Universidad de las Ciencias Informáticas

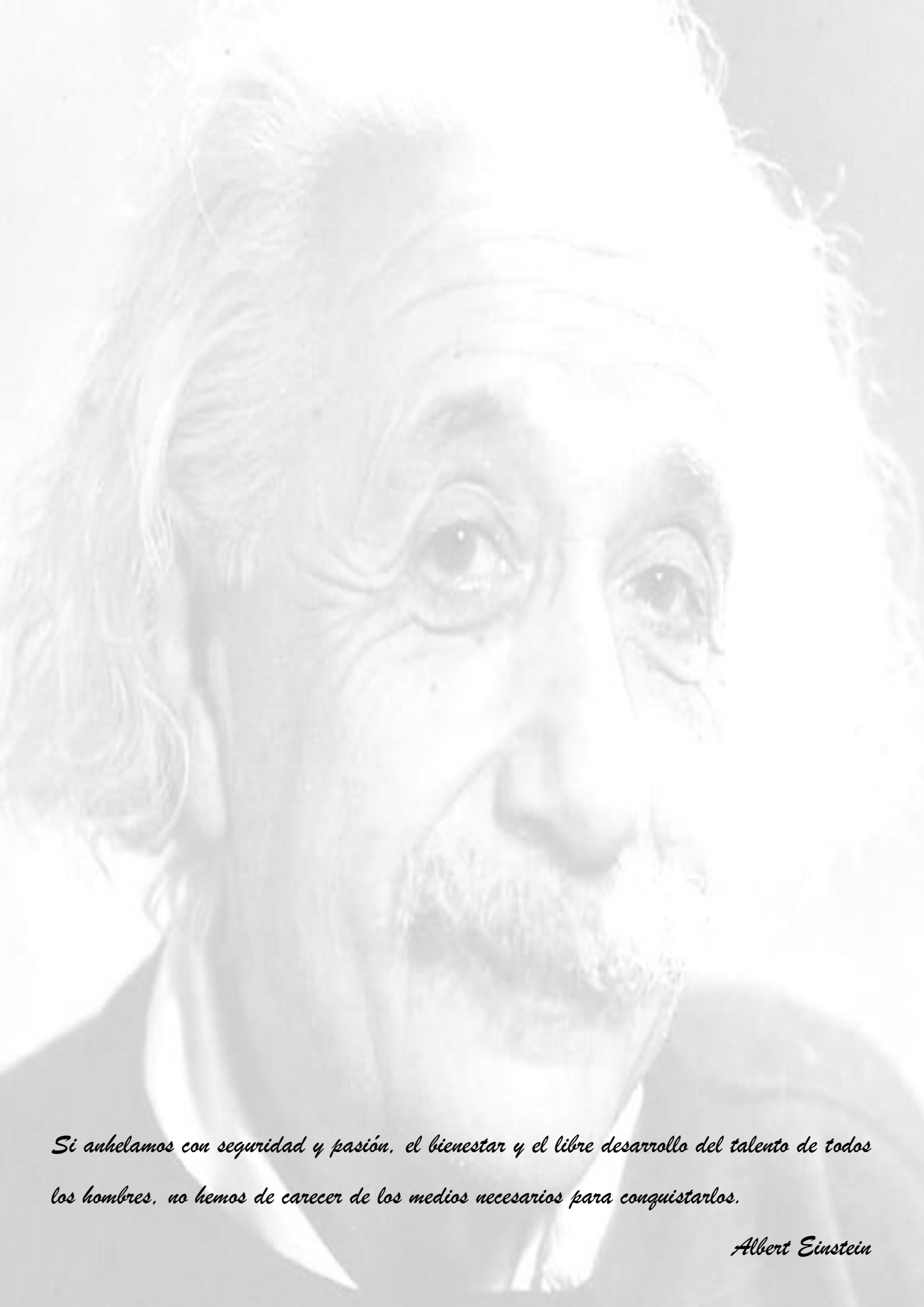
Aplicación para gestionar la información biométrica de las huellas dactilares en tarjetas inteligentes

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

Autora:
Dariana Lago Batista

Tutores: Ing. Dayron Almeida Sotolongo
Ing. Iliana Morera González

La Habana, Cuba
Año 54 de la Revolución



Si anhelamos con seguridad y pasión, el bienestar y el libre desarrollo del talento de todos los hombres, no hemos de carecer de los medios necesarios para conquistarlos.

Albert Einstein

Declaración de autoría

Declaro que soy la única autora del trabajo titulado: *"Aplicación para gestionar la información biométrica de las huellas dactilares en tarjetas inteligente*s" y autorizo a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los días _____ del mes _____ del año _____.

Dariana Lago Batista

Nombre del estudiante

Dayron Almeida Sotolongo

Nombre del tutor

Iliana Morera González

Nombre del tutor

Opinión del tutor del trabajo de diploma

Dedicatoria

A mis padres por ser mi apoyo, mi fuerza, mi ejemplo a seguir, mis guías incondicionales en el cursar de mi vida.

A mi hermanita por ser mi regalo más grande.

A ustedes mi familia, dedico mis triunfos y sacrificios, por ser las personas más importantes y hacer que mi mundo tenga sentido.

Agradecimientos

A mami, por estar ahí siempre para mi, por darme su apoyo, por enseñarme que en la vida todo lo que nos proponemos lo podemos lograr, por esa fuerza de voluntad que me ha inspirado siempre a ser una mejor persona y, por sobre todas las cosas, por todo el amor que me has dado, el cual me ha impulsado a seguir adelante día a día. Por ser mi más grande inspiración y mí musa.

A papi, por todo el amor que me das, por enseñarme que en la vida todo se gana con esfuerzo y sacrificio, por ser mí ejemplo y enseñarme a crecerme en los momentos difíciles. Por tenerme presente siempre y estar ahí para mí.

A mi hermanita, por ser mi más preciado tesoro, por sus consejos que me han ayudado mucho. Por llegar a mi vida cuando más te necesitaba.

A mi tía Leti, por ser la luz de la fuerza que me ha ayudado a levantarme siempre que he caído.

A mis abuelos: Yolanda, Nancy, Martiniano y Paco, por formar una parte importante en mi vida, por consentirme cuando era pequeña.

A mi primita María Laura, por su dulzura y cariño.

A mis primos Ale, Aldito, Juan Carlito, Carla, Camila y María Carla.

A Felipe por su apoyo y consejos, por enseñarme a no temer al cambio y ser el culpable de mis alegrías.

A mis amigos por estar siempre para mí en los buenos y malos momentos. A Lisy, Dani, Ari, Elizabeth, Yucelia, Arlenis, Lari, Lili, Ki y Albert, gracias por su apoyo incondicional. En especial a Yiri, por convertirse en una hermanita para mí y ayudarme siempre hasta en lo que no sabía.

Resumen

El mundo tecnológico de hoy está en constante cambio, por lo que se hace necesario ir revolucionando los sistemas, de forma tal que estos se adecúen a los adelantos tecnológicos. Las tarjetas inteligentes constituyen uno de los avances que han venido a transformar en gran medida muchos de los procesos de la sociedad, como por ejemplo, la gestión de la información, permitiendo a todo aquel que la utiliza un manejo mucho más rápido y sencillo. Estas pueden ser empleadas como dispositivos para gestionar información, como puede ser la relacionada con la biometría dactilar: técnica utilizada para verificar la autenticidad de las personas.

En la Universidad de las Ciencias Informáticas (UCI) existen varios centros de desarrollo de software, entre los que se encuentra el Centro de Identificación y Seguridad Digital (CISED). El departamento Tarjetas Inteligentes perteneciente a dicho centro, se dedica a desarrollar aplicaciones de este tipo, el cual a partir de la experiencia adquirida en el área de tarjetas inteligentes, ha identificado la necesidad del desarrollo de una aplicación que permita el uso de las mismas como dispositivos para gestionar la información biométrica dactilar. Esta información será usada por el *applet MoC* que se está desarrollando, el cual permitirá realizar la verificación biométrica dactilar. Dadas las características de la aplicación será posible aplicarla a cualquier *applet* de verificación biométrica.

En el presente trabajo se describen los diferentes estándares, herramientas, tecnologías y artefactos generados en el proceso de desarrollo. Además se recogen los resultados de la investigación realizada, describiéndose las principales características de los sistemas analizados, la arquitectura y el diseño de la solución propuesta.

Palabras clave: *applet***,** gestión de la información biométrica dactilar, tarjeta inteligente.

Tabla de contenido

Índice de tablas

Índice de figuras

Introducción

Hoy día el mundo se encuentra inmerso en un complejo proceso de construcción y perfeccionamiento de la sociedad tecnológica y como parte de este proceso evolutivo se hace primordial transformar y actualizar los viejos sistemas de orden político y social, por completas y modernizadas instituciones públicas que basan el flujo de sus procesos en los más novedosos adelantos tecnológicos; concibiendo sistemas más completos que hagan de una simple información todo un reporte u obtengan de una fuente de información una respuesta rápida y efectiva. La necesidad de gestionar dicha información trajo consigo que los métodos conocidos anteriormente para transmitirla y almacenarla, como libros, pergaminos, entre otros, evolucionaran hacia la era digital para alcanzar mayor capacidad y durabilidad.

Las tarjetas inteligentes constituyen uno de los avances tecnológicos que han venido a revolucionar, en gran medida, muchos de los procesos de la sociedad. Estas representan una lámina plástica de tamaño similar a las conocidas tarjetas de crédito, pero difieren de las mismas en que traen incorporados circuitos integrados que permiten almacenar información de forma segura y la ejecución de cierta lógica programada. Además, proporcionan seguridad y exactitud en la verificación de la identidad y cuando se combinan con otras tecnologías de identificación tales como los sistemas de identificación biométrica[1], pueden elevar la seguridad de los sistemas y proteger la privacidad de la información.

La identificación biométrica es esencial, puesto que brinda los beneficios de las características biológicas de no poder ser prestadas o robadas, e intrínsecamente estas características representan la identidad completa de la persona, es decir se le reconoce por su cuerpo y el mismo es enlazado a una identidad externa establecida. (Hanscan Identity Management, 2012)

El interés de la sociedad por utilizar patrones biométricos para identificar o verificar la autenticidad de las personas haya cobrado un aumento vertiginoso en las últimas décadas. Esto ha permitido que los sistemas que basan su funcionamiento en conceptos de biometría dactilar, específicamente, se hagan cada vez más comunes, siendo evidente en pasaportes y cédulas electrónicas, sistemas de acceso a instalaciones, ordenadores y teléfonos móviles por citar algunos ejemplos.

La técnica de biometría dactilar representa legalmente una de las más utilizadas como prueba fidedigna de identidad, siendo un sistema que además de ser efectivo, es cómodo

[1] Sistema de reconocimiento estadístico de patrones que establece la autenticidad de una característica fisiológica o de comportamiento que posee un usuario. (Transparencias, 2000)

de aplicar y la autenticación[2] se obtiene rápidamente, además existen numerosos estudios científicos que avalan la unicidad de la huella de una persona y, lo que es más importante, la estabilidad con el tiempo, la edad, entre otros. En estos aspectos es una técnica que les lleva mucha ventaja a las demás, debido a su siglo de existencia.

De esta forma, para un sistema de autenticación biométrica, una tarjeta inteligente se podría utilizar como:

• Dispositivo seguro que almacena la identidad del usuario, así como su patrón, dejando leer el patrón por parte del terminal que tiene permiso para ello.

• Como otro sistema más de seguridad dentro de la tarjeta inteligente, tal y como se usa, por ejemplo, el PIN (*Personal Identification Number* (Número de Identificación Personal), por sus siglas en inglés). De esta forma, se podría proteger de forma biométrica, no sólo la información almacenada dentro de la tarjeta, sino también determinadas operaciones como por ejemplo, el débito de un monedero electrónico. (Reillo, 2000)

En Cuba existen instituciones que investigan, desarrollan o utilizan la tecnología de tarjetas inteligentes, tales como: Cupet-Címex que utiliza una aplicación para el cobro de combustibles en los servicentros, la Corporación Copextel S.A, que lleva a cabo el desarrollo de un grupo de aplicaciones entre las que se encuentra el Monedero Electrónico y el CISED, perteneciente a la UCI.

El departamento Tarjetas Inteligentes perteneciente a este centro se dedica a desarrollar aplicaciones referentes a esta tecnología, con el fin de obtener productos y servicios basados en tarjetas inteligentes. Entre las soluciones desarrolladas por este se puede mencionar la aplicación para el control de acceso a la información de las entidades externas, en la cédula de identificación electrónica de la República Bolivariana de Venezuela, solución que permitió gestionar el acceso a la información que pudieran almacenar instituciones externas a la Oficina Nacional de Identificación y Extranjería (ONIDEX), actualmente SAIME, mediante la utilización de mecanismos de seguridad basados en certificados digitales. Otra de las soluciones en desarrollo es la aplicación para una infraestructura de clave pública, la cual brindará a los clientes que requieren los más altos estándares de seguridad, la creación y almacenamiento de llaves y certificados digitales en un entorno seguro dentro de la tarjeta.

La evolución de las tarjetas inteligentes ha permitido incorporar técnicas de verificación de identidad, mediante la información biométrica dactilar del portador de dicha tarjeta. Una de las formas más utilizadas es a través de la tecnología biométrica *Match on Card* (Comparación dentro de la tarjeta, por sus siglas en inglés *MoC*), la cual permite realizar

[2] Acto de establecimiento o confirmación de algo (o alguien) como auténtico, es decir probar que es verdadero.

comparaciones de huellas dactilares dentro de las tarjetas inteligentes. Para realizar este proceso es necesario contar con la información biométrica almacenada en la tarjeta, para luego realizar la verificación. En el departamento Tarjetas inteligentes se está desarrollando un *applet* para el *MoC*, el cual necesita para realizar la verificación, obtener las plantillas biométricas de la tarjeta, y para ello es necesario que se gestione previamente la información biométrica dactilar, mediante el *applet uMatchApp.* En caso que no esté disponible esta información, no se podría llevar a cabo la verificación biométrica, proceso para el cual esta es la base fundamental.

Este *applet* llevará por nombre *uMatchApp*, ya que en el departamento se acordó que todos los *applets* realizados comenzaran con la letra *u* que significa UCI y finalizaran con la palabra *App*, que significa *applet*. En este caso el nombre *Match* significa comparación, pues este *applet* es el que permitirá que se realice en un futuro la verificación de las huellas dactilares*,* por lo que se encargará de llevar a cabo todas las funcionalidades requeridas para gestionar la información biométrica dactilar.

A partir de la situación problémica anterior se plantea como **problema de la investigación**: ¿Cómo lograr la gestión de la información biométrica dactilar de manera que pueda ser utilizada por un applet de *MoC*?

En consecuencia el **objeto de estudio** de la presente investigación es: el proceso de gestión de la información biométrica.

Para dar solución al problema existente, se ha definido como **objetivo general**: Desarrollar una aplicación que gestione la información biométrica dactilar, utilizando tecnología *Java Card* y que a su vez pueda ser utilizada por un applet de *MoC.*

El **campo de acción** se centra en el proceso de gestión de la información biométrica en las tarjetas inteligentes.

Para dar solución al problema y cumplir con el objetivo general se han trazado las siguientes **tareas de investigación**:

- Descripción de los aspectos teóricos conceptuales sobre aplicaciones Applet (según tecnología Java Card).
- Análisis del protocolo de canal seguro de *Global Platform* (Plataforma global).
- Análisis del proceso de gestión de información dactilar en tarjetas inteligentes.
- Caracterización de la arquitectura de la tarjeta (Ambiente de ejecución, *Java Card Runtime Environment*, Administrador de tarjetas, *Card Manager*, Dominios de seguridad, API de *Open Platform*) como base para la implementación de la aplicación *Java Card*.
- Análisis de los estándares (*Global Platform*, PC/SC, Normas ISO) para su puesta en práctica en la implementación a realizar.

- Análisis de las herramientas que permitan el desarrollo de aplicación *Java Card*.
- Desarrollo de la aplicación *Java Card* a partir del diseño realizado.
- Realización de las pruebas de calidad a la aplicación implementada.

Para 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, es favorable utilizar como **estrategia de investigación** la exploratoria.

Los métodos científicos de investigación constituyen la forma de abordar la realidad, de estudiar la naturaleza, la sociedad y el pensamiento, con el propósito de descubrir su esencia y sus relaciones. Estos métodos se pueden clasificar en **métodos teóricos** y **métodos empíricos**.

Los **métodos teóricos** por su parte permiten estudiar las características del objeto de investigación que no son observables directamente, facilitan la construcción de modelos e hipótesis de investigación y crean las condiciones para ir más allá de las características fenomenológicas y superficiales de la realidad, es por ello que los métodos teóricos que están presentes en este trabajo de diploma son: Analítico-Sintético, Hipotético-Deductivo y Análisis Histórico-Lógico.

- **Analítico–Sintético:** Mediante este método se va a realizar un análisis de los elementos de la situación problémica, relacionándolos entre sí y vinculándolos como un todo. Además de analizar toda la teoría recopilada a través de los diferentes medios bibliográficos y poder aplicar así estos conocimientos en la práctica de manera que se adquiera una mayor preparación sobre el tema en cuestión.
- **Hipotético-Deductivo:** Este método se emplea en cada fase en la cual se obtenga un resultado y se someta a un análisis sobre la base de algunos conocimientos adquiridos por resultados anteriores.
- **Histórico-Lógico:** Este método posibilita el análisis histórico del proceso de verificación biométrica en las tarjetas inteligentes, centrado en las huellas dactilares de las personas, así como el proceso para cifrar la información dactilar que se maneja en el almacenamiento y la verificación biométrica en las tarjetas inteligentes. Se emplea durante todo el proceso de diseño y mejoramiento de las funcionalidades de la aplicación a*pplet*.

Por otro lado los **métodos empíricos** permiten, como parte del plan operativo de la investigación, determinar el método de recolección de datos y tipo de instrumento que se utilizará, es por ello que el método empírico que está presente en el actual trabajo de diploma es el siguiente:

- **Observación:** Este método es uno de los más utilizados en la presente investigación, precisamente por ser un procedimiento sencillo de realizar y que permite percibir directamente el proceso de cómo se almacena la huella en el *Applet*.

El fundamento de la presente investigación está dado debido a la importancia que tiene el desarrollo de este tipo de tecnología para el mundo entero, la cual ha constituido un impacto social. Además, como las soluciones existentes en la actualidad son privativas, se hace necesario realizar una investigación que permita al departamento Tarjetas Inteligentes utilizar una tarjeta inteligente como dispositivo que gestione la información biométrica del portador, para luego utilizar la misma en la verificación que se llevará a cabo con el *applet MoC*. De esta forma la seguridad de la información que se encuentra almacenada dentro, se elevaría en gran medida. Esto es debido a que, al almacenar la información biométrica del portador en su tarjeta, aunque se le extravíe, o sufra por robo, las probabilidades de que se logre usurpar la identidad del portador de dicha tarjeta son mínimas, pues la información almacenada dentro sólo coincidirá con su legítimo propietario.

La presente investigación consta de los siguientes capítulos:

Capítulo 1 Fundamentación Teórica: Este capítulo contiene una base teórica que fundamente el problema planteado. Se describen los conceptos fundamentales para el dominio del problema, se muestra el estudio de aplicaciones *Applets* y se hace referencia a las tecnologías y canal seguro de comunicación presente en tarjetas inteligentes, así como las herramientas actuales que se usaron y los principales estándares internacionales a utilizar.

Capítulo 2 Descripción y diseño de la solución propuesta: Se definen las clases que modelan la solución y las principales funcionalidades de cada una de ellas. Finalmente se establecen los estándares de codificación a utilizar, además de la propuesta del sistema y la arquitectura del mismo.

Capítulo 3 Validación de la solución propuesta: Se refiere al diseño de las pruebas que permitan validar la solución propuesta, detallando su objetivo, alcance y tipo, así como los valores utilizados para la ejecución de dichas pruebas.

Capítulo 1: Fundamentación Teórica

1.1 Introducción

El presente capítulo estará centrado específicamente en los puntos más técnicos de la investigación, con el cual se dará continuación al estudio preliminar realizado en el trabajo de diploma *"Diseño de un Applet y Middleware para gestionar la información biométrica contenida en la Cédula de Identificación Electrónica de la República Bolivariana de Venezuela",* que estuvo enfocado a la etapa de análisis llevada a cabo para la resolución del problema.

En este trabajo se perfilarán los conceptos fundamentales que servirán de base para el desarrollo del mismo, además se caracterizarán los estándares internacionales propuestos con anterioridad para llevar a cabo la solución propuesta, así como el protocolo a utilizar para establecer un canal seguro de comunicación. Finalmente se detallan las herramientas con las cuales se implementará la solución propuesta y las tecnologías con las cuales se trabajará.

1.2 Conceptos fundamentales asociados al dominio del problema

Para lograr un mejor entendimiento de la problemática planteada se hace necesario abordar conceptos fundamentales sobre el tema, que le permitan al lector ubicarse en el escenario adecuado.

1.2.1 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, mediante un *plugin*[3], o en aplicaciones como teléfonos móviles que soportan el modelo de programación por *Applets*. (Farlex Inc., 2012)

Los *Applets*, según el contexto del trabajo, son aplicaciones implementadas utilizando la tecnología *Java Card* (tecnología que permite implementar aplicaciones Java) y son ejecutadas dentro de las tarjetas inteligentes. (Farlex Inc., 2012)

El *applet uMatchApp* gestiona la información de las plantillas de las huellas dactilares contenidas en una tarjeta inteligente, se ejecuta en el contexto del *Java Card Runtime–Environment* (Ambiente de ejecución de Java Card, en lo adelante JCRE por sus siglas en inglés).

Este *applet* es el encargado de ejecutar las operaciones que maneja, mediante los comandos APDU (Application Protocol Data Unit o Unidades de datos de protocolos de aplicación, en lo adelante APDU, por sus siglas en inglés) que le son enviados, retornando

[3] Pequeño programa que añade alguna función a otro programa, habitualmente de mayor tamaño.

los resultados mediante APDU de respuestas. Todas las peticiones hechas al *applet* se harán mediante este tipo de comandos, ya que es la única forma que existe para comunicarse con el mismo.

Ventajas de los Applets

Los *applets* de Java suelen tener las siguientes ventajas:

- Son multiplataforma (funcionan en Linux, Windows, Mac OS, y en cualquier sistema operativo para el cual exista una Java Virtual Machine).
- El mismo applet puede trabajar en "todas" las versiones de Java, y no sólo la última versión del plugin. Sin embargo, si un applet requiere una versión posterior del Java Runtime Environment (JRE), el cliente se verá obligado a esperar durante la descarga de la nueva JRE.
- Es compatible con la mayoría de los navegadores web.
- Puede ser almacenado en la memoria caché de la mayoría de los navegadores web, de modo que se cargará rápidamente cuando se vuelva a cargar la página web, aunque puede quedar atascado en la caché, causando problemas cuando se publican nuevas versiones.
- Puede tener acceso completo a la máquina en la que se está ejecutando, si el usuario lo permite.
- Puede ejecutarse a velocidades comparables a la de otros lenguajes compilados, como C++ (dependiendo de la versión de la JVM).
- Puede trasladar el trabajo del servidor al cliente, haciendo una solución web más escalable tomando en cuenta el número de usuarios o clientes. (SlideShare Inc. , 2009)

De esta forma es muy fácil percatarse cuán necesaria es la implementación de un *applet* para llevar a cabo la solución propuesta, ya que al ser este una aplicación que corre dentro de las tarjetas inteligentes permite almacenar información sensible dentro de las mismas.

1.2.2 Middleware

Middleware es un software de computadora que conecta componentes de software o aplicaciones para que puedan intercambiar datos entre éstas, funciona como una conversión o capa de traducción. Es utilizado a menudo para soportar aplicaciones distribuidas. Esto incluye servidores web, servidores de aplicaciones, sistemas de gestión de contenido y herramientas similares. *Middleware* es especialmente esencial para tecnologías como XML, SOAP, servicios web y arquitecturas orientada a servicios. Soluciones *middleware* personalizadas se han desarrollado durante décadas para permitir a una aplicación comunicarse con otra, ya sea que se ejecuta en una plataforma diferente o viene de un proveedor distinto. (Alegsa, 2011)

El *middleware* implementado, sirve para aislar las operaciones de interacción con el *applet* contenido en la tarjeta inteligente, brindando una interpretación adecuada de las funcionalidades que realiza. Este juega un papel fundamental en la aplicación que se desarrollará, ya que servirá como módulo común entre la PC y el lector de tarjetas inteligentes, permitiendo la comunicación entre ambos.

1.2.3 Tarjetas Inteligentes

Una tarjeta inteligente (*smart card*), o tarjeta con circuito integrado (TCI), es una tarjeta de tamaño pequeño que permite la ejecución de cierta lógica programada. Aunque existe un diverso rango de aplicaciones, hay dos categorías:

- Las Tarjetas de memoria contienen sólo componentes de memoria no volátil y posiblemente alguna lógica de seguridad.
- Las tarjetas microprocesadoras contienen memoria y microprocesadores.

La percepción estándar de una *smart card* es una tarjeta microprocesadora de las dimensiones de una tarjeta de crédito (o más pequeña, como por ejemplo, tarjetas SIM o GSM[4]) con varias propiedades especiales (ej. un procesador criptográfico seguro, sistema de archivos seguro, características legibles por humanos) y es capaz de proveer servicios de seguridad (ej. confidencialidad de la información en la memoria). (Jesús Sánchez Orozco, 2004)

Las tarjetas inteligentes surgieron y fueron patentadas en la década de los setenta. Existen algunas discusiones de quién es el "inventor" original; entre los que se encuentran Juergen Dethloff de Alemania, Arimura de Japón y Roland Moreno de Francia. El primer uso masivo de las tarjetas fue para el pago telefónico público en Francia en 1983. Desde los años 70, las tarjetas inteligentes han reflejado un constante avance en cuanto a su capacidad técnica y ámbitos de aplicabilidad. El mayor auge de las tarjetas inteligentes fue en los noventa, con la introducción de las tarjetas SIM utilizadas en la telefonía móvil GSM en Europa.

La tecnología de tarjetas inteligentes ha evolucionado durante los últimos 20 años, incluyendo un incremento en las capacidades de almacenamiento y procesamiento de información, mayor seguridad, programas (software) maduros de manejo de tarjetas inteligentes, tecnologías sin contacto e integración de múltiples aplicaciones en un solo distintivo de identificación inteligente. Las tarjetas inteligentes pueden tener una variedad de aplicaciones utilizadas por organizaciones que incluyen: registrarse a Windows, administración de contraseñas, contraseñas de un solo uso (*One Time Passwords* (OTP)),

[4] Una tarjeta SIM (acrónimo en inglés de *subscriber identity module*, en español módulo de identificación del suscriptor) es una tarjeta inteligente desmontable usada en teléfonos móviles.

criptografía de correos electrónicos y de datos, firmas electrónicas, entrada única al sistema de empresas, acceso a redes inalámbricas de manera segura, autenticación biométrica, pagos de cafetería, almacenamiento de información personal, acceso basado en roles, acceso físico seguro y lealtad del cliente. (Smart Card Alliance, 2004)

Hoy en día las tarjetas inteligentes son esenciales en la solidez de la seguridad de los sistemas de manejo de identificación de las organizaciones, respaldando la sólida autenticación requerida para validar el acceso de individuos a los diferentes recursos y proveyendo un crítico primer paso para bloquear a intrusos. El trabajo de estandarización llevado a cabo por *Global Platform* y el *Government Smart Card Interoperability Specification* (GSC-IS) capacita a los que emiten tarjetas, al combinar soluciones de múltiples recursos, asegurando, por ende, interoperabilidad a larga escala y reduciendo los costos de propiedad al proveer un mercado abierto. (Smart Card Alliance, 2004)

Tipos de tarjetas según sus capacidades

Según las capacidades de su chip, las tarjetas más habituales son:

- Memoria: tarjetas que únicamente son un contenedor de ficheros pero que no albergan aplicaciones ejecutables. Estas se usan generalmente en aplicaciones de identificación y control de acceso sin altos requisitos de seguridad.
- Microprocesadas: tarjetas con una estructura análoga a la de un ordenador (procesador, memoria volátil, memoria persistente). Estas albergan ficheros y aplicaciones y suelen usarse para identificación y pago con monederos electrónicos.
- Criptográficas: tarjetas microprocesadas avanzadas en las que hay módulos hardware para la ejecución de algoritmos usados en cifrados y firmas digitales. En estas tarjetas se puede almacenar de forma segura un certificado digital (y su clave privada) y firmar documentos o autenticarse con la tarjeta sin que el certificado salga de la tarjeta (sin que se instale en el almacén de certificados de un navegador web, por ejemplo) ya que es el procesador de la propia tarjeta el que realiza la firma. Un ejemplo de estas tarjetas son las emitidas por la Fábrica Nacional de Moneda y Timbre (FNMT) española para la firma digital.

Tipos de tarjetas según la estructura de su sistema operativo

- Tarjetas de memoria: Tarjetas que únicamente son un contenedor de ficheros pero que no albergan aplicaciones ejecutables. Disponen de un sistema operativo limitado con una serie de comandos básicos de lectura y escritura de las distintas secciones de memoria y pueden tener capacidades de seguridad para proteger el acceso a determinadas zonas de memoria.

- Basadas en sistemas de ficheros, aplicaciones y comandos: Estas tarjetas disponen del equivalente a un sistema de ficheros compatible con el estándar ISO/IEC 7816 parte 4 y un sistema operativo en el que se incrustan una o más aplicaciones (durante el proceso de fabricación) que exponen una serie de comandos que se pueden invocar a través de APIs[5] de programación.
- Java Cards: Una *Java Card* es una tarjeta capaz de ejecutar mini-aplicaciones Java. En este tipo de tarjetas el sistema operativo es una pequeña máquina virtual Java (JVM) y en ellas se pueden cargar dinámicamente aplicaciones desarrolladas específicamente para este entorno. (Smart Card Alliance, 2004)

Tipos de tarjetas según la interfaz

- **Tarjeta inteligente de contacto**

Estas tarjetas disponen de unos contactos metálicos visibles y debidamente estandarizados (ISO/IEC 7816-2), por lo que deben ser insertadas en una ranura de un lector para poder operar con ellas. A través de estos contactos el lector alimenta electrónicamente a la tarjeta y transmite los datos oportunos para operar con ella conforme al estándar.

La serie de estándares ISO/IEC 7816 e ISO/IEC 7810 definen:

- La forma física (ISO 7816-1)
- La posición de las formas de los conectores eléctricos (ISO 7816-2)
- Las características eléctricas (ISO 7816-3)
- Los protocolos de comunicación (ISO 7816-3)
- La dureza de la tarjeta
- La funcionalidad
- El formato de los comandos (ADPU) enviados a la tarjeta y las respuestas retornadas por la misma.

Los lectores de tarjetas inteligentes de contacto son utilizados como un medio de comunicación entre la tarjeta inteligente y un anfitrión, como por ejemplo un ordenador. (Smart Card Alliance, 2004)

- **Tarjeta inteligente sin contacto**

El segundo grupo son las tarjetas inteligentes sin contacto, cuyo intercambio de datos se realiza por identificación por radiofrecuencia (RFID por sus siglas en inglés) e incorporan una antena interior que, mediante una señal de radio, genera la electricidad necesaria para activar el chip. Este se comunica con el lector de tarjetas mediante inducción a una tasa de transferencia de 106 a 848 Kb/s. El estándar de comunicación de tarjetas inteligentes sin

[5] Interfaz de Programación de Aplicaciones, que representa una interfaz de comunicación entre componentes software.

contacto es el ISO/IEC 14443 del 2001, en este se definen dos tipos de tarjetas sin contacto (A y B), permitidos para distancias de comunicación de hasta 10 cm. Ha habido propuestas para la ISO 14443 tipos C, D, E y F que todavía tienen que completar el proceso de estandarización. Un estándar alternativo de tarjetas inteligentes sin contacto es el ISO 15693, el cual permite la comunicación a distancias de hasta 50 cm. Las más abundantes son las tarjetas de la familia MIFARE de Philips, las cuales representan a la ISO/IEC 14443-A. (Smart Card Alliance, 2004)

Aplicaciones comerciales

Las aplicaciones fundamentales de las tarjetas inteligentes son:

- Identificación del titular de la misma.
- Pago electrónico de bienes o servicios mediante dinero virtual.
- Almacenamiento seguro de información asociada al titular.

Las aplicaciones de las tarjetas inteligentes incluyen su uso como tarjeta de crédito, SIM para telefonía móvil, tarjetas de autorización para televisión por pago, identificación de alta seguridad, tarjetas de control de acceso y como tarjetas de pago del transporte público.

Las tarjetas inteligentes también son muy utilizadas como un monedero electrónico. Estas aplicaciones disponen normalmente de un fichero protegido que almacena un contador de saldo y comandos para decrementar e incrementar el saldo (esto último sólo con unas claves de seguridad especiales, obviamente). Cuando las tarjetas son criptográficas las posibilidades de identificación y autenticación se multiplican ya que se pueden almacenar de forma segura certificados digitales o características biométricas en ficheros protegidos dentro de la tarjeta de modo que estos elementos privados nunca salgan de la tarjeta y las operaciones de autenticación se realicen a través del propio chip criptográfico de la tarjeta.

De un modo más particular, las aplicaciones más habituales son:

- Identificación digital: este tipo de aplicaciones se utilizan para validar la identidad del portador de la tarjeta en un sistema centralizado de gestión.
- Control de acceso: este tipo de aplicaciones se utilizan para restringir o permitir el acceso a una determinada área en función de distintos parámetros que pueden estar grabados en la tarjeta o pueden ser recuperados de un sistema central de gestión a partir de la identidad grabada en la tarjeta.
- Monedero electrónico: esta aplicación se utiliza como dinero electrónico. Se puede cargar una cierta cantidad de dinero (en terminales autorizados que dispongan de las claves de seguridad oportunas) y luego, sobre esta cantidad de dinero se pueden realizar operaciones de débito o consulta de modo que puede ser utilizado para el pago o cobro de servicios o bienes.
- Firma Digital: este tipo de aplicaciones permiten almacenar un certificado digital de forma segura dentro de la tarjeta y firmar con él documentos electrónicos sin que

en ningún momento el certificado (y más concretamente su clave privada) salgan del almacenamiento seguro en el que están confinados. Con estas aplicaciones se abre todo un abanico de posibilidades en el campo de la administración electrónica.

- Fidelización de clientes: Este tipo de aplicación sirve a las empresas que ofrecen servicios o descuentos especiales para clientes que hacen uso de la tarjeta para poder validar la identidad del cliente, y para descentralizar la información.
- Sistemas de Prepago: En estos sistemas, un cliente carga su tarjeta con una cierta cantidad de servicio, la cual va siendo decrementada a medida que el cliente hace uso del servicio. El servicio puede variar desde telefonía móvil hasta TV por cable, pasando por acceso a sitios web o transporte público.
- Tarjetas sanitarias: En algunos hospitales y sistemas nacionales de salud ya se está implementando un sistema de identificación de pacientes y almacenamiento de los principales datos de la historia clínica de los mismos en tarjetas inteligentes para agilitar la atención. (Smart Card Alliance, 2004)

Estructura de una tarjeta inteligente con microprocesador

Internamente, el chip de una tarjeta inteligente microprocesada se compone de:

- CPU (*Central Processing Unit*): el procesador de la tarjeta; suelen ser de 8 bits, a 5 MHz y 5 voltios. Pueden tener opcionalmente módulos hardware para operaciones criptográficas.
- ROM (*Read-Only Memory*): memoria interna (normalmente entre 12 y 30 KB) en la que se incrusta el sistema operativo de la tarjeta, las rutinas del protocolo de comunicaciones y los algoritmos de seguridad de alto nivel por software. Esta memoria, como su nombre indica, no se puede reescribir y se inicializa durante el proceso de fabricación.
- EEPROM: memoria de almacenamiento (equivalente al disco duro en un ordenador personal) en el que está grabado el sistema de ficheros, los datos usados por las aplicaciones, claves de seguridad y las propias aplicaciones que se ejecutan en la tarjeta. El acceso a esta memoria está protegido a distintos niveles por el sistema operativo de la tarjeta.
- RAM (*Random Access Memory*): memoria volátil de trabajo del procesador. (Tecnología hecha palabra, 2008)

Programación de aplicaciones para los sistemas en los que se utiliza la tarjeta inteligente

Existen varias APIs de programación estandarizadas para comunicarse con los lectores de tarjetas inteligentes desde un ordenador. Las principales son:

- PC/SC (*Personal Computer/Smart Card*), especificado por el PC/SC *Workgroup*. Existe una implementación para Microsoft Windows y también el proyecto MUSCLE

proporciona una implementación casi completa de esta especificación para los sistemas operativos GNU Linux-UNIX.

- OCF (*OpenCard Framework*), especificado por el grupo de empresas *OpenCard*. Este entorno intenta proporcionar un diseño orientado a objetos fácilmente extensible y modular. El consorcio *OpenCard* publica el API y proporciona una implementación de referencia en Java. Existe un adaptador para que OCF trabaje sobre PC/SC. (SIC-SGA, 2012)

En ambos casos, el modelo de programación que utilizan las tarjetas inteligentes está basado en protocolos de petición-respuesta. La tarjeta (su software) expone una serie de comandos que pueden ser invocados. Estos comandos interactúan con los ficheros que subyacen a cada aplicación de la tarjeta y proporcionan un resultado. Desde el terminal se invocan estos comandos a través de cualquiera de las APIs antes descritas componiendo APDU que son enviados a la tarjeta para que ésta responda.

Uso de tarjetas inteligentes en el almacenamiento de información

En la actualidad, como la seguridad se ha convertido en el problema número uno en control de acceso tanto para los mercados gubernamentales y los comerciales, muchas industrias están adoptando rápidamente el uso de una única solución integrada.

La necesidad de una estrecha identificación y autenticación está impulsando a los desarrolladores del proyecto a adoptar el uso de soluciones de tarjetas inteligentes y aplicaciones, basados en sus características de seguridad mejorada, y un potente microprocesador integrado, que puede almacenar y procesar la información selectiva en la tarjeta.

La utilización de tarjetas inteligentes proporciona como ventajas fundamentales las siguientes:

- La privacidad de los portadores de dicha tarjeta se mantiene en todo momento.
- No hay límite en el número de plantillas de huellas dactilares que pueden ser almacenados en la terminal.

Las tarjetas inteligentes presentan un costo por transacción que es menor que el de las tarjetas magnéticas convencionales. Esto es incluyendo los costos de la tarjeta, de las infraestructuras necesarias y de los elementos para realizar las transacciones. Ofrecen prestaciones muy superiores a las de una tarjeta magnética tradicional. Esta ventaja se explica por las configuraciones múltiples que puede tener, lo que permite utilizarla en distintas aplicaciones.

Además permiten realizar transacciones en entornos de comunicaciones móviles, en entornos de prepago y en nuevos entornos de comunicaciones. A estos entornos no puede acceder la tarjeta tradicional. Las mejoras en seguridad y funcionamiento permiten reducir los riesgos y costes del usuario. Nuevos servicios y aplicaciones están surgiendo, y

necesitan de esta tecnología, para los cuales las tarjetas de banda magnética no pueden brindar soluciones. Las tarjetas inteligentes son elementos que sin lugar a dudas, se convertirán en un algo cotidiano en el día a día. (Captalis, 2009)

DNI Electrónico

Quizás el cambio más significativo que han inducido las tecnologías de la información es la convergencia de los distintos soportes en los que tradicionalmente se almacenaba el conocimiento a un formato digital. Textos, fotografías, música, palabras, imágenes, entre otros, han cambiado radicalmente la forma en la que se producen, almacenan, transfieren y utilizan. Y es en este contexto (cambio de "átomos" a "bits", junto con el crecimiento exponencial de las redes de comunicaciones) dónde debemos enmarcar el nuevo DNI electrónico.

El DNI electrónico no sólo es la evolución natural de un documento de identificación que incorpora medios cada vez más eficientes para acreditar la identidad (tal y como ocurrió en su tiempo con la inclusión de la fotografía o la huella dactilar) sino que es un instrumento que proporciona de forma inequívoca e indubitada la acreditación de la identidad personal, con la garantía del Estado, tanto en el mundo físico como en el digital, añadiendo nuevas funcionalidades (identidad electrónica y firma digital) al DNI tradicional.

La infraestructura establecida para el DNI electrónico permite ofrecer en el mundo digital las certezas necesarias para establecer relaciones de confianza:

- **Autenticidad**: Que las partes que intervienen en la transacción sean efectivamente quienes dicen ser.
- **Integridad**: Que la información transmitida no haya sido alterada en la transmisión.
- **No Repudio**: Que las partes no puedan negar que la información se intercambió.

La incorporación de la identidad digital y de la firma electrónica como tecnologías de la seguridad y de la información en los procesos de negocio y de soporte de las empresas, y en particular las PYME (pequeñas y medianas empresas), permite afrontar con garantías de éxito la modernización de las mismas y mejorar su competitividad. El DNI electrónico como infraestructura básica de seguridad de la información ofrece esta oportunidad a todos los sectores económicos. (Cuerpo Nacional de Policía. DNI Electrónico., 2012)

Varios países del mundo están emitiendo DNI electrónico nacional en forma de tarjetas inteligentes, entre los que se pueden mencionar China, Qatar, Marruecos, Tailandia, Hong-Kong y Oman. Otro de los países en donde avanza la puesta en funcionamiento de una solución de DNI electrónico es Venezuela, país que planea lanzar la nueva cédula de identificación electrónica (CIE) que sería la primera de su tipo en América Latina, donde tendrá incorporado una tarjeta de memoria almacenando todos los datos del ciudadano, además de que cumpliría con la definición de tarjeta inteligente con interfaz sin contacto.

1.3 Tecnología en tarjetas inteligentes

Para el desarrollo de aplicaciones que utilizan tarjetas inteligentes se requiere del uso de especificaciones y tecnologías que permitan llevar a cabo la programación de dichas aplicaciones.

1.3.1 Tecnología Java Card

Java Card es una tecnología que permite ejecutar de forma segura pequeñas aplicaciones Java (*Applets*) en tarjetas inteligentes y similares dispositivos empotrados. *Java Card* brinda al usuario la capacidad de programar aplicaciones que se ejecutan en la tarjeta de modo que ésta tenga una funcionalidad práctica en un dominio de aplicación específico (pe. Identificación y pago). Esta tecnología se usa ampliamente en las tarjetas SIM (utilizadas en teléfonos móviles GSM (Sistema Global para Comunicaciones Móviles)) y en tarjetas monedero electrónico.

La primera tarjeta Java fue presentada en 1997 por varias empresas entre las que estaban Axalto (división de tarjetas de *Schlumberger*) y *Gemplus*. Los productos *Java Card* están basados en la Plataforma *Java Card* cuyas especificaciones han sido desarrolladas por *Sun Microsystems*. El departamento Tarjetas Inteligentes utiliza la especificación 2.1.1 de esta plataforma, liberada por *Sun* en el año 2000. Las principales características de esta tecnología son la portabilidad y la seguridad.

- Portabilidad

Java Card tiene por objeto la definición de un estándar de tarjeta inteligente que permite a la misma applet funcionar en diferentes tarjetas inteligentes, de forma muy parecida a cómo un applet de Java se ejecuta en diferentes ordenadores. Al igual que en Java, esto se consigue utilizando la combinación de una máquina virtual (la Máquina Virtual de *Java Card*), y unas librerías cuya API está especificada. La portabilidad, en todo caso, sigue siendo obviada en muchos casos por cuestiones de tamaño de la memoria, el rendimiento y tiempo de ejecución (por ejemplo para los protocolos de comunicación o algoritmos criptográficos).

- Seguridad

La tecnología *Java Card* fue desarrollada originalmente con el propósito de asegurar la información sensible almacenada en las tarjetas inteligentes. La seguridad está determinada por diversos aspectos de esta tecnología:

- Applet. Es una máquina de estados que sólo procesa los comandos recibidos a través del dispositivo lector enviando y respondiendo con códigos de estado y datos.

- Separación de applets. Las distintas aplicaciones están además, separadas unas de otras por un cortafuego que limita el acceso y control de los elementos de datos de un subprograma a otro.
- Encapsulación de datos. Los datos se almacenan en la aplicación y las aplicaciones *Java Card* se ejecutan en un entorno aislado (JVM, *Java Virtual Machine* por sus siglas en inglés), separada del sistema operativo y del equipo en que se lee la tarjeta.
- Criptografía. En esta plataforma están implementados los algoritmos criptográficos más comúnmente utilizados como DES, 3DES, AES, RSA (incluyendo el uso de criptografía de curva elíptica). Otros servicios como la firma electrónica o la generación de claves de intercambio también están soportados. (Plataforma Java Card, 2012)

Arquitectura

La tecnología *Java Card* permite a los programas escritos en el lenguaje de programación Java ejecutarse sobre tarjetas inteligentes y otros pequeños dispositivos. Los desarrolladores pueden construir y probar programas usando herramientas y ambientes de desarrollo de software estándar, luego convertirlos a un formato que puede ser instalado sobre un dispositivo que permite la tecnología *Java Card*. Aplicaciones de software para la plataforma *Java Card* son llamadas applets, o más específicamente, una applet de Java *Card* (para distinguirlas de las applets de los navegadores). Mientras la tecnología *Java Card* permite programas escritos en el lenguaje de programación Java ejecutarse sobre tarjetas inteligentes, tales dispositivos pequeños están lejos de poder soportar la completa funcionalidad de la plataforma Java. Por tanto, *Java Card* soporta solo un subconjunto de las características de la plataforma Java, el cual proporciona características que son apropiadas para escribir programas para pequeños dispositivos y preservar el potencial orientado a objeto del lenguaje.

La tecnología *Java Card* define un entorno de ejecución que soporta la memoria, comunicación, seguridad y el modelo de ejecución de las smart cards, dicho entorno cumple con el estándar internacional ISO 7816.

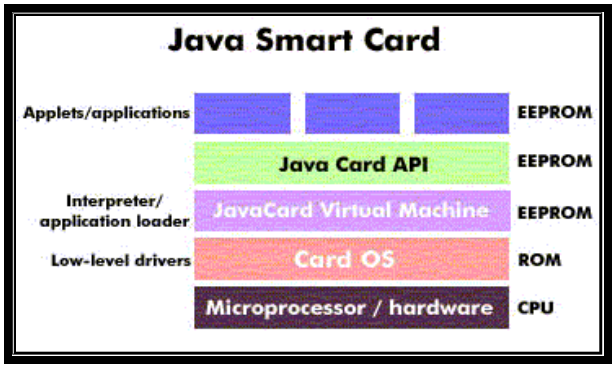


Ilustración 1. Arquitectura de una Smart Card

Debido a la división de la arquitectura de la máquina virtual, esta plataforma se distribuye entre el entorno de la *smart card* y el ordenador de escritorio. La plataforma se compone de tres partes, cada una definida en una especificación:

- El *Java Card Virtual Machine* (JCVM) *Specification*, el cual define un subconjunto del lenguaje de programación de Java y una definición de la máquina virtual deseable para las aplicaciones smart card.
- El *Java Card Runtime Environment* (JCRE) *Specification*, describe con precisión el comportamiento del entorno Java Card, incluyendo la gestión de memoria, la gestión de applets y otras características de la ejecución.
- El *Java Card Application Programming Interface* (API) *Specification*, que describe un conjunto del núcleo y extensión de los paquetes y clases de Java para programar las aplicaciones de las *smart cards*. (Plataforma Java Card, 2012)

Subconjunto del lenguaje Java Card

Debido a las restricciones de memoria, la plataforma *Java Card* solo soporta un subconjunto de características del lenguaje Java. Este subconjunto incluye características que son adecuadas para escribir programas para las smart cards y otros pequeños dispositivos, incluso preserva las posibilidades de la orientación a objetos para el lenguaje de programación Java. La siguiente figura muestra alguna de las características soportadas y no soportadas del lenguaje Java. (Plataforma Java Card, 2012)

Características soportadas de Java	Características no soportadas de Java
Tipos de datos primitivos: `boolean`, `byte`, `short`. Arrays de una dimensión. Paquetes, clases, interfaces y excepciones de Java. Características de orientación a objetos de Java. E tipo de dato entero de 32 bits es opcional.	Tipos de datos primitivos: `long`, `double`, `float`. Caracteres y cadenas. Arrays multidimensionales. Carga dinámica de clases. Gestor de seguridad. Recolector de basura. Serialización de objetos. Clonación de objetos.

Ilustración 2. Características de Java soportadas y no soportadas

Las APIs de Java Card

Las APIs de *Java Card* consisten en un juego de clases personalizadas para programar aplicaciones de smart cards acordes al modelo ISO 7816. Las APIs contienen tres paquetes centrales y un paquete de extensión. Los tres paquetes centrales son *java.lang*, *javacard.framework* y *javacard.securit,* y el paquete de extensión es *javacardx.crypto*.

Las clases en las APIs de *Java Card* son compactas y cortas, incluyen clases adaptadas de la plataforma Java para proveer soporte al lenguaje Java y a los servicios de criptografía, así como también contienen clases creadas específicamente para soportar el estándar ISO 7816 de *smart card*. (Plataforma Java Card, 2012)

Paquete java.lang

El paquete *java.lang* provee el soporte fundamental para el lenguaje Java. La clase *Object* define una raíz para la jerarquía de clases de *Java Card* y la clase *Trowable* provee un ascendiente común para todas las excepciones. Las clases de excepciones soportadas aseguran semánticas consistentes cuando ocurre un error debido a una violación del lenguaje Java. (Plataforma Java Card, 2012) Al importar este paquete se le permite al *applet uMatchApp* poder lanzar diferentes excepciones según los errores presentes durante la ejecución de la aplicación.

Object	Throwable	Exception
RuntimeException	ArithmeticException	ArrayIndexOutBoundsException
ArrayStoreException	ClassCastException	IndexOutBoundsException
NullPointerException	SecurityException	negativeArraySizeException

Tabla 1. Paquete java.lang de Java Card

Paquete javacard.framework

El paquete *javacard.framework* es esencial, el mismo provee clases, soporte e interfaces para la funcionalidad principal de un applet *Java Card*. Lo más importante es que define una clase base Applet, que provee un soporte para la ejecución del applet y su interacción con el JCRE durante el tiempo de vida del applet. Su papel respecto al JCRE es similar al de la clase Applet en el explorador de un ordenador. Otra clase importante incluida en este paquete es la clase APDU.

De acuerdo al estándar, las *smart cards* nunca inician la comunicación con el CAD, sino que sólo responden a los comandos que éste le envía. Se define dos tipos de APDU, los llamados *COMMAND* APDU, que son los que envía el CAD a la tarjeta, y los *RESPONSE* APDU, que son los que envía la tarjeta al CAD como respuesta a un *COMMAND* APDU. (Plataforma Java Card, 2012) Este paquete permite al *applet uMatchApp* poder utilizar clases y funcionalidades presentes en *Java Card*, como por ejemplo: realizar comparaciones entre arreglos de bytes, hacer copias de dichos arreglos, etc.

La siguiente tabla describe el formato de ambos tipos de APDU.

Command APDU						
Encabezado obligatorio				**Cuerpo opcional**		
CLA	INS	P1	P2	LC	Data Field	LE
Response APDU						
Cuerpo opcional				**Cola obligatoria**		
Data Field				SW1		SW2

Tabla 2. Formato del comando APDU

Paquete javacard.security

El paquete *javacard.security* provee un soporte para las funciones de criptografía soportadas en la plataforma *Java Card*. Este paquete define una clase llamada *KeyBuilder* y varias interfaces que representan claves criptográficas usadas en algoritmos simétricos, como DES, o asimétricos, como DSA o RSA. Además soporta las clases base abstractas *RandomData*, *Signature* y *MessageDigest*, que se usan para generar datos aleatorios y calcular las asimilaciones y las firmas de los mensajes. (Plataforma Java Card, 2012) El *applet* utiliza este paquete específicamente cuando va a trabajar con la función *Random Data.*

Applets de Java Card

Un *applet* de *Java Card* es un programa de Java que cumple un conjunto de convenciones que permiten ejecutarlo en el entorno de ejecución de *Java Card*. Un *applet* de *Java Card* no está pensado para ejecutarse en un explorador. Estos se pueden cargar en el entorno de ejecución después de que la tarjeta haya sido fabricada, es decir a diferencia de las aplicaciones de muchos sistemas embebidos, los applets no necesitan ser “quemados” en la ROM durante el proceso de fabricación, más bien pueden ser bajados dinámicamente a la tarjeta en un instante posterior a la fabricación. Un *applet* que se ejecuta en la tarjeta es una instancia de un *applet*, o sea un objeto del mismo, como ocurre con cualquier objeto persistente, una vez creado, un *applet* vive en la tarjeta para siempre. El entorno de ejecución de *Java Card* soporta un entorno multi-aplicación, por lo que pueden coexistir múltiples *applets* en una sola *smart card* y un *applet* puede tener múltiples instancias. (Plataforma Java Card, 2012)

Finalmente, luego de haber profundizado sobre *Java Card* se ha llegado a la conclusión de que esta es la única tecnología que permitirá desarrollar aplicaciones Java que corran dentro de las tarjetas inteligentes, siempre y cuando estas tarjetas tengan como sistema operativo *Java Card*.

1.3.2 Tecnología Biométrica

El término “biometría” proviene del griego “bio” (vida) y “metron” (medida) y se refiere a todas aquellas técnicas que permiten identificar y autenticar a las personas a través de sus características fisiológicas y de comportamiento.

Según el Diccionario de la Real Academia Española, se define *biometría* como «Estudio mensurativo o estadístico de los fenómenos o procesos biológicos». Esta definición se hace más específica cuando se utiliza el término de biometría dentro del campo de la identificación de personas. Se podría decir en este caso, que biometría es la ciencia por la que se puede identificar a una persona basándose en sus características biofísicas o de comportamiento. Expuesto en forma de ejemplos, es la ciencia que consigue reconocer a una persona mediante una imagen de su rostro o mediante la impresión de su huella dactilar.

Como es lógico, la capacidad de identificación biométrica es algo innato en los seres vivos, ya que poseen la característica de reconocer a sus semejantes. La biometría como ciencia de estudio de la individualidad de las personas, nace seriamente a finales del siglo XIX. Este término ha sido usado desde temprano en el siglo XX para referirse al campo de desarrollo de métodos estadísticos y matemáticos aplicables a problemas de análisis de datos en las ciencias biológicas. (BuenasTareas.com, 2012)

Factores Biométricos

Enfoques basados en factores biométricos abarca un grupo de tecnologías probadas y de métodos automatizados que identifican y verifican a individuos basándose en sus características personales. Estos enfoques emparejan una característica en tiempo real contra un expediente de la característica que fue creada al registrase en el sistema. Las tecnologías biométricas principales incluyen la huella digital, la cara, la geometría de la mano, el diafragma, la palma, la firma, la voz, y la piel. El proceso de pareo se realiza en tres pasos:

1) Una imagen de los datos biométricos (por ejemplo, una huella digital) se captura.
2) La imagen se convierte en una plantilla única.
3) Los algoritmos complejos comparan la plantilla con un expediente almacenado.

Como bien se hizo mención, existen diferentes técnicas de verificación biométrica, pero este trabajo sólo se enfocará en las huellas dactilares. Las tecnologías biométricas se están utilizando con mayor frecuencia como control primario o secundario para el acceso lógico.

En un escenario típico, los usuarios incorporan un nombre de usuario y colocan un dedo en un lector (en lugar de, o en adición a proporcionar una contraseña). Un servidor compara la plantilla biométrica creada por el lector con un expediente almacenado en el servidor. Como alternativa, los usuarios pueden insertar una tarjeta inteligente en un lector

de tarjetas y utilizar una huella digital para autenticar que son los titulares válidos de la tarjeta.

El biométrico capturado por el lector se compara con los datos biométricos de la tarjeta inteligente, si la información biométrica capturada iguala la información biométrica almacenada en la tarjeta, la tarjeta inteligente entonces libera la información secreta requerida para registrar al usuario en la red o para permitirle realizar la acción requerida. (Smart Card Alliance, 2004)

El escenario que se desarrollará en el presente trabajo sólo abarcará la gestión de la información biométrica dactilar, ya que no se cuenta con la tecnología necesaria para realizar la verificación biométrica de dicha información.

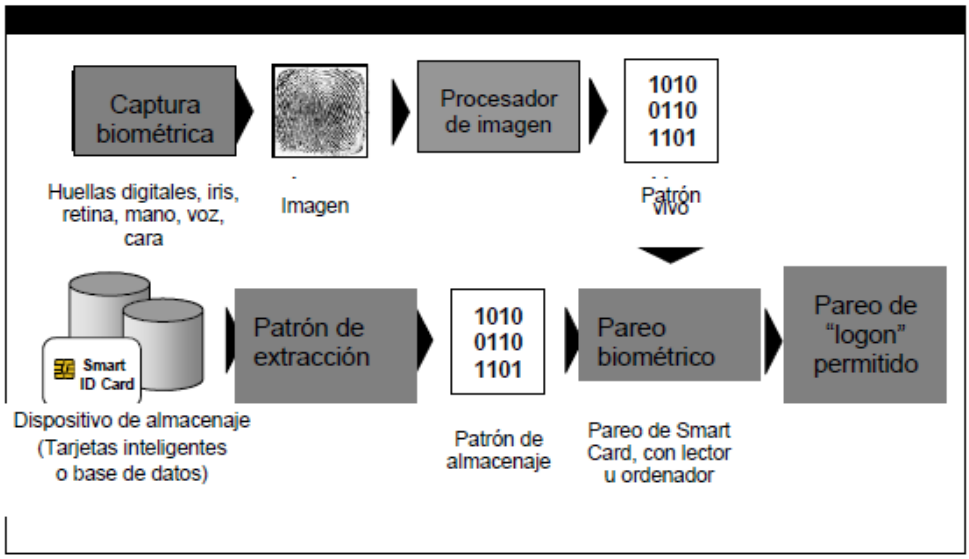


Ilustración 3. Identificación de procesos biométricos de verificación

Tecnología Match on Card

Match on Card (Comparación en tarjeta, por sus siglas en inglés MoC) es la tecnología que realiza comparaciones de huellas dactilares en tarjetas. Las tecnologías biométricas han ido fortaleciendo los mecanismos de autenticación al comparar la plantilla biométrica almacenada con la plantilla biométrica capturada al momento de la comparación (plantilla en vivo). En el caso de las tarjetas inteligentes, esta comparación se hace dentro de la tarjeta, lo cual requiere una capacidad de procesamiento interno que dependerá de la complejidad de la información biométrica a comparar (huellas dactilares, iris, tomografía facial, geometría de la mano, entre otros) y de los algoritmos usados.

- Seguridad

Una tarjeta inteligente representa un ambiente muy bien protegido y cerrado, adecuado para el almacenamiento de información personal o confidencial. Para acceder a esa información es práctico usar la biometría. Pero, ¿Es realmente infalible? Si la verificación

biométrica se realiza fuera del ambiente cerrado de la tarjeta inteligente, la verificación se pone en riesgo.

La comunicación entre la tarjeta inteligente y la unidad de la toma de decisiones puede ser interrumpida, la plantilla biométrica puede ser manipulada y la propia decisión de conceder o denegar el acceso puede ser alterada. Incluso si la plantilla biométrica está protegida por otro mecanismo de seguridad, el entorno abierto debe ser considerado como el eslabón más débil.

Desde hace muchos años, las tarjetas inteligentes han sido protegidas por PIN, pero la validación de los PIN se ha hecho siempre por la propia tarjeta. Es aquí donde entra en juego la tecnología *Match on Card*, la cual reemplaza el PIN con la biometría, y sin embargo mantiene la tarjeta inteligente a cargo de la seguridad, permitiendo de esta forma que la propia tarjeta sea la encargada de tomar la decisión de acceder o denegar el acceso, siendo esta la única forma de proteger realmente la información en una tarjeta inteligente.

- Privacidad

Con *Match on Card*, el dueño de la tarjeta inteligente controla el único registro de su plantilla biométrica. Además, no hay necesidad de una base de datos externa de huellas digitales, lo cual es bueno desde dos perspectivas: los propietarios de la tarjeta pueden disfrutar de la privacidad de no tener que dar su información biométrica, y los emisores de tarjetas no tienen que preocuparse de actualizar y mantener la base de datos. Incluso, si una tarjeta inteligente es robada o extraviada, la plantilla biométrica no se puede extraer de dicha tarjeta ya que esta se almacena de forma segura en la tarjeta inteligente junto con los datos que protege.

- Escalabilidad

Match on Card es una tecnología que crea una gran escalabilidad, además de ser adecuada para grandes sistemas donde el mantenimiento de una BD de registros biométricos sería muy costoso. Por ejemplo, la tarjeta de identificación nacional de los sistemas debe manejar no sólo miles de usuarios, sino millones de ellos, y con *Match on Card* cada ciudadano lleva su propio registro biométrico, por lo que se ofrece una solución clara y potente.

El proceso biométrico utilizando la tecnología *MoC* se divide en dos funciones a realizar: el enrolamiento y la verificación de las huellas digitales en la tarjeta. La plantilla biométrica es almacenada dentro de la tarjeta, que también realiza la comparación con la plantilla en vivo. Por tanto, se necesita una capacidad de procesamiento interno como la del microprocesador de una tarjeta inteligente, lo cual conlleva al uso de un sistema operativo que ejecute las aplicaciones de comparación necesarias. (RFID.com, 2008)

En el presente trabajo solo se desarrollará la función de enrolamiento, debido a que no se tiene la tecnología necesaria (módulo biométrico o API biométrica) para llevar a cabo la verificación de las huellas.

1.3.3 Estándares relacionados con tarjetas inteligentes

Durante la realización previa del análisis y diseño de la solución propuesta se realizó un estudio sobre los principales estándares internacionales con los cuales se llevará a cabo la solución propuesta. Estos son:

- **Estándar ISO 7816**

ISO/IEC 7816 es un estándar internacional relacionado con las tarjetas de identificación electrónicas, en especial las tarjetas inteligentes, gestionado conjuntamente por la Organización Internacional De Normalización (ISO) y Comisión Electrotécnica Internacional (IEC). Se trata de una extensión de la ISO 7810.

El objetivo de estos estándares es lograr la interoperabilidad entre distintos fabricantes de tarjetas inteligentes y lectores de las mismas, en lo que respecta a características físicas, comunicación de datos y seguridad. Estos estándares están basados en los ISO 7810 e ISO 7811, los cuales definen características físicas de tarjetas de identificación. (Ramírez, 2012) Este estándar comprende once partes, de las cuales sólo se trabajará con la parte ISO/IEC 7816-4, ya que define:

- El contenido de los pares comando-respuesta (APDU) que se intercambian a nivel de interfaz.
- Estructuras para aplicaciones y datos en la tarjeta.
- La estructura y contenido de los caracteres históricos de la respuesta de reset, los cuales describen las características de operación de la tarjeta inteligente.
- Métodos para extracción de objetos y elementos de datos de la tarjeta.
- Mecanismos para la identificación y direccionamiento de aplicaciones en la tarjeta.
- Estructuras de archivos y métodos de acceso.
- Arquitectura de seguridad para derechos de acceso a los archivos y datos en la tarjeta.
- Comandos orientados a objetos.
- Métodos de acceso a los algoritmos que procesa la tarjeta inteligente (no describe los algoritmos).
- Métodos para el intercambio seguro de mensajes.

El estándar ISO/IEC 7816-4 es independiente de la tecnología de interfaz física que se implemente en la tarjeta inteligente.

- **Estándar Global Platform**

Global Platform es un estándar que lidera mundialmente el desarrollo en temas de infraestructura de tarjetas inteligentes. Fue fundada en el año 1999 para asumir la responsabilidad de Visa Inc. *Specificacion's Open Platform*. Sus especificaciones técnicas probadas para las tarjetas, dispositivos y sistemas se consideran como el estándar de la industria para lograr implementaciones interoperables, inteligentes, flexibles y sostenibles para la tarjeta que soporte multi-aplicación y multi-actor e implementaciones multi-modelo de negocio. (Global Platform, 2012)

Canal seguro SCP01 de Global Platform

Los canales lógicos facilitan la posibilidad a más de una entidad que esté fuera de la tarjeta a comunicarse simultáneamente con varias aplicaciones en la tarjeta, cada uno dentro de su propio entorno lógicamente seguro.

El canal seguro proporciona un canal de comunicación seguro entre una tarjeta y la entidad que está fuera de la tarjeta. Este se divide en tres fases secuenciales:

- Inicialización del canal seguro: Cuando la aplicación que está corriendo en la tarjeta y la entidad que está fuera de la misma han intercambiado información suficiente que les permita realizar las funciones criptográficas requeridas.
- Operación del canal seguro: Cuando la aplicación que está corriendo en la tarjeta y la entidad que está fuera de la misma han intercambiado los datos que están dentro de la protección criptográfica de la sesión de canal seguro.
- Terminación del canal seguro: Cuando la aplicación que se encuentra en la tarjeta o la entidad que está fuera de la misma determinan que ninguna otra comunicación se requiere o se permite, a través de una sesión de canal seguro establecido.

El canal seguro es siempre inicializado por la entidad que está fuera de la tarjeta pasando desde un “host” desafío (*Random* de datos únicos establecidos en la sesión de canal seguro) hasta la tarjeta. (Global Platform, 2003)

Flujo de autenticación mutua

El anexo # 1 muestra el flujo de autenticación mutua entre un dominio de seguridad y la entidad que está fuera de la tarjeta.

- **Estándar PC/SC (*Personal Computer/Smart Card)***

PC/SC es un estándar para las tarjetas inteligentes de acceso en plataformas Windows (incluido en Windows 2000). En particular se define un API de programación, que permite a los desarrolladores trabajar de forma uniforme con lectores de tarjetas de distintos fabricantes (que cumplan con la especificación). La principal ventaja de este estándar es que las aplicaciones no tienen que estar al tanto de las actualizaciones de los lectores que

permiten la comunicación con las tarjetas inteligentes. Además, la aplicación puede funcionar con cualquier lector de cumplir con el estándar PC/SC.

La especificación se divide en 10 partes que contienen los requisitos detallados de interoperabilidad de dispositivos compatibles, información de diseño, interfaces de programación y otras. Estas partes son:

- Parte 1. Introducción y visión general de la arquitectura.
- Parte 2. Requisitos de interoperabilidad para las tarjetas y los lectores.
- Parte 3. Requisitos de interoperabilidad para los lectores conectados.
- Parte 4. Consideraciones de diseño e información de referencia de los lectores.
- Parte 5. Definición de la interfaz del Gestor de Recursos (*Resource Manager)*.
- Parte 6. Definición de la interfaz del Proveedor de Servicios (*Service Provider)*.
- Parte 7. Consideraciones de diseño para el desarrollo de aplicaciones.
- Parte 8. Recomendación para la implementación de servicios de seguridad y privacidad con tarjetas inteligentes.
- Parte 9. Lectores con capacidades extendidas.
- Parte 10. Lectores con capacidades de entrada de PIN (*Personal Identification Number* o Número de Identificación Personal) de seguridad. (Gemalto, 2009)

1.4 Tecnologías de desarrollo

El presente epígrafe hace mención de las tecnologías asociadas a la investigación, que permitieron la implementación de la aplicación propuesta.

1.4.1 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, así como la gestión del desarrollo directamente, en un ambiente familiar Java. Por lo tanto, es un desarrollador sin problemas que comprende las particularidades de *Java Card*. Además 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.

Developer Suite viene con una Suite que permite la simulación de:

- Tarjeta de Simuladores de tarjetas SIM, tarjetas USIM y tarjetas R-UIM, y las tarjetas NFC SCWS.
- Simuladores de móviles 2G, 3G y CDMA.
- Simuladores Server para 2G, 3G y SCWS OMA. (Gemalto, 2010)

1.4.2 JCardManager

El *JCardManager* es una aplicación cliente-genérico basada en terminal. Esta herramienta, permite comunicarse con los *applets* y probarlos sin necesidad de desarrollar la aplicación cliente. El *JCardManager* puede comunicarse con tarjetas reales y simuladores de tarjeta.
Las principales funciones de la *JCardManager* son:

- El acceso a la *GxpConverter* para convertir archivos de clase de formatos listos para ser cargados en una tarjeta o el GSE (simulador de tarjeta).
- La gestión de la capa de OCF comunicaciones.
- El acceso al Editor de archivos de implementación para el despliegue de registro y reproducción, lo que permite un paso de carga, la instalación y selección de *applets*.
- Realización de tarjeta de autenticación de terminales.
- La carga y la instalación de los *applets* en la tarjeta.
- Gestión de las características de cifrado de la tarjeta.
- Visualización y grabación de un seguimiento de toda la actividad entre el programa y el objetivo.
- Viendo el contenido de una tarjeta.
- El envío de comandos APDU (por sus siglas en inglés *Application Protocol Data Unit)* o comandos definidos por el usuario a un objetivo y la observación de las respuestas.
- Grabación y reproducción de secuencias de comandos para automatizar la emisión de secuencias de comandos. (Gemalto, 2010)

1.5 Applets que gestionan huellas dactilares

Durante la realización del análisis y diseño de la solución propuesta se realizó el estudio de una solución que pudiese servir de guía para la implementación del *applet uMatchApp* (*CryptoManager Applet*). A continuación se brinda algunas de las características de esta solución.

1.5.1 CryptoManager Applet

El *CryptoManager* es un *applet* que está diseñado para desarrolladores que trabajan con información biométrica y que realizan comparación de huellas dactilares. Además provee información, con la cual es posible para los clientes, configurar sus propias condiciones de acceso en orden de proteger sus datos. Este es un *applet* de servidor que controla las funcionalidades biométricas presente en la tarjeta. Además todos los datos biométricos asociados a una huella dactilar específica se almacenan en un contenedor de plantilla, es decir, la plantilla biométrica y cualquier información biométrica (pública), así como conocer

qué huella específica se ha extraído de la plantilla. Este *applet* necesita como requisitos de sistema los siguientes:

Requisitos de hardware

- Tarjeta Inteligente (*Smart Card*) y escáner de huellas dactilares.
- La aplicación *Java Card* debe ejecutar las APIs biométricas (*BioMatch*™ C o *Precise BioMatch*™ J).

Requisitos de software

- Se precisa de *BioMatch*™ *Pro Toolkit* 2.0 para generar y trabajar con las plantillas compatibles de *BioMatch*.

En la actualidad existen otras soluciones realizadas a nivel mundial que trabajan con información biométrica dactilar dentro de las tarjetas inteligentes, entre dichas soluciones se encuentran las mencionadas a continuación.

1.5.2 Precise BioManager™ J applet

Precise BioManager™ J es un *applet* de *Java Card* que administra las funcionalidades biométricas presentes en la tarjeta inteligente. Este provee una interfaz al borde de la tarjeta para aplicaciones de PC y una interfaz interna para los *applets* de seguridad de java. Las funciones claves de este *applet* son:

- El almacenamiento de plantillas biométricas
- La creación de las condiciones de acceso biométricos
- Verificación y validación de huellas dactilares
- Múltiples modos (Administración, Operacional, entre otros)
- Desbloquear las plantillas y la clave de administrador

Precise BioManager™ J brinda la posibilidad de reemplazar el PIN (Número de Identificación Personal o *Personal Identification Number* por sus siglas en inglés) por las huellas dactilares. (Precise Biometrics, 2005)

1.5.3 Precise IDStore™ J applet

Precise IDStore™ J es una aplicación de *Java Card* usada para almacenar y proteger la información biométrica sensible. El *applet* provee las funciones necesarias para administrar de forma genérica los datos en una *java card*. Los datos sensibles almacenados en las plantillas de *Precise IDStore*™ J es administrado por el applet biométrico de control de acceso de *Precise BioManager* J y protegido por la tecnología segura de *Precise Match-on-Card*™. (Precise Biometrics, 2006)

1.5.4 Conclusiones del estudio de soluciones existentes

Luego de haber estudiado las características de los diferentes *applets* antes mencionados se ha llegado a la conclusión de que estos resolverían el problema existente de utilizar una

tarjeta inteligente como dispositivo para gestionar la información biométrica dactilar de las personas; pero al ser soluciones propietarias, se niega todo tipo de acceso a su código fuente. Además, adquirirlos sería muy costoso, y se correría el riesgo de que en caso de que la compañía fabricante se fuese a la banca rota, el soporte técnico desaparecería, al igual que la posibilidad de en un futuro tener versiones mejoradas y la posibilidad de corregir los errores del mismo. (Asterisk, 2010)

Por lo antes expuesto se reafirma la necesidad de desarrollar una solución en el departamento de Tarjetas Inteligentes, perteneciente al CISED, que permita la utilización de una tarjeta inteligente como dispositivo para gestionar la información biométrica de las huellas dactilares de una persona, aplicación que será genérica a otros *applets* que realicen la verificación biométrica dactilar y necesiten de la gestión previa de dicha información, como por ejemplo: el *applet* de *MoC* que se está desarrollando en el departamento.

1.6 Conclusiones

Una tarjeta inteligente es una tarjeta de circuitos integrados, la cual tiene especificaciones estandarizadas a nivel internacional. Los estándares seleccionados para desarrollar la solución propuesta son:

- ISO/IEC 7816-4, el cual permite definir cómo serán organizados los comandos que se enviarán a la tarjeta en ámbitos de seguridad, intercambio de datos, niveles de acceso a ficheros, entre otros.
- *Global Platform*, que debido a su interoperabilidad en las tarjetas inteligentes, permite que sistemas heterogéneos puedan intercambiar procesos o datos.
- PC/SC por su parte, permite el acceso en las plataformas Windows y además facilita a los desarrolladores trabajar con lectores de tarjetas de diferentes fabricantes.

Para la gestión de la información biométrica de las huellas dactilares contenidas dentro de la tarjeta inteligente, se va a realizar la implementación del *applet uMatchApp*. Este *applet* se desarrollará utilizando la herramienta *Developer Suite* y la tecnología *Java Card.* Se establecerá además el canal seguro SCP01 definido por el estándar *Global Platform*, haciendo posible de esta forma implementar aplicaciones seguras que se ejecutan dentro de las tarjetas inteligentes, cumpliendo así con los requisitos que se hacen necesarios para la implementación de la actual solución.

Capítulo 2 Descripción y diseño de la solución propuesta

2.1 Introducción

Durante la etapa de análisis y diseño de la solución propuesta, recogida en el trabajo de diploma “Diseño de un Applet y Middleware para gestionar la información biométrica contenida en la Cédula de Identificación Electrónica de la República Bolivariana de Venezuela”, se definió XP (*Xtreme Programming*) como metodología de desarrollo a utilizar. Esta metodología permitirá desarrollar el sistema propuesto con mayor agilidad, ya que está diseñada para facilitar el trabajo de los programadores, definiendo todo el desarrollo completo incluyendo pruebas e integración.

En este capítulo, se realiza una profunda valoración de los artefactos propuestos por el analista del sistema durante la fase de Iteraciones, además se delimitan varios puntos importantes en el desarrollo de la solución propuesta tales como:

- Se describen las principales clases utilizadas en la implementación del mismo.
- Se describen los métodos que permitirán llevar a cabo la implementación.
- Se describen los comandos a utilizar para llevar a cabo la aplicación.

2.2 Valoración del análisis y diseño propuesto por el analista

En el proceso de desarrollo de software la descripción de las historias de usuarios es de suma importancia, ya que estas facilitan un mejor entendimiento de los procesos a desarrollar, además permiten una mejor comprensión del problema en cuestión y facilitan una mejor identificación de las clases y funcionalidades que serán implementadas. Las historias de usuario propuestas constituyen una entrada apropiada y un punto de partida para las actividades de implementación. Del análisis propuesto en el trabajo de diploma “Diseño de un Applet y Middleware para gestionar la información biométrica contenida en la Cédula de Identificación Electrónica de la República Bolivariana de Venezuela” se obtuvo el diagrama de clases, que permitió aumentar la comprensión del problema y contribuir a esclarecer la terminología o nomenclatura del dominio. Ver en el anexo #2 el diagrama propuesto por el analista.

Además se realiza una descripción detallada de cada una de las historias de usuario que aparecen a continuación, las cuales permitieron llevar a cabo la solución que se propone.

Inicializar comunicación
Obtener información biométrica
Enrolar información biométrica
Realizar MoC
Eliminar información biométrica

Desbloquear contenedor información biométrica
Restablecer estado de autenticación
Seleccionar Applet
Obtener propiedades
Gestionar estado
Abandonar estado virgen
Cambiar llave
Verificar llave
Finalizar comunicación

Tabla 3. Historias de Usuarios

Posterior al análisis de la propuesta de diseño del analista, que tiene una gran repercusión puesto que decide qué se asume y qué se modifica para la implementación del sistema, se pueden identificar las funcionalidades a desarrollar para que el mismo funcione correctamente, tomando como guía la descripción de las historias de usuario.

2.3 Análisis de implementaciones ya existentes que pueden ser rehusadas

Las aplicaciones que se describen a continuación sirvieron para llevar a cabo la solución propuesta, debido a que el *applet uMatchApp* utiliza ambas API, ya que estas le proveen de diferentes servicios que le son necesarios para su funcionamiento.

2.3.1 API de Java Card

La API de *Java Card* provee a los *applets* un juego de clases personalizadas que le permiten programar aplicaciones de *smart cards* acordes al modelo ISO/IEC 7816.

2.3.2 API de Global Platform

La API de *Global Platform* provee servicios a aplicaciones, por ejemplo: verificaciones del titular de la tarjeta, personalizaciones o servicios de seguridad, además de servicios de administración del contenido de la tarjeta. Provee además de un canal seguro que permite el cifrado de la información que viaja hacia el *applet* y presenta estados estandarizados que se pueden integrar a la solución y así tener un mayor control de la misma.

2.4 Estándares de codificación

Un estándar de codificación comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, que dé la impresión de que fue escrito por un solo programador, esta importante determinación se debe tomar al iniciar un proyecto haciendo que los implementadores

trabajen de forma coordinada. La legibilidad del código fuente repercute directamente en el entendimiento que pueda tener otro programador del mismo, aspecto crucial ya que todo software tiene que someterse constantemente a mantenimiento y mejora de sus funcionalidades. El mejor método para lograr que un grupo de desarrolladores mantenga un código de calidad es establecer un estándar de codificación sobre el cual se realizarán revisiones rutinarias. (msdn, 2010)

La nomenclatura que se especifica a continuación es la que se deberá llevar a cabo en la implementación del *applet uMatchApp*, teniendo en cuenta que los nombres de las clases, funcionalidades y atributos presentes en el *applet* serán escritos en idioma inglés.

2.4.1 Nomenclatura de las clases

Los nombres de las clases deben comenzar con la primera letra en mayúscula y el resto se pondrá con minúscula, en caso de que el nombre sea compuesto la primera letra de cada palabra comenzará con mayúscula.

Ejemplo:

- APDUInterpreter
- ManagerContainerBiometricData

2.4.2 Nomenclatura de las funcionalidades

En el caso de las funciones de la aplicación, se seguirá la norma de codificación de java que establece que la primera letra de la funcionalidad siempre irá en minúscula y si el nombre es compuesto la palabra que le sigue irá en mayúscula.

Ejemplo:

- storeBiometricData(byte[] biometricData, short offsetBiometricData, short lengthBiometricData)

2.4.3 Normas de comentariado

Se debe comentariar lo que se haga dentro del desarrollo, establecer las pautas que conlleven a lograr un código más legible y reutilizable y así se pueda aumentar su mantenibilidad a lo largo del tiempo.

Nomenclatura de los comentarios

Los comentarios deben ser claros y precisos de forma tal que se entienda el propósito de los que se está desarrollando, estos siempre van a estar seguido de: // en la línea donde se quiera especificar dicho comentario.

2.4.4 Estructura del comando APDU

Unidad de datos de protocolo de aplicación (*Application Protocol Data Unit*), es la unidad de comunicación entre un lector y una tarjeta. Su estructura está definida en el estándar

ISO 7816, existiendo dos tipos de categorías de APDU, APDU Command (Comando APDU) y APDU Response (APDU Respuesta).

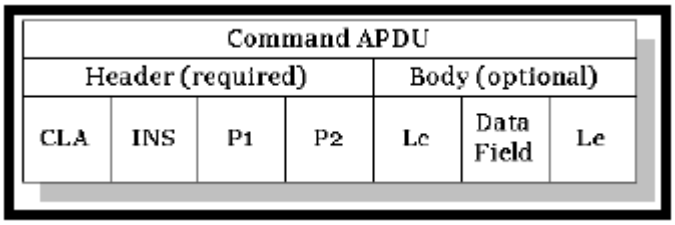


Ilustración 4. Estructura del comando APDU

Descripción del comando APDU

Command APDU		
Campo	**Tamaño**	**Descripción**
CLA	1 byte	Clase de instrucción. Indica la estructura y el formato.
INS	1 byte	Código de instrucción. Especifica la instrucción del comando.
P1	1 byte	Parámetros de la instrucción. Proveen más información sobre la instrucción.
P2	1 byte	
LC	1 byte	Número de bytes en el Data Field del APDU.
Data Field	LC bytes	Secuencia de bytes con información.
LE	1 byte	Cantidad máxima de bytes esperados como respuesta.

Tabla 4. Descripción del comando APDU

Estructura del APDU Respuesta

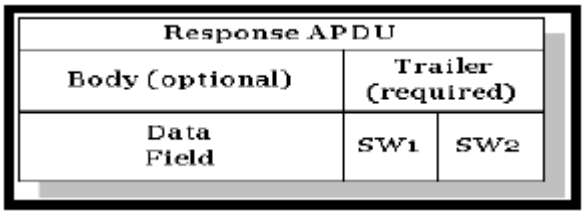


Ilustración 5. Estructura del APDU Respuesta

Descripción del comando APDU Respuesta

APDU Response		
Campo	**Tamaño**	**Descripción**
Data Field	Hasta LE bytes	Secuencia de bytes con información.
SW1	1 byte	Status Word (palabra de estado). Denotan el estado del procesamiento del comando en la tarjeta.
SW2	1 byte	

Tabla 5. Descripción del comando APDU Respuesta

2.5 Resumen de funciones

El *applet uMatchApp* es compatible con un número fijo de comandos APDU. La siguiente tabla muestra cada comando y el estado en el que se desarrollará cada uno. Estos comandos definidos a continuación son las funcionalidades que expondrá el *applet uMatchApp*, las cuales le permitirán llevar a cabo la gestión de la información biométrica dactilar.

Comando	CLA	INS	P1	P2	ESTADOS					
					Virgen	Inicialization	Administration	Operational	Locked	Transport
Comandos Biométricos										
Enroll Biometric Data	80h	30h	Templ #	Seq			x			
Verify Biometric Data	80h	32h	Templ #	Seq			x	x		
Get Biometric Data	80h	34h	Templ #	00h			x	x		
Delete Biometric Data	80h	35h	Templ #	00h			x			
Unblock Biometric Data	80h	38h	Templ #	00h			x			
Reset Authentication Stat	80h	3Ah	Templ #	00h			x	x		
Comandos Criptográficos										
Select Applet	80h	A4h	04h	00h	x	x	x	x	x	x
Get Properties	80h	40h	00h	00h	x	x	x	x	x	x
Change State Control	80h	42h	State ID	00h		x	x	x	x	x
Change Key	80h	44h	Key ID	00h		x	x			
Verify Key	80h	46h	Key ID	00h		x	x	x	x	
Leave Virgin State	80h	48h	00h	00h	x					

Ilustración 6. Comandos del applet uMatchApp

2.5.1 Diagrama de estados del ciclo de vida del applet uMatchApp

El diagrama de estados del *applet uMatchApp* muestra los estados por los que pasa el mismo durante su ciclo de vida, los estados marcados como persistentes 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. Este diagrama de estados le permitirá a todo aquel que haga uso de la aplicación *uMatchApp* entender de forma clara su funcionamiento. Ver anexo # 3 y 4.

2.6 Tratamiento de errores

El tratamiento de errores es de suma importancia para garantizar un correcto funcionamiento del sistema. Este último es el encargado de capturar todas las excepciones que son lanzadas y se le facilita un tratamiento para que no colapse. De esta forma se le da solución a los problemas que implican lanzamiento de excepciones dentro del sistema según las peticiones del usuario, capturando así los tipos de errores lanzados y mostrándole al usuario mensajes de confirmación para saber con certeza lo que está incorrecto y como debe darle solución.

La tabla muestra los mensajes generales que retornan desde el applet y una breve descripción de cada uno de ellos. Ver anexo # 5.

2.7 Descripción de las clases

La aplicación *Applet Secure Fingerprint Manager* está formada por las siguientes clases:

Clases	Descripción
ContainerBiometricData	Se almacena toda la información biométrica de una plantilla.
ManagerBiometricData	Se almacenan y gestionan los contenedores de plantillas biométricas.
StateControl	Se definen los estados del applet *uMatchApp*.
AppletProperties	Se almacenan las propiedades del applet *uMatchApp*.
APDUInterpreter	Se interpretan y ejecutan cada uno de los comandos que soporta el applet *uMatchApp*.
uMatchApp	Esta clase es la que gestiona las demás clases que componen el applet.

Tabla 6. Descripción de las clases del applet

2.7.1 Descripción de las funcionalidades

- **Enroll Biometric Data (Enrolar datos biométricos)**

Este comando es el que permite almacenar la plantilla biométrica y al cual se le pasará por parámetros el arreglo que contiene la información biométrica, así como la posición donde se va a almacenar y la longitud. Este comando se ejecutará en el estado de Administración.

Estructura del comando:

CLA	INS	P1	P2	Lc	DATA
80h	30h	# Plant.	Seq.Number	Lc	Biometric Data

Tabla 7. Estructura del comando enroll biometric data

En P1 irá el número correspondiente a la plantilla que se va a enrolar, si es la primera plantilla sería 01, si es la segunda sería 02 y así sucesivamente.

En P2 iría 00, 01 o 02 en dependencia de la información que esté almacenando, 00 es para indicar que es el inicio del enrolamiento y es ahí donde viene la biometría de cabecera, 01 indica que viene la data de referencia y 02 indica que ya se va a terminar de enviar información.

Respuesta

Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado satisfactoriamente.
6A	81h	Función no soportada.
69h	84h	Datos inválidos.
61h	00h	Posición incorrecta.

Tabla 8. Comando de respuesta enroll biometric data

Ver tabla 8 (Comandos de retorno) para otras posibles respuestas.

- **Verify Biometric Data (Verificar datos biométricos)**

Este comando permitirá verificar los datos biométricos y se ejecutará en el estado de Administración u Operacional. Este comando no se podrá ejecutar en esta primera versión del *applet*, ya que para que funcione se necesita un módulo o API biométrica dentro de la tarjeta que permita realizar la comparación de huellas dactilares y la tecnología actual que se tiene para realizar la solución no trae incluido dicho módulo biométrico.

Estructura del comando:

CLA	INS	P1	P2	Lc	DATA
80h	32h	# Plant.	Seq. Number	Lc	Biometric Data

Tabla 9. Estructura del comando verify biometric data

En P1 irá el número correspondiente a la plantilla que ya se enroló previamente y se encuentra almacenada en la tarjeta, la cual se verificará contra la que viene en el comando.

En P2 iría el número de secuencia que es usado para separar los múltiples comandos de verificación donde los datos biométricos excedan la longitud máxima de un APDU. El número de secuencia puede ser: 00 (biometría de cabecera), 01 (biometría de referencia) o 02 (fin de la verificación).

Respuesta

Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado satisfactoriamente.
6A	81h	Función no soportada.
63h	00h	Verificación fallida.

Tabla 10. Respuesta del comando verify biometric data

Ver tabla 8 (Comandos de retorno) para otras posibles respuestas.

- **Get Biometric Data (Obtener datos biométricos)**

Este comando permitirá obtener la biometría de cabecera de una plantilla específica y se ejecutará en el estado de Administración u Operacional.

Estructura del comando:

CLA	INS	P1	P2	Lc	DATA
80h	34h	# Plant.	00h	Lc	NA

Tabla 11. Estructura del comando get biometric data

En P1 irá el número correspondiente a la plantilla de la cual quiero obtener la información biométrica.

Respuesta

Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado satisfactoriamente.
61h	00h	Posición incorrecta.
6Ah	82h	Archivo no encontrado.
6A	81h	Función no soportada.

Tabla 12. Respuesta del comando get biometric data

Ver tabla 8 (Comandos de retorno) para otras posibles respuestas.

- **Delete Biometric Data (Eliminar datos biométricos)**

Este comando permitirá eliminar los datos biométricos de la plantilla especificada. Este comando se ejecutará en el estado de Administración.

Estructura del comando:

CLA	INS	P1	P2	Lc	DATA
80h	35h	# Plant.	00h	00h	NA

Tabla 13. Estructura del comando delete biometric data

En P1 irá el número correspondiente a la plantilla que se desea eliminar.

Respuesta

Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado satisfactoriamente.
6Ah	81h	Función no soportada.
6Ah	82h	Archivo no encontrado.
61h	00h	Posición incorrecta.

Tabla 14. Respuesta del comando delete biometric data

Ver tabla 8 (Comandos de retorno) para otras posibles respuestas.

- **Unblock Biometric Data**

Este comando permitirá desbloquear la información biométrica de una plantilla y se ejecutará en el estado de Administración.

CLA	INS	P1	P2	Lc	DATA
80h	38h	# Plant.	00h	00h	NA

Tabla 15. Estructura del comando unblock biometric data

El parámetro P1 es usado para especificar el número de plantilla que se va a desbloquear. Una plantilla se bloqueará si el contador de intentos fallidos de verificación alcanza su valor máximo, que sería 10.

Respuesta

Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado satisfactoriamente.
61h	00h	Posición incorrecta.
69h	85h	Condiciones no satisfechas.
6Ah	82h	Archivo no encontrado.
6Ah	81h	Función no soportada.

Tabla 16. Respuesta del comando unblock biometric data

Ver tabla 8 (Comandos de retorno) para otras posibles respuestas.

- **Reset Authentication Status (Restablecer estado de autenticación)**

Este comando permitirá resetear el contador de intentos fallidos de una plantilla biométrica específica. Este comando se ejecutará en el estado de Administración u Operacional.

CLA	INS	P1	P2	Lc	DATA
80h	3Ah	# Plant.	00h	00h	NA

Tabla 17. Estructura del comando reset authentication status

El parámetro P1 es usado para especificar el número de plantilla a la cual se le va a resetear el contador de intentos.

Respuesta

Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado satisfactoriamente.
6Ah	81h	Función no soportada

Tabla 18. Respuesta del comando reset authentication status

Ver tabla 8 (Comandos de retorno) para otras posibles respuestas.

- **Select Applet (Seleccionar el applet)**

Este comando permitirá seleccionar una instancia del *applet uMatchApp* y este se puede ejecutar durante todo el ciclo de vida del applet.

CLA	INS	P1	P2	Lc	DATA
80h	A4h	04h	00h	Lc	AID Data

Tabla 19. Estructura del comando select applet

Respuesta

Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado satisfactoriamente.
6Ah	82h	Aplicación no encontrada o inválido AID.

Tabla 20. Respuesta del comando select applet

Ver tabla 8 (Comandos de retorno) para otras posibles respuestas.

- **Get Properties (Obtener propiedades)**

Este comando permitirá conocer información general acerca del applet como por ejemplo: el número de plantillas que han sido enroladas. Este se ejecutará a lo largo de todo el ciclo de vida del *applet uMatchApp*.

CLA	INS	P1	P2	Lc	DATA
80h	40h	00h	00h	Lc	NA

Tabla 21. Estructura del comando get properties

Respuesta

Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
6Ah	86h	Parámetros P1 y/o P2 incorrectos
90h	00h	Comando procesado satisfactoriamente

Tabla 22. Respuesta del comando get properties

Ver tabla 8 (Comandos de retorno) para otras posibles respuestas.

Change State Control (Cambiar estado)

Este comando permitirá cambiar el estado del ciclo de vida del *applet uMatchApp* y se puede ejecutar en todos los estados del applet excepto en el estado Virgen.

CLA	INS	P1	P2	Lc	DATA
80h	42h	State ID	00h	00h	NA

Tabla 23. Estructura del comando change state

En P1 puede estar cualquiera de los siguientes ID de estados que aparecen en la tabla a continuación en dependencia del estado que se quiera indicar.

State ID (P1)	Nombre del estado
01h	Virgen
02h	Inicialización
04h	Administración
08h	Operacional
10h	Transporte
80h	Bloqueado

Tabla 24. State ID

Respuesta

Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado satisfactoriamente.
6Ah	81h	Función no soportada

Tabla 25. Respuesta del comando change state

Ver tabla 8 (Comandos de retorno) para otras posibles respuestas.

Change Key (Cambiar llave)

Este comando actualiza o inicializa la llave *AdminKey* del *applet uMatchApp* y se ejecutará en el estado de Inicialización o Administración.

CLA	INS	P1	P2	Lc	DATA
80h	44h	Key ID	00h	Lc	Key Data

Tabla 26. Estructura del comando change key

En P1 debe ir el ID de la llave, que en este caso sería 01h referido a la *AdminKey*.

Respuesta

Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado satisfactoriamente.
6Ah	81h	Función no soportada.

Tabla 27. Respuesta del comando change key

Ver tabla 8 (Comandos de retorno) para otras posibles respuestas.

- **Verify Key (Verificar llave)**

Este comando verifica la llave específica del *applet uMatchApp* y se ejecuta en todos los estados del applet excepto en el estado Virgen y el de Transporte.

CLA	INS	P1	P2	Lc	DATA
80h	46h	Key ID	00h	Lc	Key Data

Tabla 28. Estructura del comando verify key

En P1 debe ir el ID de la llave, que sería 01h referido a la *AdminKey* o 02h que se refiere a la *StartKey*.

Respuesta

Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado satisfactoriamente.
6Ah	81h	Función no soportada.
69h	85h	Condiciones no satisfechas

Tabla 29. Respuesta del comando verify key

Ver tabla 8 (Comandos de retorno) para otras posibles respuestas.

- **Leave Virgen State (Abandonar estado virgen)**

Este comando permite abandonar el estado Virgen y retorna el valor de la llave *StartKey*, el cual se obtiene mediante una función *RandomData*, para luego pasar al estado de Inicialización. Este comando solo se ejecuta en el estado Virgen del *applet*.

CLA	INS	P1	P2	Lc	DATA
80h	48h	00h	00h	Lc	NA

Tabla 30. Estructura del comando leave virgin state

Respuesta

Los posibles valores para SW1 y SW2 son los siguientes:

SW1	SW2	Descripción
90h	00h	Comando procesado satisfactoriamente.
6Ah	81h	Función no soportada.

Tabla 31. Respuesta del comando leave virgin state

Ver tabla 8 (Comandos de retorno) para otras posibles respuestas.

2.8 Canal seguro SCP01 de Global Platform

Para el protocolo de canal seguro 01 el nivel de seguridad establecido es un mapa de combinación de bits con los siguientes valores: *AUTHENTICATED, C_MAC, and C_DECRYPTION.* El nivel de seguridad en el canal seguro se establecerá de la siguiente forma:

- *NO_SECURITY_LEVEL*: Cuando la sesión de canal seguro se ha cerrado o aún no ha inicializado completamente.
- *AUTHENTICATED* después de haber ejecutado exitosamente el comando *EXTERNAL AUTHENTICATE*: *AUTHENTICATED* deberá estar limpia, o sea despejado, una vez cerrada la sesión de canal seguro.
- *AUTHENTICATED* y *C_MAC* después de haber ejecutado exitosamente el comando *EXTERNAL AUTHENTICATE* con P1 indicando *C_MAC* (P1='01'): *AUTHENTICATED* y *C_MAC* deberán estar limpias, o sea despejados una vez cerrada la sesión de canal seguro.
- *AUTHENTICATED*, *C_MAC* y *C_DECRYPTION* después de haber ejecutado exitosamente el comando *EXTERNAL AUTHENTICATE* con P1 indicando *C_MAC* y los comandos de encriptación (P1='03'): *AUTHENTICATED*, *C_MAC* y *C_DECRYPTION* deberán estar limpias, o sea despejados una vez cerrada la sesión de canal seguro. (Global Platform, 2003)

2.8.1 Llaves criptográficas

Llaves	Uso	Longitud	Observación
Canal seguro de llave de cifrado (S-ENC).	Autenticación del canal seguro y Encriptación (DES).	16 bytes	Obligatorio
Mensaje Código de autenticación de clave de canal seguro (S-MAC).	Verificación del canal seguro MAC (DES).	16 bytes	Obligatorio
Llave de encriptación de datos (DEK).	Descifrado de datos sensibles (DES).	16 bytes	Obligatorio

Tabla 32. Llaves del canal seguro en un dominio de seguridad

Un dominio de seguridad deberá tener al menos una llave que contenga tres claves para ser utilizadas en el uso y la inicialización del canal seguro. Estas claves son todas de doble longitud de claves DES[6] (Data Encryption Standard) y son las siguientes:

- La llave de encriptación del canal seguro (S-ENC) y la llave MAC del canal seguro (S-MAC). Estas llaves solo son usadas para generar la sesión de llaves del canal seguro durante la inicialización de este.
- La llave de datos de encriptación (DEK) para descifrar los datos sensibles, ejemplo: las llaves secretas privadas. Esta llave es una llave de longitud doble de llave DES y es usada como una llave estática. (Global Platform, 2003)

Ver anexos # 6, 7 y 8.

2.8.2 Generación y verificación de la MAC del comando APDU

El canal seguro establece el uso de la MAC en el comando EXTERNAL AUTHENTICATE. Dependiendo del nivel de seguridad definido en la inicialización del canal seguro todos los otros comandos que estén dentro del canal seguro pueden requerir mensajería segura y como tal el uso de un C-MAC.

Un C-MAC es generado por la entidad que está fuera de la tarjeta y es aplicado a través del comando APDU completo, siendo transmitido hacia la tarjeta incluyendo la cabecera (5 bytes) y el campo de datos en el mensaje de comando.

Las reglas para la modificación del comando cabecera APDU son las siguientes:

- La longitud del mensaje del comando (Lc) se incrementa en 8 para indicar la inclusión de la CMAC en el campo de datos del mensaje de comando.
- El byte de la clase deberá ser modificado para indicar que este comando APDU incluye mensajería segura. Esto se logra mediante el establecimiento de bit 3 en los

[6] DES (Data Encryption Standard) es un algoritmo de cifrado que permite cifrar la información.

bytes de la clase. Para todos los comandos definidos en esta especificación los bytes de los comandos de la clase que contienen mensajería segura será '84'. La generación y verificación de C-MAC no refleja la información del canal lógico incluido en el byte del comando de la clase. (Global Platform, 2003) Ver anexo # 9.

2.8.3 Cifrado y descifrado de los campos de datos del comando APDU

En dependencia del nivel de seguridad definido en la inicialización del canal seguro todos los comandos APDU siguientes que estén dentro del mismo pueden requerir mensajería segura y como tal del uso del C-MAC (integridad) y cifrado (confidencialidad).

Si la confidencialidad es necesaria la entidad que está fuera de la tarjeta cifra el texto claro de los campos de datos del comando, mensaje que será transmitido a la tarjeta. En caso de que el comando APDU no contenga los datos del comando original entonces no se puede cifrar la información. Antes de cifrar los datos estos deberán de ser rellenados. El relleno del campo de datos a cifrar se lleva a cabo de acuerdo a las siguientes reglas:

- La longitud del "texto claro" original, el campo de datos se anexa a la izquierda y se convierte en parte de los datos del comando.
- Si la longitud del campo de datos es un múltiplo de 8. (Global Platform, 2003)

Ver anexo # 10.

2.9 Antecedentes de la solución propuesta

El presente trabajo da continuidad al trabajo de diploma titulado: *"Diseño de un Applet y Middleware para gestionar la información biométrica contenida en la Cédula de Identificación Electrónica de la República Bolivariana de Venezuela",* el cual se encargó de realizar el análisis y diseño de la solución que se propone implementar. La solución propuesta solo abarcará el desarrollo del *applet uMatchApp*.

2.9.1 Propuesta del sistema

La aplicación que se propone es una aplicación genérica, que funcionará en cualquier tarjeta inteligente, la cual realiza la gestión de la información biométrica de las huellas dactilares y simula una verificación de los datos de referencia de la huella, ya que no se cuenta con la tecnología necesaria para llevar a cabo una válida verificación de huellas. En un futuro, mediante un módulo o API biométrica que se encuentre embebida dentro de la tarjeta, y haciendo uso de la tecnología *Match on Card,* se podrá realizar la verificación biométrica. A continuación se explica cómo funcionarán ambos procesos.

- **Proceso de enrolamiento de huellas**

El primer paso para cualquier solución de autenticación biométrica es la inscripción del usuario al sistema, lo cual no es más que el proceso de enrolamiento, donde el sistema es el encargado de capturar los datos o imágenes específicas de una huella dactilar que

serán utilizados en la autenticación biométrica, al extraer las características únicas dentro de una plantilla de referencia y luego enviar dicha información hacia el *applet uMatchApp* donde será almacenada.

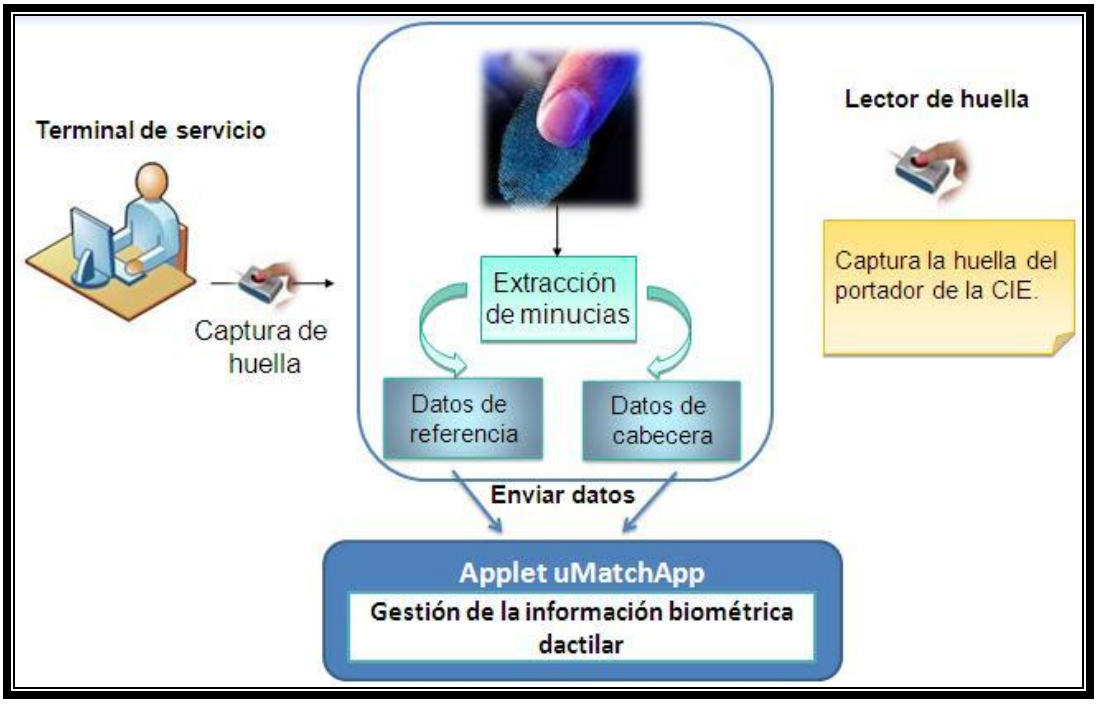


Ilustración 7. Proceso de enrolamiento de huellas

- **Proceso de verificación de huellas**

El proceso de verificación comienza cuando al usuario se le captan las huellas dactilares y se obtienen las minucias de las mismas estas son enviadas al *applet uMatchApp* donde se procede a comparar con la huella que se encuentra previamente almacenada dentro de la tarjeta inteligente (MoC), si la verificación realizada es correcta el *applet* envía un mensaje de autenticación satisfactoria, en caso contrario envía un error.

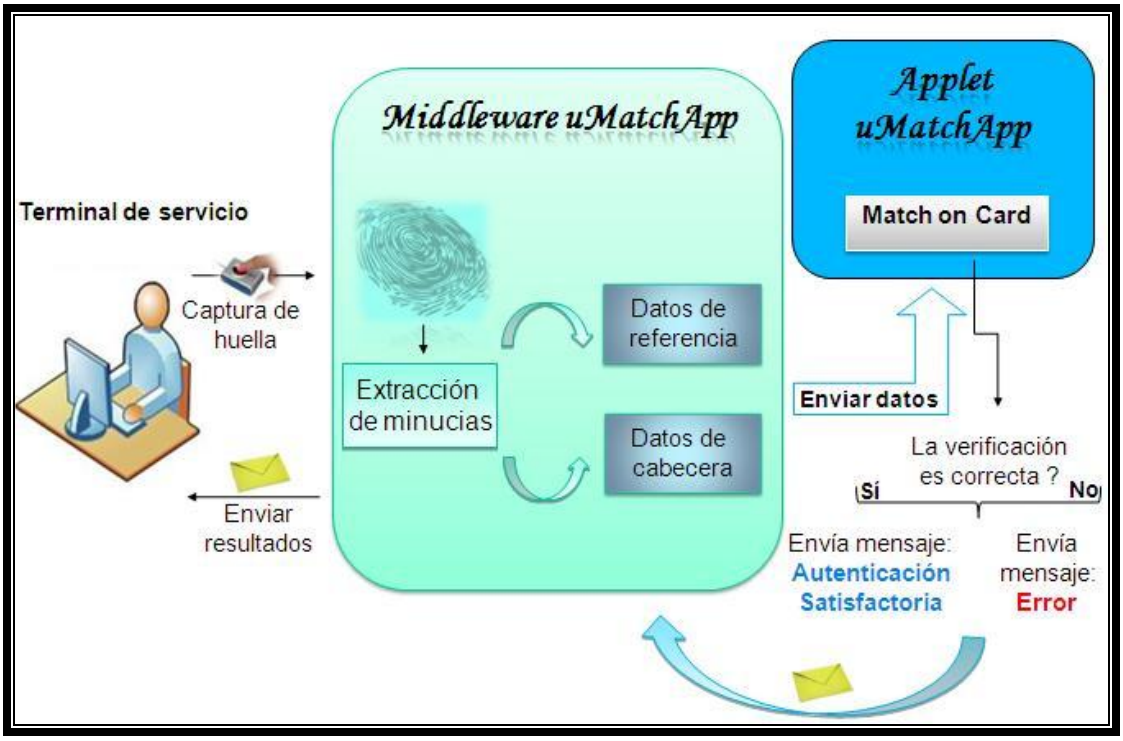


Ilustración 8. Proceso de verificación de huellas mediante la tecnología MoC

2.10 Arquitectura del sistema

El modelo de arquitectura de la solución en su vista más abstracta es un modelo Cliente - Servidor. Este modelo está basado en el principio fundamental de un cliente que realiza peticiones a otro programa, y el servidor le da respuesta. Para la solución propuesta, el *middleware* interpreta la disposición de un cliente, mientras que el *applet* funciona como servidor, dicha arquitectura refleja cómo va a funcionar la aplicación *applet uMatchApp*. Por su parte la interfaz compartida les permite saber a otros *applets* que funcionalidades expone el *applet uMatchApp*. Conjuntamente, todos los datos biométricos asociados a una huella dactilar específica se almacenarán en un contenedor de plantilla. Durante la instalación del applet un número configurable de contenedores de plantilla se asigna y se inicializan. El *applet uMatchApp* permitirá enrolar dos contenedores de plantilla. La figura a continuación muestra lo anteriormente expresado.

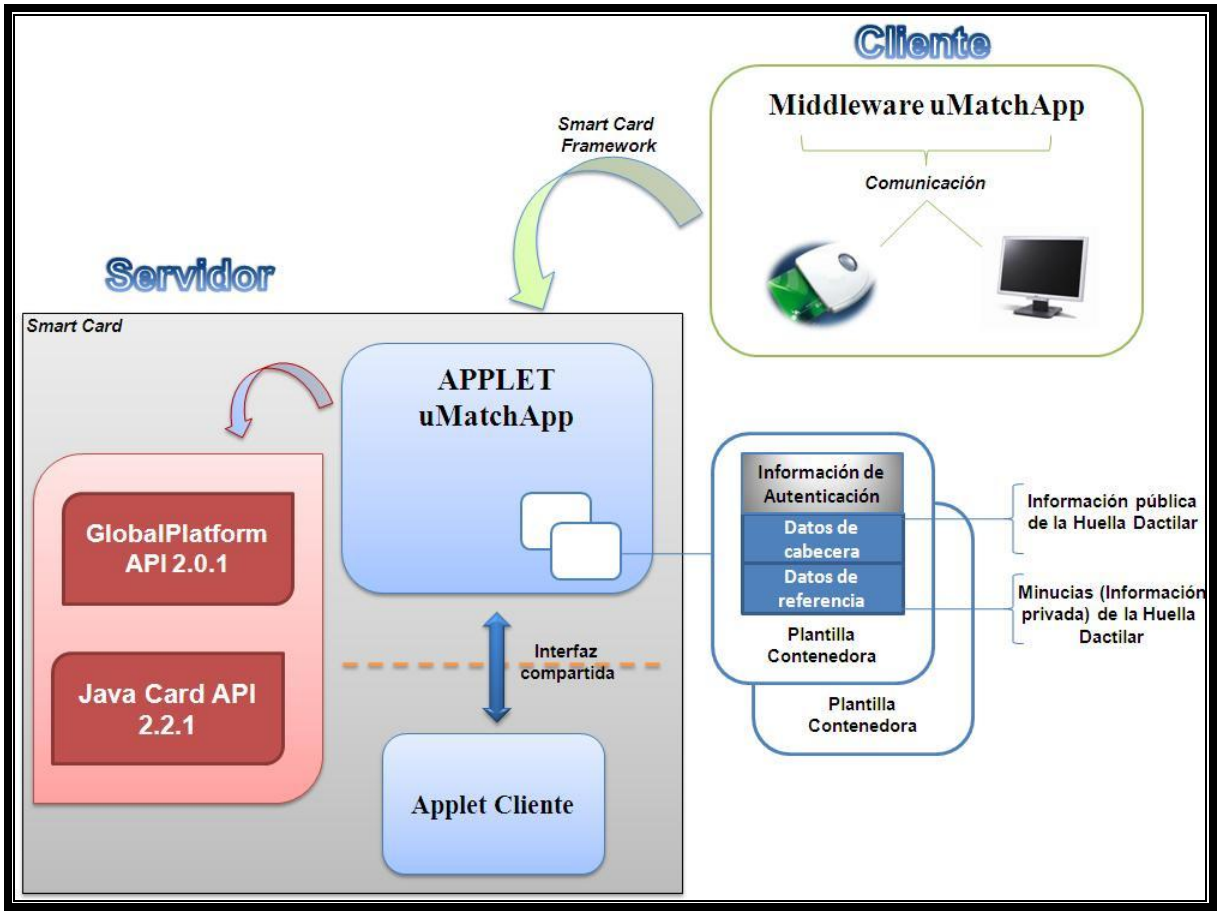


Ilustración 9. Arquitectura del applet uMatchApp

2.11 Integración entre componentes

El *middleware uMatchApp* que se desarrollará en un futuro, es un módulo común que permitirá establecer la comunicación entre un ordenador y los distintos lectores de tarjetas inteligentes, este será capaz de comunicarse con el *applet* y realizarle consultas en el orden de la información que persista en la tarjeta. En el departamento existe un *middleware* ya implementado para otro *applet* (*CryptoManager*) que gestiona la información biométrica y que además utiliza un API biométrica para realizar la comparación de la huella dactilar dentro de la tarjeta inteligente, el cual puede servir como punto de partida para el *middleware* que se implementará para el *applet uMatchApp*.

2.11.1 Vista de integración. Estilos arquitectónicos

En el sistema se puede apreciar el uso del estilo arquitectónico en capas.

- Arquitectura en capas:

Los sistemas o arquitecturas en capas constituyen uno de los estilos que aparecen con mayor frecuencia. Este estilo un conjunto de capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las

prestaciones que le brinda la inmediatamente inferior. Las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas.

El uso de las ventajas de cada uno de los patrones antes mencionados permite y lleva al sistema a obtener un alto grado de flexibilidad e integración, adaptación al cambio de interfaces de usuario ya que estos tienen a cambiar con mayor rapidez que las reglas de negocio. Permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales.

2.11.2 Descripción

La comunicación del *middleware* con el *applet* se realiza a través de capas que son necesarias para establecer un canal correcto de transmisión de la información.

En la siguiente figura se muestran las capas por donde fluye la comunicación entre el *middleware* y el *applet*.

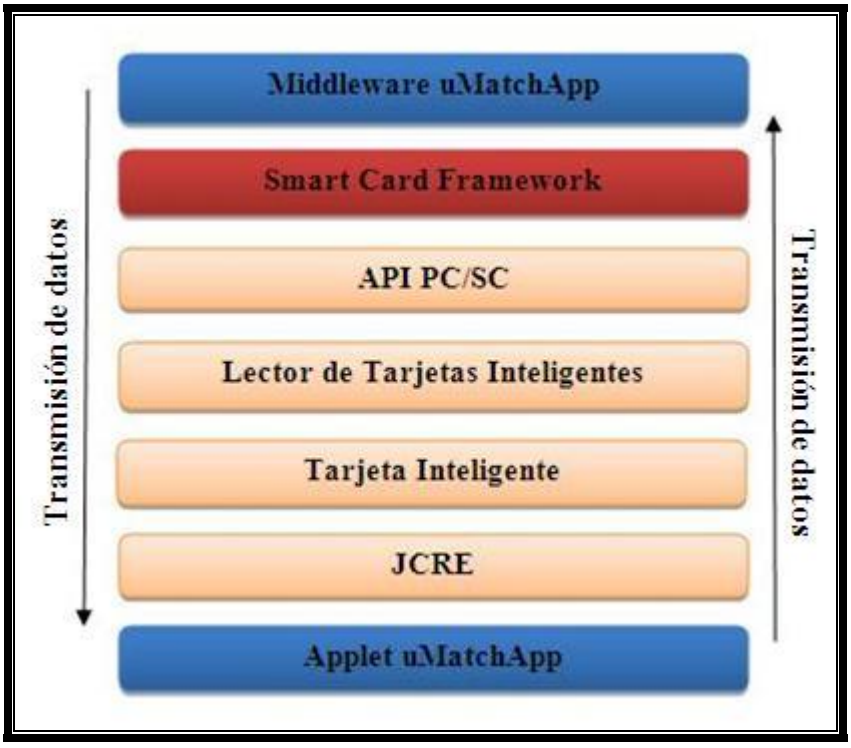


Ilustración 10. Capas de comunicación entre el middleware y el applet

Dentro de las capas por las que fluyen los datos en la comunicación entre el *middleware* y el *applet* se encuentran: el *middleware SmartCard Framework*; una API para la comunicación con los lectores de tarjetas inteligentes; el lector de tarjetas; la tarjeta inteligente; y finalmente el *Java Card Runtime Enviroment* que contiene la máquina virtual de *Java Card* y el conjunto de clases del *Framework Java Card*. De estas capas, solamente se desarrollan como parte de la solución el *applet uMatchApp*, debido a que el

resto de las capas son propias de las tecnologías que se utilizan y el *middleware* ya se encuentra implementado.

2.12 Conclusiones

Con la solución obtenida en el presente capítulo se logró utilizar una tarjeta inteligente como dispositivo para gestionar la información biométrica dactilar, aplicación que puede integrarse con un *applet* de *MoC*, permitiéndole al mismo utilizar toda la información biométrica gestionada, para realizar la verificación de esta. Por su parte las historias de usuario, la arquitectura definida para el sistema y la utilización del canal seguro SCP01 de *Global Platform,* posibilitaron el desarrollo del sistema, arrojando así, una aplicación segura utilizando un canal de comunicación cifrado.

Capítulo 3: Validación de la solución propuesta

3.1 Introducción

La prueba del software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación. El objetivo de la etapa de pruebas es garantizar la calidad del producto desarrollado. Es una actividad en la cual un sistema o componente es ejecutado bajo condiciones o requerimientos específicos, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente.

En el presente capítulo se hace una búsqueda de las técnicas de prueba de software que permitan comprobar la validez de la solución propuesta. Además se detallan los casos de prueba con sus objetivos y alcance, y se analizan los resultados obtenidos en las mismas.

3.2 Diseño de las pruebas unitarias

Las pruebas unitarias permiten probar cada unidad independiente del software. Actúan esencialmente sobre el código fuente y sobre los elementos básicos de la interfaz de cada módulo. Los casos de prueba que se generan durante las pruebas de unidad deben estar encaminados a verificar los siguientes elementos:

- Interfaz: Se prueba la interfaz del módulo para asegurar que la información fluye de forma adecuada hacia y desde la unidad de programa que está siendo probada.
- Estructuras de datos locales: Se examinan las estructuras de datos locales para asegurar que los datos que se mantienen temporalmente, conservan su integridad durante todos los pasos de ejecución del algoritmo.
- Condiciones límites: Se prueban las condiciones límites para asegurar que el módulo funciona correctamente en los límites establecidos como restricciones de procesamiento.
- Caminos independientes: Se ejercitan todos los caminos independientes de la estructura de control para asegurar que todas las sentencias del módulo se ejecutan por lo menos una vez.
- Caminos de manejo de errores: Se prueban todos los caminos de manejo de errores.

Con las pruebas unitarias es posible aislar una parte del código de manera que pueda ser analizado. Un ejemplo de esto es evaluar las funciones o métodos, a los cuales se les realiza una entrada de datos para obtener los datos de salida correctos. Este tipo de pruebas validan la forma en la que las funciones y métodos trabajan en cada caso particular. (Alberto Elers Pérez, 2010)

3.3 Tipos de pruebas de software

La calidad de un sistema está determinada, entre otras cosas, por la coincidencia entre lo que se programó y los requisitos establecidos en la primera fase. Para comprobar el grado de cumplimiento de estos requisitos se usan las pruebas del sistema. La prueba de unidad es la prueba enfocada a los elementos probables más pequeños del software, consiste en una prueba estructural (o caja blanca), lo cual requiere conocer el diseño interno de la unidad puesto que verifica la lógica interna y el flujo de datos; y una prueba de especificación (de caja negra), basada sólo en la especificación del comportamiento externamente visible de la unidad.

Estas definen un conjunto amplio de acciones de comprobación que abarcan todas las características que determinan la calidad de un software. Se comprueban las funcionalidades diseñando casos de prueba que definen cómo proceder. Estos casos de prueba incluyen los juegos de datos a usar que son los válidos o esperados y los no válidos o no esperados por el programa. Las pruebas se deben aplicar durante todo el ciclo de vida del software e invariablemente se le debe dedicar una gran parte del esfuerzo total del desarrollo. Se deben planificar correctamente desde el inicio y establecer qué hacer, cómo hacer, quién va a hacer y en qué condiciones hacer las comprobaciones.

El objetivo de los casos de prueba es forzar al máximo el sistema en los puntos críticos para encontrar fallos y detectar defectos. Se debe escoger los tipos de prueba que se adapten mejor al sistema que se va a probar. Para esto se debe tener en cuenta el lenguaje de programación, el proceso de desarrollo, las características de los desarrolladores, el tipo de funcionalidad que se implementa, la plataforma en que se ejecutan los procesos, los errores más importantes, si la aplicación es de escritorio o web, si realiza conexiones a bases de datos entre otras observaciones. (Alberto Elers Pérez, 2010)

3.3.1. Pruebas de Caja Blanca

Las pruebas de Caja Blanca se nombran de esta forma porque revisan la parte interna del software, específicamente sobre el código fuente. Se basan en el examen minucioso de los detalles procedimentales. Se comprueban los caminos lógicos del sistema generando casos de prueba que ejerciten las estructuras condicionales y los bucles. Existen varios métodos que analizan diferentes partes del programa y se complementan entre sí para garantizar la calidad del sistema. La técnica del camino básico se utiliza para comprobar la complejidad lógica de un diseño procedimental, permite diseñar casos de prueba para cubrir todas las sentencias de un programa a partir de la obtención de un conjunto de caminos independientes.

La complejidad ciclomática, como resultado fundamental de estas pruebas, acota la cantidad mínima de casos de prueba que se deben ejecutar. La prueba de las Condiciones es un método que se encamina hacia la ejercitación de las condiciones. Se basa en el principio de que si un conjunto de casos de prueba es capaz de ejercitar todas las condiciones contenidas en un bloque de código, este mismo conjunto serviría para encontrar más errores en el programa que no tengan que ver directamente con las condiciones. La prueba del Flujo de Datos verifica la validez en el uso de las variables para manipular los datos de la aplicación. Selecciona los casos de prueba atendiendo a las definiciones y los usos de las variables. El procedimiento indica que se debe encontrar las sentencias donde se define cada variable y las sentencias donde se hace uso de las mismas. Luego se encuentran las cadenas de definición, uso que representan el ciclo de vida de las variables y se diseñan casos de prueba que las ejecuten en su totalidad. La prueba de los Bucles se centra en la validez de las estructuras cíclicas o bucles. El objetivo es probar el comportamiento de estas estructuras en sus valores límites de iteración. Los bucles se clasifican en cuatro tipos: Bucles simples, Bucles anidados, Bucles concatenados y Bucles no estructurados. Aunque la esencia es la misma, cada tipo se prueba de forma diferente. De lo anterior pudiera pensarse que las pruebas de Caja Blanca logran enmendar todos los errores del programa. La desventaja de estas es entonces de tipo logística ya que resulta imposible abarcar todo el código fuente de un sistema medianamente grande. El tiempo necesario para realizarlas sería considerable y se torna compleja su aplicación sobre algoritmos críticos. (Alberto Elers Pérez, 2010)

Diseño de casos de prueba de caja blanca aplicados

El objetivo fundamental del diseño de casos de prueba es conseguir un conjunto de pruebas que tengan la mayor probabilidad de encontrar los defectos del software.

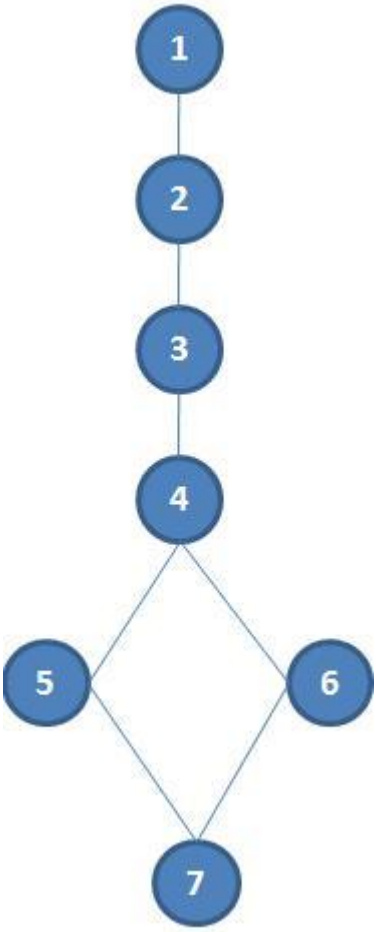
Un caso de prueba es un conjunto de entradas de pruebas, condiciones de ejecución y resultados esperados desarrollados para verificar una función esperada.

- **Caso de prueba cambiar llave**

De acuerdo al segmento de código correspondiente a la historia de usuario: “Cambiar llave (ChangeKey)” se le realiza la prueba de caja blanca.

```
private void changeKey(APDU apdu)
{
        byte[] apduBuffer = apdu.getBuffer();                                   (1)
        if(apdu.getCurrentState()==apdu.STATE_INITIAL)                          (2)
        apdu.setIncomingAndReceive();                                           (3)
        if(apduBuffer[ISO7816.OFFSET_P1] == (byte)0x01 &&
        apduBuffer[ISO7816.OFFSET_P2] == (byte)0x00 &&
        apduBuffer[ISO7816.OFFSET_LC]==(byte)0x08)                              (4)
        {
```

```
        Util.arrayCopyNonAtomic(apduBuffer,
        (short)(ISO7816.OFFSET_CDATA + 1), _AdminKey,
        (short)ISO7816.OFFSET_CLA,
        (short)apduBuffer[ISO7816.OFFSET_LC]);                (5)
        _currentlyAuthenticatedKeys = (byte) 0x01;
                                    (5)
        _VerifyAdminKey = true;
                                    (5)
          }
        else
        ISOException.throwIt(ISO7816.SW_FUNC_NOT_SUPPO
RTED);          (6)
}               (7)
```



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.

V (G) = 7 – 7 + 2

V (G) = 2

Caminos: 1-2-3-4-5-7, 1-2-3-4-6-7

- **Camino**: 1-2-3-4-5-7

Caso de prueba: ChangeKey

Entrada: Se verifica que el ID de la llave entrada en P1 sea 01, referido a la llave de Administración, que el parámetro P2 lo que tenga sea 00 y que la longitud de los datos de la llave sea de 8 bytes.

Resultados: La llave se modifica o se inicializa satisfactoriamente y se muestra el APDU Respuesta 90 00.

Condiciones: Para ejecutar este comando se debe estar en el estado de Inicialización o Administración.

- **Camino**: 1-2-3-4-6-7

Caso de prueba: ChangeKey

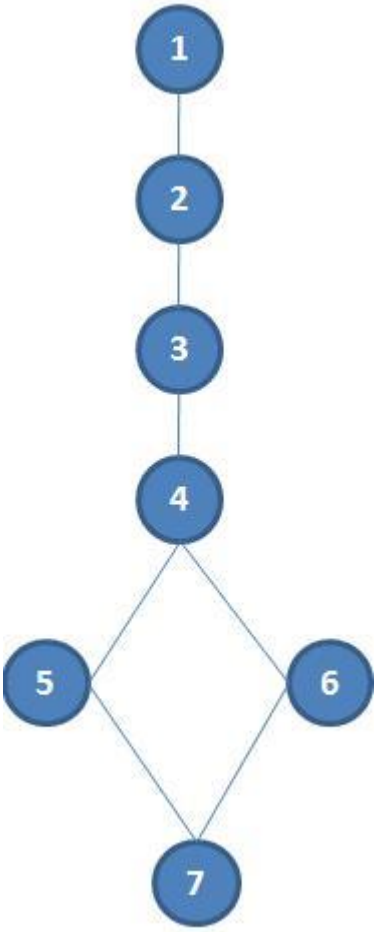
Entrada: Se verifica que el ID de la llave entrada en P1 sea 01, referido a la llave de Administración, que el parámetro P2 lo que tenga sea 00 y que la longitud de los datos de la llave sea de 8 bytes.

Resultados: Se verifica si se cumple alguno de los datos requeridos por la entrada, al no cumplirse por lo menos uno de ellos se muestra el APDU Respuesta 6A 81.

Condiciones: Para ejecutar este comando se debe estar en el estado de Inicialización o Administración.

- **Caso de prueba abandonar estado virgen**

De acuerdo al segmento de código correspondiente a la historia de usuario: "Abandonar estado virgen (LeaveVirginState)" se le realiza la prueba de caja blanca.




```
private void leaveStateVirgin(APDU apdu)
{
        byte[] apduBuffer = apdu.getBuffer();                    (1)
        if(apdu.getCurrentState()==apdu.STATE_INITIAL)           (2)
        apdu.setIncomingAndReceive();                            (3)
        if(apduBuffer[ISO7816.OFFSET_P1] == (byte)0x00 &&
        apduBuffer[ISO7816.OFFSET_P2] == (byte)0x00)             (4)
        {
                _State = StateControl._Initialization;           (5)
                _RandomData.generateData(_StartKey,
                (short)ISO7816.OFFSET_CLA,
                _LenghtStartKey);                                (5)
                Util.arrayCopyNonAtomic(_StartKey,
                (short)ISO7816.OFFSET_CLA, apduBuffer,
                (short)ISO7816.OFFSET_CLA, _LenghtStartKey);
                (5)
                apdu.setOutgoingAndSend((short)ISO7816.OFFS
                ET_CLA, _LenghtStartKey); (5)
        }
        else
                ISOException.throwIt(ISO7816.SW_FUNC_NOT_
                SUPPORTED);                                      (6)
}                                                                (7)
```

Complejidad Ciclomática

V (G) = A – N + 2

V (G) = 7 – 7 + 2

V (G) = 2

Caminos: 1-2-3-4-5-7, 1-2-3-4-6-7

- **Camino**: 1-2-3-4-5-7

Caso de prueba: LeaveVirginState

Entrada: Se verifica que los parámetros P1 y P2 sean 00.

Resultados: Se deja el applet en el estado de inicialización y se crea la llave de inicio (StartKey) satisfactoriamente, en el simulador se muestra el APDU Respuesta 90 00.

Condiciones: Para ejecutar este comando se debe estar en el estado Virgen.

- **Camino**: 1-2-3-4-6-7

Caso de prueba: LeaveVirginState

Entrada: Se verifica que los parámetros P1 y P2 sean 00.

Resultados: Uno de los parámetros (P1, P2) están incorrectos, por lo que en el simulador se muestra el APDU Respuesta 6A 81.

Condiciones: Para ejecutar este comando se debe estar en el estado Virgen.

- **Caso de prueba obtener información biométrica**

De acuerdo al segmento de código correspondiente a la historia de usuario: “Obtener información biométrica (GetBiometricData)” se le realiza la prueba de caja blanca.

```
private void getBiometricData(APDU apdu)
{
        byte []apdubuffer = apdu.getBuffer();                                   (1)
        if(apdu.getCurrentState()==apdu.STATE_INITIAL)                          (2)
                apdu.setIncomingAndReceive();                                   (3)
        if(_isEliminated == false)                                              (4)
        {
                byte[] prueba =
               _ManagerBiometricData.GetContainerBiometricData(apdubuffer[ISO7816.OFFSET
               _P1]).GetBiometricHeader();                                      (5)
                ContainerBiometricData container =
               _ManagerBiometricData.GetContainerBiometricData(apdubuffer[ISO7816.OFFSET
               _P1]);                                                           (5)
                if(apdubuffer[ISO7816.OFFSET_P2] == (byte)0x00)                 (6)
                {
                        if(container.GetIsEnrolled() == true)                   (7)
                        {
                                Util.arrayCopy(prueba, (short)0, apdubuffer, (short)0,
                                (short)prueba.length);                          (8)
                                apdu.setOutgoingAndSend((short)0, (short)prueba.length); (8)
                        }
                        else
                                ISOException.throwIt(ISO7816.SW_FILE_NOT_FOUND); (9)
                }
                else
                        ISOException.throwIt(ISO7816.SW_FUNC_NOT_SUPPORTED);    (10)
        }
        else
                ISOException.throwIt(ISO7816.SW_FILE_NOT_FOUND);                (11)
}                                                                               (12)
```

Complejidad Ciclomática

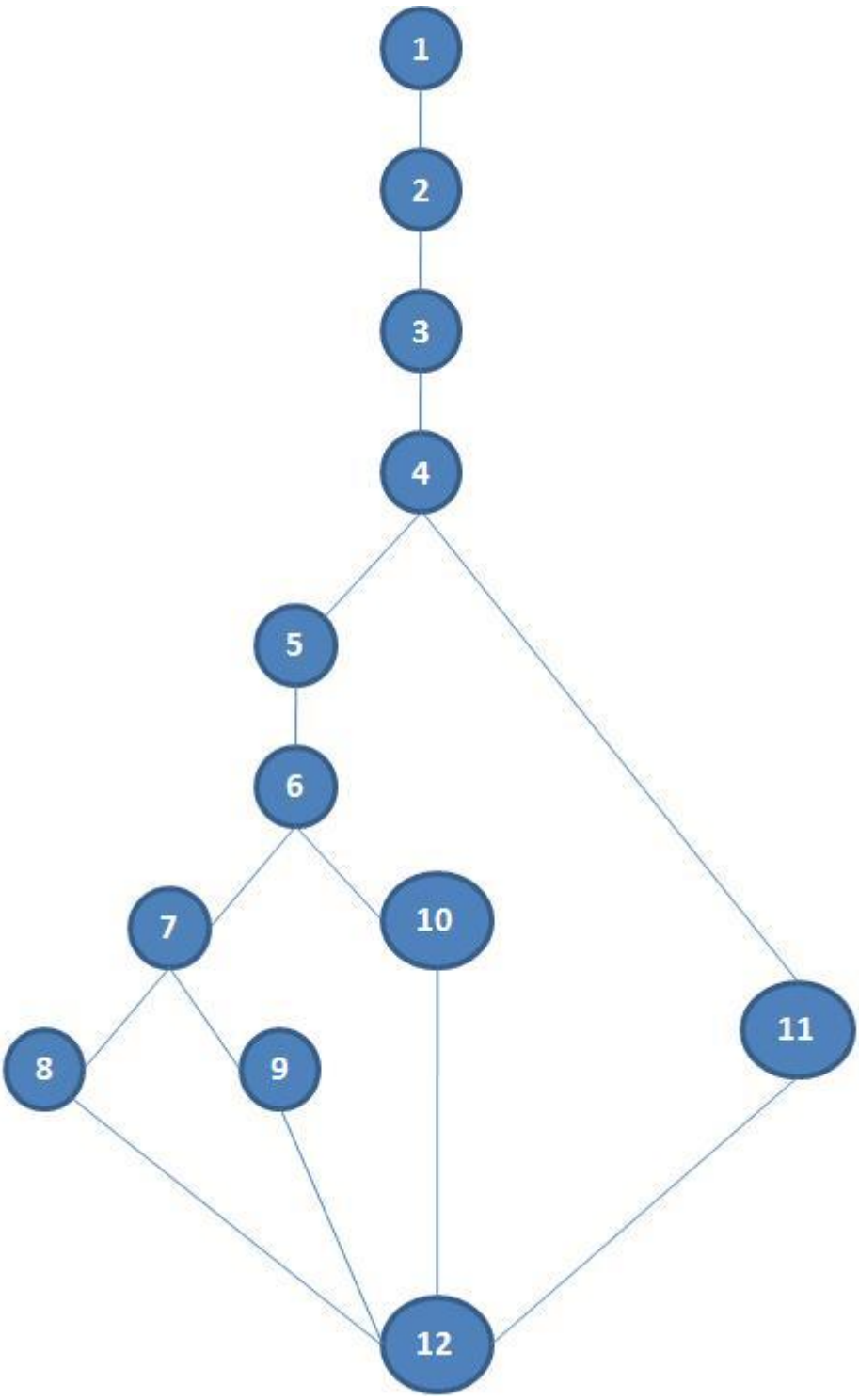
V (G) = A – N + 2

V (G) = 14 – 12 + 2

V (G) = 4

Caminos: 1-2-3-4-11-12, 1-2-3-4-5-6-10-12, 1-2-3-4-5-6-7-8-12, 1-2-3-4-5-6-7-9-12

- **Camino**: 1-2-3-4-11-12



Caso de prueba: getBiometricData

Entrada: Se verifica que la plantilla de la cual se quiere obtener la biometría de cabecera exista.

Resultados: Como la plantilla escogida no existe se muestra en el simulador el APDU Respuesta 6A 82.

Condiciones:

Para ejecutar este comando se debe estar en el estado Administración u Operacional.

- **Camino**: 1-2-3-4-5-6-7-8-12

Caso de prueba: getBiometricData

Entrada: Se verifica que la plantilla de la cual se quiere obtener la biometría de cabecera exista, además los datos entrados en el parámetro P2 deben ser 00 y la plantilla debe haber sido enrolada.

Resultados: Se obtienen los datos biométricos de la cabecera de la plantilla especificada en P1 y en el simulador se muestra el APDU Respuesta 90 00.

Condiciones: Para ejecutar este comando se debe estar en el estado Administración u Operacional.

3.3.2 Pruebas de Caja Negra

Las Pruebas de Caja Negra deben su nombre a los elementos que estas revisan y las condiciones en que se hace la revisión. Estas 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 antes determinadas acciones y los datos de salida para determinados datos de entrada. Los casos de prueba de la caja negra pretenden demostrar que:

- Las funciones del software son operativas
- La entrada se acepta de forma correcta
- Se produce una salida correcta
- La integridad de la información externa se mantiene

A continuación se derivan conjuntos de condiciones de entrada que utilicen todos los requisitos funcionales de un programa.

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

Diseño de casos de pruebas de caja negra

La prueba verifica que el ítem 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 corran bien.

- **Caso de prueba Abandonar estado virgen (Leave Virgin State)**

Escenario	Descripción	CLA	INS	P1	P2	Le	Data	Respuesta del sistema	Flujo central
SC1	Abandonar estado Virgen correctamente	80	48	00	00	-	-	El sistema muestra el APDU Respuesta **90 00**, que indica que se ejecutó el comando correctamente con 8 bytes de información extra.	Se abre el simulador *JCardManager*, en el panel de la izquierda se escoge la opción *Authenticate* que está dentro de la carpeta *op*, que se encuentra dentro de la carpeta *Platform* y ambas aparecen dentro de la carpeta raíz *USim Card R5.* Una vez seleccionada la opción, se desmarca el *check box Use default key file* y se debe cargar un fichero que va a permitir que el canal seguro se autentique contra el *applet uMatchApp* y no contra el *cardManager,* y se pincha en la pestaña *Go.* Luego en el panel de la izquierda se escoge la opción *Send* APDU *Secure* y se envía el comando APDU que indica que se va a abandonar el estado virgen para proceder a los demás estados del applet. En el simulador *JCardManager* se muestra una ventanita en forma de comando APDU

									donde el usuario del sistema introduce los datos correspondientes (CLA, INS, P1, P2, Lc y Data) para ejecutar dicho comando APDU. Al pinchar en la pestaña Go del simulador se muestra un mensaje señalizado en color verde indicando que los datos son correctos.
SC2	Abandonar estado Virgen incorrectamente	80	48	01-99	01-99	-	-	El sistema muestra el APDU Respuesta **6A 81** que indica "Función no soportada", lo que significa que el comando está incorrecto.	Se abre el simulador *JCardManager*, en el panel de la izquierda se escoge la opción *Authenticate* que está dentro de la carpeta *op*, que se encuentra dentro de la carpeta *Platform* y ambas aparecen dentro de la carpeta raíz *USim Card R5.* Una vez seleccionada la opción, se desmarca el *check box Use default key file* y se debe cargar un fichero que va a permitir que el canal seguro se autentique contra el *applet uMatchApp* y no contra el *cardManager,* y se pincha en la pestaña *Go.* Luego en el panel de la izquierda se escoge la opción *Send APDU* y se envía el comando APDU que indica que se va a abandonar el estado virgen para proceder a

									los demás estados del applet. En el simulador *JCardManager* se muestra una ventanita en forma de comando APDU donde el usuario del sistema introduce los datos correspondientes de forma incorrecta. Al pinchar en la pestaña Go del simulador se muestra un mensaje señalizado en color rojo indicando que los datos son incorrectos.

Tabla 33. Caso de prueba Leave virgin state

- **Caso de prueba Verificar llave (Verify Key)**

Escenario	Descripción	CLA	INS	P1	P2	Le	Data	Respuesta del sistema	Flujo central
SC1	Verificar llave StartKey (llave de inicio) correctamente	80	46	02	00	-	# de 8 bytes que contiene la información de la StartKey. (Este número se	El sistema muestra el APDU Respuesta **90 00**, que indica que se ejecutó el comando correctamente.	Se abre el simulador *JCardManager*, en el panel de la izquierda se escoge la opción *Authenticate* que está dentro de la carpeta *op*, que se encuentra dentro de la carpeta *Platform* y ambas aparecen dentro de la carpeta raíz *USim Card R5.* Una

							obtiene del APDU Respuesta que viene cuando se ejecuta el caso de prueba *Leave Virgin State*).		vez seleccionada la opción, se desmarca el *check box Use default key file* y se debe cargar un fichero que va a permitir que el canal seguro se autentique contra el *applet uMatchApp* y no contra el *cardManager,* y se pincha en la pestaña *Go.* Luego en el panel de la izquierda se escoge la opción *Send APDU* y se envía el comando APDU que indica que se va a verificar la llave de inicio (StartKey). En el simulador *JCardManager* se muestra una ventanita en forma de comando APDU donde el usuario del sistema introduce los datos correspondientes (CLA, INS, P1, P2, Lc y Data) para ejecutar dicho comando. Al pinchar en la pestaña Go del simulador se muestra un mensaje señalizado en color verde indicando que los datos son

									correctos.
SC2	Verificar llave AdminKey (llave de administración) correctamente	80	46	01	00	-	00 00 00 00 00 00 00 00	El sistema muestra el APDU Respuesta **90 00**, que indica que se ejecutó el comando correctamente.	Se abre el simulador *JCardManager*, en el panel de la izquierda se escoge la opción *Authenticate* que está dentro de la carpeta *op*, que se encuentra dentro de la carpeta *Platform* y ambas aparecen dentro de la carpeta raíz *USim Card R5.* Una vez seleccionada la opción, se desmarca el *check box Use default key file* y se debe cargar un fichero que va a permitir que el canal seguro se autentique contra el *applet uMatchApp* y no contra el *cardManager,* y se pincha en la pestaña *Go.* Luego en el panel de la izquierda se escoge la opción *Send APDU* y se envía el comando APDU que indica que se va a verificar la llave de administración (AdminKey). En el simulador *JCardManager* se

									muestra una ventanita en forma de comando APDU donde el usuario del sistema introduce los datos correspondientes para ejecutar dicho comando. Al pinchar en la pestaña *Go* del simulador se muestra un mensaje señalizado en color verde indicando que los datos son correctos.
SC3	Verificar llave AdminKey (llave de administración) incorrectamente	80	46	05	01	-	00 00 00 00 00 00 00 00	El sistema muestra el APDU Respuesta **69 85** que indica “Condiciones no satisfechas”, lo que significa que el comando está incorrecto.	Se abre el simulador *JCardManager*, en el panel de la izquierda se escoge la opción *Authenticate* que está dentro de la carpeta *op*, que se encuentra dentro de la carpeta *Platform* y ambas aparecen dentro de la carpeta raíz *USim Card R5.* Una vez seleccionada la opción, se desmarca el *check box Use default key file* y se debe cargar un fichero que va a permitir que el canal seguro se autentique contra el *applet uMatchApp* y no contra el

									cardManager, y se pincha en la pestaña *Go.* Luego en el panel de la izquierda se escoge la opción *Send APDU* y se envía el comando APDU que indica que se va a verificar la llave de administración (AdminKey). En el simulador *JCardManager* se muestra una ventanita en forma de comando APDU donde el usuario del sistema introduce los datos correspondientes para ejecutar dicho comando. Al pinchar en la pestaña Go del simulador se muestra un mensaje señalizado en color rojo indicando que los datos son incorrectos.
SC4	Verificar llave StartKey (llave de inicio) incorrectamente	80	46	03	06	-	# de 8 bytes que contiene la información de la StartKey.	El sistema muestra el APDU Respuesta **69 85** que indica “Condiciones no satisfechas”, lo que significa que el comando está incorrecto.	Se abre el simulador *JCardManager*, en el panel de la izquierda se escoge la opción *Authenticate* que está dentro de la carpeta *op*, que se encuentra dentro de la carpeta *Platform* y

							(Este número se obtiene del APDU Respuesta que viene cuando se ejecuta el caso de prueba *Leave Virgin State*).		ambas aparecen dentro de la carpeta raíz *USim Card R5.* Una vez seleccionada la opción, se desmarca el *check box Use default key file* y se debe cargar un fichero que va a permitir que el canal seguro se autentique contra el *applet uMatchApp* y no contra el *cardManager,* y se pincha en la pestaña *Go.* Luego en el panel de la izquierda se escoge la opción *Send APDU Secure* y se envía el comando APDU que indica que se va a verificar la llave de inicio (StartKey). En el simulador *JCardManager* se muestra una ventanita en forma de comando APDU donde el usuario del sistema introduce los datos correspondientes para ejecutar dicho comando. Al pinchar en la pestaña Go del simulador se

									muestra un mensaje señalizado en color rojo indicando que los datos son incorrectos.
SC5	Verificar llave AdminKey (llave de administración) incorrectamente	80	46	01	00	-	-	El sistema muestra el APDU Respuesta **69 85** que indica “Condiciones no satisfechas”, lo que significa que el comando está incorrecto.	Se abre el simulador *JCardManager*, en el panel de la izquierda se escoge la opción *Authenticate* que está dentro de la carpeta *op*, que se encuentra dentro de la carpeta *Platform* y ambas aparecen dentro de la carpeta raíz *USim Card R5.* Una vez seleccionada la opción, se desmarca el *check box Use default key file* y se debe cargar un fichero que va a permitir que el canal seguro se autentique contra el *applet uMatchApp* y no contra el *cardManager,* y se pincha en la pestaña *Go.* Luego en el panel de la izquierda se escoge la opción *Send APDU Secure* y se envía el comando APDU que indica que se va a

									verificar la llave de inicio (StartKey). En el simulador *JCardManager* se muestra una ventanita en forma de comando APDU donde el usuario del sistema introduce los datos correspondientes para ejecutar dicho comando. Al pinchar en la pestaña Go del simulador se muestra un mensaje señalizado en color rojo indicando que los datos son incorrectos.
SC6	Verificar llave StartKey (llave de inicio) incorrectamente	80	46	02	00	-	-	El sistema muestra el APDU Respuesta **69 85** que indica “Condiciones no satisfechas”, lo que significa que el comando está incorrecto.	Se abre el simulador *JCardManager*, en el panel de la izquierda se escoge la opción *Authenticate* que está dentro de la carpeta *op*, que se encuentra dentro de la carpeta *Platform* y ambas aparecen dentro de la carpeta raíz *USim Card R5.* Una vez seleccionada la opción, se desmarca el *check box Use default key file* y se debe cargar un fichero

									que va a permitir que el canal seguro se autentique contra el *applet uMatchApp* y no contra el *cardManager,* y se pincha en la pestaña *Go.* Luego en el panel de la izquierda se escoge la opción *Send APDU Secure* y se envía el comando APDU que indica que se va a verificar la llave de inicio (StartKey). En el simulador *JCardManager* se muestra una ventanita en forma de comando APDU donde el usuario del sistema introduce los datos correspondientes para ejecutar dicho comando. Al pinchar en la pestaña Go del simulador se muestra un mensaje señalizado en color rojo indicando que los datos son incorrectos.

Tabla 34. Caso de prueba verify key

- **Caso de prueba Cambiar llave (Change Key)**

Nota: Para realizar este caso de prueba se tiene que haber verificado antes la llave de administración.

Ver anexo # 11.

- **Caso de prueba Cambiar estado (Change State Control)**

Ver anexo # 12.

3.4 Conclusiones

A partir de las pruebas realizadas al software en el presente capítulo se logró verificar el correcto funcionamiento del mismo y la conformidad con los requisitos definidos inicialmente para su desarrollo. Las técnicas utilizadas permitieron no solo validar y verificar el funcionamiento interno del software, sino externo también, ya que se realizaron pruebas unitarias utilizando la técnica de caja blanca, y pruebas al sistema con la técnica de caja negra. Los resultados obtenidos a partir de la ejecución de ambas técnicas demuestran que el producto final cuenta con la calidad deseada y esperada por los clientes.

Conclusiones generales

Con la investigación realizada se logró cumplir el objetivo general, desarrollando una solución capaz de permitir el uso de una tarjeta inteligente como dispositivo para gestionar la información biométrica dactilar, y brindar este servicio a un *applet* de *MoC* para realizar la verificación biométrica, cumpliendo con los estándares internacionales ISO/IEC 7816-4, *Global Platform* y PC/SC. Concluyendo se puede plantear que:

- El análisis de las tecnologías, estándares y herramientas existentes, posibilitó conocer las características de las mismas, facilitando así un desarrollo exitoso del *applet uMatchApp*.
- La definición de la arquitectura permitió identificar las clases y funcionalidades que estuvieron presentes en el desarrollo de la solución.
- Las pruebas realizadas arrojaron como resultado que el *applet uMatchApp* cumple satisfactoriamente con los requisitos definidos para el sistema.

Recomendaciones

Al finalizar la presente investigación se plantean las siguientes recomendaciones para las siguientes versiones de la solución:

- Implementar la funcionalidad de autenticación asimétrica, que permita trabajar con una llave pública RSA, de forma tal que esta pueda ser enviada hacia el *middleware* y este cifrar la información con una llave privada, evitando así ataques de tipo *YesCard*. De esta forma se aumentaría en gran medida la seguridad de la aplicación.
- Integrar el *applet uMatchApp* con el *applet* que se está desarrollando de *MoC*, el cual usará la información biométrica dactilar gestionada por el *applet uMatchApp* para realizar la verificación biométrica.
- Comercializar la aplicación en un futuro, como uno de los productos del Centro CISED de la Universidad de las Ciencias Informáticas.

Referencias bibliográficas

Alberto Elers Pérez, Jorge L. Sariol Pérez. 2010. Implementación del Módulo Getión de Actividades del Tutor Virtual de evaluación para el Aprendizaje Autónomo de Idiomas. *Implementación del Módulo Getión de Actividades del Tutor Virtual de evaluación para el Aprendizaje Autónomo de Idiomas.* [En línea] Junio de 2010. http://repositorio_institucional.uci.cu/jspui/bitstream/ident/TD_03452_10/1/TD_03452_10.pdf.

Alegsa. 2011. Programación Orientada a Objetos. *Programación Orientada a Objetos.* [En línea] 24 de Mayo de 2011. http://www.alegsa.com.ar/Dic/middleware.php .

Asterisk. 2010. Software Propietario. *Software Propietario.* [En línea] 2010. http://bibing.us.es/proyectos/abreproy/11969/fichero/Memoria%252F05_Capitulo03.pdf..

BuenasTareas.com. 2012. Antropometría y Biometría. *Antropometría y Biometría.* [En línea] Abril de 2012. http://www.buenastareas.com/ensayos/Antropometr%C3%ADa-y-Biometr%C3%ADa/3859354.html.

Captalis. 2009. Tarjetas Inteligentes. *Tarjetas Inteligentes.* [En línea] Marzo de 2009. http://blog.captalis.com/2009/03/17/tarjetas-inteligentes/.

Cuerpo Nacional de Policía. DNI Electrónico. 2012. DNI Electrónico. *DNI Electrónico.* [En línea] 2012. http://www.dnielectronico.es/seccion_empresas/index.html.

Farlex Inc. 2012. The Free Dictionary. *The Free Dictionary.* [En línea] 2012. http://www.thefreedictionary.com/applet.

Gemalto. 2010. Developer Suite. *Developer Suite.* [En línea] 22 de Noviembre de 2010. http://translate.google.co.ve/translate?hl=es&sl=en&u=http://developer.gemalto.com/home/dev-tools/developer-suite.html&ei=PkbFTrj0IqXg0QHD6vzgDg&sa=X&oi=translate&ct=result&resnum=1&ved=0CB4Q7gEwAA&prev=/search%3Fq%3Dgemalto,%2Bdeveloper%2Bsuite%26hl%3Des.

2009. Estándar PC/SC. *Estándar PC/SC.* [En línea] 2009. http://translate.google.co.ve/translate?hl=es&sl=en&u=http://www.gemalto.com/techno/pcsc/&ei=8ZRzTr6HD8PcgQfM7vziDA&sa=X&oi=translate&ct=result&resnum=1&ved=0CBoQ7gEwAA&prev=/search%3Fq%3Destandar%2BPC/SC%26hl%3Des%26biw%3D1024%26bih%3D605%26prmd%3Dimvns..

2010. JCard Manager. *JCard Manager.* [En línea] 2010. http://www.gemalto.com.

Global Platform. 2003. *GlobalPlatform Card Specification version 2.1.1.* 2003.

2012. The Standar for Managing Applications on Secure Chip Technology. *The Standar for Managing Applications on Secure Chip Technology.* [En línea] 2012. http://www.globalplatform.org/.

Hanscan Identity Management. 2012. Biocriptología® la Tecnología definitiva para la Gestión de Autentificaciones. *Biocriptología® la Tecnología definitiva para la Gestión de Autentificaciones.* [En línea] 2012. http://www.hanscan.com/es/identificacion-personas.

Identificación biométrica y su unión con las tarjetas inteligentes. **Reillo, Raúl Sánchez. 2000.** 2000, Ágora SIC.

Jesús Sánchez Orozco, José R. Sánchez Orozco, Benito Recuero Díaz, Francisco José Seva Mora, Álvaro Rendón Pérez. 2004. Tarjetas Inteligentes. *Tarjetas Inteligentes.* [En línea] 20 de Mayo de 2004. http://www.google.com.cu/url?sa=t&rct=j&q=definicion+de+tarjeta+inteligente&source=web&cd=3&ved=0CF4QFjAC&url=http%3A%2F%2Fwww.dte.us.es%2Ftec_inf%2Fitis%2Fperi_int%2Ftrabajos%2FTarjetasInteligentes.ppt&ei=q0fGT8vCG4XpgAfbo-HpBg&usg=AFQjCNEdU1T1IkPK3KFNyv.

Jimeno, Francisco José Díez. 2009. Sistema integrado de control de accesos. [En línea] Septiembre de 2009. http://e-archivo.uc3m.es/bitstream/10016/5813/1/PFC_FJ_Diez_Jimeno.pdf..

msdn. 2010. Revisiones de código y estándares de codificación. *Revisiones de código y estándares de codificación.* [En línea] 2010. http://msdn.microsoft.com/es-es/library/aa291591(v=vs.71).aspx.

Plataforma Java Card. 2012. Plataforma Java Card. *Plataforma Java Card.* [En línea] 2012. http://bibing.us.es/proyectos/abreproy/11458/fichero/PFC%252FCapitulo04+-+La+plataforma+Java+Card.pdf.

Precise Biometrics. 2005. Precise BioMatch™ J 3.0 Manual. [En línea] 2005. http://info@precisebiometrics.com.

2006. Precise IDStore™ J 1.1 Manual. [En línea] 2006. http://info@precisebiometrics.com.

Ramírez, Julio César. 2012. Estándares ISO/IEC 7816. [En línea] 2012. http://www.tataware.com/tesis_maestria/13%20-%20Anexo%20B%20-%20Est%E1ndares%20ISO_IEC%207816.pdf.

RFID.com. 2008. Precise Biometrics presenta la tecnología Match-on-Card . *Precise Biometrics presenta la tecnología Match-on-Card .* [En línea] 14 de Febrero de 2008. http://www.rfid-spain.com/articulo/52043/biometria/precise-biometrics-presenta-la-tecnologia-match-on-card-que-ayuda-a-conservar-los-datos-biometricos-de-forma-segura..

SIC-SGA. 2012. TCM: Ingeniería de integración de sistemas. Smart Cards. *TCM: Ingeniería de integración de sistemas. Smart Cards.* [En línea] 2012. http://www.asesga.org/noticias/Toni%20Biedma.pdf.

SlideShare Inc. . 2009. Características y ventajas Java. *Características y ventajas Java.* [En línea] 2009. http://www.slideshare.net/bat1820/java-ventajas-y-caracteristicas.

Smart Card Alliance. 2004. Acceso lógico seguro: El papel de las tarjetas inteligentes en una autenticación más sólida. *Acceso lógico seguro: El papel de las tarjetas inteligentes en una autenticación más sólida.* [En línea] Octubre de 2004. http://www.smartcardalliance.org/latinamerica/translations/Logical_Access_Security_Spanish.pdf.

Tecnología hecha palabra. 2008. Tecnología al instante. *Tecnología al instante.* [En línea] Agosto de 2008. http://www.tecnologiahechapalabra.com/tecnologia/glosario_tecnico/articulo.asp?i=2895..

Bibliografía

Aguilera, Yarisel. 2011. Diseño de un Applet y Middleware para gestionar la información biométrica contenida en la Cédula de Identificación Electrónica de la República Bolivariana de Venezuela. Ciudad de la Habana : Universidad de las Ciencias Informáticas, 2011.

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.

Boehm, Barry W. 1995/2. 1995/2.

Building_a_Chain_of_Trust_Spanish. **Smart Card Alliance.**

EcuRed. Altova Umodel - EcuRed. [En línea]

http://www.ecured.cu/index.php/Altova_Umodel.

2011. Tarjetas inteligentes-EcuRed. [En línea] 2011.

http://www.ecured.cu/index.php/Tarjetas_Inteligentes.

EcuRed. Applet - EcuRed. [En línea] http://www.ecured.cu/index.php/Applet.

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

Gemalto. 2010. Developer Suite. *Developer Suite.* [En línea] 22 de Noviembre de 2010. http://translate.google.co.ve/translate?hl=es&sl=en&u=http://developer.gemalto.com/home/dev-tools/developer-suite.html&ei=PkbFTrj0IqXg0QHD6vzgDg&sa=X&oi=translate&ct=result&resnum=1&ved=0CB4Q7gEwAA&prev=/search%3Fq%3Dgemalto,%2Bdeveloper%2Bsuite%26hl%3Des.

2009. Estándar PC/SC. *Estándar PC/SC.* [En línea] 2009. http://translate.google.co.ve/translate?hl=es&sl=en&u=http://www.gemalto.com/techno/pcsc/&ei=8ZRzTr6HD8PcgQfM7vziDA&sa=X&oi=translate&ct=result&resnum=1&ved=0CBoQ7gEwAA&prev=/search%3Fq%3Destandar%2BPC/SC%26hl%3Des%26biw%3D1024%26bih%3D605%26prmd%3Dimvns..

2010. JCard Manager. *JCard Manager.* [En línea] 2010. http://www.gemalto.com.

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

Gemalto. What is Global Platform?. [En línea]

http://www.gemalto.com/nfc/global_platform.html.

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

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

Proyecto de Taller V Programación de JavaCards. **Daniel Perovich, Leonardo Rodríguez,Martín Varela. 2001.** 2001.

Sun Microsystem. 2012. Java Card 2.1 API Specifications. [En línea] 2012.

http://java.sun.com/products/javacard/javacard21.html..

Sun Microsystems. 2012. Java Card 2.1 Runtime Environment (JCRE) Specification. [En línea] 2012. 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.

PC/SC Workgroup. 2010. PC/SC Workgroup. [En línea] 2010. http://www.pcscworkgroup.com

Plataforma Java Card. 2012. Plataforma Java Card. *Plataforma Java Card.* [En línea] 2012. http://bibing.us.es/proyectos/abreproy/11458/fichero/PFC%252FCapitulo04+-+La+plataforma+Java+Card.pdf.

Precise Biometrics. 2005. Precise BioMatch™ J 3.0 Manual. [En línea] 2005. http://info@precisebiometrics.com.

2006. Precise IDStore™ J 1.1 Manual. [En línea] 2006. http://info@precisebiometrics.com.

Tecnología hecha palabra. 2008. Tecnología al instante. *Tecnología al instante.* [En línea] Agosto de 2008. http://www.tecnologiahechapalabra.com/tecnologia/glosario_tecnico/articulo.asp?i=2895..

Glosario de términos

APDU: Protocolo de Unidad de Datos de Aplicaciones.

API: Una interfaz de programación de aplicaciones o API (en inglés *Application Programming Interface*) es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro *software* como una capa de abstracción.

Applet: Es un componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo un navegador web. Los *Applets*, según el contexto del trabajo, son aplicaciones implementadas utilizando la tecnología *Java Card* y son ejecutadas dentro de las tarjetas inteligentes.

Autenticación: Acto de establecimiento o confirmación de algo (o alguien) como auténtico, es decir probar que es verdadero. La autenticación de un objeto puede significar la confirmación de su procedencia, mientras que la autenticación de una persona consiste en verificar su identidad.

CAD: Tarjeta de dispositivo de aceptación.

CLA: Clase que define las funcionalidades del applet uMatchApp.

Data: campo de datos del comando APDU

Global Platform: Estándar que permite a sistemas heterogéneos intercambiar procesos o datos.

INS: Instrucción, número particular definido por cada funcionalidad del applet uMatchApp.

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.

Lc: Longitud del comando de datos del applet uMatchApp.

Le: Longitud esperada en los datos de respuesta del comando APDU.

Match on Card (*MoC* por sus siglas en inglés): es una tecnología que realiza comparaciones de huellas dactilares en tarjetas inteligentes.

Middleware: Software que funciona como una conversión o capa de traducción. Los *middlewares* son soluciones personalizadas que se han desarrollado durante décadas para permitir a una aplicación comunicarse con otra, ya sea que se ejecuta en una plataforma diferente o viene de un proveedor distinto.

P1: Parámetro 1 del applet uMatchApp.

P2: Parámetro 2 del applet uMatchApp.

PC/SC: (en inglés *Personal Computer*/*Smart Card*) es un estándar para las tarjetas inteligentes de acceso en plataformas Windows (incluido en Windows 2000). Este define un API de programación, que permite a los desarrolladores trabajar de forma uniforme con lectores de tarjetas de distintos fabricantes (que cumplan con la especificación).

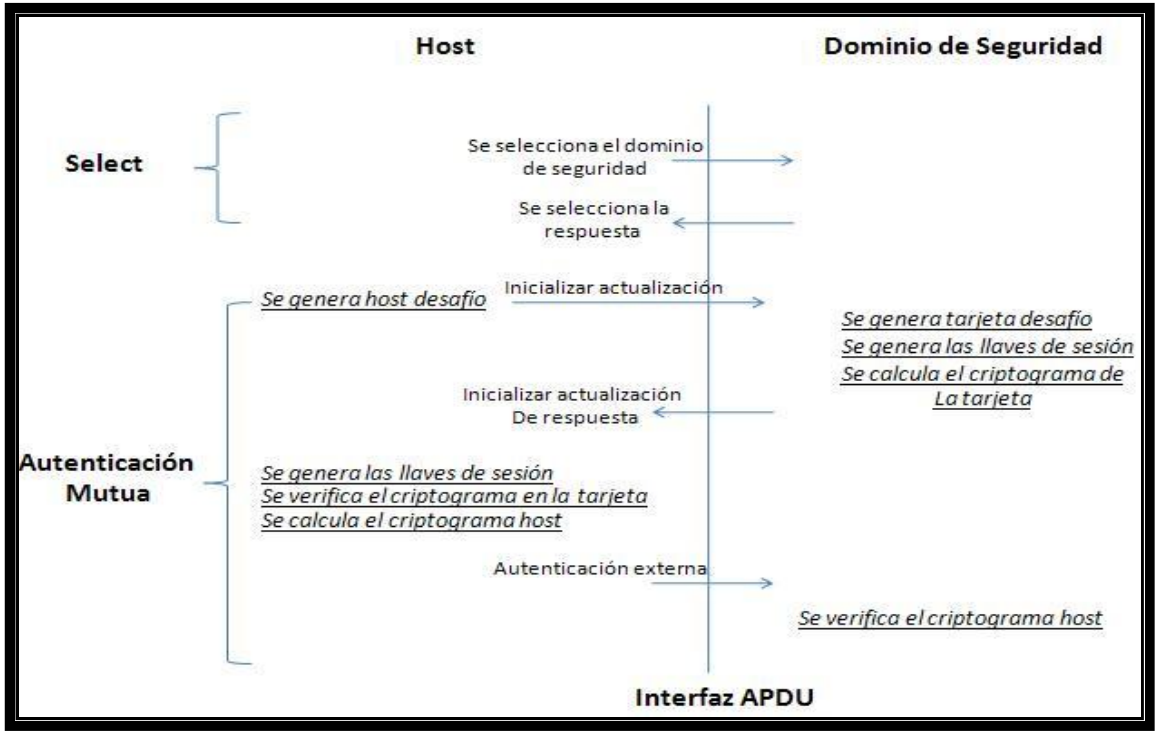
SCP01: Es un canal que proporciona un canal de comunicación seguro entre una tarjeta y la entidad que está fuera de la tarjeta.

Sistema de identificación biométrica: Sistema de reconocimiento estadístico de patrones que establece la autenticidad de una característica fisiológica o de comportamiento que posee un usuario.

SW: Palabra de Estado.

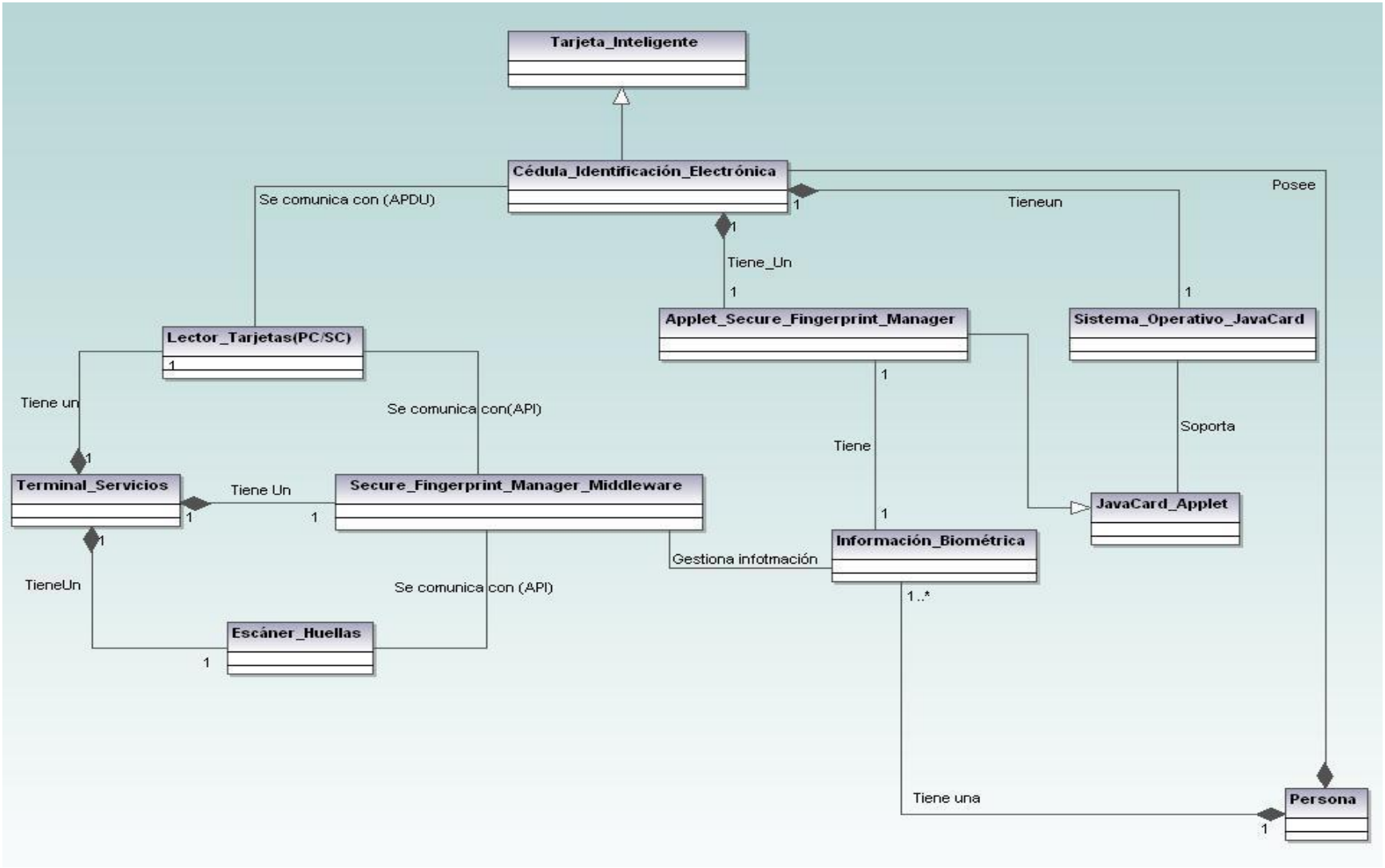
Anexos

Anexo # 1: Flujo de Autenticación Mutua del canal seguro SCP01 de Global Platform



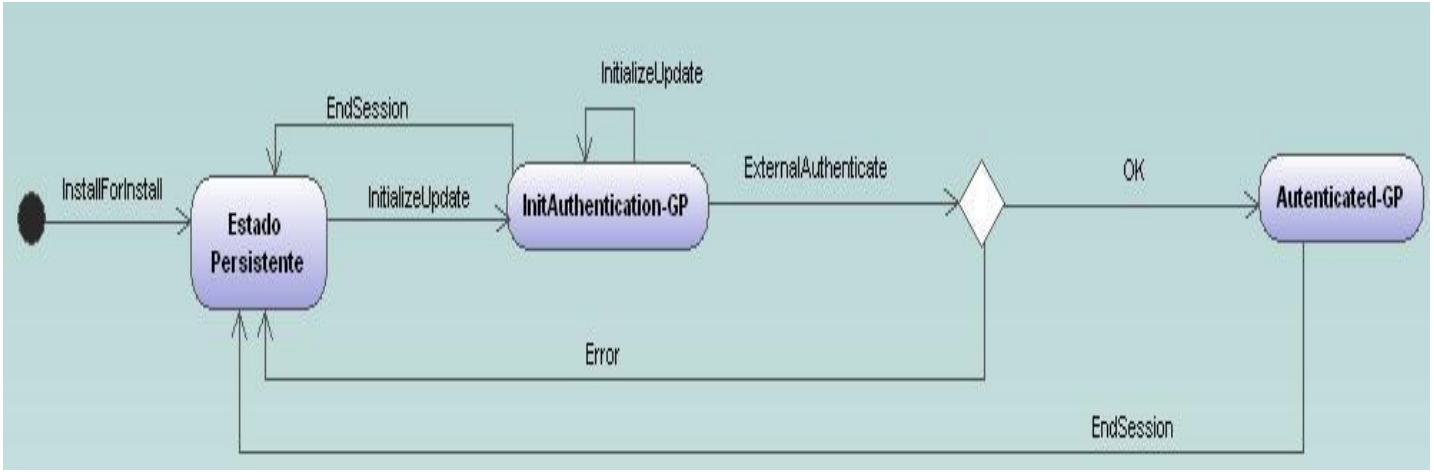
Anexo 1: Flujo de Autenticación Mutua

Anexo # 2: Diagrama de clases del modelo de dominio



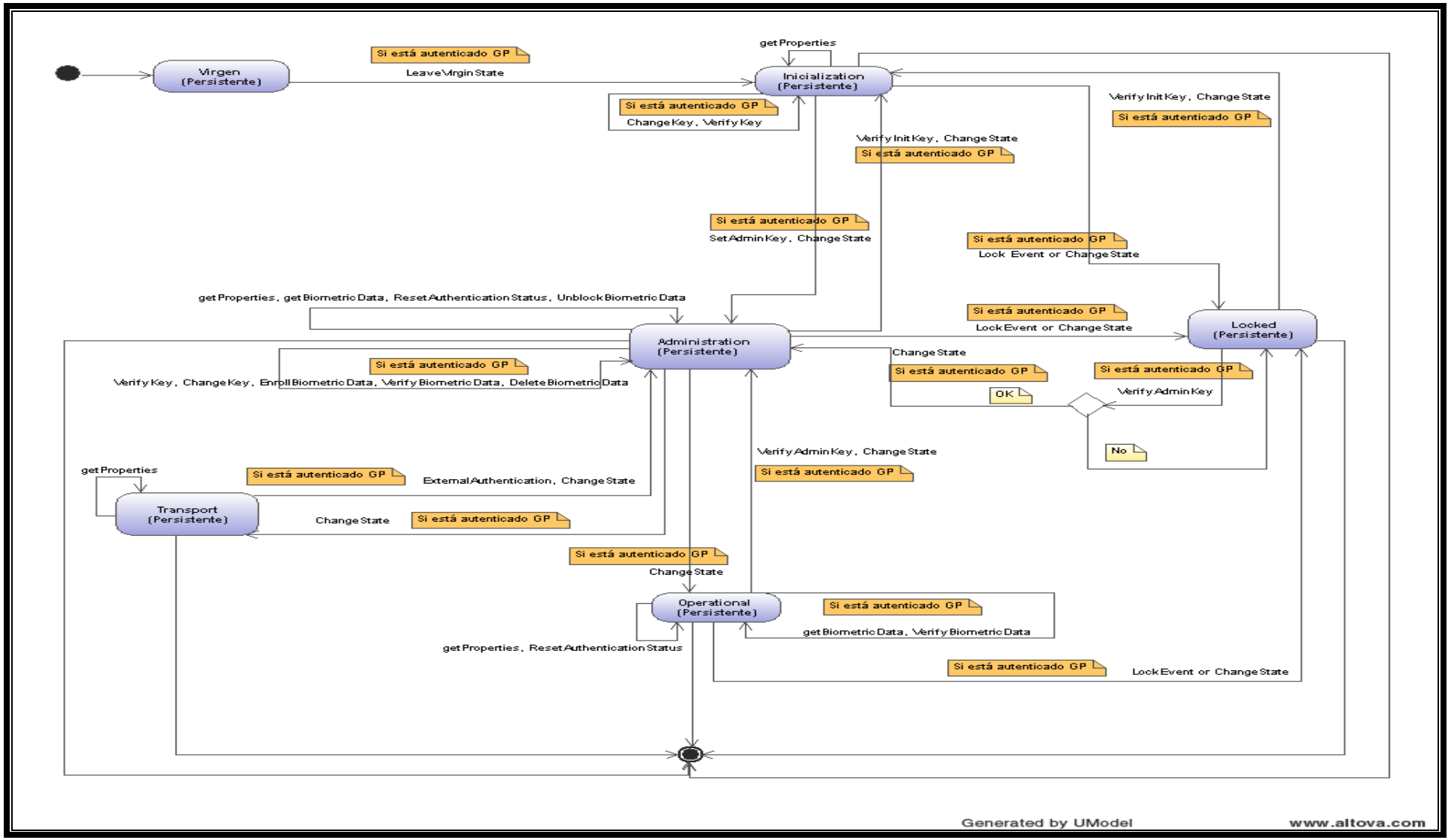
Anexo 2: Diagrama de clases del modelo de dominio

Anexo # 3: Diagrama de estados de la autenticación de *Global Platform*



Anexo 3: Diagrama de estados de la autenticación de *Global Platform*

Anexo # 4: Diagrama de estados del *applet uMatchApp*



Anexo 4: Diagrama de estados del *applet uMatchApp*

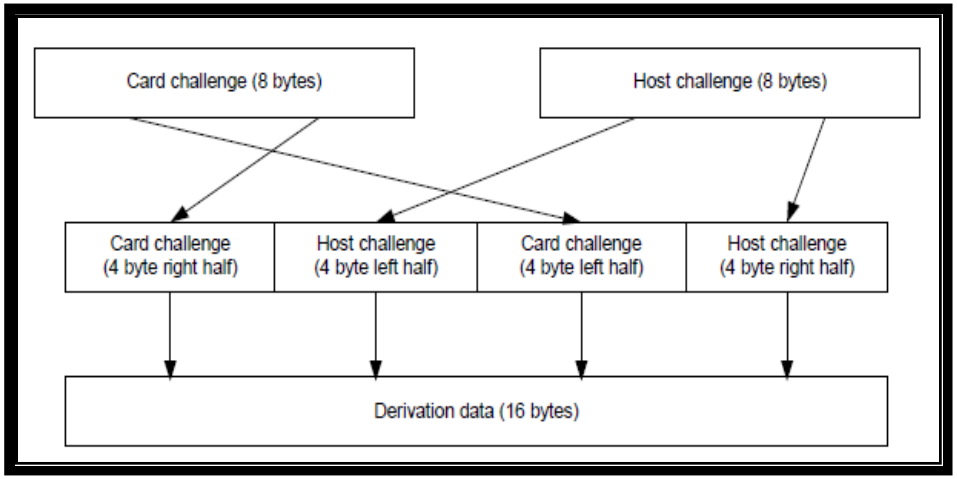
Anexo # 5: Comandos de retornos de respuesta del *applet uMatchApp*

Comandos generales de retorno		**Mensaje**	**Descripción**
SW1	**SW2**		
63h	00h	SW_AUTH_FAILED	Autenticación fallida
67h	00h	SW_WRONG_LENGTH	Longitud incorrecta
69h	82h	SW_SECURITY_NOT_SATISFIED	Condiciones de seguridad no satisfechas
69h	84h	SW_DATA_INVALID	Datos inválidos
69h	85h	SW_CONDITIONS_NOT_SATISFIED	Condiciones de uso no satisfechas
69h	86h	SW_COMMAND_NOT_ALLOWED	Comandos no permitidos
6Ah	80h	SW_WRONG_DATA	Datos incorrectos
6Ah	81h	SW_FUNC_NOT_SUPPORTED	Funciones no soportadas
6Ah	82h	SW_FILE_NOT_FOUND	Archivo no encontrado
6Ah	86h	SW_INCORRECT_PARAMETERS_P1P2	Parámetros de comandos incorrectos

6Dh	00h	SW_INS_NOT_SUPPORTED	Instrucción de código no soportado/inválido
6Eh	00h	SW_CLA_NOT_SUPPORTED	Clases no soportadas
90h	00h	SW_SUCCESS	Operación exitosa

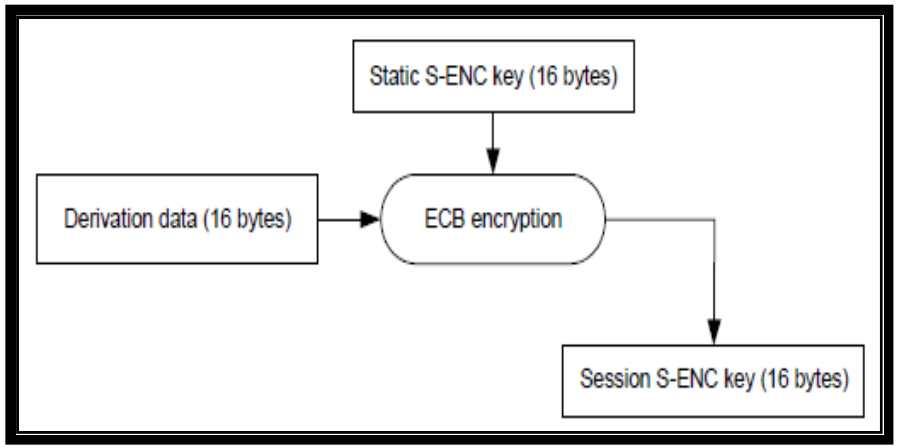
Anexo 5: Comandos de retornos de respuesta del *applet uMatchApp*

Anexo # 6: Clave de Sesión - Paso 1 - Generar datos de derivación



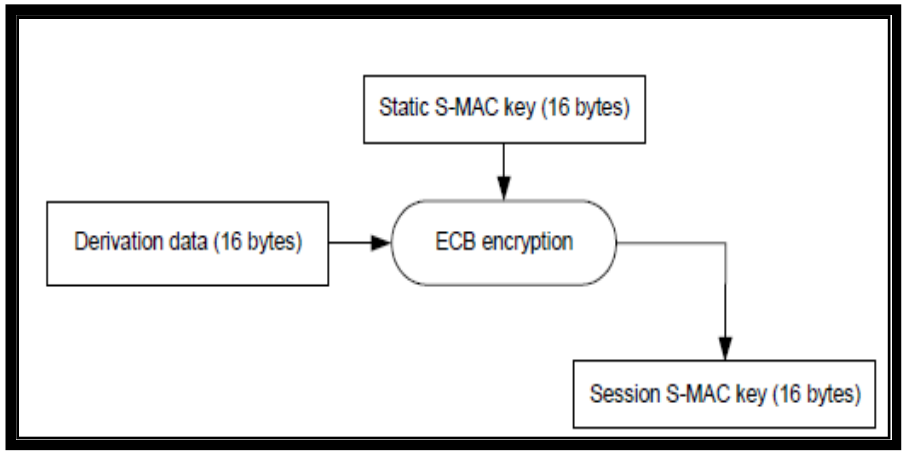
Anexo 6: Clave de Sesión - Paso 1 - Generar datos de derivación

Anexo # 7: Clave de Sesión - Paso 2 – Creación de la llave de sesión S-ENC



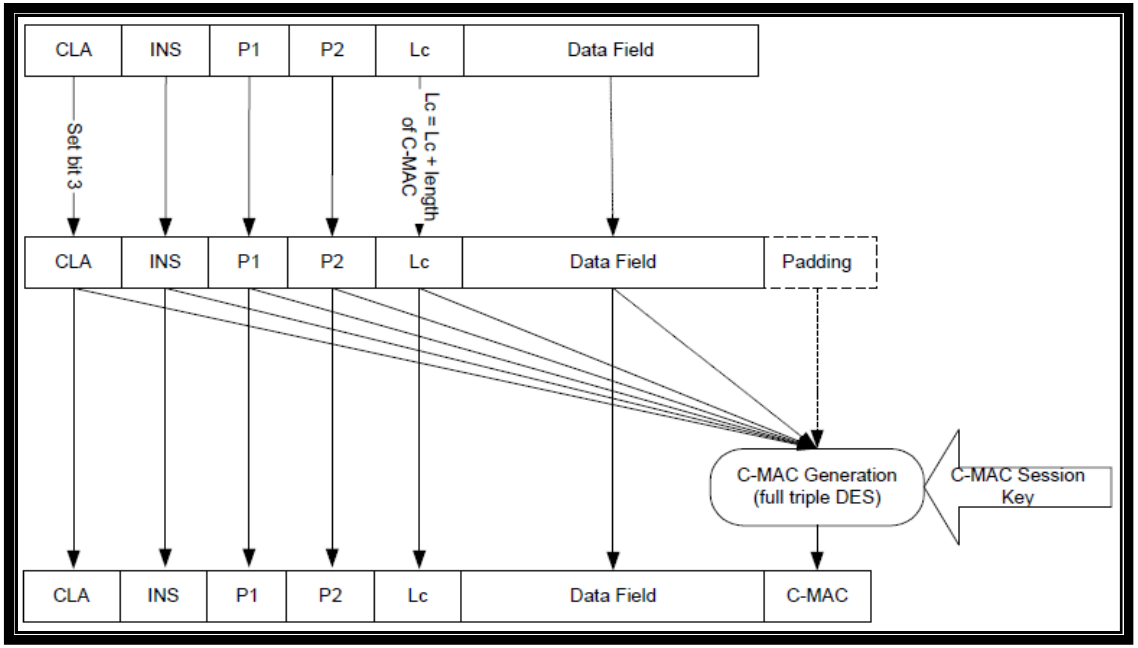
Anexo 7: Clave de Sesión - Paso 2 – Creación de la llave de sesión S-ENC

Anexo # 8: Clave de Sesión - Paso 3 – Creación de la llave de sesión S-MAC



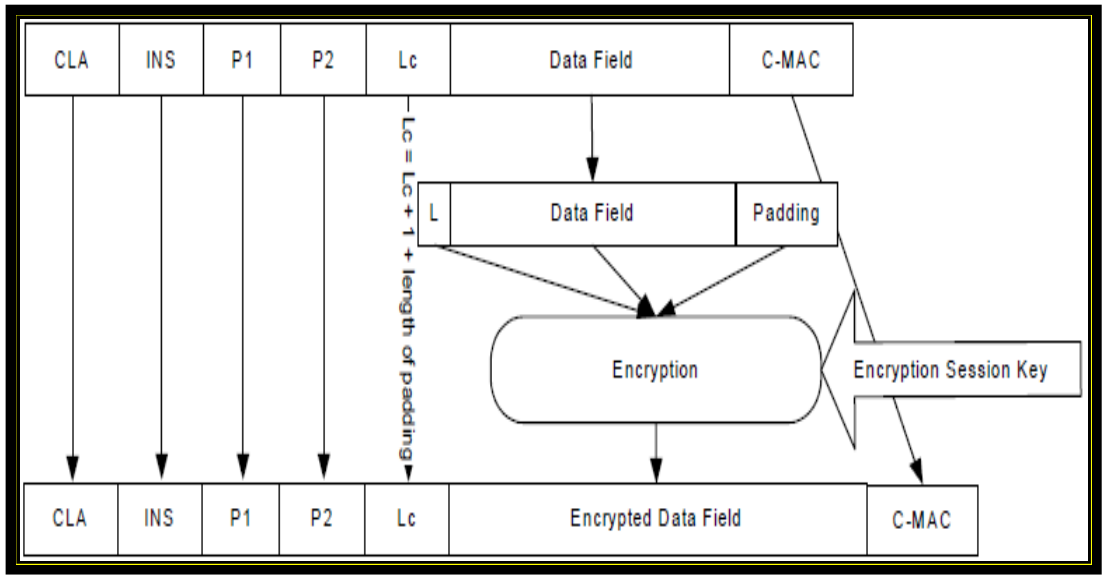
Anexo 8: Clave de Sesión - Paso 3 – Creación de la llave de sesión S-MAC

Anexo # 9: Generación y verificación de la MAC del comando APDU



Anexo 9: Generación y verificación de la MAC del comando APDU

Anexo # 10: Cifrado del campo de datos del comando APDU



Anexo 10: Cifrado del campo de datos del comando APDU

Anexo # 11: Caso de prueba Change Key

Escenario	Descripción	CLA	INS	P1	P2	Le	Data	Respuesta del sistema	Flujo central
SC1	Cambiar llave correctamente	80	44	01	00	-	14 12 00 01 05 44 11 12	El sistema muestra el APDU Respuesta **90 00**, que indica que se ejecutó el comando correctamente.	Se abre el simulador *JCardManager*, en el panel de la izquierda se escoge la opción *Authenticate* que está dentro de la carpeta *op*, que se encuentra dentro de la carpeta *Platform* y ambas aparecen dentro

									de la carpeta raíz *USim Card R5.* Una vez seleccionada la opción, se desmarca el *check box Use default key file* y se debe cargar un fichero que va a permitir que el canal seguro se autentique contra el *applet uMatchApp* y no contra el *cardManager,* y se pincha en la pestaña *Go.* Luego en el panel de la izquierda se escoge la opción *Send APDU Secure* y se envía el comando APDU que indica que se va a cambiar la llave. En el simulador se muestra una ventanita en forma de comando APDU donde el usuario del sistema introduce los datos correspondientes (CLA, INS, P1, P2, Lc y Data) para ejecutar dicho comando. Al pinchar en la pestaña Go del simulador se muestra un mensaje señalizado en color verde indicando que los datos son correctos.
SC2	Cambiar llave incorrectamente	80	44	01	00	-	-	El sistema muestra el APDU Respuesta **6A 81** que indica	Se abre el simulador *JCardManager*, en el panel de la izquierda se escoge la

								“Función no soportada”, lo que significa que el comando está incorrecto.	opción *Authenticate* que está dentro de la carpeta *op*, que se encuentra dentro de la carpeta *Platform* y ambas aparecen dentro de la carpeta raíz *USim Card R5.* Una vez seleccionada la opción, se desmarca el *check box Use default key file* y se debe cargar un fichero que va a permitir que el canal seguro se autentique contra el *applet uMatchApp* y no contra el *cardManager,* y se pincha en la pestaña *Go.* Luego en el panel de la izquierda se escoge la opción *Send APDU* y se envía el comando APDU que indica que se va a cambiar la llave. En el simulador se muestra una ventanita en forma de comando APDU donde el usuario del sistema introduce los datos correspondientes de forma incorrecta. Al pinchar en la pestaña Go del simulador se muestra un mensaje señalizado en color rojo indicando que los datos son incorrectos.

SC4	Cambiar llave incorrectamente	80	44	00	00	-	-	El sistema muestra el APDU Respuesta **6A 81** que indica "Función no soportada", lo que significa que el comando está incorrecto.	Se abre el simulador *JCardManager*, en el panel de la izquierda se escoge la opción *Authenticate* que está dentro de la carpeta *op*, que se encuentra dentro de la carpeta *Platform* y ambas aparecen dentro de la carpeta raíz *USim Card R5.* Una vez seleccionada la opción, se desmarca el *check box Use default key file* y se debe cargar un fichero que va a permitir que el canal seguro se autentique contra el *applet uMatchApp* y no contra el *cardManager,* y se pincha en la pestaña *Go.* Luego en el panel de la izquierda se escoge la opción *Send APDU Secure* y se envía el comando APDU que indica que se va a cambiar la llave. En el simulador se muestra una ventanita en forma de comando APDU donde el usuario del sistema introduce los datos correspondientes de forma incorrecta. Al pinchar en la pestaña Go del simulador se muestra un mensaje señalizado en color

									rojo indicando que los datos son incorrectos.
SC5	Cambiar llave incorrectamente	80	44	01	00	-	14 12 00 01 05 44 11 12	El sistema muestra el APDU Respuesta **69 82** que indica "Condiciones de seguridad no satisfechas", lo que significa que aún no se ha verificado la llave de administración y hasta que ese comando no se ejecute no se puede ejecutar este caso de prueba.	Se abre el simulador *JCardManager*, en el panel de la izquierda se escoge la opción *Authenticate* que está dentro de la carpeta *op*, que se encuentra dentro de la carpeta *Platform* y ambas aparecen dentro de la carpeta raíz *USim Card R5.* Una vez seleccionada la opción, se desmarca el *check box Use default key file* y se debe cargar un fichero que va a permitir que el canal seguro se autentique contra el *applet uMatchApp* y no contra el *cardManager,* y se pincha en la pestaña *Go.* Luego en el panel de la izquierda se escoge la opción *Send APDU Secure* y se envía el comando APDU que indica que se va a cambiar la llave. En el simulador se muestra una ventanita en forma de comando APDU donde el usuario del sistema introduce los datos correspondientes de forma incorrecta. Al

									pinchar en la pestaña Go del simulador se muestra un mensaje señalizado en color rojo indicando que las condiciones no han sido satisfechas.

Anexo 11: Caso de prueba Change Key.

Anexo # 12: Caso de prueba Change State Control

Escenario	Descripción	CLA	INS	P1	P2	Le	Data	Respuesta del sistema	Flujo central
SC1	Cambiar estado correctamente	80	42	01 ó 02 ó 04 ó 08 ó 10 ó 80	00	-	-	El sistema muestra el APDU Respuesta **90 00**, que indica que se ejecutó el comando correctamente.	Se abre el simulador *JCardManager*, en el panel de la izquierda se escoge la opción *Authenticate* que está dentro de la carpeta *op*, que se encuentra dentro de la carpeta *Platform* y ambas aparecen dentro de la carpeta raíz *USim Card R5.* Una vez seleccionada la opción, se desmarca el *check box Use default key file* y se debe cargar un fichero que va a permitir que el canal seguro se autentique contra el *applet uMatchApp* y no contra el *cardManager,* y se pincha en la pestaña *Go.* Luego en el panel de la izquierda se escoge la opción *Send APDU Secure* y se

									envía el comando APDU que indica que se va a cambiar de estado. En el simulador se muestra una ventanita en forma de comando APDU donde el usuario del sistema introduce los datos correspondientes (CLA, INS, P1, P2, Lc y Data) para ejecutar dicho comando. **Nota:** Si lo que vendrá en P1 es 04 indicando que se cambiará para el estado de inicialización, primero debe haber verificado correctamente la llave de administración. Al pinchar en la pestaña Go del simulador se muestra un mensaje señalizado en color verde indicando que los datos son correctos.
SC2	Cambiar estado incorrectamente	80	42	01 ó 02 ó 04 ó 08 ó	12	-	-	El sistema muestra el APDU Respuesta **6A 81** que indica "Función no soportada", lo que significa que el comando está incorrecto.	Se abre el simulador *JCardManager*, en el panel de la izquierda se escoge la opción *Authenticate* que está dentro de la carpeta *op*, que se encuentra dentro de la carpeta *Platform* y ambas aparecen dentro de la carpeta raíz *USim Card R5.* Una vez seleccionada la opción, se desmarca el *check box Use default key*

				10 ó 80					*file* y se debe cargar un fichero que va a permitir que el canal seguro se autentique contra el *applet uMatchApp* y no contra el *cardManager,* y se pincha en la pestaña *Go.* Luego en el panel de la izquierda se escoge la opción *Send APDU Secure* y se envía el comando APDU que indica que se va a cambiar de estado. En el simulador se muestra una ventanita en forma de comando APDU donde el usuario del sistema introduce los datos correspondientes de forma incorrecta. Al pinchar en la pestaña Go del simulador se muestra un mensaje señalizado en color rojo indicando que los datos son incorrectos.
SC3	Cambiar estado incorrectamente	80	42	00	00	-	-	El sistema muestra el APDU Respuesta **6A 81** que indica "Función no soportada", lo que significa que el comando está incorrecto.	Se abre el simulador *JCardManager*, en el panel de la izquierda se escoge la opción *Authenticate* que está dentro de la carpeta *op*, que se encuentra dentro de la carpeta *Platform* y ambas aparecen dentro de la carpeta raíz *USim Card R5.* Una vez seleccionada la opción, se

									desmarca el *check box Use default key file* y se debe cargar un fichero que va a permitir que el canal seguro se autentique contra el *applet uMatchApp* y no contra el *cardManager,* y se pincha en la pestaña *Go.* Luego en el panel de la izquierda se escoge la opción *Send APDU Secure* y se envía el comando APDU que indica que se va a cambiar de estado. En el simulador se muestra una ventanita en forma de comando APDU donde el usuario del sistema introduce los datos correspondientes de forma incorrecta. Al pinchar en la pestaña Go del simulador se muestra un mensaje señalizado en color rojo indicando que los datos son incorrectos.
SC4	Cambiar estado incorrectamente	80	42	05	00	-	-	El sistema muestra el APDU Respuesta **6A 81** que indica "Función no soportada", lo que significa que el comando está incorrecto.	Se abre el simulador *JCardManager*, en el panel de la izquierda se escoge la opción *Authenticate* que está dentro de la carpeta *op*, que se encuentra dentro de la carpeta *Platform* y ambas aparecen dentro de la carpeta raíz *USim Card R5.*

									Una vez seleccionada la opción, se desmarca el *check box Use default key file* y se debe cargar un fichero que va a permitir que el canal seguro se autentique contra el *applet uMatchApp* y no contra el *cardManager,* y se pincha en la pestaña *Go.* Luego en el panel de la izquierda se escoge la opción *Send APDU Secure* y se envía el comando APDU que indica que se va a cambiar de estado. En el simulador se muestra una ventanita en forma de comando APDU donde el usuario del sistema introduce los datos correspondientes de forma incorrecta. Al pinchar en la pestaña Go del simulador se muestra un mensaje señalizado en color rojo indicando que los datos son incorrectos.
SC5	Cambiar estado incorrectamente	80	42	04	00	-	12 11 14 15 16	El sistema muestra el APDU Respuesta **6A 81** que indica “Función no soportada”, lo que significa que el comando está incorrecto.	Se abre el simulador *JCardManager*, en el panel de la izquierda se escoge la opción *Authenticate* que está dentro de la carpeta *op*, que se encuentra dentro de la carpeta *Platform* y ambas aparecen

									dentro de la carpeta raíz *USim Card R5.* Una vez seleccionada la opción, se desmarca el *check box Use default key file* y se debe cargar un fichero que va a permitir que el canal seguro se autentique contra el *applet uMatchApp* y no contra el *cardManager,* y se pincha en la pestaña *Go.* Luego en el panel de la izquierda se escoge la opción *Send APDU Secure* y se envía el comando APDU que indica que se va a cambiar de estado. En el simulador se muestra una ventanita en forma de comando APDU donde el usuario del sistema introduce los datos correspondientes de forma incorrecta. Al pinchar en la pestaña Go del simulador se muestra un mensaje señalizado en color rojo indicando que los datos son incorrectos.

Anexo 12: Caso de prueba Change State Control