, 8

Figura 10: Grafo de flujo del caso de prueba obtener reto.

Casos de prueba para cada camino**.**

Camino: 1, 3, 8.

Entrada: Comando *GetChallange* con **le** igual a 8.

Salida: Excepción especificando que las condiciones de seguridad necesarias no están satisfechas.

Precondiciones: No haber especificado llaves para la autenticación mutua.

Camino: 1, 2, 3, 8.

Entrada: Comando *GetChallange* con **le** igual a 8.

Salida: Excepción especificando que las condiciones de seguridad necesarias no están satisfechas.

Precondiciones: Haber realizado ya una autenticación mutua.

Camino: 1, 2, 4, 5, 6, 8.

Entrada: Comando *GetChallange* con **le** distinto de 8.

Salida: Excepción especificando que la longitud esperada en la respuesta es incorrecta.

Precondiciones: haber especificado llaves para la autenticación mutua y no haber realizado una autenticación mutua.

Camino: 1, 2, 4, 5, 7, 8.

Entrada: Comando Get Challange con **le** igual a 8.

Salida: 8 bytes aleatorios seguido de 90 00 especificando que la ejecución del comando terminó con éxito.

Precondiciones: haber especificado llaves para la autenticación mutua y no haber realizado una autenticación mutua.

El anexo 39 muestra otro caso de prueba de caja blanca.

3.4.2 Pruebas de Caja Negra o funcionales

Las Pruebas de Caja Negra, deben su nombre a los elementos que éstas revisan y las condiciones en que se hace la revisión. Se basan en los requerimientos funcionales del sistema y se llevan a cabo desde el exterior de la aplicación. Este tipo de prueba es importante a la hora de medir el grado de cumplimiento de los requerimientos solicitados por el cliente y se aplican sobre la interfaz de la aplicación observando las respuestas del sistema ante determinadas acciones y los datos de salida para determinados datos de entrada.

Las pruebas de caja negra pretenden encontrar estos tipos de errores:

- Funciones incorrectas o ausentes
- Errores en la interfaz
- Errores en estructuras de datos o en accesos a bases de datos externas
- Errores de rendimiento
- Errores de inicialización y de terminación (Pressman, 2002)

3.4.2.1 Diseño de casos de pruebas de caja negra

La prueba verifica que el elemento que se está probando, cuando se dan las entradas apropiadas produce los resultados esperados. Los datos de prueba se escogerán atendiendo a las especificaciones del problema, sin importar los detalles internos del programa, a fin de verificar que el programa o la aplicación se ejecuten correctamente.

Para cada uno de los casos de prueba se utiliza el simulador *JCardManager*. Éste muestra una ventana en forma de comando *APDU* donde el usuario del sistema introduce los datos correspondientes (CLA, INS, P1, P2, Lc y Data) para ejecutar un comando *APDU*.

3.4.2.1.1 Caso de prueba verificar PIN

Escenario	Descripción	CLA	INS	P1	P2	Le	Data	Respuesta del sistema	Flujo central
SC1	Verificar el *PIN* correctamente.	00	20	00	Identificador del *PIN* correcto.	-	Valor del *PIN correcto.*	El sistema muestra el *APDU* Respuesta **90 00**, que indica que se ejecutó el comando correctamente.	Se escribe en el simulador los datos del comando para verificar el *PIN y se presiona la pestaña Go,* el simulador muestra el mensaje **90 00** indicando que el *PIN* se verificó correctamente.
SC2	Verificar el *PIN* incorrectamente.	00	20	00	Identificador del *PIN* correcto.	-	Valor del *PIN* incorrecto.	El sistema muestra el *APDU* Respuesta **63 CX** donde X es la cantidad de intentos restantes.	Se escribe en el simulador los datos del comando para verificar el *PIN y se presiona la pestaña Go,* el simulador muestra el mensaje **63 CX** indicando que el *PIN* no se verificó correctamente y quedan X intentos restantes.
SC3	Verificar el *PIN* incorrectamente.	00	20	00	Identificador del	-	Valor del *PIN* corre	El sistema muestra el *APDU* Respuesta	Se escribe en el simulador los datos del comando para

					PIN incorrecto.		cto.	**6A 88** indicando que el identificador del *PIN* no existe.	verificar el *PIN y se presiona la pestaña Go,* el simulador muestra el mensaje **6A 88** indicando que el *PIN* no se verificó correctamente.

Tabla 19: Caso de prueba de Caja negra 1

3.4.2.1.2 Caso de prueba Firmar Datos

Escenario	Descripción	CLA	INS	P1	P2	Le	Data	Respuesta del sistema	Flujo central
SC1	Se firman datos correctamente.	00	2A	9A	9E	-		El sistema muestra el *APDU* Respuesta **61 XX**, donde XX es la cantidad de bytes de la firma que quedan por recoger en el buffer de salida del *applet*.	Se envía el comando MSE-Set para seleccionar la llave de firma, se envía el comando PSO-HASH con el hash de los datos que se desea firmar, se introducen los datos del comando PSO-Compute Digital Signature y se presiona la pestaña *Go.* El simulador muestra el mensaje 61 XX.

SC2	Se firman datos incorrectamente.	00	2A	9A	9E	-	Se envía incorrecto.	El sistema muestra el *APDU* Respuesta **67 00** expresando que la longitud de los datos es incorrecta, esta debe ser igual a 0.	Se envía el comando MSE-Set para seleccionar la llave de firma, se envía el comando PSO-HASH con el hash de los datos que se desea firmar, se introducen los datos del comando PSO-Compute Digital Signature y se presiona la pestaña *Go.* El simulador muestra el mensaje **67 00** indicando incorrecta longitud de datos.
SC3		00	2A	9A	9E	-		El sistema muestra el *APDU* Respuesta **69 85** expresando que no se ha enviado ningún hash a la tarjeta o no se	Se envía el comando MSE-Set para seleccionar la llave de firma, se introducen los datos del comando PSO-Compute Digital Signature y se presiona la

								cumplen las condiciones de seguridad para usar la llave privada involucrada.	pestaña *Go.* El simulador muestra el mensaje **69 85** indicando que no se ha enviado ningún has a la tarjeta o no se cumplen las condiciones de seguridad para usar la llave privada involucrada.

Tabla 20: Caso de prueba de Caja Negra 2

3.4.2.2 Resultado de las pruebas

En las pruebas de caja negra realizadas a las funcionalidades que requieren entrada de datos se identificaron 11 casos de prueba en total para las 5 iteraciones con un número variable de escenarios para cada caso. En la primera iteración se realizaron 2 casos de prueba detectándose 5 no conformidades a las cuales se les dio solución. En la segunda iteración se realizaron 2 casos de prueba detectándose 3 no conformidades, las mismas fueron resueltas en su totalidad. En la tercera iteración se realizaron 3 casos de prueba detectándose 4 no conformidades. En la cuarta iteración se realizaron 2 casos de prueba detectándose 2 no conformidades y en la quinta iteración se realizaron 2 casos de prueba detectándose 2 no conformidades. En las cinco iteraciones efectuadas se detectaron un total de 16 no conformidades, las cuales en su mayoría respondían a errores de bajo impacto en el correcto funcionamiento de la aplicación y todas tuvieron solución un tiempo máximo de 3 días. Posteriormente se realizó una iteración adicional para verificar que no existieran no conformidades, la misma arrojó un resultado de cero no conformidades, lo que indica que la aplicación para tarjetas inteligentes en una Infraestructura de Clave Pública presenta buena calidad.

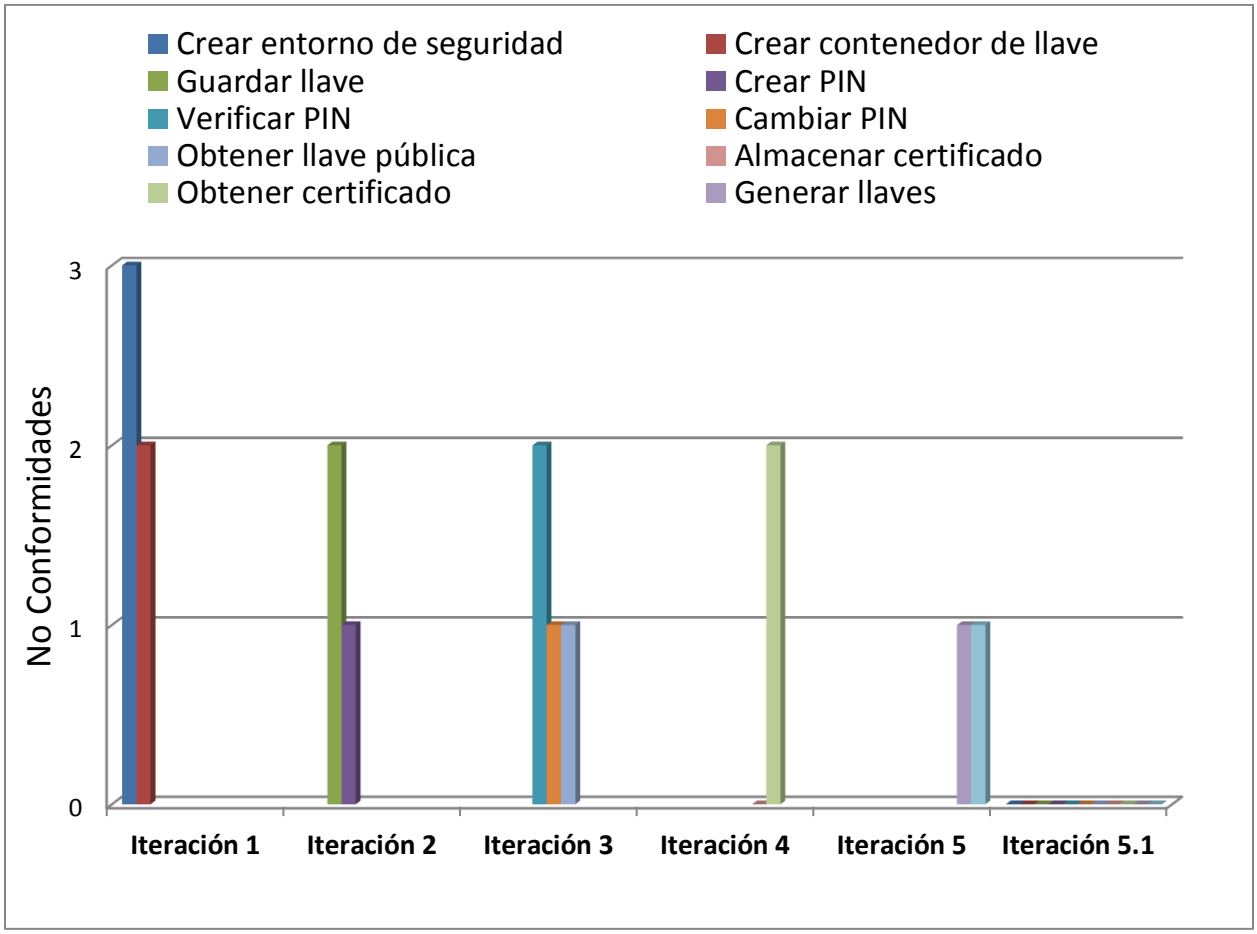


Figura 11: Resultado de las pruebas de caja negra

3.5 Conclusiones

- A partir de los comandos soportados por el applet y la implementación de los mismos, fue posible realizar un diagrama de estados para la aplicación uSafeApp el cual fue utilizado como guía para implementar las transiciones entre los estados del applet.

- La descripción de la estructura de los objetos de seguridad y los comandos APDU facilitó la comprensión y posterior implementación de los mecanismos de seguridad del applet.

- Las pruebas realizadas a la aplicación validaron y verificaron las funcionalidades descritas, comprobando que el sistema cumple con el nivel de calidad requerido.

Conclusiones generales

- La aplicación de los métodos teóricos y el análisis bibliográfico permitieron concretar el marco teórico en correspondencia con el uso de las tarjetas inteligentes en una Infraestructura de Clave Pública.
- Se desarrolló una aplicación que puede ser utilizada en cualquier sistema que requiera incrementar su seguridad mediante el uso de una PKI.
- La aplicación uSafeApp cumple con los estándares ISO/IEC 7816-4 y PKCS #15 permitiendo el uso de tarjetas como dispositivo criptográfico.
- La aplicación desarrollada podrá ser comercializada en el futuro, como uno de los productos del Centro de Identificación y Seguridad Digital de la Universidad de Ciencias Informáticas.

Recomendaciones

Se recomienda:

- Implementar el middleware correspondiente al *applet* uSafeApp cumpliendo con el estándar PKCS #11.
- En desarrollos posteriores el *applet* uSafeApp pueda soportar otros algoritmos de criptografía asimétrica, específicamente en algoritmos basados en curvas elípticas para así aumentar sus posibilidades de uso en ambientes reales en la actualidad.
- Integrar el *applet* uSafeApp con el de Match On Card desarrollado en el departamento de tarjetas inteligentes del CISED, para aumentar la seguridad, incluyendo verificación biométrica en la autenticación por PIN o en el desbloqueo del mismo.

Bibliografía citada

Alliance, Smart Card. 2006. *Tarjetas Inteligentes y Sistemas de Identificación Seguros: Construyendo una Cadena de Confianza.* 2006.

ARX. 2012. ARX. [Online] 2012. http://www.arx.com.

Eclipse. Plataforma Eclipse: Comunidad en español de Eclipse IDE, el entorno de desarrollo multiplataforma más completo del mundo. [Online] [Cited: Noviembre 28, 2011.] http://plataformaeclipse.com/.

Effing, Wolfgang and Rankl, Wolfgang. 2002. *Smart Card Handbook Third Edition: John Wiley & Sons Ltd.* 2002.

Finkenzeller, Klaus. 2010. *RFID HANDBOOK Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field.* 2010.

Gemalto. 2010. DeveloperSuite. [Online] 2010. http://www.gemalto.com/.

gemalto. JCardManager 4.0 home page. *gemalto security to be free.* [Online] gemalto. [Cited: april 26, 2012.] http://www.gemalto.com/products/jcardmanager/.

Gemalto. What is Global Platform?. [Online] [Cited: diciembre 5, 2011.] http://www.gemalto.com/nfc/global_platform.html.

Gemalto. 2011. Gemalto. [Online] 2011. http://www.gemalto.com.

GlobalPlatform. 2003. *Card Specification Version 2.1.1.* 2003.

ISO. 2005. *Organization, security and commands for interchange.* 2005.

Ivar Jacobson, James Rumbaugh, Grady Booch. 2000. *El lenguaje unificado de modelado. Manual de referencia.* 2000.

Joskowicz, Ing. José. 2008. *Reglas y Prácticas en eXtreme Programming.* 2008.

Lucena López, Manuel José. 1999. *Criptografía y Seguridad en Computadores.* 1999.

Oracle. Applet (Java 2 Platform SE 5.0). [Online] [Cited: Diciembre 8, 2011.] http://docs.oracle.com/javase/1.5.0/docs/api/java/applet/Applet.html.

Perovich, Daniel, Rodríguez, Leonardo and Martín, Varela. 2001. Proyecto de Taller V Programación de JavaCards. [Online] Marzo 22, 2001. http://www.fing.edu.uy/inco/grupos/mf/Proyectos/Grado/JavaCard/progjavacard.pdf.

Pressman, Roger S. 2002. *Ingeniería del Software. Un enfoque práctico.* 2002.

proyectos Ágiles. Qué es SCRUM | proyectos Ágiles. [Online] [Cited: Diciembre 10, 2011.] http://www.proyectosagiles.org/que-es-scrum.

RSA Laboratories. 1999. *PKCS #15 v1.0: Cryptographic Token Information Format Standard.* 1999.

RSA Laboratories. 2002. *PKCS #1 v2.1: RSA Cryptography Standard.* 2002.

Smart Card Operating Systems: Past, Present. **2003.** 2003.

SUN. Bienvenido a NetBeans y www.netbeans.org, Portal del IDE Java de Código Abierto.. [Online] [Cited: Noviembre 30, 2011.] http://netbeans.org/index_es.html.

Sun Microsystem. 1999. Java Card 2.1 API Specifications. [Online] 1999. http://java.sun.com/products/javacard/javacard21.html..

Sun Microsystems. 1999. Java Card 2.1 Runtime Environment (JCRE) Specification. [Online] 1999. http://java.sun.com/products/javacard/javacard21.htm.

SUN. http://java.sun.com/javacard/overview.jsp. [Online] http://java.sun.com/javacard/overview.jsp.

Tanenbaum, Andrew S. 2003. *Computer Network Tanenbaum 4ed.* 2003.

Wells, Don. 2011. Extreme Programming. [Online] 2011. http://www.extremeprogramming.org/.

Bibliografía consultada

Albrecht, Allan. 1979. 1979.

Alliance, Smart Card. 2006. *Tarjetas Inteligentes y Sistemas de Identificación Seguros: Construyendo una Cadena de Confianza.* 2006.

Almeida Sotolongo, Dayron y Sáez Vilar, Joel. 2009. *Solución para el control de acceso a la información de las entidades externas, en la cédula de identificación electrónica de la República Bolivariana de Venezuela.* Ciudad de la Habana : Universidad de las Ciencias Informáticas, 2009.

Altova, Inc. 2010. ALTOVA. [En línea] 2010. http://www.altova.com/umodel.htm.

ARX. 2012. ARX. [En línea] 2012. http://www.arx.com.

BlueZ PKCS#15 An implementation for Open Platform Java Cards. **IBM Corporation. 2003.** 2003, Vol. Rev 2.01.

Eclipse. Plataforma Eclipse: Comunidad en español de Eclipse IDE, el entorno de desarrollo multiplataforma más completo del mundo. [En línea] [Citado el: 28 de Noviembre de 2011.] http://plataformaeclipse.com/.

Effing, Wolfgang and Rankl, Wolfgang. 2002. *Smart Card Handbook Third Edition: John Wiley & Sons Ltd.* 2002.

Finkenzeller, Klaus. 2010. *RFID HANDBOOK Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field.* 2010.

Gemalto. 2010. DeveloperSuite. [En línea] 2010. http://www.gemalto.com/.

gemalto. JCardManager 4.0 home page. *gemalto security to be free.* [En línea] gemalto. [Citado el: 26 de april de 2012.] http://www.gemalto.com/products/jcardmanager/.

Gemalto. What is Global Platform?. [En línea] [Citado el: 5 de diciembre de 2011.] http://www.gemalto.com/nfc/global_platform.html.

Gemalto. 2011. Gemalto. [En línea] 2011. http://www.gemalto.com.

GlobalPlatform. 2003. *Card Specification Version 2.1.1.* 2003.

ISO. 2005. *Organization, security and commands for interchange.* 2005.

Ivar Jacobson, James Rumbaugh, Grady Booch. 2000. *El lenguaje unificado de modelado. Manual de referencia.* 2000.

Joskowicz, Ing. José. 2008. *Reglas y Prácticas en eXtreme Programming.* 2008.

Kniberg, Henrik. 2007. *Scrum y XP desde las trincheras.* 2007.

Letelier, Patricio y Panadés, María Carmen. 2004. *Métodologías ágiles para el desarrollo de software:.* Valencia : s.n., 2004.

Lucena López, Manuel José. 1999. *Criptografía y Seguridad en Computadores.* 1999.

Oracle. Applet (Java 2 Platform SE 5.0). [En línea] [Citado el: 8 de Diciembre de 2011.] http://docs.oracle.com/javase/1.5.0/docs/api/java/applet/Applet.html.

Pascual, Juan Carlos. 2010. PKI: seguridad en la red. Microsoft. [En línea] 2010. http://www.microsoft.com/spain/enterprise/perspectivas/numero_5/seguridad.mspx.

Perovich, Daniel, Rodríguez, Leonardo y Martín, Varela. 2001. Proyecto de Taller V Programación de JavaCards. [En línea] 22 de Marzo de 2001. http://www.fing.edu.uy/inco/grupos/mf/Proyectos/Grado/JavaCard/progjavacard.pdf.

Pressman, Roger S. 2002. *Ingeniería del Software. Un enfoque práctico.* 2002.

proyectos Ágiles. Qué es SCRUM | proyectos Ágiles. [En línea] [Citado el: 10 de Diciembre de 2011.] http://www.proyectosagiles.org/que-es-scrum.

Quiñones Bondartchuk, Roberto. 2009. *Firma Digital de Documentos utilizando Smart Cards.* Ciudad de la Habana : Universidad de las Ciencias Informáticas, 2009.

RSA Laboratories. 1999. *PKCS #15 v1.0: Cryptographic Token Information Format Standard.* 1999.

RSA laboratories. 1999. *RSA Laboratories - PKCS #15: Cryptographic Token Information Format Standard.* 1999.

RSA Laboratories. 2002. *PKCS #1 v2.1: RSA Cryptography Standard.* 2002.

Smart Card Alliance. About Smart Cards : Introduction : Primer - Smart Card Alliance. [En línea] [Citado el: 16 de Diciembre de 2011.] http://www.smartcardalliance.org/pages/smart-cards-intro-primer.

Smart Card Operating Systems: Past, Present. **2003.** 2003.

SUN. Bienvenido a NetBeans y www.netbeans.org, Portal del IDE Java de Código Abierto.. [En línea] [Citado el: 30 de Noviembre de 2011.] http://netbeans.org/index_es.html.

Sun Microsystem. 1999. Java Card 2.1 API Specifications. [En línea] 1999. http://java.sun.com/products/javacard/javacard21.html..

Sun Microsystems. 1999. Java Card 2.1 Runtime Environment (JCRE) Specification. [En línea] 1999. http://java.sun.com/products/javacard/javacard21.htm.

SUN. http://java.sun.com/javacard/overview.jsp. [En línea] http://java.sun.com/javacard/overview.jsp.

Tanenbaum, Andrew S. 2003. *Computer Network Tanenbaum 4ed.* 2003.

Wells, Don. 2011. Extreme Programming. [En línea] 2011. http://www.extremeprogramming.org/.

Wells., Don. 2011. Extreme Programming. [En línea] 2011. http://www.extremeprogramming.org/.

Glosario de términos

APDU: Unidad de Datos del Protocolo de aplicación.

Applet: Aplicación que se ejecutan dentro de las tarjetas inteligentes y gestiona la información almacenada en ellas.

CAD: *Card Accepting Device* (Dispositivo que acepta tarjetas inteligentes).

DF: Fichero dedicado.

EF: Fichero elemental.

ISO: (International Organization for Standardization): La Organización Internacional para la Normalización es el organismo encargado de promover el desarrollo de normas internacionales de fabricación, comercio y comunicación para todas las ramas industriales a excepción de la eléctrica y la electrónica. Su función principal es la de buscar la estandarización de normas de productos y seguridad para las empresas u organizaciones a nivel internacional.

Llaves simétricas: Es un tipo de llaves que consiste en que haya una contraseña con la cual sea posible codificar y decodificar los mensajes que se envían entre dos o más partes. Esta contraseña, aplicada a una función junto con el mensaje original, produce una salida tan difícil de descifrar como sea posible. Para descifrarla, se aplica de nuevo la misma función, obteniendo el mensaje original.

MF: Fichero maestro.

PIN: Número de Identificación Personal.

PKCS: (Public-Key Cryptography Standards) se refiere a un grupo de estándares de criptografía de clave pública concebidos y publicados por los laboratorios de RSA.

SW: Palabra de Estado.

TLV: Etiqueta, Longitud y Valor.

Anexos

Anexos 1: Cronograma de trabajo

No.	Acciones a realizar	Responsable	Fecha de entrega
1.	Descripción de las soluciones existentes sobre el uso de tarjetas inteligentes en una Infraestructura de Clave Pública.	Kenly Rodríguez	14/12/2011
2.	Caracterización de la tecnología *Java Card*.	Yoan Cruz	14/12/2011
3.	Análisis de los principales aspectos de la norma ISO/IEC 7816 relacionados con la comunicación con tarjetas inteligentes, la estructura para almacenamiento de información y los modelos de seguridad.	Kenly Rodríguez	14/12/2011
4.	Análisis del estándar PKCS#15 relacionado con la estructura de la información para un dispositivo criptográfico.	Yoan Cruz	14/12/2011
5.	Análisis de los principales algoritmos de criptografía simétricos y asimétricos, protocolos de autenticación, algoritmos de firma digital y certificados digitales.	Yoan Cruz	14/12/2011
6.	Análisis y diseño de la aplicación *Java Card* para tarjetas inteligentes, cumpliendo los estándares analizados.	Kenly Rodríguez	10/2/2012
7.	Implementación de la aplicación *Java Card* a partir del diseño realizado.	Yoan Cruz	15/04/2012
8.	Pruebas de calidad a la aplicación desarrollada.	Kenly Rodríguez	15/04/2012

Tabla 21: Cronograma de trabajo

Anexo 2: HU Crear contenedor de llave

Historia de Usuario	
Número: HU_3	**Nombre de Historia de Usuario**: Crear contenedor de llave
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Desarrollador	**Iteración asignada**: 1
Prioridad en negocio: Alta	**Puntos estimados:** 1.0
Riesgo en desarrollo: Alto	**Puntos reales:** 1.2
Descripción: El *applet* debe permitir crear los contenedores de llaves durante la fase de personalización	
Observaciones: El *applet* responderá con un error en caso que: • El contenedor de la llave exista. • El *applet* ya este personalizado.	

Tabla 22: HU Crear contenedor de llaves

Anexo 3: HU Guardar llave

Historia de Usuario	
Número: HU_4	**Nombre de Historia de Usuario**: Guardar llave
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Desarrollador	**Iteración asignada**: 2
Prioridad en negocio: Alta	**Puntos estimados:** 1.2
Riesgo en desarrollo: Alto	**Puntos reales:** 1.5
Descripción: El *applet* debe permitir guardar llaves simétricas y asimétricas en la tarjeta para su posterior uso, poniéndolas en los contenedores previamente creados para esas llaves.	
Observaciones: El *applet* responderá con un error en caso que: • El contenedor de la llave no exista. • No se cumplan las condiciones de seguridad para actualizar el contenedor de llave seleccionado.	

Tabla 23: HU Guardar llave

Anexo 4: HU Establecer Canal Seguro

Historia de Usuario	
Número: HU_5	**Nombre de Historia de Usuario**: Establecer Canal Seguro

Modificación de Historia de Usuario Número: Ninguna	
Usuario: Desarrollador	**Iteración asignada**: 2
Prioridad en negocio: Alta	**Puntos estimados:** 1.5
Riesgo en desarrollo: Alto	**Puntos reales:** 1.4
Descripción: El *applet* debe permitir la creación de un canal de intercambio de información de forma segura entre el *Middleware* y el *Applet de PKI*.	
Observaciones: El *applet* responderá con un error en caso que: • No se encuentren en el *applet* las llaves necesarias para la autenticación. • La información brindada por los dispositivos que aceptan tarjetas inteligentes (*CAD* por sus siglas en inglés) no coincida con la generada por la tarjeta.	

Tabla 24: HU Establecer Canal Seguro

Anexo 5: HU Crear PIN

	Historia de Usuario
Número: HU_6	**Nombre de Historia de Usuario**: Crear *PIN*
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Desarrollador	**Iteración asignada**: 2
Prioridad en negocio: Alta	**Puntos estimados:** 0.5
Riesgo en desarrollo: Medio	**Puntos reales:** 1.0
Descripción: El *applet* debe permitir crear durante la fase de personalización los *PINs* que se utilizarán para proteger el acceso a los distintos objetos.	
Observaciones: El *applet* responderá con un error en caso de que el *PIN* exista.	

Tabla 25: HU Crear *PIN*

Anexo 6: HU Verificar PIN

	Historia de Usuario
Número: HU_7	**Nombre de Historia de Usuario**: Verificar *PIN*
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Desarrollador	**Iteración asignada**: 3
Prioridad en negocio: Alta	**Puntos estimados:** 0.5
Riesgo en desarrollo: Medio	**Puntos reales:** 0.8
Descripción: El *applet* debe permitir que el usuario introduzca un *PIN* para autenticarse permitiéndole el acceso a los objetos protegidos por ese *PIN*.	

Observaciones: El *applet* responderá con un mensaje de error si: • No se encuentra el *PIN* especificado. • El *applet* no se encuentre en la fase de aplicación. • No se cumplen las condiciones de seguridad para verificar el *PIN*. • La cantidad de intentos restantes en caso de que el *PIN* sea incorrecto.

Tabla 26: HU Cambiar *PIN*

Anexo 7: HU Cambiar PIN

	Historia de Usuario
Número: HU_8	**Nombre de Historia de Usuario**: Cambiar *PIN*
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Desarrollador	**Iteración asignada**: 3
Prioridad en negocio: Media	**Puntos estimados:** 0.9
Riesgo en desarrollo: Medio	**Puntos reales:** 1
Descripción: El *applet* debe brindar la posibilidad de modificar un *PIN* si el usuario ya se ha autenticado con este *PIN* o ha presentado la *clave personal de desbloqueo(PUK por sus siglas en inglés)*	
Observaciones: El *applet* responderá con un mensaje de error si: • No se encuentra en la fase de aplicación. • Si no se encuentra el *PIN* que se intenta cambiar. • No se cumplen las condiciones de seguridad necesarias para cambiar el *PIN*.	

Tabla 27: HU Verificar *PIN*

Anexo 8: HU Obtener llave pública

	Historia de Usuario
Número: HU_9	**Nombre de Historia de Usuario**: Obtener llave pública
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Desarrollador	**Iteración asignada**: 3
Prioridad en negocio: Alta	**Puntos estimados:** 0.9
Riesgo en desarrollo: Medio	**Puntos reales:** 1.3
Descripción: El *applet* debe permitir obtener una llave pública almacenada o generada dentro de la tarjeta.	
Observaciones: El *applet* responderá con un mensaje de error en caso que:	

• No se cumplan las condiciones de seguridad para obtener la llave. • No se encuentre la llave especificada.

Tabla 28: HU Obtener llave pública

Anexo 9: HU Almacenar certificado

Historia de Usuario	
Número: HU_10	**Nombre de Historia de Usuario**: Almacenar certificado
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Desarrollador	**Iteración asignada**: 4
Prioridad en negocio: Alta	**Puntos estimados:** 1.5
Riesgo en desarrollo: Alto	**Puntos reales:** 1.6
Descripción: El *applet* debe permitir que se almacenen certificados de hasta 3 kb en los contenedores de certificados correspondientes.	
Observaciones: El *applet* responderá con un mensaje de error en caso de que no quede espacio para almacenar el certificado.	

Tabla 29: HU Almacenar certificado

Anexo 10: HU Obtener certificado

Historia de Usuario	
Número: HU_11	**Nombre de Historia de Usuario**: Obtener certificado
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Desarrollador	**Iteración asignada**: 4
Prioridad en negocio: Alta	**Puntos estimados:** 0.7
Riesgo en desarrollo: Alto	**Puntos reales:** 1.0
Descripción: El *applet* debe permitir obtener un certificado de hasta 3 kb que se ha guardado previamente en la tarjeta.	
Observaciones: El *applet* responderá con un mensaje de error en caso de que no se cumplan las condiciones de seguridad necesarias para obtener el certificado.	

Tabla 30: HU Obtener certificado

Anexo 11: HU Generar llaves

Historia de Usuario	
Número: HU_12	**Nombre de Historia de Usuario**: Generar llaves
Modificación de Historia de Usuario Número: Ninguna	

Usuario: Desarrollador	**Iteración asignada**: 5
Prioridad en negocio: Alta	**Puntos estimados:** 0.8
Riesgo en desarrollo: Alto	**Puntos reales:** 1.1
Descripción: El *applet* debe permitir generar pares de llaves *RSA* dentro de la tarjeta.	
Observaciones: El *applet* responderá con un mensaje de error en caso que: • No se cumplan las condiciones de seguridad necesarias para modificar los valores de los contenedores de llaves pública y privada, involucrados en la generación. • No exista el contenedor de la llave privada que se intenta actualizar mediante la generación del par llaves.	

Tabla 31: HU Generar llaves

Anexo 12: HU Firmar datos

	Historia de Usuario
Número: HU_13	**Nombre de Historia de Usuario**: Firmar datos
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Desarrollador	**Iteración asignada**: 5
Prioridad en negocio: Alta	**Puntos estimados:** 1.2
Riesgo en desarrollo: Alto	**Puntos reales:** 1.5
Descripción: El *applet* debe permitir realizar la firma digital de un hash *SHA-1* realizado por el *middleware*, correspondiente a los datos que se desea firmar, utilizando una llave privada *RSA* especificada por el usuario.	
Observaciones: El *applet* responderá con un mensaje de error en caso que: • La llave seleccionada no exista. • No se cumplan las condiciones de seguridad para usar esa llave. • No se encuentre ninguna información para firmar.	

Tabla 32: HU Firmar Datos

Anexo 13: Plan de entregas

Entregable	Fin Iteración 1	Fin Iteración 2	Fin Iteración 3	Fin Iteración 4	Fin Iteración 5
Aplicación para tarjetas inteligentes en una Infraestructura de Clave Pública	Enero 2012	Marzo 2012	Abril 2012	Mayo 2012	Junio 2012

Tabla 33: Plan de entregas

Anexo 14: Estimación de tiempo

Historia de Usuario	Estimación
Crear Fichero	**1**
Crear entorno de seguridad	**0.9**
Crear contenedor de llave	**1**
Guardar llave	**1.2**
Establecer Canal Seguro	**1.5**
Crear *PIN*	**0.5**
Cambiar *PIN*	**0.9**
Verificar *PIN*	**0.9**
Obtener llave pública	**0.9**
Almacenar certificado	**0.9**
Obtener certificado	**0.9**
Generar llaves	**0.9**
Firmar Datos	**0.9**

Tabla 34: Estimación de tiempo de las historias de usuario.

Anexo 15: Plan de iteraciones

Iteración	No.HU	Historia de Usuario	Duración estimada (semanas)
1	**HU_1**	Crear Fichero	**12**
	HU_2	Crear entorno de seguridad	
	HU_3	Crear contenedor de llave	
2	**HU_4**	Guardar llave	**8**
	HU_5	Establecer Canal Seguro	
	HU_6	Crear PIN	
3	**HU_7**	Verificar PIN	**8**
	HU_8	Cambiar PIN	

	HU_9	Obtener llave pública	
4	**HU_10**	Almacenar certificado	**8**
	HU_11	Obtener certificado	
5	**HU_12**	Generar llaves	**4**
	HU_13	Firmar Datos	

Tabla 35: Plan de iteraciones

Anexo 16: Fragmento 1 del diagrama de clases.

Figura 12: Fragmento 1 del diagrama de clases

Anexo 17: Fragmento 2 del diagrama de clases.

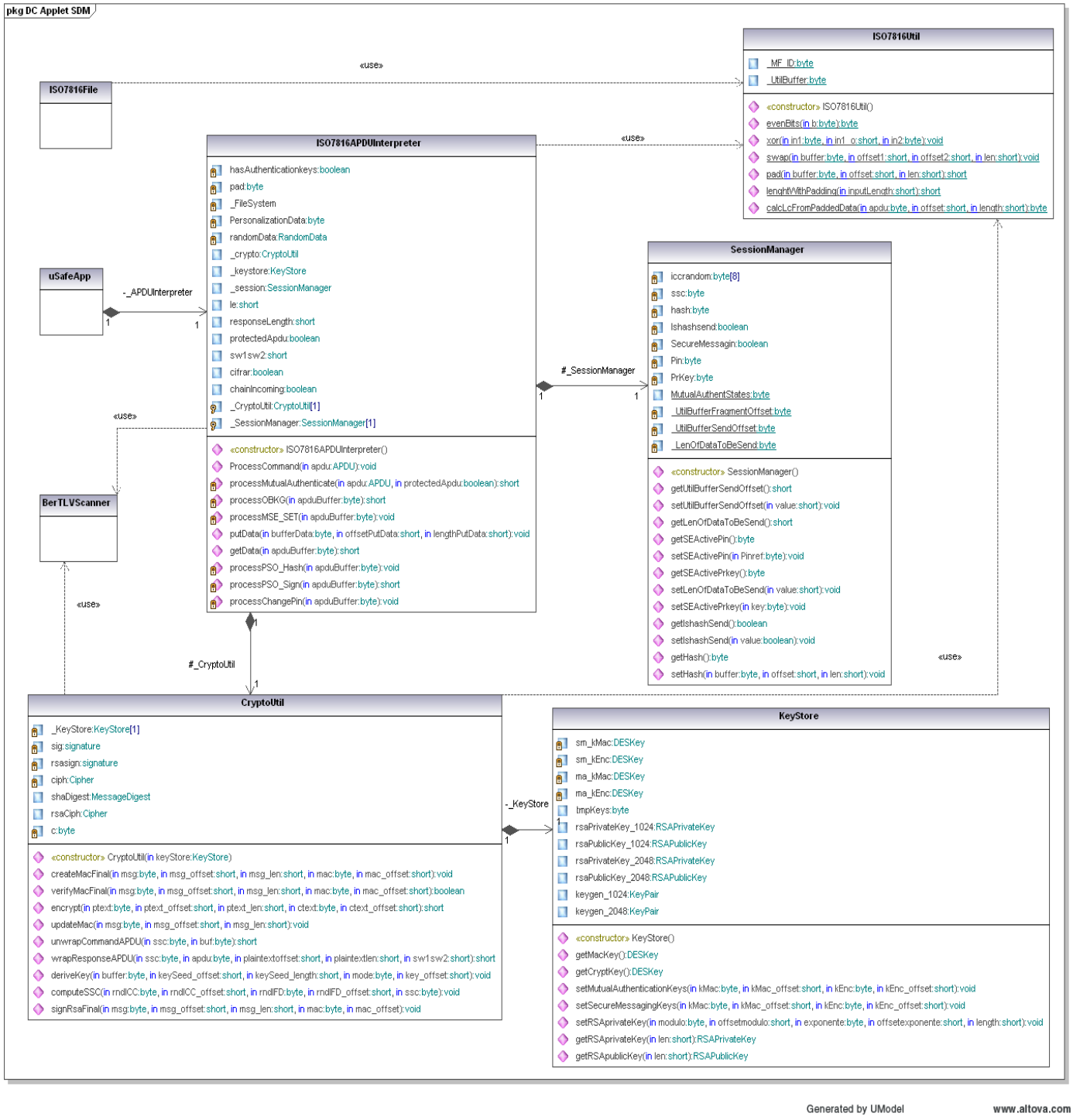


Figura 13: Fragmento 2 del diagrama de clases.

Anexo 18: Descripción de la clase SecurityEnvironment.

Nombre: SecurityEnvironment
Tipo de clase: controladora

Atributo	**Tipo**
_IDSE	byte
_LCS	byte
_CRTCollection	List<byte[]>
Para cada responsabilidad:	
Nombre:	SecurityEnvironment(byte idSE, bytelcsSE)
Descripción:	Crea un objeto de la clase SecurityEnvironment
Nombre:	getSEID()
Descripción:	Obtiene el id del entorno de seguridad
Nombre:	getLCS()
Descripción:	Obtiene el ciclo de vida del entorno de seguridad
Nombre:	setSEID(byte value)
Descripción:	Establece el id del entorno de seguridad
Nombre:	setLCS(byte value)
Descripción:	Establece el ciclo de vida para el que es válido el entorno de seguridad
Nombre:	getCRT()
Descripción:	Obtiene una plantilla de referencia de control
Nombre:	AddCRT(byte[] newRCT,short offsetRCT, short lengthRCT)
Descripción:	Agrega una plantilla de referencia de control

Tabla 36: Descripción de la clase SecurityEnvironment

Anexo 19: Descripción de la clase DataObject.

Nombre: DataObject	
Tipo de clase: controladora	
Atributo	**Tipo**
_Data	Byte[]
Longitud	short
Verifiedflag	byte
Para cada responsabilidad:	
Nombre:	DataObject()
Descripción:	Crea un objeto de la clase
Nombre:	addDataDO(byte[] dataSDO,short offsetdataSDo, short lengthDataSDO,short lengthcontainer)
Descripción:	Agrega datos al objeto de datos
Nombre:	getDataDObject()
Descripción:	Obtiene la información contenida en el Objeto de Datos
Nombre:	updateDataDO(byte[] dataSDO,short offsetdataSDo,short offsetcontainer, short lengthDataSDO)
Descripción:	Actualiza el Objeto de Datos copiando una cantidad de datos x a partir de una posición.
Nombre:	getlongitud()
Descripción:	Obtiene la longitud real del Objeto de Datos.
Nombre:	getVerifiedFlag()
Descripción:	Obtiene si el objeto ha sido verificado en el caso de los PIN.
Nombre:	setVerifiedFlag(byte value)
Descripción:	Establece el valor del atributo verifiedflag en el caso de los PIN.
Nombre:	setVerifiedFlag()
Descripción:	Establece si el PIN ha sido correctamente presentado o no.

Tabla 37: Descripción de la clase DataObject

Anexo 20: Descripción de la clase ISO7816APDUInterpreter.

Nombre: ISO7816APDUInterpreter	
Tipo de clase: controladora	
Atributo	**Tipo**
hasAuthenticationkeys	boolean
CREATE_FILE_INS	byte
SELECT_FILE_INS	byte
READ_BINARY_INS	byte
UPDATE_BINARY_INS	byte
PUT_DATA_INS	byte
OBKG_INS	byte
PSO_INS	byte
MSE_INS	byte
VERIFY_INS	byte
GET_DATA_SYS_INS	byte
GET_DATA_TLV_INS	byte
CHANGE_REFERENCE_DATA_INS	byte
RESET_RETRY_COUNTER_INS	byte
GET_RESPONSE_INS	byte
END_PERSONALIZATION_INS	byte
SW_INTERNAL_ERROR	short
INS_MUTUAL_AUTHENTICATE	byte
INS_GET_CHALLENGE	byte
INS_INTERNAL_AUTHENTICATE	byte
Pad	byte
_FileSystem	ISO7816FileSystem
PersonalizationData	byte
randomData	RandomData
_crypto	CryptoUtil
_keystore	KeyStore
_session	SessionManager
KEY_LENGTH	short
KEYMATERIAL_LENGTH	short
RND_LENGTH	short
MAC_LENGTH	short
CLA_PROTECTED_APDU	byte
SW_OK	short
SW_SM_DO_MISSING	short
SW_SM_DO_INCORRECT	short
Le	short
responseLength	short
protectedApdu	boolean
sw1sw2	short
Cifrar	boolean
chainIncoming	boolean
_CryptoUtil	CryptoUtil
_SessionManager	SessionManager
Para cada responsabilidad:	
Nombre:	ISO7816APDUInterpreter(ISO7816FileSystem fileSystem)
Descripción:	Crea un objeto de la clase.
Nombre:	ProcessCommand(APDU apdu)
Descripción:	Se encarga de procesar los comandos entregados al intérprete.
Nombre:	processGetChallenge(APDU apdu, boolean protectedApdu, short le)

Descripción:	Procesa los comandos GetChallange.
Nombre:	processMutualAuthenticate(APDU apdu, booleanprotectedApdu)
Descripción:	Procesa los comandos MutualAuthenticate.
Nombre:	processOBKG(byte[] apduBuffer)
Descripción:	Procesa los comandos Generate Public Key Pair.
Nombre:	processMSE_SET(byte[] apduBuffer)
Descripción:	Procesa los comandos MSE-SET.
Nombre:	processVerify(byte[] apduBuffer)
Descripción:	Procesa los comandos Verify.
Nombre:	putData(byte[] bufferData, short offsetPutData, short lengthPutData)
Descripción:	Procesa todos los comandos PUT Data
Nombre:	getData(byte[] apduBuffer)
Descripción	Obtiene los datos especificados en el apduBuffer.
Nombre:	processPSO_Hash(byte[] apduBuffer)
Descripción	Almacena el hash enviado en el apduBuffer
Nombre:	processPSO_Sign(byte[] apduBuffer)
Descripción:	Firma el hash almacenado previamente
Nombre:	processChangePin(byte[] apduBuffer)
Descripción:	Procesa los comandos Change Refence Data
Nombre:	processResetRetryCounter(byte[] apduBuffer)
Descripción	Desbloquea un PIN bloqueado por exceder el número de intentos incorrectos.

Tabla 38: Descripción de la clase ISO7816APDUInterpreter.

Anexo 21: Descripción de la clase CryptoUtil.

Nombre: CryptoUtil	
Tipo de clase: controladora	
Atributo	**Tipo**
ENC_MODE	byte
MAC_MODE	byte
PAD_INPUT	byte
DONT_PAD_INPUT	byte
PAD_DATA	byte
_KeyStore	KeyStore
Sig	signature
Rsasign	signature
Ciph	Cipher
shaDigest	MessageDigest
rsaCiph	Cipher
C	byte
Para cada responsabilidad:	
Nombre:	CryptoUtil(KeyStore keyStore)
Descripción:	Crea un objeto de la clase
Nombre:	createMacFinal(byte[] msg, shortmsg_offset, shortmsg_len,byte[] mac, shortmac_offset)
Descripción:	Crea un código de autenticación de mensaje MAC (por sus siglas en ingles), de 8 bytes, utilizando el algoritmo DES y las especificaciones de ISO9797-1.
Nombre:	verifyMacFinal(byte[] msg, shortmsg_offset, shortmsg_len, byte[] mac, shortmac_offset)
Descripción:	Verifica un código de autenticación de mensaje MAC de 8 bytes, utilizando el algoritmo DES y las especificaciones de ISO9797-1.
Nombre:	decryptInit()

Descripción:	Inicializa el objeto de la clase Cipher en modo de descifrado.
Nombre:	decryptFinal(byte[] ctext, shortctext_offset,shortctext_len, byte[] ptext, shortptext_offset)
Descripción:	Descifra un arreglo de datos.
Nombre:	encryptInit()
Descripción:	Inicializa el objeto de la clase Cipher en modo de cifrado.
Nombre:	encryptInit(byte padding, byte[] ptext, shortptext_offset, shortptext_len)
Descripción:	Inicializa el objeto de la clase Cipher en modo de cifrado.
Nombre:	encrypt(byte[] ptext, shortptext_offset, shortptext_len,byte[] ctext, shortctext_offset)
Descripción:	Cifra un bloque de datos y espera por el resto, si el bloque no tiene el tamaño correcto, se almacena y espera por los datos restantes.
Nombre:	encryptFinal(byte[] ptext, shortptext_offset,shortptext_len, byte[] ctext, shortctext_offset)
Descripción:	Cifra todos o el último bloque de datos y almacena el resultado.
Nombre:	updateMac(byte[] msg, shortmsg_offset, shortmsg_len)
Descripción:	Acumula la firma de los datos entrados.
Nombre:	initMac(byte mode)
Descripción:	Inicializa el objeto sig utilizado para calcular el código de autenticación de mensaje
Nombre:	unwrapCommandAPDU(byte[] ssc, byte[] buf)
Descripción:	Descifra un comando.
Nombre:	getApduBufferOffset(shortplaintextLength)
Descripción:	Obtiene la cantidad de bytes que tendrá la cabecera del comando cifrado.
Nombre:	wrapResponseAPDU(byte[] ssc, byte[] apdu, shortplaintextOffset, shortplaintextLen, short sw1sw2)
Descripción:	Cifra un comando para enviarlo al terminal.
Nombre:	deriveKey(byte[] buffer, shortkeySeed_offset,shortkeySeed_length, byte mode, shortkey_offset)
Descripción:	Deriva las llaves de sesión a partir de las llaves simétricas de autenticación mutua
Nombre:	incrementSSC(byte[] ssc)
Descripción:	Incrementa el contador de secuencia.
Nombre:	computerSSC(byte[] rndICC, shortrndICC_offset,byte[] rndIFD, shortrndIFD_offset, byte[] ssc)
Descripción:	Calcula el contador de secuencia.
Nombre:	OBKG(byte[] apdubuffer,shortkeylen)
Descripción:	Genera un par de llaves RSA dentro de la tarjeta.
Nombre:	InitRsaSign(shortlen,byte mode)
Descripción:	Inicializa el objeto rsasign con la llave que se utilizará para firmar.
Nombre:	signRsaFinal(byte[] msg, shortmsg_offset, shortmsg_len, byte[] mac, shortmac_offset)
Descripción:	Firma los datos utilizando los algoritmos RSA y SHA, siguiendo las indicaciones de PKCS#1

Tabla 39: Descripción de la clase CryptoUtil.

Anexo 22: Descripción de la clase KeyStore.

Nombre: KeyStore	
Tipo de clase: controladora	
Atributo	**Tipo**
sm_kMac	DESKey

sm_kEnc	DESKey
ma_kMac	DESKey
ma_kEnc	DESKey
tmpKeys	byte
rsaPrivateKey_1024	RSAPrivateKey
rsaPublicKey_1024	RSAPublicKey
rsaPrivateKey_2048	RSAPrivateKey
rsaPublicKey_2048	RSAPublicKey
keygen_1024	KeyPair
Keygen_2048	KeyPair
Para cada responsabilidad:	
Nombre:	KeyStore()
Descripción:	Crea un objeto de la clase
Nombre:	getMacKey()
Descripción:	Obtiene la llave de mac actual del keystore.
Nombre:	getCryptKey()
Descripción:	Obtiene la llave de cifrar actual del keystore.
Nombre:	setMutualAuthenticationKeys(byte[] kMac, shortkMac_offset, byte[] kEnc, shortkEnc_offset)
Descripción:	Establece las llaves para la autenticación mutua, de estas llaves se derivan las llaves de sesión.
Nombre:	setSecureMessagingKeys(byte[] kMac, shortkMac_offset, byte[] kEnc, shortkEnc_offset)
Descripción:	Establece las llaves de sesión.
Nombre:	setRSAprivateKey(byte[] modulo,shortoffsetmodulo,byte[] exponente,shortoffsetexponente,short length)
Descripción:	Establece la llave privada que será usada en la siguiente operación de firma.
Nombre:	getRSAprivateKey(shortlen)
Descripción:	Obtiene la llave privada que está actualmente en uso.
Nombre:	getRSApublicKey(shortlen)
Descripción:	Obtiene la llave pública que está actualmente en uso.

Tabla 40: Descripción de la clase KeyStore

Anexo 23: Descripción de la clase BerTLVScanner.

Nombre: BerTLVScanner	
Tipo de clase: controladora	
Atributo	**Tipo**
UNIVERSAL_CLASS	short
APPLICATION_CLASS	short
CONTEXT_SPECIFIC_CLASS	short
PRIVATE_CLASS	short
Tag	short
tagClass	short
isPrimitive	boolean
valueOffset	short
valueLength	short
Para cada responsabilidad:	
Nombre:	readTag(byte[] in, short offset, short length)
Descripción:	Identifica un *tag* de un TLV y avanza el cursor en la secuencia de bytes
Nombre:	readLength(byte[] in, short offset, short length)
Descripción:	Identifica una longitud en un TLV y avanza el cursor en la secuencia de bytes

Nombre:	skipValue(byte[] in)
Descripción:	Identifica la posición del *tag* del siguiente tlv.

Tabla 41: Descripción de la clase: BerTLVScanner

Anexo 24: Descripción de la clase ISO7816Util.

Nombre: ISO7816Util	
Tipo de clase: controladora	
Atributo	**Tipo**
_MF_ID	byte
_OffsetEFSize	short
_OffsetEFFDB	short
_OffsetEFID	short
_OffsetEFValueLCS	short
_OffsetEFLengthSecurityAttribute	short
_OffsetEFAccessMode	short
_OffsetEFSecurityCondition	short
_OffsetDFID	short
_OffsetDFLengthSecurityAttribute	short
_OffsetDFAccessMode	short
_OffsetDFSecurityCondition	short
_OffsetDFLenghtName	short
_UtilBuffer	byte
_OffsetDFName	short
Para cada responsabilidad:	
Nombre:	ISO7816Util()
Descripción:	Crea un objeto de la clase
Nombre:	evenBits(byte b)
Descripción:	Cuenta el número de bits con valor 1 en un byte
Nombre:	xor(byte[] in1, short in1_o, byte[] in2, short in2_o, byte[] out, shortout_o, shortlen)
Descripción:	Realiza xor entre dos arreglos de bytes y guarda el resultado en out
Nombre:	swap(byte[] buffer, short offset1, short offset2, shortlen)
Descripción:	Intercambia 2 segmentos de arreglo de la misma longitud en en arreglo de bytes.
Nombre:	pad(byte[] buffer, short offset, shortlen)
Descripción:	Rellena un arreglo de bytes con máximo 8 bytes y mínimo 1 comenzando con 0x80 y seguido por 7 o menos ceros, esta operación de relleno está definida en el método de relleno 2 del estándar ISO/IEC 9797-1.
Nombre:	lengthWithPadding(shortinputLength)
Descripción:	Devuelve la longitud que tendría un arreglo de longitud inputLength incluyendo el relleno.
Nombre:	calcLcFromPaddedData(byte[] apdu, short offset, short length)
Descripción:	Calcula la longitud de un bloque de datos ignorando el relleno.

Tabla 42: Descripción de la clase ISO7816Util

Anexo 25: Descripción de la clase SessionManager.

Nombre: SessionManager	
Tipo de clase: controladora	
Atributo	**Tipo**
iccrandom	byte
Ssc	byte
Hash	byte

Ishashsend	boolean
SecureMessagin	boolean
Pin	byte
PrKey	byte
MutualAuthentStates	byte
_UtilBufferFragmentOffset	byte
_UtilBufferSendOffset	byte
_LenOfDataToBeSend	byte
Para cada responsabilidad:	
Nombre:	SessionManager()
Descripción:	Crea un objeto de la clase
Nombre:	getUtilBufferFragmentOffset()
Descripción:	Obtiene la posición donde se debe copiar los datos durante la entrada de comandos en cadena.
Nombre:	setUtilBufferFragmentOffset(short value)
Descripción:	Establece la posición donde se debe copiar los datos durante la entrada de comandos en cadena.
Nombre:	getUtilBufferSendOffset()
Descripción:	Obtiene la posición desde donde se debe enviar datos durante el envío de comandos en cadena.
Nombre:	setUtilBufferSendOffset(short value)
Descripción:	Establece la posición desde donde se debe enviar datos durante el envío de comandos en cadena.
Nombre:	getLenOfDataToBeSend()
Descripción:	Obtiene la longitud de los datos que se van a enviar en cadena.
Nombre:	setLenOfDataToBeSend(short value)
Descripción:	Establece la longitud de los datos que se van a enviar en cadena.
Nombre:	getICCrandom()
Descripción:	Obtiene los 8 bytes almacenados, que se generaron durante el comando Get Challange.
Nombre:	setICCrandom(byte[] buffer,shortoffset,shortlen)
Descripción:	Almacena los 8 bytes generados durante el comando Get Challange.
Nombre:	getSSC()
Descripción:	Obtiene el contador de secuencia utilizado en la mensajería segura.
Nombre:	setSSC(byte[] buffer,shortoffset,shortlen)
Descripción:	Establece el contador de secuencia utilizado en la mensajería segura.
Nombre:	getSEActivePin()
Descripción:	Obtiene el PIN que se encuentra activo para verificar.
Nombre:	setSEActivePin(bytePinref)
Descripción:	Activa un PIN permitiendo su verificación.
Nombre:	getSEActivePrkey()
Descripción:	Obtiene la llave privada que se encuentra activa para ser usada en una operación de firma.
Nombre:	setSEActivePrkey(byte key)
Descripción:	Activa una llave privada para su posterior uso en una operación de firma.
Nombre:	getMAStates()
Descripción:	Obtiene el estado en que se encuentra la autenticación mutua.
Nombre:	setMAStates(byte state)
Descripción:	Establece el estado en que se encuentra la autenticación mutua.
Nombre:	getSecureMessagin()
Descripción:	Obtiene si se está usando o no, mensajería segura (si el comando viene cifrado o no).

Nombre:	setSecureMessagin(boolean value)
Descripción:	Establece si se está usando o no, mensajería segura.
Nombre:	getIshashSend()
Descripción:	Obtiene si hay o no un hash en la tarjeta, para ser firmado.
Nombre:	setIshashSend(boolean value)
Descripción:	Establece si hay o no un hash en la tarjeta, para ser firmado.
Nombre:	getHash()
Descripción:	Obtiene el hash que se encuentra almacenado en la tarjeta para firmarlo.
Nombre:	setHash(byte[] buffer,shortoffset,shortlen)
Descripción:	Almacena un hash en la tarjeta para firmarlo.

Tabla 43: Descripción de la clase SessionManager

Anexo 26: COMANDO GENERATE PUBLIC KEY PAIR

Este comando no crea los objetos de datos del par de llaves, pero puede ser usado para inicializar o actualizar un par de llaves existente. Almacena la llave privada y pública en el *applet* y devuelve el valor del módulo de la llave pública correspondiente de respuesta al terminal.

El formato del comando es el siguiente:

CLA	INS	P1	P2	Lc	Data	Le
CLA	46h	02h	P2	Lc	Data	Le

Tabla 44: Formato del comando GENERATE PUBLIC KEY PAIR

Donde

CLA: 00h Etapa de personalización sin mensajería segura.
04h Etapa de personalización con mensajería segura.

P2: es el ID de la llave privada a ser actualizada.

LC: es la longitud del exponente de la llave.

Data: contiene el objeto de datos del exponente de la llave en formato TLV como se muestra:

Etiqueta	Longitud	Etiqueta	Longitud	Valor
7F49h	Longitud del exponente TLV	82h	01h-08h	Exponente público

Tabla 45 Ejemplo de data: GENERATE PUBLIC KEY PAIR

Le: es la longitud esperada de la respuesta.

Respuesta:

La respuesta es el módulo de la llave pública en formato TLV como se muestra:

Etiqueta	Longitud	Etiqueta	Longitud	Valor
7F49h	Lmod+2	81h	Lmod	Módulo

Tabla 46: Módulo de la llave pública en formato TLV

La tarjeta retorna los códigos de estado SW1 y SW2. Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado sin error.
67h	00h	Longitud incorrecta de Lc.
69h	82h	Estado de seguridad no satisfecho: La actualización del atributo de seguridad para la llave privada no se ha cumplido. Error durante la mensajería segura.
6Ah	80h	Data incorrecta, por ejemplo: Incorrecta etiqueta o longitud de etiqueta (incluyendo TLVs usados en mensajería segura). Incorrecta longitud del comando.
6Ah	86h	Incorrectos parámetros P1 y P2

Tabla 47: Posibles valores para SW1 y SW2

Anexo 27: COMANDO PUT DATA

Este comando crea los siguientes objetos de datos en la etapa de personalización:

- Entorno de seguridad
- *PIN*
- Llaves secretas DES
- Llaves públicas RSA
- Llaves privadas RSA

Las condiciones de uso y la estructura de los comandos para cada tipo de objeto son diferentes, por lo que son descritas por separado.

COMANDO PUT DATA Entorno de seguridad

Este comando es usado para crear los entornos de seguridad del *applet*.

El formato del comando es el siguiente:

CLA	INS	P1	P2	Lc	Data
CLA	DBh	00h	FFh	Lc	Data

Tabla 48: Descripción del comando PUT DATA Entorno de seguridad

Donde:

CLA: 00h Transmite el final o solo una porción de los datos normalmente y ejecuta el comando.

04h Transmite el final o solo una porción de los datos con mensajería segura y ejecuta el comando.

10h Transmite la primera porción de datos en modo cadena normalmente y espera por la porción final.

14h Transmite la primera porción de datos en modo cadena con mensajería segura y espera por la porción final.

Lc: indica la longitud del campo Data.

Data: como el ejemplo que se describe a continuación:

En este entorno de seguridad hay dos objetos *PIN*: *PINA* y *PINB*. Cada *PIN* debe ser codificado en una plantilla de autenticación separada. El SE1 es codificado de la siguiente manera:

- SEID = 01h
- LCS = 04h, significa Operacional
- CRT = A4h, significa Plantilla de autenticación (AT)

<table>
<tr><th>Etiqueta</th><th>Longitud</th><th colspan="3">Valor</th><th>Descripción</th></tr>
<tr><td>80h</td><td>01h</td><td colspan="3">01h</td><td>Etiqueta, longitud y valor del SEID (SE#1).</td></tr>
<tr><td>8Ah</td><td>01h</td><td colspan="3">04h</td><td>Etiqueta, longitud y valor del LCS (04= Operacional).</td></tr>
<tr><td>A4h</td><td>06h</td><td colspan="3"></td><td>Etiqueta y longitud de la plantilla de autenticación.</td></tr>
<tr><td colspan="2" rowspan="3"></td><td>Etiqueta</td><td>Longitud</td><td>Valor</td><td></td></tr>
<tr><td>83h</td><td>01h</td><td>Referencia del PIN</td><td>TLV del PIN A (Valor de AT).</td></tr>
<tr><td>95h</td><td>01h</td><td>08h</td><td>Clasificador de uso CRT (el valor 08h indica autenticación de usuario (AT)).</td></tr>
<tr><td>A4h</td><td>06h</td><td colspan="3"></td><td>Etiqueta y longitud de la plantilla de autenticación.</td></tr>
<tr><td colspan="2"></td><td>83h</td><td>01h</td><td>Referencia</td><td>TLV del PIN B (Valor de</td></tr>
</table>

			del *PIN*	AT).
	95h	01h	08h	Clasificador de uso CRT (el valor 08h indica autenticación de usuario).

Tabla 49: Ejemplo de data: PUT DATA Entorno de seguridad

COMANDO PUT DATA *PIN*

Este comando es usado para crear objetos de datos de tipo *PIN*. Solo puede crear un *PIN* a la vez.

El formato del comando es el siguiente:

CLA	INS	P1	P2	Lc	Data
CLA	DBh	00h	FFh	Var.	Data

Tabla 50: Descripción del comando PUT DATA *PIN*

Donde:

CLA: 00h Transmite el final o solo una porción de los datos normalmente y ejecuta el comando.

04h Transmite el final o solo una porción de los datos con mensajería segura y ejecuta el comando.

10h Transmite la primera porción de datos en modo cadena normalmente y espera por la porción final.

14h Transmite la primera porción de datos en modo cadena con mensajería segura y espera por la porción final.

Lc: indica la longitud del campo Data.

Data: como el ejemplo que se describe a continuación:

Suponga que se desean crear dos *PIN*, *PIN*1 y *PIN*2, donde *PIN*2 debe ser presentado para cambiar o resetear el *PIN*1.

*PIN*1 tiene los siguientes valores:

Referencia del *PIN*	= 81h (PIN#1).
Límites de intentos	=03h.
Contador de desbloqueo	= A5h (no existe límite)
Referencia del *PIN* de desbloqueo	=82h (*PIN*#2 debe ser presentado para desbloquear al *PIN*#1).
Contador de uso	= FFh (no existe límite).

Valor del *PIN*1	= 32h 31h 36h 33h 38h 34h 31h 33h.
Atributos de seguridad	RESET RETRY COUNTER= SM definido en el SE#1.
	CHANGE REFERENCE DATA= SM y *PIN*2 definido en el SE#1.
	VERIFY = Libre.

Tabla 51: Valores de *PIN*1

*PIN*2 tiene los siguientes valores:

Referencia del *PIN*	= 82h (*PIN*#2).
Límites de intentos	=04h.
Contador de desbloqueo	= 00h (no puede ser desbloqueado).
Referencia del *PIN* de desbloqueo	=FFh (no comprobado).
Contador de uso	= 05h (puede ser usado solo 5 veces).
Valor del *PIN*1	= 31h 32h 35h 37h 31h 35h 33h 37.
Atributos de seguridad	RESET RETRY COUNTER= Nunca.
	CHANGE REFERENCE DATA= Nunca.
	VERIFY = Libre.

Tabla 52: Valores de *PIN*2

Los dos *PIN*s deben ser creados usando comandos PUT DATA separados. El campo de datos para cada *PIN* es el siguiente:

PIN#1

Etiqueta	**Longitud**	**Plantilla del *PIN***		
FFh 20h	1Bh	**Etiqueta**	**Longitud**	**Valor**
		83h	01h	Referencia del *PIN*= 81h.
		8Ch	04h	Atributos de seguridad = F0h 42h 62h 00h F0h: El byte AM indica que hay bytes SC para los comandos RESET RETRY COUNTER, CHANGE REFERENCE DATA y VERIFY. 42h: Byte SC definido para RESET RETRY COUNTER (SM definido en el SE#1). 62h: Byte SC definido para CHANGE

			REFERENCE DATA (SM y *PIN* definido en el SE#1). 00h: Byte SC definido para VERIFY (Libre).
	DFh 21h	04h	Atributos del *PIN* = 03h FFh A5h 86h 03h: Límite de intentos. FFh; Contador de uso (no existe límite). A5h: Contador de desbloqueo (no existe límite). 86h: Referencia del *PIN* de desbloqueo (*PIN*#2).
	DFh 22h	08h	Valor del *PIN*= 30h 31h 32h 33h 34h 35h 36h 37h .

Tabla 53: Comando de datos del *PIN*1

Para este ejemplo el valor de Lc es 1Eh

PIN#2

Etiqueta	**Longitud**	**Plantilla del *PIN***		
FFh 20h	1Bh	**Etiqueta**	**Longitud**	**Valor**
		83h	01h	Referencia del *PIN*= 82h.
		8Ch	04h	Atributos de seguridad = 90h 43h 90h: Byte AM que indica que hay un byte SC para VERIFY solamente 43h: SC definido para VERIFY en SE#1.
		DFh 21h	04h	Atributos del *PIN* = 04h 05h 00h FFh. 04h: Límite de intentos 05h: Contador de uso (cinco veces). 00h: Contador de desbloqueo (no puede ser desbloqueado). FFh: Referencia del *PIN* de desbloqueo (no comprobado).
		DFh 22h	0Ch	Valor del *PIN*= 39h 38h 37h 36h 35h 34h 33h 32h 31h 30h 30h30h.

Tabla 54: Comando de datos del *PIN*2

COMANDO PUT DATA Llaves Secretas

Estas llaves son usadas durante la autenticación mutua simétrica. Estas deben ser creadas como un par e inicializadas durante la etapa de personalización.

El formato del comando es el siguiente:

CLA	INS	P1	P2	Lc	Data
CLA	DBh	00h	FFh	Lc	Data

Tabla 55: Descripción del comando PUT DATA Llaves Secretas

Donde:

CLA: 00h Transmite el final o solo una porción de los datos normalmente y ejecuta el comando.

04h Transmite el final o solo una porción de los datos con mensajería segura y ejecuta el comando.

10h Transmite la primera porción de datos en modo cadena normalmente y espera por la porción final.

14h Transmite la primera porción de datos en modo cadena con mensajería segura y espera por la porción final.

Lc: indica la longitud del campo Data

Data: como el TLV que se describe en el ejemplo a continuación:

Etiqueta	Longitud	Valor	Descripción	No. de bytes
A4h	03h		Uso de la llave: Plantilla de autenticación.	2
		83 01 xxh	Donde: xx = id de la llave.	3
8C	02h	82h[2]XXh[3]	TLV de los atributos claves de seguridad.	19
DF23h	10h	K.ENC	Valor de K.ENC.	19
DF24h	10h	K.MAC	Valor de K.MAC.	19
			Total de bytes	43

Tabla 56 Ejemplo de data: PUT DATA -Creación de llave secreta

COMANDO PUT DATA Llaves Privadas

[2]82h es el valor del byte de modo de acceso

[3]XXh es el valor del byte de la condición de seguridad

Según el estándar PKCS#1 una llave privada puede ser representada de dos formas, la primera consiste en el par (módulo (N), exponente privado (d)) y la segunda consiste en cinco elementos (p, q, qInv, dP,dQ). La primera forma es la utilizada por la aplicación, puesto que la API criptográfica de *Java Card* recibe estos elementos para crear los objetos llave privada. Las llaves privadas son creadas durante la etapa de personalización usando un comando PUT DATA. (RSA Laboratories., 2002)

El formato del comando es el siguiente:

CLA	INS	P1	P2	Lc	Data
CLA	DBh	00h	FFh	Lc	Data

Tabla 57: Comando PUT DATA Llaves Privadas

Donde:

CLA: 00h Transmite el final o solo una porción de los datos normalmente y ejecuta el comando.

04h Transmite el final o solo una porción de los datos con mensajería segura y ejecuta el comando.

10h Transmite la primera porción de datos en modo cadena normalmente y espera por la porción final.

14h Transmite la primera porción de datos en modo cadena con mensajería segura y espera por la porción final.

Lc: indica la longitud del campo Data

Data: como el TLV que se describe en el ejemplo a continuación:

<table>
<tr><th>Etiqueta</th><th>Longitud</th><th colspan="3">Valor</th><th>Descripción</th><th>No. de bytes</th></tr>
<tr><td>A4 o B6h</td><td>03h</td><td colspan="3"></td><td>AT (llave de autenticación SK.ICC.AUT).</td><td>2</td></tr>
<tr><td></td><td></td><td>84h</td><td>01h</td><td>ID Llave</td><td>TLV de la llave privada.</td><td>3</td></tr>
<tr><td>DFh 25h</td><td>02h</td><td colspan="3">Longitud de la llave</td><td>TLV de la longitud máxima de la llave
08h 00h indica 2048 bytes
04h 00h indica 1024 bytes</td><td>5</td></tr>
<tr><td>DFh 26h</td><td>01h</td><td colspan="3">XXh</td><td>TLV para el uso de llaves privadas
Bit
<table><tr><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td></tr></table></td><td>4</td></tr>
</table>

			RFU RFU RFU RFU RFU RFU RFU RFU b1 = 1 indica llave para firma digital (DS). b2 = indica llave de utilidad. Una llave puede ser ambas cosas (de utilidad y para firma digital).	
8Ch	03h		Etiqueta y longitud del atributo de seguridad (bytes AM y SC).	2
		8Ah	Byte modo de acceso	1
		00h	Byte de condición de seguridad para PSO. 00h significa que ninguna condición fue especificada.	1
		27h	Byte de condición de seguridad para PUT DATA y GENERATE PUBLIC KEY PAIR (27 h significa que EXTERNAL AUTHENTICATE es requerida).	1
			Total de bytes	**19**

Tabla 58 Ejemplo de data: PUT DATA -Creación de llave privada

COMANDO PUT DATA Llaves Públicas

Una llave pública contiene dos elementos: un módulo N y un exponente público e. Es creada durante la etapa de personalización en dos etapas:

- Un comando PUT DATA para crear la llave
- Uno o dos comandos PUT DATA para inicializar los datos de cada uno de los elementos

El formato del comando es el siguiente:

CLA	INS	P1	P2	Lc	Data
CLA	DBh	00h	FFh	Lc	Data

Tabla 59: Descripción del comando PUT DATA Llaves Públicas

Donde:

CLA: 00h Transmite el final o solo una porción de los datos normalmente y ejecuta el comando.

04h Transmite el final o solo una porción de los datos con mensajería segura y ejecuta el comando.

10hTransmite la primera porción de datos en modo cadena normalmente y espera por la porción final.

14h Transmite la primera porción de datos en modo cadena con mensajería segura y espera por la porción final.

Lc: indica la longitud del campo Data

Data: como el TLV que se describe en el ejemplo a continuación:

Ejemplo para la creación de la llave pública

Etiqueta	Longitud	Valor			Descripción	No. de bytes
B6h	03h				Etiqueta y longitud de la plantilla DST.	2
		83h	01h	ID Llave	TLV del identificador de la llave.	3
DFh 25h	02h	Longitud máxima de la llave			TLV de la longitud máxima de la llave. 08h 00h indica 2048 bytes 04h 00h indica 1024 bytes	5
8Ch	02h				Etiqueta y longitud del atributo de seguridad (bytes AM y SC).	2
		82h			Byte modo de acceso.	1
		27h			Byte condición de seguridad para PUT DATA (el valor 27 h indica que EXTERNAL AUTHENTICATION es requerida).	1
					Total de bytes	**14**

Tabla 60 Ejemplo de data: PUT DATA -Creación de llave pública

Ejemplo de estructura para los elementos módulo y exponente:

Etiqueta	Longitud	Valor			Descripción	No. de bytes
A4h o B6h	03h				Etiqueta o longitud de la plantilla AT y DST.	2
		83h	01h	IDLlave	TLV de llave pública.	3
7Fh 49h	06h				Etiqueta y longitud de la plantilla de llave pública.	3

		Etiqueta	Longitud	Valor		
		81h	8180h	Var.	TLV del módulo (valor codificado en 128 bytes).	131
		82h	04h	Primo[4]	TLV del exponente (valor codificado en 4 bytes).	6
					Total de bytes	**145**

Tabla 61 Ejemplo de data: PUT DATA –Escribiendo el valor del módulo y exponente

Los dos elementos (módulo y exponente) son enviados en un comando PUT DATA. En este ejemplo, para una llave de 1024 bits, los datos son 145 bytes, entonces el valor de Lc en el comando es 91h.

Respuesta:

La respuesta para los cinco comandos PUT DATA de la etapa de personalización es agrupada a continuación:

La tarjeta retorna los códigos de estado SW1 y SW2. Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado sin error.
67h	00h	Incorrecta longitud, Lc.
69h	82h	Estado de seguridad no satisfecho, por ejemplo: El atributo de seguridad no ha sido satisfecho. Error durante la mensajería segura.
69h	85h	Condiciones de uso no satisfechas, por ejemplo: Intentos para crear un *PIN* con un ID de *PIN* que ya existe. Llave secreta o privada ya existen (cuando se está intentando crear la llave). Llave secreta o privada no existe (cuando está tratando de escribir el valor de un elemento). El *PIN* a ser creado está protegido por un *PIN* que todavía no se ha creado.
6Ah	80h	Datos incorrectos, por ejemplo, incorrecta etiqueta o longitud.
6Ah	84h	Memoria insuficiente (número máximo permitido de llaves privadas excedido).

[4]El exponente puede ser cualquier valor pero debe ser un número primo. El número no puede exceder los 8 bytes de longitud

6Ah	86h	Incorrectos parámetros P1 y P2.
6Ah	88h	Datos de referencias no encontrados (Etiquetas especificadas en P1 y P2).

Tabla 62: Posibles valores para SW1 y SW2

Anexos 28: COMANDO SELECT FILE

El comando SELECT FILE selecciona un archivo DF o EF.

El formato del comando es el siguiente:

CLA	INS	P1	P2	Lc	Data	Le
CLA	A4h	RefCtrl	RefCtrl	Lc	Data	Le

Tabla 63: Descripción del comando SELECT FILE

Donde:

CLA: 00h Transmite normalmente

04h Transmite con mensajería segura

P1: Es un parámetro de control de referencia. La configuración de bits es la siguiente:

P1	**b8**	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**
RFU.	0	0	0	0	x	x	x	x
Seleccionar por la ruta desde el MF.					1	0	0	0
Seleccionar EF por el ID del archivo bajo el DF actual.					0	0	1	0
Seleccionar MF, DF o EF por el ID del archivo.					0	0	0	0

Tabla 64: ejemplo de la configuración de bits para P1

P2: Es un parámetro de control de referencia que especifica el formato de la respuesta o que no existe respuesta. La configuración de bits es la siguiente:

P2	**b8**	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**
RFU.	0	0	0	0				
Opción FCI.					x	x		
Devolver plantilla FCI.					0	0		
Devolver plantilla FCP.					0	1		
Sin respuesta.					1	1		
Primera o única ocurrencia del archivo.							0	0

Tabla 65: Ejemplo de la configuración de bits para P2

Lc: es la longitud del campo *Data* como se indica a continuación:

02h Para la selección del archivo usando el ID.

02h o 04h Para la selección por camino.

Data: contiene la referencia del archivo y es uno de los siguientes:

ID del archivo: identificador del archivo a ser seleccionado.

Camino: es la concatenación de los ID del MF o DF actual hasta el ID del archivo a ser seleccionado, no incluyendo el ID del MF o DF actual.

Le: es la longitud de los datos que deben ser devueltos, y está determinado por la configuración de P2.

Respuesta:

La tarjeta retorna los datos solicitados por el parámetro de control de referencia P2, seguido por los códigos de estado SW1 y SW2. Los datos posibles de respuesta son: la plantilla FCI, la plantilla FCP o no existe (P2 = 0Ch).

La plantilla FCP contiene los parámetros de control del archivo que fueron especificados cuando fue creado el archivo.

La información FCI retornada por el comando (P2 = 00h) es mostrada a continuación:

Desplazamiento	Datos	Descripción
0	6Fh	Etiqueta de la plantilla FCI.
1	L	Longitud de la data FCI.
2	83h	Etiqueta del ID del archivo.
3	02h	Longitud del ID del archivo.
4-5	ID del archivo	Valor del ID del archivo.
6	8Ch	Etiqueta de atributos de seguridad.
7	L	Longitud de atributos de seguridad.
8	AM	Byte Modo de Acceso.
9-(8+X)	SC	Bytes Condición de Seguridad (X).
9+X	84h	Etiqueta del nombre del DF.
10+X	L	Longitud del nombre del DF.
11+X-	Nombre del DF	Valor del nombre del DF (hasta 16 bytes).

Tabla 66: FCI para DFs

Desplazamiento	Datos	Descripción
0	6Fh	Etiqueta de la plantilla FCI.

1	L	Longitud de la data FCI.
2	81h	Etiqueta de tamaño de archivo.
3	02h	Longitud de tamaño de archivo.
4-5	Tamaño del archivo	Valor de tamaño de archivo.
6	82h	Etiqueta de FDB.
7	01h	Longitud de FDB.
8	FDB	Valor de FDB.
9	83h	Etiqueta del ID del archivo.
10	02h	Longitud del ID del archivo.
11-12	ID del archivo	Valor del ID del archivo.
13	8Ah	Etiqueta del byte de estado del ciclo de vida del archivo.
14	01h	Longitud del byte de estado del ciclo de vida del archivo.
15	Var	Valor del byte de estado del ciclo de vida del archivo.
16	8Ch	Etiqueta de atributos de seguridad.
17	L	Longitud de atributos de seguridad.
18	AM	Byte Modo de Acceso.
19-(18+X)	SC	Bytes Condición de Seguridad (X).

Tabla 67: FCI para EFs

Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado sin error. No retornados datos FCI.
67h	00h	Longitud incorrecta de Lc.
69h	82h	Estado de seguridad no satisfecho, error durante el envío de mensaje seguro.
6Ah	82h	Archivo no encontrado.

6Ah	86h	Incorrectos parámetros P1 y P2.

Tabla 68: Posibles valores para SW1 y SW2

Anexos 29: COMANDO GET CHALLENGE

Este comando genera una respuesta que es usada por el comando MUTUAL AUTHENTICATE (para la autenticación mutua simétrica). El número aleatorio generado por el comando es válido solamente para el próximo comando.

Este comando es ejecutado sin mensajería segura

El formato del comando es el siguiente:

CLA	INS	P1	P2	Le
80h	84h	00h	00h	08h

Tabla 69: Descripción del comando GET CHALLENGE

Respuesta:

La respuesta es un reto de ocho bytes RND.ICC

Después de la respuesta, la tarjeta retorna los códigos de estado SW1 y SW2. Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado sin error.
67h	00h	Longitud incorrecta.
69h	82h	Condiciones de seguridad no satisfechas.
69h	86h	Comando solo soportado en fase de aplicación.

Tabla 70: Posibles valores para SW1 y SW2

Anexo 30: COMANDO GET DATA

Este comando recupera uno de los siguientes conjuntos de datos:

- Los datos de personalización del *applet* actualizados con el comando END PERSONALIZATION.
- El contenido de un Entorno de seguridad especificado.

La respuesta de los datos en formato TLV para los datos de personalización y de un entorno de seguridad.

El formato del comando es el siguiente:

CLA	INS	P1	P2	Lc	Data	Le
CLA	INS	P1	P2	Lc	Data	Le

Tabla 71: Descripción del comando GET DATA

Los valores dependen del tipo de objeto a ser recuperado. La siguiente tabla muestra los posibles valores:

Objetos de datos	INS	P1	P2	Lc	Data	Le
Datos de personalización del *applet.*	CAh	DFh	32h	Lc	Estructura TLV	Le
Datos de un Entorno de Seguridad o Llave pública.	CBh	00h	FFh	Lc	Estructura TLV	Le

Tabla 72: Valores de GET DATA

Respuesta:

La respuesta es codificada como se muestra. Para los datos de personalización del *applet*

Etiqueta	Longitud	Valor	Descripción
DF32h	Eh	'AAAAMMDDhhmmss'	Fecha de personalización del *applet* codificada en ASCI.

Tabla 73: Estructura de respuesta de GET DATA – Datos de personalización

Para los datos de un Entorno de Seguridad (Ejemplo que contiene un *PIN* solamente en el AT):

Etiqueta	Longitud	Valor		
7B	03	80	01	ID

Tabla 74: Ejemplo que contiene un PIN solamente en el AT

Respuesta:

Etiqueta	Longitud	Valor				Descripción
7Bh	Lse					Etiqueta y longitud del SE actual.
		Etiqueta	**Longitud**	**Valor**		Descripción
		A4h	06h			Etiqueta y longitud del AT.
			Etiqueta	**Longitud**	**Valor**	
			A4h	06h	Referencia del *PIN*	Etiqueta y longitud del AT.
			95h	01h	08h	Calificador de uso CRT (valor 08h

				indica autenticación del usuario).

Tabla 75: Estructura de respuesta de GET DATA – Entorno de Seguridad

Para obtener una llave pública entera:

Etiqueta	**Longitud**	**Valor**			**Descripción**			
B6h	03				Etiqueta y longitud de la plantilla de firma a la que pertenece la llave.			
		Etiqueta	**Longitud**	**Valor**	Descripción			
		83h	01h	ID	Etiqueta y longitud e id de la llave.			
79h 4Fh				**Etiqueta**	**Longitud**	**Valor**		
	Tag							
	80h						Etiqueta para obtener la llave entera.	

Tabla 76: Para obtener una llave pública entera

Respuesta:

Etiqueta	**Longitud**	**Valor**			**Descripción**		
B6h	03				Etiqueta y longitud de la plantilla de firma a la que pertenece la llave.		
		Etiqueta	**Longitud**	**Valor**	Descripción		
		83h	01h	ID	Etiqueta y longitud e id de la llave.		
79h 4Fh	L			**Etiqueta**	**Longitud**	**Valor**	
				81h	Lmod	módulo	TLV del módulo.
				82h	Lexp	exponente	TLV del exponente.

Tabla 77: Respuesta para obtener una llave pública entera

Los posibles valores para SW1 y SW2 son los siguientes:

SW1	**SW2**	**Descripción**
90h	00h	Comando procesado sin error.

69h	82h	Estado de seguridad no satisfecho, error durante mensajería segura.
6Ah	86h	Incorrectos parámetros P1 y P2.

Tabla 78: Posibles valores para SW1 y SW2

Anexo 31: COMANDO MUTUAL AUTHENTICATE

Este comando permite a la tarjeta y el terminal autenticarse el uno con el otro, verificando la presencia de las dos llaves secretas K.ENC y K.MAC. Para adicionar seguridad los valores deK.ENC y K.MAC almacenados en la tarjeta pueden ser diversificados.

Es necesario que haya sido enviado un comando GET CHALLENGE antes de este comando con el fin de obtener el RND.ICC.

El formato del comando es el siguiente:

CLA	INS	P1	P2	Lc	Data	Le
80h	82h	00h	00h	00h o 48h	Data	48h

Tabla 79: Descripción del comando MUTUAL AUTHENTICATE

Donde:

Lc: es la longitud de la *Data* (48h = 72 bytes).

Data: es S' || MAC

Donde:

S = RND.IFD || SN.IFD || RND.ICC || SN.ICC || TRnd

RND.IFD: es un reto de 8 bytes generados por la terminal.

SN.IFD: es el número de serie del terminal.

RND.ICC: es un reto de 8 bytes generados por la tarjeta.

SN.ICC: es el número de serie del chip.

TRnd: es un número aleatorio de 32 bytes generado por el terminal.

Le: es la longitud de la respuesta (siempre 72 bytes (48h))

Respuesta:

Donde la Data es SS' || MAC.

Después de la respuesta, la tarjeta retorna los códigos de estado SW1 y SW2. Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado sin error.
67h	00h	Incorrecta longitud.
69h	85h	Condiciones del comando no satisfechas.
6Ah	80h	Datos no encontrados.

6Ah	86h	Incorrectos parámetros P1 y P2.

Tabla 80 Ejemplo de data: MUTUAL AUTHENTICATE

Anexo 32: COMANDO MSE: Set

Este comando actualiza un CRT en el Entorno de Seguridad actual.

En la etapa de aplicación, este comando puede ser utilizado antes o después de la autenticación mutua. La mensajería segura es opcional.

El formato del comando es el siguiente:

CLA	INS	P1	P2	Lc	Data
CLA	22h	41h	P2	Lc	Data

Tabla 81: Descripción del comando MSE: Set

Donde

CLA: 00h Transmite normalmente.

0Ch Transmite con mensajería segura.

P2: es la etiqueta del CRT a ser actualizado como se muestra:

A4h Plantilla de autenticación (AT).

B6h Plantilla de la firma digital (DST).

Lc: es la longitud del campo Data.

Data: es la concatenación de los objetos de datos de control de referencia (CRDOs) a ser actualizados. No puede exceder los 30 bytes.

CRDOs: son elementos contenidos en el CRT.

Respuesta:

Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado sin error.
67h	00h	Incorrecta longitud.
69h	82h	Estado de seguridad no satisfecho, error durante la mensajería segura.
6Ah	86h	Incorrectos parámetros P1 y P2.
6Ah	80h	Datos de plantilla incorrectos.

Tabla 82: Posibles valores para SW1 y SW2

Anexo 33: COMANDO PUT DATA

Este comando permite actualizar los siguientes objetos de datos en la etapa de aplicación:

- Llaves secretas
- Llaves públicas RSA

- Llaves privadas RSA

Nota: *Para actualizar un objeto de tipo PIN deben usarse los comandos CHANGE REFERENCE DATA o RESET RETRY COUNTER.*

Las condiciones de uso y la estructura de los comandos para cada tipo de objeto son diferentes, por lo que son descritas por separado.

COMANDO PUT DATA Llaves secretas

Estas llaves son usadas durante la autenticación mutua simétrica, por lo que deben ser conocidas por el terminal y deben ser únicas para cada tarjeta. Estas llaves son creadas e inicializadas durante la etapa de personalización. En la etapa de aplicación el comando PUT DATA debe ser usado solo para actualizar el valor de las llaves.

El formato del comando es el siguiente:

CLA	INS	P1	P2	Lc	Data
CLA	DBh	00h	FFh	Lc	Data

Tabla 83: Descripción del comando PUT DATA Llaves secretas

Donde:

CLA: 00h Transmite el final o solo una porción de datos con mensajería segura y ejecuta el comando.

1Ch: Transmite la primera porción de datos en el modo cadena con mensajería segura y espera por la porción final.

Lc: indica la longitud del campo de datos

Data: es codificado en el siguiente **TLV** como se muestra:

Etiqueta	Longitud	Valor	Descripción	No. de bytes
A4h	03h		Uso de la llave: Plantilla de autenticación.	2
		83 01 xxh	Donde xx= idLlave.	3
DF23h	10h	K.ENC	Valor de K.ENC.	19
DF24h	10h	K.MAC	Valor de K.MAC	19
Total de bytes				**43**

Tabla 84 Ejemplo de data: PUT DATA -Actualización de llave secreta

En este ejemplo, los datos son 43 bytes, por lo que el valor de Lc es 2Bh

COMANDO PUT DATA Llave privada

Esta llave es creada e inicializada durante la etapa de personalización. En la etapa de aplicación el comando PUT DATA debe ser usado solo para actualizar el valor de la llave.

El formato del comando es el siguiente:

CLA	INS	P1	P2	Lc	Data
CLA	DBh	00h	FFh	Lc	Data

Tabla 85: Descripción del comando PUT DATA Llave privada

Donde:

CLA: 00h Transmite el final o solo una porción de datos con mensajería segura y ejecuta el comando.

1Ch Transmite la primera porción de datos en el modo cadena con mensajería segura y espera por la porción final.

Lc: indica la longitud del campo Data.

Data: se describe a continuación.

Data para llaves privadas

Etiqueta	**Longitud**	**Valor**			**Descripción**	**No. De Bytes**
A4h o B6h	03h				AT(llave de autenticación SK.ICC.AUT) DST (firma digital y/o llave de utilidad).	2
		84h	01h	XXh	TLV del ID de la llave privada (XXh es el valor del ID de la llave).	**3**
7Fh 48h	XXh	Etiqueta y longitud de los elementos privados				**3**
		Etiqueta	**Longitud**	**Valor**		
		92h	Variable	Mod	Módulo de la llave privada.	**128**
		93h	Variable	Exp	Exponente de la llave privada.	**128**
					Total de bytes	**264**

Tabla 86 Ejemplo de data: PUT DATA -Actualización de llave privada

En el ejemplo, para una llave de 1024 bits (el módulo y el exponente son de 128 bytes)

COMANDO PUT DATA Llaves Públicas

El formato del comando es el siguiente:

CLA	INS	P1	P2	Lc	Data
CLA	DBh	00h	FFh	Lc	Data

Tabla 87: Descripción del comando PUT DATA Llaves Públicas

Donde:

CLA: 00h Transmite el final o solo una porción de datos con mensajería segura y ejecuta el comando.

1Ch Transmite la primera porción de datos en el modo cadena con mensajería segura y espera por la porción final.

Lc: indica la longitud de del campo Data.

Data: se describe a continuación.

Data para llaves públicas

Etiqueta	Longitud	Valor			Descripción	No. De Bytes
A4h o B6h	03h				Etiqueta y longitud de la plantilla AT o DST.	2
		83h	01h	ID Llave	TLV de la llave pública.	3
7Fh 49h	06h	Etiqueta y longitud de los elementos privados.				3
		Etiqueta	**Longitud**	**Valor**		
		81h	8180h	Var.	TLV del módulo (valor codificado en 128 bytes).	131
		82h	04h	Primo[5]	TLV del exponente (valor codificado en 4 bytes).	**6**
					Total de bytes	145

Tabla 88 Ejemplo de data: PUT DATA -Actualización de llave pública

[5] El exponente puede ser cualquier valor pero debe ser un número primo. El número no puede exceder los 8 bytes de longitud

Los dos elementos (módulo y exponente) son enviados en un comando PUT DATA. En este ejemplo, para una llave de 1024 bits, los datos son de 145 bytes, por lo que el valor de Lc es de 91h.

Respuesta:

La respuesta para los tres comandos PUT DATA de la etapa de aplicación es agrupada a continuación.

La tarjeta retorna los códigos de estado SW1 y SW2. Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado sin error.
67h	00h	Incorrecta longitud, Lc. Lc debe ser menos de 230 bytes cuando use mensajería segura en la etapa de aplicación.
69h	82h	Estado de seguridad no satisfecho, por ejemplo: El atributo de seguridad no ha sido satisfecho. Error durante la mensajería segura.
69h	85h	Condiciones de uso no satisfechas, por ejemplo: Llave secreta o privada no existe (cuando está tratando de escribir el valor de un elemento). Incorrecto estado del ciclo de vida del *applet.*
6Ah	86h	Incorrectos parámetros P1 y P2.
6Ah	80h	Datos del comando incorrectos.

Tabla 89: Respuesta para los tres comandos PUT DATA de la etapa de aplicación

Anexo 34: COMANDO VERIFY

Este comando autentica a un usuario comparando el *PIN* introducido en el comando (*PIN* de verificación) con el *PIN* de referencia.

Si el comando VERIFY tiene éxito, ocurren las siguientes acciones:

- El contador de intentos del *PIN* de referencia es actualizado con el número de intentos.
- La bandera de validación del *PIN* de referencia es actualizada a verdadero.
- El contador de uso del *PIN* de referencia es decrementado en 1, a menos que su valor sea FFh (no existen límites en el número de veces que el *PIN* puede ser usado) o 00h (ya ha sido usado el máximo de veces permitido).

Por otra parte, si el comando VERIFY falla, ocurren las siguientes acciones:

- El contador de intentos del *PIN* de referencia es decrementado en 1.
- La bandera de validación del *PIN* de referencia es actualizada a falso.

El formato del comando es el siguiente:

CLA	INS	P1	P2	Lc	Data
CLA	20h	00h	P2	Lc	*PIN* Verificación.

Tabla 90: Descripción del comando VERIFY

Donde:

CLA: 00h Transmite normalmente

0Ch Transmite con mensajería segura

P2: es la referencia al *PIN* que se desea verificar (81h-8Fh)

Lc: 00h permite averiguar si el *PIN* ya se ha presentado de manera exitosa.

08h-10h Debe ser igual a L_{PIN} (longitud del *PIN*).

Data: contiene el valor del *PIN* de verificación (8-16 bytes)

Respuesta:

Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado sin error. No retornados datos FCI.
63h	Cxh	*PIN* de referencia no verificado. X intentos restantes.
67h	00h	Longitud incorrecta de Lc. Lc debe ser menos de 248 bytes cuando usa mensajería segura.
69h	83h	Método de autenticación bloqueado (*PIN* de referencia bloqueado).
69h	84h	Límite de intentos o Contador de uso del *PIN* de referencia ha alcanzado cero.
6Ah	86h	Incorrectos parámetros P1 y P2.
6Ah	88h	Datos referenciados no encontrados.

Tabla 91: Posibles valores para SW1 y SW2

Anexo 35: COMANDO PSO-HASH

Este comando almacena en la tarjeta los datos que se firmarán posteriormente.

El formato del comando en texto plano es el siguiente:

CLA	INS	P1	P2	Lc	Data
00h	2Ah	90h	0Ah	14h	**Hash**

Tabla 92: Descripción del comando GET CHALLENGE

Hash: 20 bytes que representan el hash (SHA1) de los datos que se desea firmar posteriormente.

Respuesta:

Después de la respuesta, la tarjeta retorna los códigos de estado SW1 y SW2. Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado sin error.
67h	00h	Longitud incorrecta.

Tabla 93: Posible valores para SW1 y SW2

Anexo 36: COMANDO PSO-Compute Digital Signature

Este comando firma un hash previamente enviado a la tarjeta y devuelve el resultado.
El formato del comando en texto plano es el siguiente:

CLA	INS	P1	P2
00h	2Ah	9Eh	9Ah

Tabla 94: Descripción del comando GET CHALLENGE

Respuesta:

Después de la respuesta, la tarjeta retorna los códigos de estado SW1 y SW2. Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
61h	XXh	Comando procesado sin error con XX bytes de respuesta.
67h	00h	Longitud incorrecta.
69h	85h	En el caso de que no se halla enviado un hash previamente o no se cumplan las condiciones de seguridad necesarias para utilizar la llave de firma seleccionada.

Tabla 95: Posibles valores para SW1 y SW2

Anexo 37: Descripción de la clase ISO7816FileSystem

Nombre: ISO7816FileSystem	
Tipo de clase: controladora	
Atributo	**Tipo**
TAG_FCI	byte
TAG_FILE_SIZE	byte
TAG_SECURITY_ENVIRONMENT_ID	byte
TAG_SECURITY_ENVIRONMENT_LCS	byte
TAG_CRT_AT	byte
TAG_CRT_KAT	byte
TAG_CRT_CCT	byte
TAG_CRT_DST	byte
TAG_CRT_CT	byte
TAG_PUT_DATA_PIN	byte
TAG_PUT_DATA_SECRETKEY	byte

TAG_PUT_DATA_PUBLICKEY	byte
_MasterFile	ISO7816File
_SelectedFile	ISO7816File
_SelectSecurityEnvironment	SecurityEnvironment
_SecurityEnviromentCollection	List<SecurityEnviroment>
_DataObjectCollection	List<DataObject>
_sesion	SessionManager
_fileSystem	ISO7816FileSystem

Para cada responsabilidad:	
Nombre:	ISO7816FileSystem()
Descripción:	Crea un objeto de la clase
Nombre:	getInstance()
Descripción:	Obtiene o crea la única instancia posible de la clase.
Nombre:	getSession()
Descripción:	Obtiene el objeto de la sesión actual
Nombre:	Init()
Descripción:	Inicializa el Sistema de ficheros creando el DF raíz o MF(Master file)
Nombre:	getSelectedFile()
Descripción:	Obtiene el fichero seleccionado
Nombre:	getSelectedSecurityEnvironment()
Descripción:	Obtiene el entorno de seguridad seleccionado
Nombre:	selectFileID(byte [] bufferData, short offsetFileID, short lenFileID)
Descripción:	Selecciona el fichero indicado por un identificador debajo del DF seleccionado
Nombre:	selectFileByPath(byte [] bufferData, short offsetFilePath, short lenFilePath)
Descripción:	Selecciona el fichero indicado por el camino proporcionado partiendo del MF(Master file) o del DF seleccionado
Nombre:	verifyFileID(byte [] bufferData, short offsetFileID, short lenFileID)
Descripción:	Verifica si existe el fichero con el identificador proporcionado, debajo del DF seleccionado.
Nombre:	createFile(byte [] bufferData, short offsetDataBuffer, short lenDataBuffer,boolean sensitive)
Descripción:	Crea un fichero, puede ser un DF o un EF
Nombre:	verifyFileCreation(byte[] bufferData, short offsetDataFCI, short FCIIDOffset)
Descripción:	Verifica si se puede crear un fichero debajo del DF seleccionado
Nombre:	readData(byte[] bufferData, short offsetReadData, short lengthReadData, boolean verify)
Descripción:	Lee una cantidad de bytes del EF seleccionado
Nombre:	updateBinary(byte[] bufferData, short offsetFileUpdateData, short lengthUpdateData,boolean verify)
Descripción:	Actualiza una cantidad de bytes a partir de una posición dada en el EF seleccionado
Nombre:	createOrUpdateSecretKey(byte[] bufferData,short offsetPutData,short lengthPutData,BERTLVParser Data)
Descripción:	Crea si no existe o actualiza en caso de que exista un par de llaves secretas o privadas
Nombre:	createOrUpdatePublicKey(byte[] bufferData,short offsetPutData,short lengthPutData,BERTLVParser Data)
Descripción:	Crea si no existe o actualiza en caso de que exista una llave pública
Nombre:	securityConditionVerify(byte SecurityConditionByte)
Descripción:	Verifica si se cumplen las condiciones de seguridad para utilizar un objeto o un fichero
Nombre:	VerifyUserAuthentication(byte se)
Descripción:	Verifica si existe algún PIN autenticado en el entorno de seguridad dado

Nombre:	getDOContainer(byte id,byte tag)
Descripción:	Obtiene el contenedor de una llave o un PIN
Nombre:	existTag(byte tag,BERTLVParser Data)
Descripción:	Verifica si existe un tag dado en un Objeto de Datos
Nombre:	selectSecurityEnvironment(byte idSecurityEnvironment)
Descripción:	Selecciona un entorno de seguridad
Nombre:	createSecurityEnvironment(byte[] bufferData, BERTLVParser dataComponentSE)
Descripción:	Crea un entorno de seguridad
Nombre:	createPinDataObject(byte[] bufferData, short offsetDO, short lenghtDO,BERTLVParser data)
Descripción:	Crea un objeto PIN
Nombre:	createNonSecurityDO(byte[] bufferData, short offsetDataSDO, short lenghtDO)
Descripción:	Crea un Objeto de Datos que no sea PIN, llave o entorno de seguridad
Nombre:	getSecurityEnvironment(byte id)
Descripción:	Obtiene un entorno de seguridad
Nombre:	SelectMF()
Descripción:	Pone como fichero seleccionado al fichero raíz o MF

Tabla 96: Descripción de la clase ISO7816FileSystem

Anexo 39: Caso de prueba de caja Blanca de un segmento de código correspondiente a las historias de usuario Almacenar certificado y Obtener certificado.

De acuerdo al siguiente segmento de código correspondiente a las historias de usuario "*Almacenar certificado*" y "*Obtener certificado*", se realiza la prueba de caja blanca.

```
Public void selectFileID(byte [] bufferData, short offsetFileID, short lenFileID)
{
ISO7816File file = null;(1)
if(_SelectedFile == null)(2)
        ISOException.throwIt(ISO7816.SW_FILE_INVALID);(3)
if(lenFileID!=2)(4)
ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);(5)
if (Util.arrayCompare(bufferData, (short)offsetFileID, ISO7816Util._MF_ID, (short)0,
(short)lenFileID) == 0)(6)
{
file = _MasterFile; (7)
        }
else
{
        if (_SelectedFile.getISO7816FileType())(8)
{
file = _SelectedFile.getISO7816File(bufferData, offsetFileID, lenFileID);(9)
```

```
        }
        else
{
        file = _SelectedFile.getParent().getISO7816File(bufferData, offsetFileID, lenFileID);(10)
        }
if(file == null)(11)
        ISOException.throwIt(ISO7816.SW_FILE_INVALID);(12)
}
_SelectedFile = file; (13)
}(14)
```

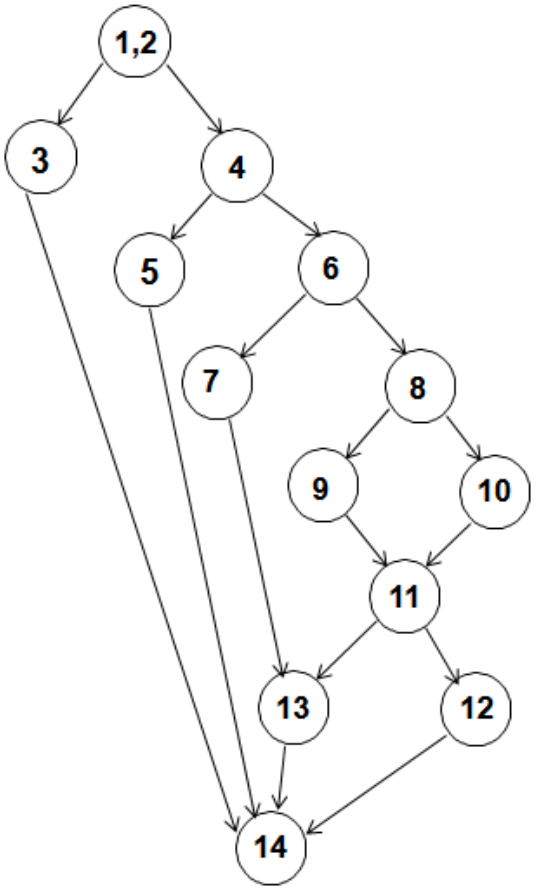


Figura 14: Grafo de flujo del caso de prueba seleccionar fichero.

V (G) = A – N + 2

V (G) = P + 1

A: *Número* de aristas del grafo.

N: *Número de n*odos.

P: *Número* de nodos predicados.

A=17

N=13

P=5

V (G) = 6

Caminos:

1, 2, 3, 14

1, 2, 4, 5, 14

1, 2, 4, 6, 7, 13, 14

1, 2, 4, 6, 8, 9, 11, 13, 14

1, 2, 4, 6, 8, 10, 11, 13, 14

1, 2, 4, 6, 8, 10, 11, 12, 14

Complejidad ciclomática.

V (G): *Número de regiones del grafo.*

Casos de prueba para cada camino**.**

Camino: 1, 2, 3, 14.

Entrada: Arreglo de bytes que tiene a partir de la posición *offsetFileID* un identificador de *lengthFileID* bytes de longitud.

Salida: Excepción especificando que no se puede seleccionar el fichero con el id especificado porque no hay ningún fichero seleccionado en ese momento.

Precondiciones: Ninguna.

Camino: 1, 2, 4, 5, 14.

Entrada: Arreglo de bytes que tiene a partir de la posición *offsetFileID* un identificador de *lenFileID* bytes de longitud donde *lenFileID es diferente de 2*.

Salida: Excepción especificando que la longitud del id es incorrecta.

Precondiciones: Debe haber un fichero seleccionado.

Camino: 1, 2, 4, 6, 7, 13, 14.

Entrada: Arreglo de bytes que tiene a partir de la posición *offsetFileID* un identificador de *lenFileID* bytes de longitud donde el id es igual a 3F00.

Salida: Se selecciona el fichero raíz o MF.

Precondiciones: Debe haber un fichero seleccionado.

Camino: 1, 2, 4, 6, 8, 9, 11, 13, 14.

Entrada: Arreglo de bytes que tiene a partir de la posición *offsetFileID* un identificador de *lenFileID* bytes de longitud donde *lenFileID es igual a 2 y el identificador es distinto de 3F00*.

Salida: Se selecciona el fichero con el id especificado.

Precondiciones: Debe haber un DF seleccionado.

Camino: 1, 2, 4, 6, 8, 10, 11, 13, 14.

Entrada: Arreglo de bytes que tiene a partir de la posición *offsetFileID* un identificador de *lenFileID* bytes de longitud donde *lenFileID es igual a 2 y el identificador es distinto de 3F00*.

Salida: Se selecciona el fichero con el id especificado.

Precondiciones: Debe haber un EF seleccionado.

Camino: 1, 2, 4, 6, 8, 10, 11, 12, 14.

Entrada: Arreglo de bytes que tiene a partir de la posición *offsetFileID* un identificador de *lenFileID* bytes de longitud donde *lenFileID es igual a 2 y el identificador es distinto de 3F00*.

Salida: Excepción especificando que el fichero no se encontró.

Precondiciones: Debe haber un fichero seleccionado.