ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Proyecto Fin de Carrera

VIRTUALIZACION DE TARJETAS MIFARE CLASSIC EN UN ENTORNO JAVA CARD

(MIFARE Classic cards virtualization in a Java Card environment)

Para acceder al Titulo de

INGENIERO DE TELECOMUNICACIÓN

Autor: Jose María Carcedo Aldecoa

Julio – 2017

INGENIERÍA DE TELECOMUNICACIÓN

CALIFICACIÓN DEL PROYECTO FIN DE CARRERA

Realizado por: Jose María Carcedo Aldecoa
Director del PFC: Jorge Lanza Calderón
Título: "Virtualización de tarjetas MIFARE Classic en un entorno Java Card"
Title: "MIFARE Classic cards virtualization in a Java Card enviroment"

Presentado a examen el día: 31 de Julio de 2017

para acceder al Título de

INGENIERO DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Sánchez González, Luis
Secretario (Apellidos, Nombre): Lanza Calderón, Jorge
Vocal (Apellidos, Nombre): García Gutiérrez, Alberto Eloy

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente Fdo.: El Secretario

Fdo.: El Vocal Fdo.: El Director del PFC
(sólo si es distinto del Secretario)

Vº Bº del Subdirector Proyecto Fin de Carrera Nº (a asignar por Secretaría)

AGRADECIMIENTOS

Me gustaría agradecer el siguiente documento en primer lugar a mis padres, Antonio e Isabel, que siempre me han apoyado y aguantado desde que empecé la carrera hace unos cuantos años ya y, aunque su fe haya sido puesta a prueba más veces de las que quiero reconocer, siguen ahí intentando que mejore cada día.

En segundo lugar estarían mis hermanos Javier y Juan los cuales siempre me han ofrecido su ayuda incluso antes de necesitarla y siempre han intentado que de lo mejor de mí mismo.

También me gustaría agradecerle a mi director de proyecto que, aunque no siempre nos hemos puesto de acuerdo y de los problemas de comunicación que hemos tenido durante el desarrollo de este proyecto, hemos conseguido sacarlo delante de forma satisfactoria.

Por último me gustaría hacer una mención especial a la contribución de mis amigos que, sobretodo moralmente, me han apoyado durante todo este tiempo, sin dudar de mí y siempre con una palabra de ánimo.

RESUMEN

En los últimos años la evolución tanto de las propias tarjetas inteligentes como de los aplicativos que hacen uso de ellas ha sido muy rápida. Así, cada día son más las tarjetas que usamos en nuestra vida cotidiana. Esta amplia adopción, especialmente desde la implantación de las tecnologías sin contactos, ha supuesto que ya no exista un temor en su uso y se acepte la seguridad en ellas inherente.

Aunque existe una alta heterogeneidad en cuanto a aplicación final, muchas de ellas comparten la misma tecnología. En este sentido, parece razonable considerar que una misma tarjeta pueda dar soporte a varias aplicaciones de forma simultánea, independientemente del fabricante y del ámbito de la aplicación.

La tecnología Java Card surge con este propósito, definiéndose como una plataforma segura, portable y multiaplicación. Estas capacidades multiaplicación dan respuesta a la necesidad anteriormente descrita.

El presente trabajo, basándose en una tarjeta Java Card, plantea el diseño e implementación de una solución para dar soporte a la creación y gestión de múltiples tarjetas inteligentes virtuales sobre una única tarjeta. Partiendo de la premisa de que todas la tarjetas virtuales emularán el comportamiento de una tarjeta MIFARE Classic 1Kb, la aplicación subyacente contará con los mecanismos necesarios que garanticen la correcta personalización y uso de las diferentes tarjetas virtuales, restringiendo el acceso a funcionalidades y datos dependiendo del perfil del usuario.

SUMMARY

In the last decade, the technological evolution of smart cards and their ecosystem of applications has been really fast. On our daily life, more and more card are been used. The initial reluctance to use them has steadily disappeared, especially upon the launch of contactless cards. In this sense, the inherent security features has also been widely accepted.

Even though the heterogeneity considering the application scope is huge, most of the smart cards share the same underlying technology. In this sense, it seems sensible to consider that just one card can hold multiple application simultaneously, regardless of the manufacturer or the scope of the application.

Java Card technology emerges for this purpose, defining itself as a secure, portable and multi-application platform. These multi-application capabilities address the needs described above.

This work, based on a Java Card smart card, proposes the design and implementation of a solution that supports the creation and management of multiple virtual cards on a single smart card. Considering that all virtual cards will emulate the behavior of a MIFARE Classic 1KB, the underlying application will have to provide all the necessary mechanisms to guarantee the secure personalization and use of the different virtual cards, restricting access to functionalities and data depending on the user profile.

ÍNDICE

Índice de contenido

Índice de Figuras

Índice de tablas

1 INTRODUCCIÓN

La inmersión en la tecnología es uno de los factores más característicos de la sociedad actual. Cada vez es más común la utilización de herramientas digitales e inteligentes como forma de simplificar y facilitar la realización de cualquier actividad. No es difícil comprobar cómo apenas hay personas que no lleven siempre encima un teléfono móvil con conexión a internet o una tarjeta de crédito con un chip incorporado.

En los últimos años hemos sido testigos del desarrollo y penetración de la tecnología en nuestra vida diaria y como se ha ido incorporando a elementos cotidianos tales como neveras que ya disponen de acceso a internet para hacer la compra de forma automática, aparcamientos totalmente automatizados donde dejas el coche en la entrada y lo aparca solo o los navegadores para que no volvamos a perdernos nunca entre otros muchos ejemplos.

Pero sin duda alguna dos de los grandes elementos tecnológicos que han revolucionado nuestra sociedad son internet y los dispositivos móviles (*smatphones*, *tablets* u ordenadores portátiles entre otros). El primero de ellos es tan importante que la ONU (Organización de las Naciones Unidas) ha declarado el acceso a internet como un derecho humano altamente protegido [1]. El uso de internet tiene una penetración media del 67% de la población mundial (en España esa cifra sube hasta el 87%) y en el caso de los dispositivos móviles nos encontramos con índices de penetración en *smartphones* del 43% a nivel mundial (71% si hablamos de España) y un crecimiento del 21% en el año 2013 hasta el 37% en el año [2] .

Entre estos elementos tecnológicos se encuentran también las tarjetas inteligentes, tarjetas que incorporan un chip que les permite ser capaces de realizar operaciones, de imponer características y de dotar de seguridad a la comunicación a la vez que sirven de nexo entre el usuario y la entidad u organización emisora de dicha tarjeta. Este tipo de tarjetas están en auge actualmente llegando a 9600 millones de dispositivos, en 2016, que las incorporan en el mercado mundial con una previsión de crecimiento del 4% en 2017 [3].

Desde las primeras tarjetas inteligentes desarrolladas por la empresa Bull (ahora llamada Bull & Atos technologies) a finales de los 80 y puestas en marcha como tarjetas telefónicas para cabinas como sustitutas de las monedas y que consistían en un microcontrolador y un segmento de memoria hasta las que utilizamos actualmente, con una CPU mucho más potente, una memoria más extensa y funcionalidades añadidas como sistema de encriptación

complejos y sistemas de comunicación en proximidad la funcionalidad y las posibilidades de estas las han convertido en un objeto cotidiano que todo el mundo utiliza hoy en día.

Este desarrollo ha impulsado el uso de las tarjetas inteligentes en multitud de mercados, lo que ha redundado en el desarrollo del ecosistema de servicios alrededor de ellas. Así las administraciones públicas de multitud de países han impulsado su uso en los documentos de identidad (por ejemplo el DNI -Documento Nacional de Identidad- electrónico en España) o las entidades bancarias como reemplazo de las tarjetas bancarias de banda magnética.

Gracias a estas mejoras y al abaratamiento de las mismas se están empezando a usar como alternativa principal en servicios destinado al público general, por ejemplo sistemas de transportes, donde encontraríamos la Tarjeta Transporte Público de la comunidad de Madrid o la *Oyster Card* en Londres por ejemplo, sistemas de control de accesos en edificios o tarjetas universitarias con multitud de aplicaciones como la tarjeta TUI de la Universidad de Cantabria.

En todos estos servicios las ventajas que ofrecen las tarjetas inteligentes a los usuarios frente a los métodos tradicionales son evidentes. Se elimina por ejemplo la necesidad de utilizar dinero en efectivo, entre otros casos para pagar los billetes en los transportes públicos o para obtener algún tentempié en una máquina expendedora. Otra ventaja sería en combinación con la tecnología sin contactos que hace que no sea necesario siquiera sacar la tarjeta de la cartera o el bolso para poder utilizarla.

Uno de los principales problemas del uso extendido de las tarjetas inteligentes es la consecuente acumulación de tarjetas en nuestras carteras que, no solo presenta los problemas típicos de dificultad a la hora de localizar una tarjeta determinada, sino los derivados de los extravíos, robos, etc. y los potenciales usos fraudulentos.

Dada la extensión y las múltiples aplicaciones que pueden tener las tarjetas inteligentes este es un problema que debería ser atajado de tal forma que las ventajas que se obtienen por usar estos elementos no lleguen a suponer un estorbo. El problema radica en cómo disminuir el número de tarjetas físicas que un usuario cualquiera puede llegar a tener y al mismo tiempo mantener todas las ventajas de cada una de ellas así como respetar cuestiones tales como la seguridad y confidencialidad de la información y del usuario. Además hay que tener en cuenta que cada emisor requiere de unas características y un entorno de aplicación específicos, por lo que el elemento común que compartan todas las tarjetas debe adaptarse a todas estas necesidades.

En otros ámbitos se está siguiendo esta misma aproximación de integrar cada vez más datos e información sobre un mismo dispositivo o en un mismo emplazamiento como concepto lógico, si bien la gestión podría ser distribuida, prueba de ello serían los *smartphones* o teléfonos inteligentes. Estos dispositivos, que cada vez son más comunes entre la población, integran multitud de herramientas que no hace demasiado tiempo eran independientes, cualquier *smatphone* actual es capaz de tomar fotos y grabar vídeo, navegar por internet, utilizarse como linterna, como mapa, dispone de calculadora, calendario, agenda, radio, grabadora de voz y puede contener multitud de aplicaciones descargables que van desde juegos a herramientas tales como nivel, regla o brújula entre otros.

La tendencia de futuro, cuya presencia empieza a hacerse patente, es facilitar la vida de las personas con el uso de la tecnología. En el caso concreto de las tarjetas inteligentes desde

hace años se están sustituyendo las llaves de los hoteles por tarjetas para acceder a las habitaciones, así como multitud de empresas que ya las utilizan para gestionar el control de acceso, también es más frecuente encontrarlas en multitud de aplicaciones para micro pagos siendo el más habitual el transporte público. Las tarjetas inteligentes están cada vez más aceptadas entre la población.

1.1 OBJETIVOS Y MOTIVACIÓN

Como hemos comentado se está viendo como es cada vez más habitual que los distintos objetos cotidianos se comuniquen e integren entre si y como los avances en el campo de las tarjetas inteligentes están haciéndolas cada vez más versátiles, potentes y asequibles lo que creemos va a suponer un problema para el usuario en cuanto al volumen de elementos físicos va a tener que cargar encima.

El presente proyecto plantea solventar dicha situación reduciendo al máximo el número de tarjetas físicas que una persona cualquiera llevaría, a una sola tarjeta física sin que las funcionalidades y servicios proveídos por las distintas tarjetas inteligentes.

Así, el objetivo es, partiendo de una tarjeta inteligente de uso regular, crear una aplicación que compartimente su espacio y permita almacenar distintas tarjetas inteligentes virtuales en una misma tarjeta física.

1.2 CONTENIDO DE LA MEMORIA

Esta memoria se ha estructurado en seis capítulos que se describen brevemente a continuación.

El primer capítulo es la introducción donde exponemos brevemente la situación que nos ha conducido a plantear este proyecto y que esperamos conseguir de él. Nos sirve para familiarizarnos con el tema a tratar.

El segundo capítulo trata sobre las tarjetas inteligentes. Introduce su historia y evolución, para posteriormente analizar sus fundamentos, estructura y características. También se incluyen en este capítulo las especificaciones, estándares y sistemas operativos que operan con tarjetas inteligentes.

El tercer capítulo se centra en los sistemas de comunicaciones en proximidad o, más específicamente, en las tarjetas inteligentes sin contactos. Al igual que en el capítulo anterior se exponen sus características y arquitectura. Además ahonda en dos temas más específicos de nuestro proyecto, la familia de tarjetas inteligentes MIFARE y el sistema MIFARE4Mobile.

El cuarto capítulo trata sobre la especificación del proyecto. Recoge todas las herramientas que hemos utilizado para la realización del proyecto y la información básica para su correcto funcionamiento.

El quinto capítulo se centra en el desarrollo, describiendo los pasos y decisiones que se han llevado a cabo durante la creación de la parte práctica del proyecto, la aplicación para la tarjeta inteligente.

En el último capítulo, conclusiones y líneas futuras, se resumen los resultados que obtenidos durante la fase de desarrollo y como estos cumplen los objetivos inicialmente establecidos. También introduce la previsión de potenciales líneas de desarrollo futuro, basándose en proyectos similares de acuerdo con cómo se encuentra ahora el estado del arte y hacia donde está tendiendo.

TARJETAS INTELIGENTES 2

El uso de las tarjetas está arraigado en nuestra vida cotidiana como elemento de identificación, pago, fidelización, etc. Genéricamente estas tarjetas son de plástico e incluyen información del dueño o usuario de la misma, así como de la entidad emisora. Desde el primer carnet o tarjeta la evolución ha sido constante, fruto de la cual se han incluido un mayor número de capacidades y funcionalidades, al tiempo que se las ha dotado de mecanismos que eviten los usos fraudulentos e incrementen la seguridad en el acceso y almacenaje de información.

El concepto de tarjeta, considerado como dispositivo de pago, surgió como idea en 1887 por el escritor Edward Bellamy en su novela *Looking Backward* (Mirando Atrás) [4]. No es hasta 1934, de la mano de American Airlines y la Asociación del transporte aéreo, cuando aparecen las primeras tarjetas en un formato similar al actual, con una tarjeta llamada Air Travel Card [1], destinadas a la identificación de las compañías aéreas y sus clientes y que sigue actualmente en circulación bajo la red de pago Universal Air Travel Plan. En 1958 *Bank of America* lanza la primera tarjeta de crédito, BankAmericard [2], con el objetivo de reunir bajo un mismo modelo a gran cantidad de comercios en lugar de que cada uno utilizara el suyo propietario. Este primer paso sirvió de base para impulsar el uso masivo de las tarjetas a escala global. La BankAmericard cambiaría su nombre en 1977 pasando a llamarse Visa e impulsaría a la competencia a lanzar su Everything Card que posteriormente conocida como MarterCard.

El siguiente gran paso que experimentaron las tarjetas fue la inclusión de una pequeña banda magnética que permitía el almacenamiento y la lectura y escritura de información codificada en la misma, cuyo uso se masifico y extendió con el establecimiento, en 1970, de la norma ISO7811 [5]. En 1971 las primeras tarjetas magnéticas se pusieron a disposición del público general, principalmente en el sector bancario aunque también en el de los seguros, hospitales, etc. Estas tarjetas, que contaban con una única banda magnética, se quedaron rápidamente obsoletas ya que la información que se podía codificar era muy reducida para los requisitos que las grandes compañías necesitaban.

[1] https://en.wikipedia.org/wiki/Universal_Air_Travel_Plan

[2] http://about.bankofamerica.com/en-us/our-story/our-history-and-heritage.html#fbid=bH_-HmVH0EK/hashlink=technology-innovations

La estructura y codificación de la banda magnética se encuentra especificada en las partes 2, 6, 7 y 8 de la norma ISO7811, que definen un total de 3 bandas magnéticas con una capacidad teórica total aproximada de 1607 bits. El limitado volumen de información y el hecho de que únicamente puedan almacenar datos impulsaron el desarrollo de una nueva tecnología que dotara a las tarjetas de capacidades de cómputo.

Las tarjetas con circuito integrado (ICC, *Integrated Chip Cards*) fueron patentadas en 1968 y 1969 la parte correspondiente a la tarjeta con chip [6][7], en 1974 el concepto de tarjeta de memoria [8] y en 1976 la patente de una tarjeta inteligente con procesador y memoria [9].

Las tarjetas inteligentes son la evolución natural de las tarjetas tradicionales a las que añade la posibilidad de ejecutar pequeños programas y una mayor capacidad de memoria. Todo ello extiende los potenciales usos especialmente en entornos que requieren alta seguridad. Por motivos de compatibilidad con ciertos sistemas se ha mantenido la banda magnética, si bien con un uso cada vez más residual.

2.1 Arquitectura de una tarjeta inteligente

La gran mayoría de las tarjetas inteligentes se basan en la arquitectura de máquina de Von Neumann [10] constituida por una CPU (*Central Processing Unit* o Unidad Central de Procesos), un espacio de memoria para almacenar los datos y programas, una interfaz I/O (entrada/salida) para comunicarse con el exterior y un bus de comunicación entre todas las partes. En la Figura 2-1 se muestra la arquitectura descrita.

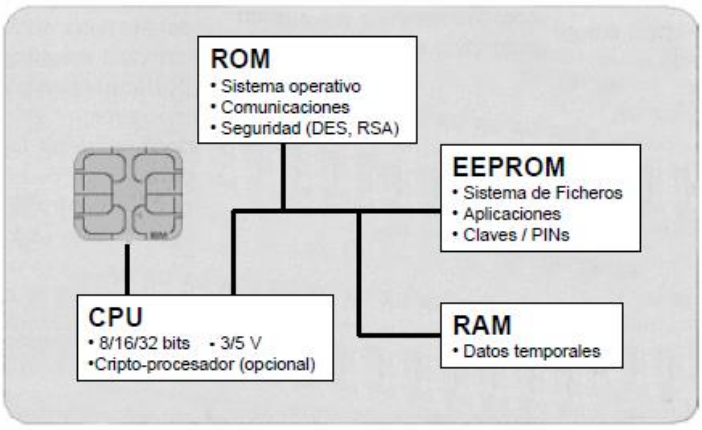


Figura 2-1 - Arquitectura de una tarjeta inteligente

La CPU es, en la mayoría de los casos, un procesador de 8 bits aunque existen y se utilizan también de 16 y 32 bits que se encarga de realizar las operaciones matemáticas y de procesar la información. En ocasiones puede incluir un coprocesador criptográfico que mejora sustancialmente el rendimiento de las operaciones de seguridad y criptografía.

Las tarjetas disponen de tres tipos distintos de memoria: ROM (*Read Only Memory* o memoria de solo lectura) donde se almacena el sistema de operativa y datos permanentes de usuario y aplicaciones, EEPROM (*Electrically Erasable Programmable ROM* o ROM programable

mediante borrado eléctrico), que aloja información variable pero persistente entre usos y RAM (*Random Access Memory* o Memoria de Acceso Aleatorio), memoria más rápida pero volátil que aloja datos específicos de cada sesión.

En las tarjetas inteligentes es muy común la fabricación de la CPU y la memoria sobre el mismo chip físico dificultando la intercepción de las señales que intercambian lo que se traduce en una alta seguridad física de los elementos almacenados en memoria.

2.2 Características físicas

La norma ISO7810 [11] recoge las características físicas de las tarjetas de identidad. Especifica no solo las dimensiones que deben de tener sino otros aspectos como la tolerancia al doblado, toxicidad, resistencia química, rango de temperatura y humedad, resistencia a la luz y resistencia al pelado de las capas. Encontramos cuatro tipos de tarjetas definidas en el estándar, como se refleja en la Figura 2-2:

- ID-3 utilizadas principalmente para pasaportes y visados.
- ID-2 utilizada para algunos documentos de identificación, como el francés.
- ID-1 usado para las tarjetas bancarias, la mayoría de tarjetas de fidelización y de identificación, como el DNIe o el carnet de conducir español.
- ID-000 dedicado a las tarjetas SIM utilizadas en los teléfonos móviles pero no incluye las modificaciones posteriores de formatos micro-SIM y nano-SIM.

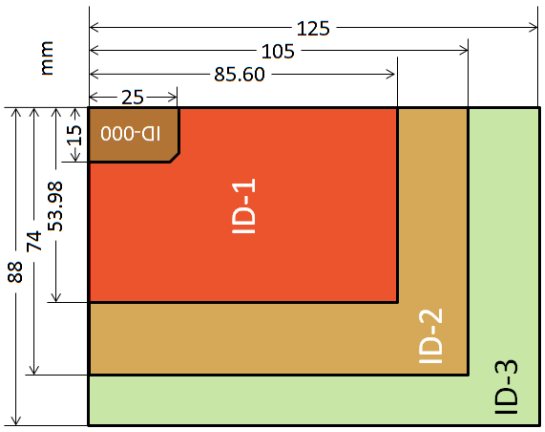


Figura 2-2 - Dimensiones de las tarjetas según la norma ISO7810

Por otro lado, la parte más reconocible de en la mayoría de las tarjetas inteligentes sería los contactos. Éstos están estandarizados en disposición y funcionalidad en la parte 2 de la norma ISO7816 [12], si bien existen diferentes patrones de presentación. Estos contactos son el enlace entre el dispositivo lector y la tarjeta proveyendo a esta de la alimentación que necesita

y permitiendo llevar a cabo la transmisión de los datos, la funcionalidad de los contactos es la que se muestra en la Figura 2-3.

Figura 2-3 - Distribución y uso de los distintos contactos

- Vcc se utiliza para suministrar la alimentación al chip. El voltaje que se aplica es de 3V o 5V. En telefonía móvil se emplean 3V.
- RST es el contacto por el que se envía la señal de reset al microprocesador.
- CLK proporciona una señal externa de reloj que se tomará como referencia para la generación del reloj interno.
- GND es la conexión de masa.
- Vpp es un conector en desuso, ya que el voltaje externo que proporcionaba para poder grabar y borrar las memorias EEPROM de las tarjetas antiguas, actualmente se genera internamente. No obstante, se mantiene por motivos de compatibilidad.
- I/O se emplea para transferir datos entre el dispositivo lector y la tarjeta en modo half-duplex. El puerto de entrada/salida consiste en un simple registro a través del cual la información es transferida bit a bit.
- RFU (Reserved for Future Use) son contactos sin uso actual y están reservados para uso futuro

2.3 PROTOCOLOS DE COMUNICACIÓN

La comunicación con la tarjeta inteligente se hace a través de un dispositivo lector de tarjetas inteligentes o CAD (*Card Acceptance Device, Dispositivo de Aceptación de Tarjetas*). Un lector es el dispositivo que se conecta a un ordenador u otro terminal móvil o fijo y que permite establecer un canal de comunicación entre la tarjeta y el terminal. Aunque los lectores no suelen tener inteligencia suficiente para procesar datos si suelen tener algún sistema de detección y corrección de errores.

Existiendo un único contacto para el intercambio de datos en las tarjetas con contacto y utilizando una única frecuencia en las sin contacto, la comunicación entre tarjeta y CAD es half-duplex y se sigue un modelo de operación tipo maestro-esclavo en el que la tarjeta toma el rol del esclavo, actuando de forma pasiva al no realizar ninguna acción esperando a recibir un comando del CAD.

El intercambio de datos se realiza mediante un protocolo que estructura la información en paquetes de datos llamados APDU (*Application Protocol Data Unit,* Unidad de Datos del Protocolo de Aplicación) pudiendo ser estos de dos tipos: APDU comando y APDU respuesta.

2.3.1 ENTORNO CON CONTACTOS

En las tarjetas con contactos la comunicación está estandarizada mediante la norma ISO7816 en sus partes 3 y 4.

Cuando la tarjeta se introduce en el lector y sus contactos se conectan, ésta envía la secuencia ATR (*Answer To Reset*, Respuesta al Reinicio) hacia el terminal. El ATR contiene información sobre los parámetros que necesita la tarjeta para establecer la comunicación entre lector y tarjeta. Consta de cinco pasos: el carácter de inicio TS, el byte de formato T0, bytes de interfaz TA_i, TB_i, TC_i, TD_i cuya existencia está subordinada al valor del byte T0, los bytes históricos T_i opcionales y también subordinados al valor de T0 y el byte de control TCK.

Las APDU se envían encapsuladas en el protocolo de transporte definido en la norma ISO7816 en su parte 3. Los datos encapsulados bajo este protocolo se llaman TPDU (*Transmission Protocol Data Units*, Unidad de Datos del Protocolo de Transmisión).

De todos los protocolos de transporte definidos en la norma los dos más utilizados en las tarjetas inteligentes son los protocolos T=0 y T=1. Ambos protocolos son de transmisión half-duplex asíncrona diferenciándose en que el protocolo T=0 está orientado al byte mientras que T=1 está orientado al bloque, es decir, la unidad más pequeña de datos que se puede procesar y transmitir sería de un byte para el T=0 y de un bloque, una secuencia indefinida de bytes consecutivos, para el T=1.

Una vez iniciada la comunicación toda la información será transmitida a través de mensajes APDU. El formato de los mensajes APDU viene definido en la parte 4 de la norma ISO7816 pudiendo dividir los mensajes en dos estructuras: los mensajes enviados por el CAD hacia la tarjeta, también llamados APDU Comando (C-APDU) y las respuestas a estos comandos generadas por la tarjeta o APDU Respuesta (R-APDU). En toda comunicación siempre ha de existir una pareja comando-respuesta.

Tabla 2-1 - Comando APDU

Cabecera obligatoria				Campos adicionales		
CLA	INS	P1	P2	Lc	DATOS	Le

La APDU Comando, tal como se muestra en la Tabla 2-1, se compone de dos partes, una cabecera de envío obligatorio y un cuerpo de envío opcional. La cabecera consta de 4 campos, cada uno de ellos con una longitud de 1 byte:

- CLA (clase de instrucción) identifica la categoría del comando enviado y la respuesta esperada.

- INS (código de instrucción) instrucción del comando dentro de esa categoría,
- P1 y P2 (parámetro 1 y 2 respectivamente) utilizados para proporcionar información adicional al comando.

El cuerpo se compone de 3 campos:

- Lc, con una longitud de 1 byte, indica la longitud (en bytes) del campo de datos,
- Campo de datos, tiene una longitud variable y contiene los datos enviados por el comando
- Le que con una longitud de 1 byte indica la longitud (en bytes) de los datos que se esperan recibir en la respuesta.

Tabla 2-2 - Respuesta APDU

Campo opcional	Trailer obligatorio
DATOS	SW

La APDU Respuesta, mostrada en la Tabla 2-2 se divide en dos campos:

- Datos: es un campo opcional cuya longitud está determinada en el campo Le de la APDU comando y es donde se encuentran los datos generados por la tarjeta en respuesta al comando.
- SW: consta de dos campos, SW1 y SW2, de 1 byte de longitud cada uno que, en conjunto, forman la llamada *status word* e indica si ha ocurrido algún error durante el procesamiento de la APDU comando. El valor más habitual de la *status word* es "0x9000" que significa que el comando se ha procesado y ejecutado correctamente.

2.3.1.1 Protocolo de comunicación de la parte sin contactos

Al contrario de lo que sucede con la parte con contactos, la parte sin contactos, aunque cuenta con un estándar, el ISO14443 [13], en la mayoría de las tarjetas que hemos tomado como referencia, solo se aplican las tres primeras partes de dicho protocolo, a saber; la parte 1 referente a las características físicas, la parte 2 que recoge la forma de las señales, su potencia, su modulación, etc y la parte 3, que cuenta con los mecanismos de inicialización y anticolisión. La parte 4 recoge los protocolos de transmisión, donde se utilizan los comandos APDU como en la parte con contactos, aunque esta última parte no está implementada en todas las tarjetas, usando únicamente un protocolo propietario o un protocolo mixto.

También hay que incluir, entre todas las opciones que tenemos, los dos tipos de tarjetas que específica la parte 2 de la norma ISO14443, las tarjetas tipo A y tipo B que se diferencian en el tipo de modulación que utilizan.

Tanto la estructura como los protocolos de comunicación de la parte sin contactos los explicaremos más detenidamente en el capítulo 3-Sistemas de comunicaciones en proximidad, dedicado a este aspecto.

2.4 SISTEMAS OPERATIVOS, ESTÁNDARES Y ESPECIFICACIONES

Para facilitar el uso y desarrollo de las tarjetas inteligentes de forma masiva necesitamos herramientas que nos permitan trabajar bajo una misma base, es decir, bajo un mismo estándar y especificaciones estructuradas que permitan la comunicación entre dispositivos de distintos operadores.

En la parte de los estándares tenemos tres principales que pertenecen al mundo de las tarjetas inteligentes: el estándar ISO7810 que especifica las dimensiones físicas de las tarjetas, el estándar ISO7816 para las tarjetas inteligentes con contactos y el estándar ISO14443 para las tarjetas inteligentes sin contactos.

El estándar ISO7810 ha sido ya comentado, para más información consultar el capítulo 2.2-Características físicas.

El estándar ISO7816 denominado "*Identification cards – Integrated circuits cards with contacs*" (Tarjetas de identificación – Tarjetas de circuitos integrados con contactos) es el que define las características de las tarjetas con contactos. Los aspectos más importantes que este estándar cubre son los referentes a las características físicas de las tarjetas inteligentes, la estructura de los comandos y los propios comandos estandarizados, las señales eléctricas y la aplicación de la criptografía.

El estándar ISO14443 denominado "*Identification cards – Contactless integrated circuits cards – Proximity cards*" (Tarjetas de identificación – Tarjetas de circuitos integrados sin contactos – Tarjetas de proximidad) define las características de las tarjetas en proximidad. Si bien su extensión es más reducida que el estándar ISO7816 y no especifica unos comandos de comunicación estandarizados (aunque puede encapsular APDU). El estándar recoge las características físicas de las tarjetas, las señales de radio a utilizar, los protocolos de inicialización y anticolisión y los de transmisión.

A pesar de la existencia de los citados estándares, estos no bastan para asegurar el uso masivo de las tarjetas inteligentes. Para ello necesitamos dos sistemas adicionales que nos permitan un uso genérico de las aplicaciones por lo que necesitamos unas tarjetas independientes del fabricante y hardware que se vaya a utilizar.

El primero es una tecnología que nos permita el desarrollo de aplicaciones dentro de aplicaciones bajo una misma especificación de un sistema abierto. En este caso disponemos de varías tecnologías que han ido apareciendo con los años como podrían ser MULTOS [14] (desarrollado por un consorcio de empresas, desde fabricantes a desarrolladores), Windows Powered Smart Cards (desarrollado por Microsoft e integrado en su SDK) y Java Card [15]

(basado en tecnología Java y desarrollado por Sun Microsystems). La última de estas tecnologías es la que presenta un uso más extendido y por ese motivo, el presente proyecto basará los desarrollos en esta tecnología.

El segundo sistema es una especificación estándar para la gestión de la tarjeta. GlobalPlatform ofrece una especificación [16] que se centra en la infraestructura de la tarjeta proveyendo un sistema de seguridad y una arquitectura de gestión de la tarjeta común para cualquier tecnología que se utilice y asegurando que se pueda realizar una migración futura a otra tecnología si la situación lo requiere.

Además y debido a la complejidad que supondría generar todo el entorno de GlobalPlatform para poder comunicarnos con la tarjeta, utilizaremos el programa GPShell que nos servirá como puente de unión en la comunicación entre nuestro ordenador y la tarjeta inteligente.

2.4.1 JAVA CARD

El lenguaje de programación Java es sin duda uno de los más ampliamente utilizados del mundo desde sistemas embebidos hasta supercomputadores, con más de 9 millones de desarrolladores y cerca de 10 mil millones de dispositivos (entre ellos 5 mil millones de Java Cards en uso) [3].

Uno de los principales factores de la expansión de este lenguaje es su filosofía WORA (*Write Once Run Anywhere*) que se basa en código que puede generarse en una plataforma y ejecutarse en la misma u otra sin que, para ello, tenga que modificarse.

Recogida como una de las tecnologías de Java, Java Card permite desarrollar aplicaciones basadas en Java para tarjetas inteligentes y dispositivos embebidos con pequeños segmentos de memoria. Entre estos dispositivos cabe destacar las tarjetas SIM de los teléfonos móviles o las tarjetas bancarias con chip incorporado.

2.4.1.1 LIMITACIONES DE JAVA CARD

A pesar de que su estructura es igual que las demás ediciones de Java, Java Card al estar pensada para dispositivos portátiles, en cierta parte autónomos y con una limitada cantidad de memoria y poder de procesado, se creó con una serie de limitaciones comparada con otras ediciones. Estas limitaciones, pensadas para reducir la carga de procesamiento y almacenaje del dispositivo sobre el que fueran implementados deben ser tenidas en cuenta a la hora de programar. En la Tabla 2-3 hacemos referencia las diferencias entre Java y Java Card.

[3] http://www.java.com/en/about/

Tabla 2-3 - Diferencias de elementos soportados entre Java Card y Java

Elementos Java soportados	Elementos Java no soportados
• Tipos de datos primitivos pequeños: boolean, byte, short y, opcionalmente, int. • Arrays unidimensionales. • Paquetes, clases, interfaces y excepciones. • Orientación a objetos	• Tipos de datos primitivos grandes: long, double y float. • Arrays multidimensionales, caracteres y strings. • Carga dinámica de clases. • Recolector de basura. • Multiproceso. • Serialización y clonación de objetos. • Gestor de seguridad.

Algunas de estas limitaciones no han influido en nuestro proyecto como podrían ser la imposibilidad de crear varios hilos, la de clonación de objetos o que no esté soportada el método de finalización.

2.4.1.2 Paquetes, clases y métodos únicos de Java Card

Al igual que tiene limitaciones, Java Card dispone de algunas clases únicas para la comunicación y gestión de tarjetas inteligentes. Las más relevantes en nuestro caso son:

- Paquete javacard.framework: Este paquete contiene las clases que se encargan de la comunicación entre tarjeta y programa.
- Paquetes javacard.security y javacardx.crypto: En combinación estos dos paquetes son los encargados del cifrado de datos, una funcionalidad necesaria en cualquier elemento que contenga datos privados.
- Paquete javacardx.external: Nos permite acceder a subsistemas de memoria que no son directamente accesibles por el entorno Java Card.

2.4.2 GLOBALPLATFORM

GlobalPlatform [4] es una organización que promueve una implementación global de la infraestructura de las tarjetas inteligentes. Su objetivo es asegurar el desarrollo de las tarjetas inteligentes multi-aplicación fomentando la compatibilidad entre los distintos fabricantes y usuarios.

La iniciativa Open Platform [5] creada por Visa Internacional ha evolucionado hacia GlobalPlatform definiendo una especificación para la gestión de las tarjetas que fuera independiente del hardware, fabricante y de las aplicaciones que contuviera. Esta especificación provee de un sistema de seguridad y una arquitectura de gestión común que protege la infraestructura de la tarjeta.

Gracias a esta especificación los emisores de tarjetas pueden crear tarjetas multi-aplicación con la tecnología de tarjetas que precisen asegurándose además que podrán migrar a una nueva tecnología en caso de ser necesario sin que la infraestructura se vea comprometida.

2.4.2.1 ARQUITECTURA DE GLOBALPLATFORM

La arquitectura de GlobalPlatform está diseñada para proveer a los emisores de tarjetas inteligentes de un sistema de gestión asegurando que las aplicaciones y los sistemas de gestión exteriores a la tarjeta no dependerán del hardware y las interfaces añadidas por los vendedores en las tarjetas.

Aunque GlobalPlatform se basa en la suposición de que únicamente existirá un emisor por tarjeta da la posibilidad de ceder y compartir parte del control sobre parte de la tarjeta a terceros autorizados por el emisor de la misma. Si bien el emisor siempre tendrá el control total, aquellos autorizados podrán gestionar sus propias aplicaciones dentro de la tarjeta con total control sobre sus aplicaciones.

Para asegurar la independencia de las aplicaciones y los sistemas de gestión externos a la tarjeta la arquitectura de GlobalPlatform se divide en una serie de componentes que se pueden observar en la Figura 2-4.

[4] https://www.globalplatform.org/
[5] http://www.opengroup.org/

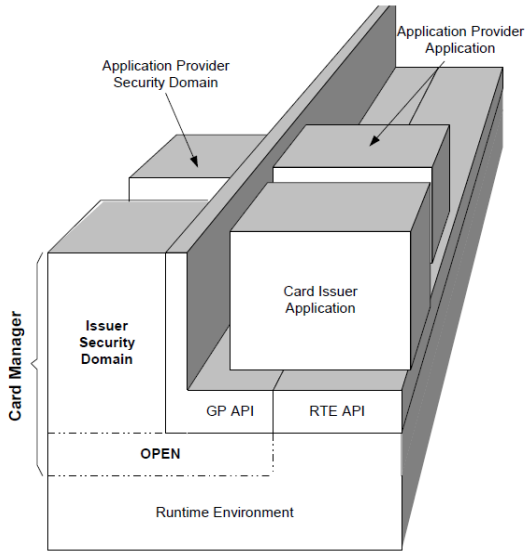


Figura 2-4 - Arquitectura de una tarjeta GlobalPlatform.

Imagen obtenida de la especificación de GlobalPlatform

Aparte de los componentes específicos de GlobalPlatform todas las aplicaciones deben ser implementadas en un entorno de ejecución (RTE, *Runtime Environment*) que incluya una interfaz de programación de aplicaciones (API, *Application Programming Interface*) que no dependa del hardware con el objetivo de permitir la portabilidad de las aplicaciones.

El gestor de la tarjeta (*Card Manager*) es el componente principal de GlobalPlatform y actúa como el administrador central de la tarjeta. Adicionalmente se crean otros Dominios de Seguridad (*Security Domains*), con sus propias políticas de seguridad y claves, de forma que se garantiza la independencia entre el emisor de la tarjeta y los proveedores de aplicaciones, por ende entre las diferentes aplicaciones residentes en la tarjeta.

El gestor de la tarjeta se compone de tres entidades distintas para desempeñar todas sus funciones:

- Entorno de GlobalPlatform (OPEN)
- Dominio de seguridad del emisor
- Método de verificación del titular de la tarjeta

2.4.2.1.1 Entorno de GlobalPlatform (OPEN)

El OPEN se encarga de gestión la carga de las aplicaciones así como su instalación en la tarjeta. Es responsable de asegurar las medidas de seguridad para la carga e instalación de contenido, entre las que se encuentran verificar el código de la aplicación y que exista autorización por parte del emisor de la tarjeta para la carga e instalación.

Entre sus funciones también cuenta la selección de aplicaciones, así como la gestión de los comandos APDU, que se redirigirán a la aplicación seleccionada previamente. La existencia de

canales lógicos en una tarjeta permite la selección simultánea de varias aplicaciones como aplicaciones seleccionadas

2.4.2.1.2 Dominio de seguridad del emisor

El dominio de seguridad del emisor, presente por defecto en todas las tarjetas Java Card, está directamente vinculado al emisor y ofrece las capacidades de cargar, instalar y borrar aplicaciones que propias como de terceros.

Cualquier dominio de seguridad es una aplicación con permisos especiales, que proporciona servicios de gestión de claves, cifrado y descifrado, generación de firma digital y verificación de su dueño (si es un dominio de seguridad del emisor de la tarjeta, de un proveedor de aplicaciones o de una autoridad controladora).

2.4.2.1.3 Método de verificación del usuario de la tarjeta

El método de verificación del usuario de la tarjeta (CVM, *Card Verification Method*) es un mecanismo opcional que el gestor de la tarjeta puede proveer y que puede ser utilizado por todas las aplicaciones dentro de la tarjeta. La especificación de GlobalPlatform provee de un método para implementar un servicio de número global de identificación personal (global PIN) a la tarjeta para satisfacer los requisitos de la verificación del usuario. Los servicios ofrecidos por el CVM pueden ser disfrutados por cualquier aplicación si bien su gestión se reservara únicamente a aplicaciones con privilegios para ello.

2.4.2.2 CANALES SEGUROS

GlobalPlatform define protocolos para el establecimiento de canales seguros que aseguren la transferencia de comandos APDU mediante un proceso de autenticación mutua. La utilización de canales seguros por parte de una aplicación está condicionada por el dominio de seguridad en el que se hospede.

Una sesión de canal seguro se divide en tres fases:

- Fase de iniciación: la aplicación y la entidad externa intercambian información para realizar las funciones de cifrado necesarias en la comunicación segura. Esta iniciación siempre requiere la autenticación de la entidad externa por parte de la aplicación.
- Fase de operación: el intercambio de datos entre la entidad externa y la aplicación se encuentra cifrado según la protección elegida durante la iniciación.

- Fase de finalización: cuando la entidad externa o la aplicación no precisa de más datos o la comunicación deja de estar permitida se finaliza el canal seguro. La finalización puede ser solicitada por la entidad externa o la aplicación a través de comandos específicos, aunque también se da por finalizada una sesión con el consecuente cierre del canal cuando se produce un error en la comunicación segura entre entidades.

La especificación de GlobalPlatform describe dos protocolos de canal seguro aunque dispone varios identificadores de protocolo de canal seguro reservados para uso futuro. Los dos protocolos definidos se denominan Protocolo de Canal Seguro 01 y 02 (SCP01 y SCP02 por sus siglas en inglés respectivamente) con los identificadores de protocolo 01 y 02 respectivamente.

El protocolo SCP02 es una evolución del SCP01 que permite más opciones de implementación por un lado, permite que la inicialización del canal seguro se realice tanto de forma implícita como explicita y tanto la tarjeta como la entidad externa puedan tomar el rol de las entidades de envío y recepción seguras (en el SCP01 solo podía iniciarse de forma explícita y solo la entidad externa podía ejercer de entidad de envío seguro). Nos centraremos en el SCP02 ya que es el protocolo más completo y el que utilizaremos durante la realización del proyecto.

El protocolo SCP02 proporciona tres niveles de seguridad:

- Autenticación de entidad: la tarjeta autentica la entidad externa y esta puede autenticar la tarjeta demostrando que la entidad externa tiene conocimiento sobre los mismos secretos que la tarjeta.
- Integridad y autenticación del origen de los datos: la entidad receptora (puede ser la tarjeta o la entidad externa) aseguran que los datos recibidos provienen de una entidad emisora autenticada en el orden correcto y sin haber sufrido ninguna alteración.
- Confidencialidad: los datos transmitidos por la entidad emisora a la entidad receptora no serán accesibles por una entidad no autorizada.

2.4.3 GPSHELL

GPSHELL es un intérprete de scripts que nos permite comunicarnos con una tarjeta inteligente a través de un ordenador e implementa la especificación de tarjeta de GlobalPlatform. Incorpora las librerías de GlobalPlatform, que proporciona un API para las funciones específicas de GlobalPlatform, y el plugin de conexión PC/SC, que implementa el estándar PC/SC (Personal Computer/ Smart Card) para integrar las tarjetas inteligentes en el entorno de los ordenadores permitiendo la comunicación entre el ordenador y un lector de tarjetas inteligentes conectado al mismo.

GPSHELL nos permite instalar, desinstalar, seleccionar y listar aplicaciones, establecer canales seguros y comprobar el estado de la tarjeta entre otras funciones.

SISTEMAS DE COMUNICACIONES EN PROXIMIDAD

3

A diferencia de las tarjetas de banda magnética o las tarjetas de contactos, las tarjetas basadas en un sistema de comunicación en proximidad no precisan introducirse en un dispositivo lector sino que, para funcionar, únicamente necesitan estar en el entorno del dispositivo lector , entre unos pocos centímetros y varias decenas de metros según la tecnología utilizada, lo que facilita su uso, por ejemplo, pudiendo utilizarla sin sacar la tarjeta de la cartera y no sufre tanto desgaste como los contactos en las tarjetas de contactos o la banda magnética en las tarjetas de banda magnética.

Cada vez es más común encontrarse este tipo de tarjeta en sectores como en seguridad, con tarjetas sin contactos para el de control de acceso; en banca, con las tarjetas de crédito o débito que incorporan comunicación sin contacto; o en el transporte público y peajes, en forma de abonos o tarjetas monedero entre otros.

Sus características físicas son las mismas que las de las tarjetas con contactos, las tarjetas sin contacto incluyen una antena o espira entre las capas de plástico que las permiten habilitar el canal de comunicación entre el lector y el chip de la tarjeta.

3.1 ARQUITECTURA DE UNA TARJETA INTELIGENTE SIN CONTACTOS

La arquitectura de una tarjeta inteligente sin contactos podría dividirse en dos grandes bloques, tal como se muestra en la Figura 3-1 dividirla en dos: el micro controlador y la interfaz de radiofrecuencia.

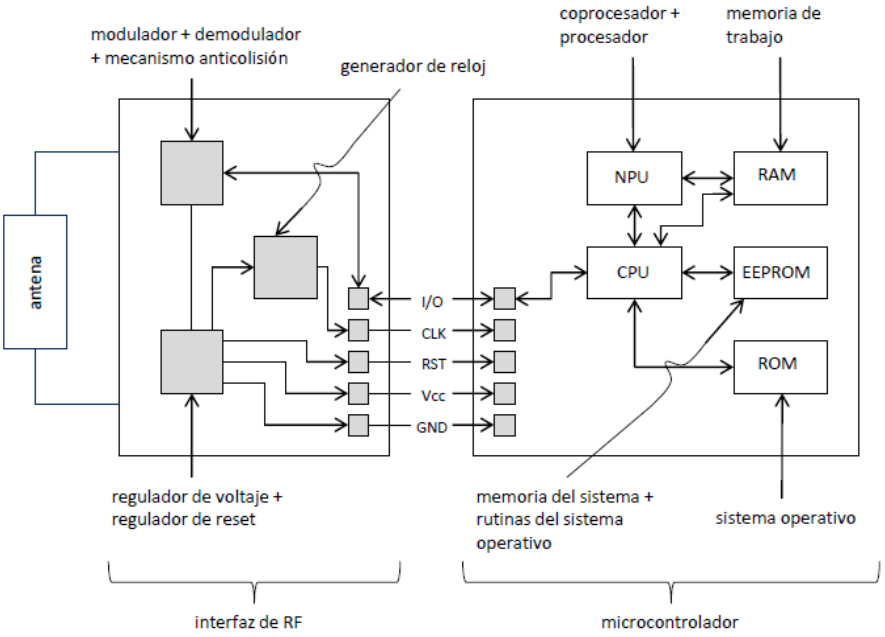


Figura 3-1 - Arquitectura típica de una tarjeta sin contactos

La arquitectura de la parte del microcontrolador es similar a la descrita en la sección 2.1, implementado una máquina de Von Newman. En este caso, tanto la señal de activación de la tarjeta como la de datos se transfieren a través de la interfaz de radiofrecuencia.

La interfaz de radiofrecuencia consta de tres bloques: un primer bloque que contiene el modulador, demodulador y el mecanismo de anticolisión, un segundo bloque generador de la señal de reloj y un tercer bloque que consta de un regulador de tensión y un generador de *reset*. El primer bloque se encarga de demodular la señal recibida y modular la señal a transmitir según las especificaciones de la norma ISO14443 en su parte 2 y también posee el mecanismo de anticolisión tal y que cumple con sus funciones tal y como específica la norma ISO14443 en su parte 3. El segundo bloque se compone del generador de reloj que genera la señal de reloj para el microcontrolador. Por último, el tercer bloque, se encarga de regular la tensión que recibe la tarjeta, a través de la antena, de los campos magnéticos de alta frecuencia producidos por el lector y suministra corriente a la tarjeta; y del generador de *reset* que será el encargado de generar la señal de *reset* de la tarjeta.

Fuera de los dos bloques encontraríamos la antena, una serie de hilos metálicos dispuestos entre las dos capas de plástico que conforman la tarjeta sin contactos, según específica la norma ISO14443 en su parte 1.

3.2 METODOLOGÍAS DE COMUNICACIÓN SIN CONTACTOS

En los métodos de comunicación nos encontramos con dos grandes grupos: las tarjetas de vecindad y las tarjetas de proximidad.

Las primeras se rigen según la norma ISO15693 [17] y están diseñadas para ser tarjetas operadas a una cierta distancia teniendo una distancia de lectura máxima de entre un metro y un metro y medio. Trabajan a 13,56 MHz, con subportadoras a 423,75 kHz obteniendo unas tasas binarias inferior y superior de 6,62 kbit/s y 26,48 kbit/s respectivamente.

Las segundas trabajan bajo la norma ISO14443, diseñadas para operar a muy corta distancia su rango efectivo máximo oscila entre los cinco y diez centímetros. Al igual que las tarjetas de vecindad trabajan a 13,56 MHz pero con subportadoras a 847,5 kHz con una tasa binaria de 106 kbit/s.

Aparte de estos dos grandes grupos encontramos una variedad de tarjetas que podríamos denominar híbridas. Estas tarjetas no entran completamente en ninguno de los dos grupos sino que combinan parte de las normas que los rigen con partes propietarias. La mayoría se basan en la norma ISO14443, sobretodo en sus tres primeras partes, utilizando protocolos propietarios para el resto.

Entre las diferentes tarjetas que se incluyen en el último grupo podemos destacar a FeliCa [6], LEGIC RF [18], y MIFARE [19] como las más importantes.

FeliCa es una tecnología de tarjetas inteligentes, desarrollada por Sony, y principalmente usado en tarjetas monedero y abonos de transporte. Está ampliamente extendida en Asia, sobretodo en Japón donde es un estándar de facto y es utilizado en gran parte de los transportes públicos del país [7]. Su despliegue ha llegado incluso a las universidades estadounidenses [8]. FeliCa fue propuesta para incluirse en la norma ISO14443 como tarjeta de tipo C pero fue rechazada. FeliCa utiliza codificación Manchester a 212 kbit/s en la banda de 13,56 MHz y con una distancia máxima entre tarjeta y lector de 10 cm.

LEGIC_RF es un estándar creado por la compañía Legic, con sede en Suiza, cuya tecnología se centra en combinar y gestionar múltiples aplicaciones entre las que destacan el control de accesos, pagos sin contacto, tickets electrónicos, control de tiempos y asistencia, etc. El estándar de Legic también trabaja en la banda de los 13.56 MHz. Aunque la tecnología Legic tiene bastante en común con la norma ISO14443, ahonda más en la seguridad, haciendo especial hincapié en el reconocimiento mutuo entre la tarjeta y el receptor y en el intercambio de señales RF cifradas. Una de las grandes ventajas de las tarjetas Legic es su versatilidad ya que no solo se basan en su estándar LEGIC_RF, sino que también son compatibles con las normas ISO14443 e ISO15693.

[6] https://www.sony.net/Products/felica/business/products/index.html
[7] http://www.sony.net/Products/felica/casestudy/index.html
[8] http://www.sony.net/Products/felica/casestudy/e-ID.html

La tecnología MIFARE es muy utilizada para tarjetas de memoria sin contactos. El presente proyecto se centra en la emulación de una tarjeta perteneciente a esta familia por lo que a continuación se analiza en mayor profundidad la familia de tarjetas sin contacto MIFARE.

3.3 FAMILIA MIFARE

La familia MIFARE es la tecnología de tarjetas sin contactos más extendida con 150 millones de lectores vendidos y 10000 millones de tarjetas en circulación. Todas las tarjetas de la familia MIFARE implementan la norma ISO14443 (tipo A), si bien solo la implementan al completo las familias DESFire o SmartFX, las demás (familias Classic, Ultralight y Plus) solo incorporan hasta la parte 3, sustituyendo la parte 4 por un protocolo de comunicación propietario.

Los productos que podemos encontrar son: MIFARE Ultralight [20], MIFARE Classic [21], MIFARE Plus [22], MIFARE DESFire [23] y SmartMX [24].

MIFARE Ultralight es un tipo de tarjeta con poca memoria especialmente diseñada para usarse en entornos donde se requiera un uso limitado de la tarjeta, bien en número de usos, tiempo de uso, lugar específico o una combinación de las tres. También son utilizados en aplicaciones de bajo coste y gran volumen de usuarios.

MIFARE Classic es una tarjeta con más capacidad que la MIFARE Ultralight y mayor capacidad de gestión de dicha memoria, aunque está también dirigido a aplicaciones de bajo coste y gran volumen de usuarios. Este tipo de tarjeta será la que utilizaremos en nuestro proyecto por lo que las características de la misma serán expuestas con detalle más adelante.

MIFARE Plus es una mejora de MIFARE Classic. Su base es la misma pero proporciona mejoras de seguridad al implementar autenticación y cifrado de los datos basándose en estándares abiertos como AES (*Advance Encryption Standard*). Además permite autenticación múltiple de sectores y acceso múltiple a bloques.

MIFARE DESFire está orientado a desarrolladores y operadores de sistemas que requieran soluciones con tarjetas inteligentes sin contactos. Usa algoritmos de seguridad DES (*Data Encryption Standard*), aunque las últimas especificaciones también incluyen AES (*Advance Encryption Standard*) para la implementación de la seguridad en la transmisión de datos. MIFARE DESFire se caracteriza por tener una organización de la memoria flexible, una transmisión de datos rápida y muy segura e interoperabilidad con las infraestructuras para tarjetas sin contactos ya existentes.

Por último, SmartMX son tarjetas inteligentes preparadas para soportar múltiples aplicaciones con un alto rendimiento y seguridad de los datos a un bajo coste, asegurando la independencia de las aplicaciones entre sí. SmartMX incorpora una interfaz dual (cuenta tanto con interfaz sin contacto como con contacto) e incorpora un chip de encriptación para encriptación simétrica (DES y AES) y asimétrica (PKI y ECC).

3.3.1 MIFARE CLASSIC

En nuestro proyecto utilizaremos las dos versiones de tarjetas que nos ofrece MIFARE Classic, la de 4 kBytes será la tarjeta física con la que contaremos y sobre esta mapearemos las de 1 kByte.

La única diferencia entre estas dos tarjetas es la capacidad de memoria de que disponen y la forma en la que esta está estructurada, la de 4kB tiene una estructura un poco más compleja como veremos más adelante.

El documento de especificación de ambas tarjetas es el MF1S70yyX/V1 [21] sobre el que nos hemos basado y al que referimos si se desean conocer más en detalle las características y funcionamiento que expondremos en los siguientes apartados.

3.3.1.1 CARACTERÍSTICAS MIFARE CLASSIC

La tarjeta se compone de tres grandes bloques: una interfaz de radio frecuencia que sirve para comunicarse con el exterior, una unidad de control digital que podríamos entenderla como el cerebro de la tarjeta y una memoria de tipo EEPROM

La arquitectura de las tarjetas MIFARE Classic sigue el patrón de las tarjetas inteligentes sin contactos descritas en 3.1-Arquitectura de una tarjeta inteligente sin contactos. Además de la interfaz de RF y la memoria EEPROM cuenta con una Unidad de Control Digital un poco más avanzada que la estructura del micro controlador antes expuesta. En concreto cuenta con tres módulos más de los que aparecen en la estructura básica:

- El módulo de autenticación: Un vez la tarjeta es seleccionada se encarga, mediante una autenticación en 3 pasos, de establecer una comunicación segura entre la tarjeta y el PCD. A partir del primer paso de la autenticación toda la comunicación entre los dos lados estará cifrada.
- Módulo de Control y ALU: Se encarga de las operaciones de incremento y decremento de los valores que se encuentran almacenados en un formato especialmente redundante en la memoria.
- Módulo Criptográfico: La seguridad en el intercambio seguro de datos corre a cargo de este módulo que se encarga de cifrar los datos con el cifrado tipo CRYPTO1.

3.3.1.2 COMUNICACIÓN Y SEGURIDAD MIFARE CLASSIC

La comunicación con la tarjeta MIFARE Classic se realiza mediante comandos-respuestas. Los comandos son generados por el PCD y analizados por la Unidad de Control Digital de la tarjeta.

El flujo de inicialización y autenticación entre la tarjeta MIFARE Classic y el PCD es el que se muestra a continuación en la Figura 3-2.

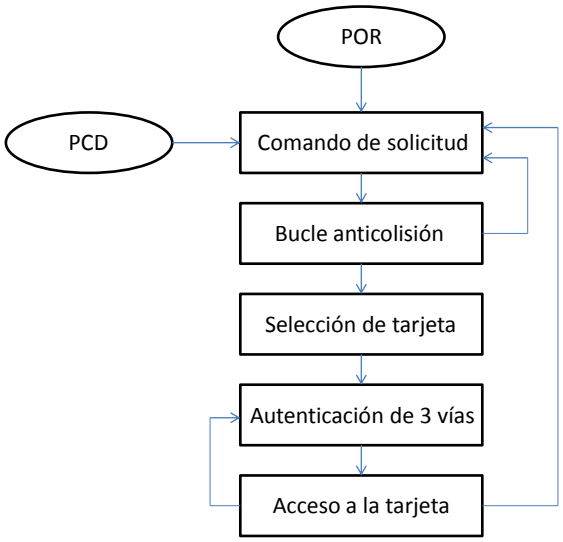


Figura 3-2 - Flujo de inicialización y autenticación de la tarjeta MIFARE Classic

Al aproximarse la tarjeta al PCD esta se activa, también llamado *Power On Reset* (POR). Una vez activa puede responder al comando de solicitud, enviado por el PCD a todas las tarjetas a su alcance, con el código de respuesta ATQA (recogido en la norma ISO14443 para tarjetas de tipo A). La respuesta de la tarjeta se utiliza para que el PCD se percate de su presencia y pueda seleccionarla.

Una vez se ha enviado el ATQA se entra en el bucle anticolisión. En el bucle anticolisión el PCD lee los números de serie de las tarjetas que le han enviado un ATQA y seleccionar una de ellas para comenzar con las operaciones. Las tarjetas que no han sido seleccionas se quedan en modo espera hasta que el PCD mande nuevamente el comando de solicitud.

Seleccionada una tarjeta comienza el proceso de autenticación en 3 pasos mediante el envío de retos cifrados para comprobar la identidad de las dos partes.

Tras la transmisión del primer reto aleatorio, la comunicación entre PCD y tarjeta estará encriptada mediante el algoritmo de encriptación Crypto1, algoritmo propietario de la familia MIFARE Classic.

El algoritmo de encriptación Crypto1 [25] es un cifrado de flujo. Cuenta con un generador de bit pseudo-aleatorio formado por un registro de desplazamiento linear realimentado (LFSR) de 48 bits y dos niveles de filtros no lineales. Entre la información utilizada por este algoritmo se encuentran: la clave seleccionada por el PCD, el número de serie de la tarjeta y los retos utilizados durante la fase de autenticación.

El algoritmo de encriptación Crypto1 era utilizado en todos los productos de la familia MIFARE Classic como base para su seguridad, el problema es que el algoritmo tiene ciertas vulnerabilidades. En diciembre de 2007 aparecieron los primeros artículos sobre los problemas que presentaba Crypto1 y que, mediante ingeniería inversa, consiguieron ir desvelando el algoritmo y dieron pie a los primeros intentos de ataque a las tarjetas que culminaron en

ataques satisfactorios contra la tarjeta donde se logra, sin tener información previa, las claves secretas de los *Sector Trailer* [26][27][28][29]

Este hecho obligó a realizar mejoras en materia de seguridad en las tarjetas MIFARE, donde empezaron a incluir en sus modelos, además del Crypto1, algoritmos de encriptación estandarizados y ya probados como son el AES o el DES.

3.3.1.3 Distribución de la memoria MIFARE Classic

Existen tarjetas MIFARE Classic con capacidades de memoria que van desde 1Kb hasta los 4Kb. Todas ellas muestran una distribución estructurada de la memoria, dividida en bloques con características de acceso similares formando lo que se conocen como sectores. Para el caso de una tarjeta MIFARE Classic de 4Kb, la memoria está organizada en 32 sectores de 64 bytes divididos en 4 bloques de 16 bytes y 8 sectores de 256 bytes con 16 bloques de 16 bytes, tal como se refleja en la Figura 3-3.

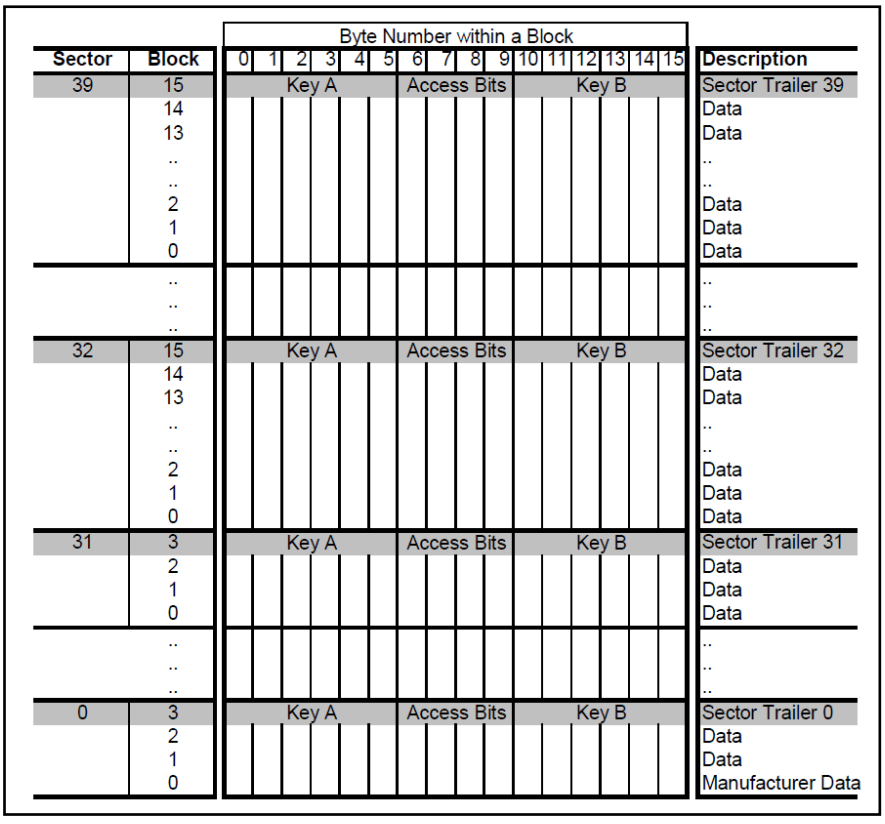


Figura 3-3 - Distribución de la memoria

Imagen obtenida de MF1S70yyX/V1

Cada sector cuenta con un bloque especial llamado *Sector Trailer* que está siempre situado en la última posición del sector (bloque 3 para los sectores del 0 al 30 y bloque 15 del sector 31 al 39) que almacena dos claves secretas (Key A y Key B) y las condiciones de acceso al sector.

Para el caso de la tarjeta MIFARE Classic de 1kB la distribución de su memoria corresponde con los 16 primeros sectores (del sector 0 al sector 15) de la tarjeta de 4kB.

3.3.1.3.1 Manufacturer Block

El primer bloque del primer sector (Bloque 0, Sector 0) de la memoria, denominado *Manufacturer Block*. El contenido de dicho bloque incluye el número de serie de la tarjeta, un byte de integridad y un conjunto de datos específicos del fabricante. Por motivos de seguridad este bloque se encuentra protegido contra escritura.

3.3.1.3.2 Data Block

La mayor parte de los sectores se componen de bloques de datos, el sector 0 contiene 2 bloques de datos, los sectores del 1 al 31 contienen 3 bloques y los sectores del 32 al 39, 15 bloques.

Con un tamaño de 16 bytes por bloque, los bloques de datos pueden ser configurados de dos formas: bloques de datos binarios y bloques de valor.

Los bloques binarios admiten operaciones de lectura y escritura autenticada, mientras que sobre los bloques valor que almacenan un valor numérico con un formato específico que habilita la detección y corrección de errores, se admite adicionalmente operaciones de, incremento, decremento, y operaciones atómicas.

3.3.1.3.3 Sector Trailer

El *Sector Trailer* es un tipo de bloque especial en el que se definen las condiciones de acceso a los bloques del sector y las claves necesarias para autenticar las diferentes operaciones. Tal y como se aprecia en la Figura 3-3, el bloque contiene la KeyA, las condiciones de acceso y la KeyB, en ese orden.

Ambas claves (Key A y Key B) se componen de 6 bytes y la KeyB es opcional pudiendo usarse como bytes de datos. La KeyA nunca podrá leerse, mientras que la KeyB por el contrario al ser opcional y poder usarse como datos podrá leerse siempre que se cumplan determinadas condiciones.

Las condiciones de acceso consisten en los 4 bytes restantes del bloque de los cuales solo los 3 primeros llevan los bits de acceso, mientras que el byte 9 puede utilizarse para almacenar datos. La combinación de los valores de estos 3 bits de acceso determina las operaciones (lectura, escritura, incremento, decremento, transferencia y restauración) que pueden realizarse sobre cada bloque y la clave (KeyA o KeyB) con la que pueden hacerlo.

3.4 MIFARE4MOBILE

El grupo industrial MIFARE4Mobile [9] está, compuesto por empresas líderes en sistemas NFC entre las que se incluyen Gemalto, Giesecke & Devrient, NXP Semiconductors, Oberthur Technologies y STMicroelectronics. Surge a raíz de dar respuesta a la necesidad de dotar de una solución interoperable para el uso de tarjetas de la familia MIFARE en entornos móviles NFC.

El elemento que permite asegurar la interoperatibilidad y sirve como referente único es la especificación técnica MIFARE4Mobile [30], con el objetivo de gestionar aplicaciones basadas en la tecnología MIFARE en dispositivos móviles. La especificación provee a operadores de telefonía móvil, gestores de servicios autorizados (TSM, *Trusted Service Manager*) y proveedores de servicios de una interfaz programable única y multi-operador para gestionar y abastecer de forma remota servicios basados en MIFARE en elementos de seguridad embebidos y tarjetas SIM en dispositivos móviles que implementen la tecnología NFC.

M4M gestiona las aplicaciones MIFARE mediante la creación y utilización de tarjetas virtuales (VC, *Virtual Cards*), donde carga estas aplicaciones, localizadas en una implementación MIFARE, un componente del elemento seguro del teléfono móvil.

Para el correcto funcionamiento de M4M debe asegurarse la existencia de un elemento seguro en el dispositivo móvil y el fabricante del mismo deberá incluir en él:

- Una implementación MIFARE con licencia en el elemento seguro donde el SO (Sistema Operativo) de la tarjeta provea del API compatible.
- Incluir instalada una aplicación Java Card que implemente M4M (el núcleo de M4M que consiste en el Gestor de Tarjetas Virtuales y el Gestor de Servicios).
- Habilitar dominios de seguridad de GP, donde se crearan las instancias de M4M.
- Asegurarse de que existe un modo de comunicación remoto entre el M4M instanciado y las entidades apropiadas que lo necesiten.

3.4.1 ARQUITECTURA DE MIFARE4MOBILE

Bajo la estructura de M4M nos encontramos con tres tipos de accesos a las tarjetas virtuales según la entidad que acceda: los Proveedores de Servicios, gestores de las aplicaciones MIFARE que ofrecen sus servicios al usuario, los TSM, sobre los que los proveedores de servicios delegan las responsabilidades de acceso a las aplicaciones alojadas en la tarjeta, y los usuarios finales, quienes disfrutan de los servicios y aplicaciones, sin tener acceso a su gestión directa.

[9] https://www.mifare4mobile.org/

La arquitectura de M4M recogida en el documento M4M_Arch [31], reflejada en la Figura 3-4, se divide en dos grandes bloques dentro del elemento seguro del dispositivo móvil: MIFARE4Mobile Framework y la implementación MIFARE.

Dada la complejidad y extensión de M4M únicamente entraremos en más detalle en los apartados que se relacionan directamente con el proyecto, pudiendo consultarse el resto de detalles en las documentaciones pertinentes que se irán especificando.

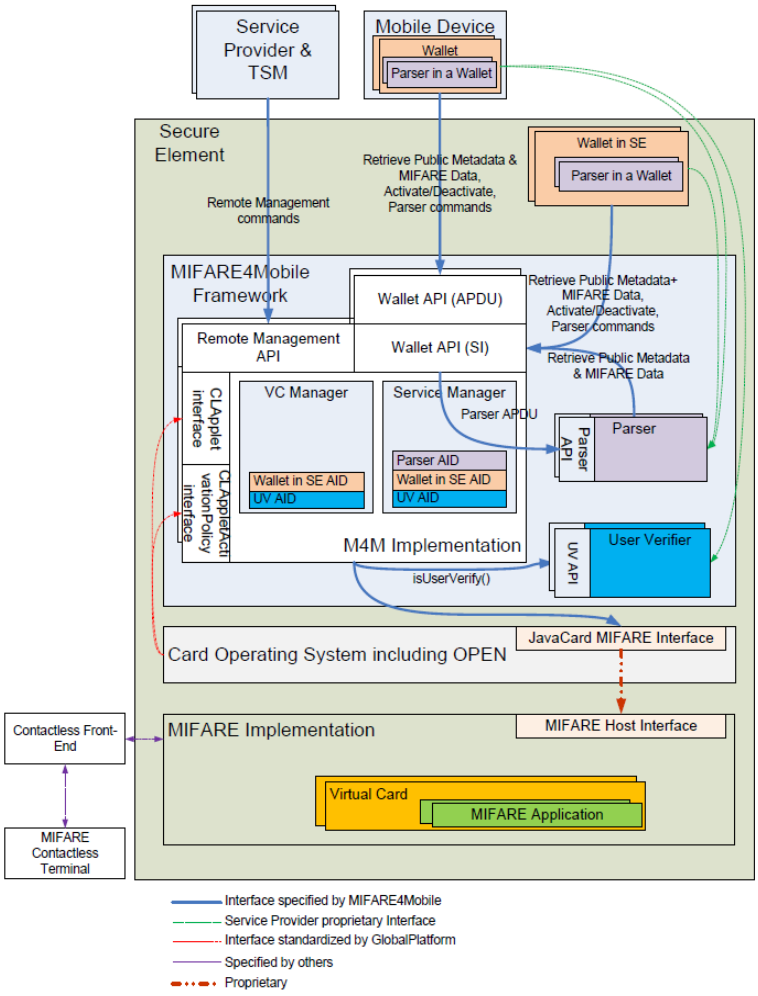


Figura 3-4 - Arquitectura MIFARE4Mobile

Imagen obtenida de M4M_Arch

3.4.1.1 Implementación MIFARE

La implementación MIFARE es un componente hardware y/o software que implementa la lógica de MIFARE y gestiona la memoria requerida para soportar una o varias ranuras para tarjetas virtuales donde estas podrán ser cargadas. La implementación MIFARE puede soportar las tecnologías MIFARE Classic y MIFARE DESFire.

Tal y como se muestra en la Figura 3-4, la implementación MIFARE cuenta con dos interfaces de comunicación:

- Interfaz propietaria: Encargada de comunicarse con el sistema operativo JavaCard, para gestionar los ciclos de vida de la tarjeta y los contenidos de las tarjetas virtuales; y de proveer al resto de las capacidades de la implementación MIFARE.
- Interfaz sin contactos: Vía de comunicación para el intercambio de mensajes de MIFARE entre la implementación MIFARE y la interfaz NFC.

Las ranuras para tarjetas virtuales son áreas de almacenamiento donde la implementación MIFARE puede cargar sus tarjetas virtuales. Cada ranura puede estar ocupada o sin ocupar por una única tarjeta virtual y, puesto que la implementación MIFARE permite tantas tarjetas virtuales como su memoria permita y las dimensiones de estas tarjetas pueden ser variables también es variable el número de ranuras para tarjetas virtuales.

Las tarjetas virtuales a todos los efectos pueden ser utilizadas como su correspondiente física y se caracteriza por:

- Tipo de tarjeta MIFARE, MIFARE Classic o MIFARE DESFire.
- Tamaño de la memoria pudiendo ser de 1, 2,4 u 8 kB.
- UID (número de serie de la tarjeta) que puede tener una longitud de 4, 7 o 10 bytes. El UID puede ser heredada del elemento seguro o establecida por el proveedor tecnológico de la implementación MIFARE. Ante este hecho nos encontramos que las tarjetas virtuales pueden tener todas el mismo UID, distintos UIDs o algunas tenerlo igual y otras distinto.
- Parámetros del protocolo sin contactos, utilizados en la norma ISO14443.
- Entrada VC que identifica de forma única una tarjeta virtual dentro del elemento seguro.

Estas características son asignadas durante la creación de la tarjeta virtual, que puede realizarse durante la producción del elemento seguro o en cualquier momento que este esté en circulación. La tarjeta virtual puede ser creada o cargada en una ranura para tarjetas virtuales tan pronto como la implementación MIFARE tenga el suficiente espacio disponible.

Una aplicación MIFARE es una colección de datos, claves y condiciones de acceso cargadas en una tarjeta virtual, llamada tarjeta virtual asociada. La aplicación pertenece y es gestionada por un proveedor de servicios y será cargada en una y solo una tarjeta virtual aunque una misma tarjeta virtual pueda tener cargadas multitud de aplicaciones.

Los recursos utilizados por una aplicación MIFARE dependen del tipo de su tarjeta virtual asociada, MIFARE Classic o MIFARE DESFire. Así una aplicación MIFARE asociada a una tarjeta virtual MIFARE Classic consiste en uno o varios sectores de la tarjeta asignados por el propietario de la tarjeta virtual. Por su parte, una asociada a una tarjeta virtual MIFARE DESFire estará compuesta por una o varias aplicaciones MIFARE DESFire creadas por el propietarios de la tarjeta virtual.

3.4.1.2 MIFARE4MOBILE FRAMEWORK

El MIFARE4Mobile *Framework* se compone de varias aplicaciones Java Card, corriendo en el sistema operativo dentro del elemento seguro, que se encargan de la gestión de las tarjetas virtuales y las aplicaciones MIFARE.

Como se puede observar en la Figura 3-4, M4M *Framework*, recogido en M4M_Arch, se divide en varios componentes siendo los más importantes para nosotros:

- Gestor de Tarjetas Virtuales (*VC Manager*): Es una aplicación Java Card que contiene la representación lógica de una tarjeta virtual, implementada físicamente en la implementación MIFARE. El Gestor de Tarjetas Inteligentes es una aplicación sin contactos definida en la especificación de servicios sin contactos de GlobalPlatform. El Gestor de Tarjetas Virtuales únicamente gestionara una tarjeta virtual, del mismo modo que una tarjeta virtual solo puede ser gestionada por un único Gestor de Tarjetas Virtuales.
- Verificador de usuario (*User Verifier*): Es una aplicación Java Card opcional que permite a un proveedor de servicios o al emisor del elemento seguro definir un método para verificar la identidad del usuario final. Su uso restringe la activación de una tarjeta virtual o la obtención de datos de aplicación.

Además el componente *VC Manager* tiene acceso al API de Gestión Remota (*Remote Management API*), una interfaz dedicada a la gestión remota de las aplicaciones MIFARE y las tarjetas virtuales.

3.4.2 GESTIÓN REMOTA DE MIFARE4MOBILE

La gestión remota de M4M provee a los Proveedores de Servicio y los TSM de:

- Funciones para gestionar de forma remota el contenido de las tarjetas virtuales y las aplicaciones MIFARE.
- Personalización y actualización de los metadatos, tanto privados como públicos, de un *VC Manager*.
- La gestión de los datos, claves y condiciones de acceso en la interfaz sin contactos de las aplicaciones MIFARE.

Una sesión de gestión remota de un *VC Manager* o es iniciada por el propietario del dominio de seguridad (SDM, *Security Domain Manager*), que establece un canal seguro para comunicarse con el dominio de seguridad (SD, *Security Domain*) correspondiente al *VC Manager* objetivo y servirá de vínculo entre este y el SDM.

El inicio de la sesión tiene como objetivo indicar la aplicación objetivo, donde irán destinados los comandos de GlobalPlatform. En estos comandos irán envueltos los comandos

M4M (en el comando los comandos y en su respuesta las respuestas con la palabra de estado). Pueden ir envueltos uno o varios comandos M4M en un solo comando .

En el caso de una aplicación MIFARE, la gestión remota permite la personalización, actualización y lectura de datos de la aplicación MIFARE propiedad de un *Service Manager*. Las condiciones de acceso también pueden leerse con un acceso mediante gestión remota.

Una actualización mediante gestión remota puede verse interrumpida por una pérdida de tensión en el elemento seguro, por una sesión sin contactos que haya sido iniciada mientras se realizaba la actualización o por la ejecución del comando para actualizar ha fallado.

Frente a una pérdida de tensión habría que esperar a que el elemento seguro recobrara la tensión y que la situación de los componentes de M4M volviera a restablecerse al estado en el que se encontraban cuando se inició la actualización e iniciar una nueva sesión de gestión remota.

En el caso de una sesión sin contactos activa durante la actualización, el *Service Manager* se encargara de rechazar todos los comandos MIFARE por lo que la sesión de actualización tendrá que generarse nuevamente una vez termine la sesión sin contactos.

La gestión remota de los datos de una tarjeta virtual MIFARE Classic se lleva a cabo mediante tres comandos:

- Lectura de un sector: Con el comando se pueden leer uno, varios o todos los bloques de un sector concreto incluso el *Sector Trailer* (las claves por seguridad no se pueden leer y aparecen como ceros).
- Actualización de un sector: Con el comando pueden actualizarse los valores de uno, varios o todos los bloques de un sector específico.
- Reinicio de un sector: Con el comando puede reiniciarse uno, varios o todos los sectores. El reinicio de un sector supone poner a cero los bloques de datos, salvo el bloque 0 del sector 0 que no sufre ninguna modificación, y el *Sector Trailer* vuelve a los valores que se pasaron como parámetro al *VC Manager* para la creación de la tarjeta virtual.

La gestión remota que vamos a aplicar en nuestro proyecto se basa en estos comandos por lo que serán ampliados en la sección 4.4.2.

ESPECIFICACIÓN 4

En este capítulo recogeremos las especificaciones que hemos tenido que desarrollar, partiendo de lo que hemos explicado en el capítulo anterior y que hemos utilizado para la consecución de nuestros objetivos. Describiremos, de forma general, las especificaciones en las que nos hemos basado y, en detalle, las modificaciones que hemos tenido que realizar para suplir las carencias que estas tenían. También expondremos los comandos propios que hemos creado por necesidades puntuales del proyecto.

Para la realización del proyecto contaremos con una tarjeta inteligente con interfaz dual, es decir, una tarjeta inteligente que cuenta tanto con una interfaz con contactos como con una sin contactos. Mediante la interfaz con contactos accedemos a la implementación de GlobalPlatform que incluye el sistema operativo Java Card. La interfaz sin contactos nos permite acceder a una sección de memoria del tipo MIFARE Classic de la forma tradicional aparte de poder acceder a la implementación de GlobalPlatform. El acceso a la memoria MIFARE Classic desde la parte sin contactos no sería de la forma tradicional sino que necesitaríamos hacer uso de Java Card.

La memoria MIFARE Classic la utilizaremos como la base de una tarjeta virtual que hemos generado y es sobre la que realizaremos las pruebas, simularemos sus accesos desde nuestra aplicación y comprobaremos como su comportamiento como tarjeta virtual es el mismo que sobre una tarjeta real.

Para comenzar expondremos brevemente las necesidades que nos han surgido de cara a la especificación del problema y analizaremos hasta qué punto nos sirven las especificaciones ya existentes y que puntos tendremos que mejorar a partir de ahí.

Seguiremos con una breve explicación de las clases y métodos de Java Card, haciendo especial énfasis en aquellos más específicos de esta versión de Java y que más han afectado a nuestro proyecto.

Continuaremos con los comandos específicos de GlobalPlatform, incluidos como una librería propia en Java Card. De estos comandos únicamente utilizaremos dentro de nuestra aplicación los referentes a los canales seguros.

Expondremos los comandos de M4M, que serán la base de nuestros comandos APDU. El uso de estos comandos vendrá determinado según quien acceda a la aplicación, mediante distintos niveles de seguridad que explicaremos más adelante, y las opciones que tenga disponibles. Pondremos especial atención a los comandos propios de la tarjeta MIFARE Classic y a los comandos de gestión remota.

Por último detallaremos los comandos que son únicos para este proyecto, tanto aquellos que, basados en comandos ya existentes, hayamos tenido que modificar como aquellos que se hayan tenido que crear especialmente para nuestra aplicación.

4.1 ASPECTOS GENERALES DE LA ESPECIFICACIÓN

Como ya hemos comentado anteriormente el objetivo que estamos persiguiendo es, teniendo una tarjeta inteligente física, emular de forma virtual varias tarjetas del tipo MIFARE Classic, manteniendo la seguridad e independencia de los datos entre ellas.

De cara a la creación, mantenimiento y acceso a dichas tarjetas virtuales nos basaremos en la estructura de MIFARE4Mobile y utilizaremos Java Card para implementar nuestro código.

Para mantener las tarjetas virtuales utilizaremos la memoria MIFARE Classic para comprobar, sobre una tarjeta real, que la implementación virtual se realiza con éxito.

A la vista de lo anteriormente expuesto necesitamos crear una aplicación que cumpla con los siguientes requisitos:

1. La aplicación debe reservar espacio de memoria suficiente para alojar todas las tarjetas virtuales que especifique.
2. La aplicación deberá tener un control preciso de todos los movimientos en la memoria para asegurar la confidencialidad de los datos de las tarjetas virtuales.
3. Por seguridad el intercambio de datos entre el dispositivo externo y la aplicación debe estar cifrado con un método seguro.
4. Dado el uso de claves privadas y distintos grados de uso y pertenencia, tanto de la tarjeta inteligente como de las distintas tarjetas virtuales, designaremos tres niveles distintos de seguridad, a saber: nivel de emisor, de gestor y de usuario.
5. Aunque puedan estar adaptados los comandos y respuestas APDU deberán estar lo más estandarizados posible según las distintas especificaciones de referencia.
6. Las tarjetas virtuales generadas deberán reflejar el comportamiento y estructura de su homóloga real.

El primero de los requisitos depende de la tarjeta física de que dispongamos y nos permitirá disponer de más o menos tarjetas virtuales según su capacidad. Para nuestro proyecto hemos dispuesto un límite de cuatro tarjetas virtuales MIFARE Classic de 1kB que en la práctica mapearemos sobre una tarjeta real MIFARE Classic de 4kB.

Para el segundo nos hemos basado en un sistema de *offsets* que limita los accesos a la tarjeta virtual seleccionada, de forma que nunca se pueda exceder su rango.

Respecto al tercer requisito haremos que todos los comandos y respuestas APDU vayan cifrados mediante procedimientos ya testeados que nos aseguran su eficiencia.

Los tres niveles de seguridad a los cuales hace referencia el cuarto requisito permiten reconocer quien se comunica con la tarjeta, manteniendo la confidencialidad de los datos, las claves y la integridad de las tarjetas virtuales. Los tres niveles de seguridad disponen de distintos accesos y acciones dentro de la tarjeta virtual. Los diferentes niveles de seguridad son:

- Nivel de seguridad de emisor: es el fabricante de la tarjeta física. Se encarga de la creación y eliminación de las tarjetas virtuales y utiliza las claves del dominio de seguridad de emisor. No tiene acceso a la gestión de las tarjetas virtuales una vez creadas ni a los datos que estas contienen.
- Nivel de seguridad de gestor: es el propietario de la tarjeta virtual. No tiene capacidad de crear las tarjetas virtuales pero si de gestionarla aunque únicamente aquellas de las que sea propietario. Se encarga de establecer las condiciones de acceso de los bloques y las claves de acceso a los mismos. También dispone de acceso a los datos contenidos en la tarjeta virtual.
- Nivel de seguridad de usuario: es el usuario final y el propietario de la tarjeta física. Únicamente tiene acceso a los datos de las tarjetas virtuales según se lo permitan las condiciones de acceso de cada una.

Los dos últimos requisitos se basaran en el uso lo más cercano posible de las especificaciones de GlobalPlatform y MIFARE4Mobile para el quinto y las de MIFARE Classic para el sexto.

En los siguientes apartados veremos hasta qué punto las especificaciones que utilizamos como base nos sirven para cumplir estos requisitos y cuales hemos tenido que modificar o crear.

4.2 JAVA CARD

Java Card es la herramienta principal para la creación de nuestra aplicación. Nos proporciona todas las herramientas para trabajar específicamente en tarjetas inteligentes, entre ellas la librería específica de GlobalPlatform.

En Java Card tenemos que destacar el uso de cuatro paquetes en particular: `framework`, `external`, `security` y `crypto`.

El paquete `framework` es el que contiene lo referente a las APDU, las aplicaciones y lo recogido en la parte 4 de la norma ISO7816. Es el paquete que nos permite gestionar los flujos

de entrada y salida, así como su formato, con las APDU, gestiona la instalación de la aplicación dentro de la tarjeta y contiene los valores de las constantes estandarizadas.

El paquete es el que contiene la interfaz de comunicación para acceder a segmentos de memoria que no son directamente direccionables por el entorno de ejecución Java Card, es decir, sirve para acceder desde la aplicación a la memoria MIFARE Classic y poder leer y escribir en ella.

Los paquetes y son los que contienen las herramientas que necesitaremos para cifrar todos nuestros datos. Se utiliza desde la generación de la clave MF_Password, como veremos en el apartado 5.1 - Aproximación a MIFARE Classic desde JavaCard, hasta el cifrado de los comandos y respuestas APDU.

Utilizaremos la versión 2.2.2 de Java Card para el desarrollo de la aplicación.

4.3 GlobalPlatform

GlobalPlatform será la base que nos ayude a gestionar internamente la instalación, ejecución y comunicación de las aplicaciones dentro de la tarjeta inteligente. Se encargara de proporcionarnos seguridad entre la tarjeta inteligente y el usuario y entre las distintas aplicaciones existentes dentro de la tarjeta.

En nuestra aplicación únicamente utilizaremos directamente de la especificación de GlobalPlatform lo referente al establecimiento y mantenimiento de los canales seguros, que realizaremos entre GPShell como entidad externa y la aplicación que hemos creado.

Crearemos tres canales seguros independientes, uno por cada nivel de seguridad, donde la principal diferencia será la gestión de dichos canales y las claves de cifrado.

Para la creación y gestión de los canales seguros utilizaremos la API de GlobalPlatform disponible para Java Card aunque únicamente en el nivel de seguridad de emisor, se utilizara la API de serie debido a que es en el único nivel de seguridad en el que se usan las claves y el dominio de seguridad que la tarjeta trae de serie.

La creación de los canales seguros se realizara tal y como se recoge en la especificación de GlobalPlatform. Para el caso del dominio de seguridad del gestor se implementara manualmente una versión modificada por nosotros y para el nivel de seguridad de usuario se utilizara otro tipo de canal seguro, desarrollado por nosotros. La implementación de los canales seguros se desarrollará en el apartado 5.3-Desarrollo de la seguridad en el acceso.

4.4 MIFARE4MOBILE

Como ya hemos comentado intentaremos emular parte de la finalidad que persigue esta tecnología por lo que adaptaremos nuestros comandos y códigos a los especificados por M4M.

En este capítulo expondremos las limitaciones que nos presenta M4M y describiremos brevemente los comandos que hemos utilizado de M4M. Los comandos de M4M se pueden encontrar en sus especificaciones M4M_Classic_HI [32], para la tarjeta MIFARE Classic y M4M_RM_API [33] para el acceso remoto.

4.4.1 LIMITACIONES DE MIFARE4MOBILE

MIFARE4Mobile se trata de una tecnología muy completa cuyo objetivo es adaptar distintos tipos de tarjetas virtuales de la familia MIFARE en una tarjeta que se encuentre dentro de un dispositivo móvil pero mediante una serie de comandos e instrucciones propietarias que no reflejan fielmente el comportamiento de la tarjeta real. En nuestro proyecto nos centramos únicamente en tarjetas MIFARE Classic donde intentamos emular de forma real su comportamiento al mismo tiempo que aumentamos las medidas de seguridad en la comunicación.

La diferencia más significativa entre el comportamiento real y el comportamiento en M4M en las tarjetas MIFARE Classic la encontramos con la utilización de las condiciones de acceso. El acceso de M4M a los bloques de un sector se realiza mediante el uso de una clave denominada MF_PASSWORD específica de cada sector, detallada en la especificación AN02105 [34], que se deriva de las claves KeyA y KeyB del sector. Con este tipo de acceso M4M ignora todas las medidas de control de los sectores, tanto las condiciones de acceso como la matriz de condiciones de acceso descrita en AN02105. La falta de este control puede generar problemas en bloques cuya información pueda ser sensible ya que, aunque se puede bloquear contra escritura con las condiciones de acceso, al no hacer uso de estas, dicha información podría ser alterada.

Con el fin de solventar esta peculiaridad hemos incluido el procesamiento de las condiciones de acceso para cada lectura y escritura de un bloque. Por defecto esta funcionalidad viene desactivada y debe activarla el administrador de la aplicación a través de los parámetros de instalación que se incluyen durante la instalación de la aplicación en la tarjeta. La activación o desactivación de dicha funcionalidad es permanente, es decir, una vez seleccionado esta decisión se mantiene durante la vida de la aplicación y únicamente se puede alterar desinstalando y volviendo a instalar la aplicación.

Otra diferencia sería el acceso a la información del *Sector Trailer*. MIFARE Classic especifica que al intentar leer el *Sector Trailer* la KeyA siempre se devuelve como ceros pero, según la configuración de las condiciones de acceso, la KeyB puede no ser necesaria para la gestión de los accesos a los bloques del sector y por tanto podría ser utilizada como espacio para datos.

En MIFARE4Mobile esta opción no está disponible. Según la especificación de MIFARE4Mobile al realizar una lectura de un *Sector Trailer*, únicamente devolverá en claro el valor de las condiciones de acceso, las dos claves KeyA y KeyB se devolverán como ceros. La solución que hemos planteado para resolver este problema es activar las condiciones de acceso, para determinar la necesidad o no de una KeyB, y, si estas permiten leer la KeyB, la lectura del *Sector Trailer* devolverá la KeyA como ceros y las condiciones de acceso y la KeyB con su valor.

La última limitación que nos encontramos es en el campo de la seguridad. M4M cuenta con distintos comandos y herramientas según quien acceda a la aplicación. En el caso de los gestores de las tarjetas lo hacen a través de los comandos de gestión remota que si se cifran de extremo a extremo pero, cuando el acceso lo hacen los usuarios la información va en claro y por tanto una tercera persona podría acceder a ella.

Nuestra solución ha sido cifrar toda la comunicación que haya hacia y desde la tarjeta. Para este cifrado utilizaremos distintos métodos según cómo accedamos a la tarjeta. Dicho métodos se explicaran en detalle en el apartado 5.3-Desarrollo de la seguridad en el acceso.

4.4.2 Comandos de MIFARE4Mobile para MIFARE Classic

Los comandos M4M diseñados para MIFARE Classic, a los que denominaremos comandos locales, forman una pareja comando-respuesta donde únicamente se efectúa una instrucción por pareja, es decir, únicamente tienes un sector objetivo.

Los comandos locales son los que se utilizan cuando se tiene acceso físico a la tarjeta y por tanto se considera que se dispone del suficiente tiempo y recursos para operar sobre ella. Debido a esto serán utilizados por el nivel de seguridad de usuario ya que es el que dispone de acceso físico a la tarjeta.

El nivel de seguridad de gestor utilizara los comandos de gestión remota, que explicaremos más adelante y el nivel de seguridad de emisor utilizara comandos específicos para tarjetas virtuales que detallaremos en el apartado 4.5.1-Comandos del nivel de seguridad de emisor.

Todos los comandos vienen detallados en el anexo 7.1-Comandos APDU.

4.4.2.1 Comando readSector

El comando se utiliza para leer uno o varios bloques no necesariamente consecutivos de un sector especificado.

En caso de que entre esta lectura se incluya el *Sector Trailer* se devolverá, para esta, 16 bytes con los bytes del 0 al 5 y del 10 al 15, que son los correspondientes a las claves KeyA y KeyB respectivamente, puesto a cero.

Los bloques que se van a leer vendrán determinados por la combinación del número de sector a leer y un BlockBitMap que determinará los bloques dentro de dicho sector. Los bloques definidos por el BlockBitMap no podrán exceder los límites del sector (no se podrá leer un bloque 4 en un sector que solo tenga bloques del 0 al 3) y por tanto para sectores de 4 bloques el valor del BlockBitMap siempre será igual o inferior a 0x000F y si se detecta que es mayor se devolverá un error de parámetro incorrecto.

4.4.2.2 Comando writeSector

El comando se utiliza para escribir uno o varios bloques no necesariamente consecutivos de un sector especificado.

La estructura del comando es igual a la del comando salvo porque añade además un campo con los datos a escribir.

La escritura de los bloques es una escritura no atómica, es decir, si durante la escritura hay una interrupción como una desconexión o una pérdida de corriente algunos bloques pueden haber sido actualizados pero no necesariamente haberlo sido todos.

4.4.3 Gestión Remota

La gestión remota se reserva para que los gestores de aplicaciones accedan a la tarjeta sin necesidad de estar en contacto con esta. Gracias a la gestión remota se consigue actualizar las aplicaciones con un gran nivel de seguridad y con una gestión más eficiente de comandos al poder mandar más de una instrucción en un único comando APDU reduciendo la sobrecarga en los comandos y respuestas y el tiempo en el canal.

Para el uso de los comandos de gestión remota se requiere establecer un canal seguro con una seguridad mínima con unas claves que nos asegure cifrado tanto en comando como respuesta, claves que se derivan de las claves de sesión. Gracias a estas medidas nos aseguramos que únicamente los gestores de la aplicación puedan recurrir a estos comandos.

4.4.3.1 Comandos Gestión Remota

Los comandos en Gestión Remota tienen la particularidad de estar encapsulados dentro de un comando STORE DATA de GlobalPlatform lo que supone una estructura más compleja que la de un comando APDU normal.

La Figura 4-1 expone de manera gráfica este formato de encapsulamiento de los comandos de Gestión Remota.

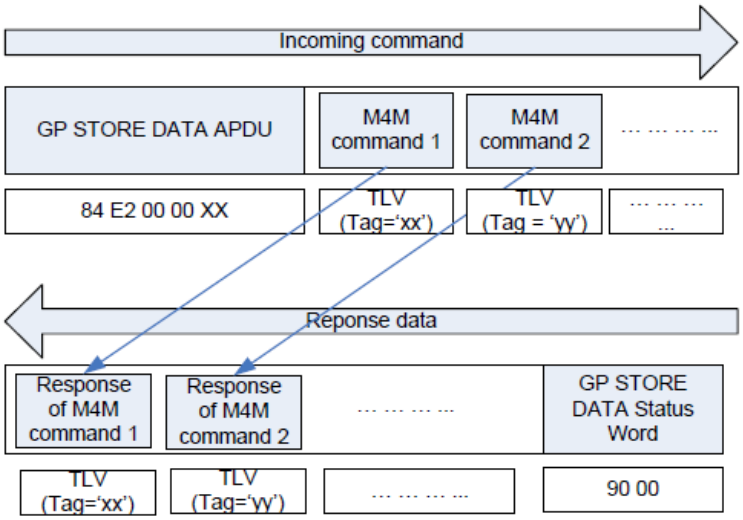


Figura 4-1 - Formato de los comandos de Gestión Remota

Imagen obtenida de M4M_RM_API

Los comandos APDU se envían con la cabecera estipulada para el comando STORE DATA de GlobalPlatform:

- CLA = "0x84": código de CLA para los mensajes APDU con seguridad.
- INS = "0xE2": código de la instrucción STORE DATA especificada por GlobalPlatform.
- P1 y P2 = "0x00": puesto que los comandos irán encapsulados en el campo datos, el campo de ambos parámetros se mantiene a cero.
- Lc: Longitud del campo de datos y por tanto variable.

Cada uno de los comandos de Gestión Remota se envían concatenados en el campo de datos codificados en formato TLV (Tag Length Value) consistentes en:

- Tag (1 byte): código del comando.
- Length (1 byte): longitud del campo de datos.
- Value (variable): contiene los datos del comando. Este campo es de longitud variable y depende del comando que se esté enviando.

Las respuestas APDU también se reciben con todos los comandos de Gestión Remota concatenados, con las respuestas correspondientes a cada comando en el mismo orden en el que se enviaron estos y codificados también en formato TLV de la forma:

- Tag (1 byte): código del comando al que responde.
- Length (1 byte): longitud del campo de datos.
- Value (variable): se divide en dos partes concatenadas, los primeros dos bytes corresponden a la palabra de estado de la respuesta y los bytes restantes a los bytes de datos de la respuesta, si los hubiera.

Tras las respuestas de los comandos de Gestión Remota se encuentra la palabra de estado de la respuesta del comando STORE DATA de GlobalPlatform que, a menos que haya habido un error en la recepción o el procesado de la APDU comando, tomará el valor "0x9000".

Existen tres comandos de gestión remota: Read MIFARE Classic Sector y Update MIFARE Classic Sector cuyas funcionalidades y datos son iguales que las de los comandos locales y y

también incluye el comando Reset MIFARE Classic Sector que formatea un sector y lo devuelve al valor original con el que se creó al instalar la tarjeta virtual.

4.5 COMANDOS PROPIOS

A pesar de haber intentado adaptar formatos y protocolos ya conocidos y utilizados universalmente, nos hemos visto en la obligación de diseñar algunos comandos y situaciones que no están recogidas en ninguno de los casos anteriores.

En todos los casos no existía una encriptación de las respuestas APDU que hemos tenido que añadir y por lo tanto, el dispositivo externo que vaya a comunicarse con nuestra aplicación deberá contar con los medios para descifrar la respuesta.

En cuanto a los comandos, en el casos del nivel se seguridad de emisor, hemos tenido que crear los comandos de creación de las tarjetas virtuales ya que los que M4M nos facilitaba [35] eran mucho más complejos de lo que necesitábamos.

Al poder crear tarjetas virtuales también hemos tenido que crear unos comandos que nos permitan seleccionar una u otra tarjeta virtual así como dejar de seleccionarla una vez hayamos dejado de trabajar con ella.

En el nivel de seguridad de usuario para poder cifrar los datos y hacerlo de forma sencilla para el usuario contamos con un número de identificación personal (PIN) que nos servirá de base para cifrar y descifrar los mensajes APDU. Puesto que solo nos basamos en un número PIN de ocho bytes para cifrar, en lugar de las 3 claves de 16 bytes que se utilizan en los canales seguros de GlobalPlatform, utilizaremos un método de cifrado distinto al que se utiliza en los otros dos niveles de seguridad y se tendrá que tener en cuenta en el dispositivo externo que se conecte. En este método de cifrado utilizaremos el número PIN como base para las claves de cifrado.

Al igual que en los comandos heredados de M4M, los comandos detallados se pueden encontrar en el anexo 7.1-Comandos APDU

4.5.1 COMANDOS DEL NIVEL DE SEGURIDAD DE EMISOR

Como ya hemos comentado M4M sí que dispone de comandos de creación y eliminación de tarjetas virtuales, que en nuestro caso estará limitado a tarjetas MIFARE Classic de 1kb. En cuanto al establecimiento de la comunicación se hará mediante el programa GPSHELL.

Teniendo esto en cuenta no tenemos la necesidad de usar los comandos tan complejos de M4M y podemos crear comandos más simples y sencillos para nuestro caso concreto.

En primer lugar, se define un comando de creación e instalación de una tarjeta virtual. Para este proyecto, se reserva el espacio equivalente a cuatro tarjetas virtuales.

Los espacios de memoria se asignan en orden ascendente, seleccionando el primer hueco disponible. En caso de no tener espacio libre se reportará el consiguiente error.

Como vamos a utilizar el espacio de memoria de una tarjeta MIFARE Classic de 4kB, las tarjetas virtuales que crearemos las mapearemos en esa memoria, pudiendo almacenar hasta 4 tarjeta virtuales independientes del tipo MIFARE Classic de 1kB.

Para la creación de una tarjeta virtual necesitaremos suministrar los bloques *Sector Trailer* de cada uno de los sectores. Esto es necesario de cara a los comandos de reset de los sectores donde se requiere restaurar el valor del *sector trailer* al que se facilita durante la instalación. También es necesario suministrar el *manufacturer block* del sector 0 y las tres claves necesarias para el establecimiento del canal seguro.

Como todos los datos requeridos ocupan más de una APDU, cada comando de creación e instalación de tarjeta virtual se compondrá de dos APDUs comando, con sus respectivas respuestas. El primer comando APDU estará formado por el *manufacturer block*, el *Sector Trailer* del sector 0 y las tres claves de canal seguro, en ese orden. El segundo comando contendrá los *Sector Trailer* de los 15 sectores restantes y su respuesta incluirá, aparte de la palabra de estado, un byte que indica el hueco de tarjeta virtual utilizado. Una vez se reciba la última respuesta se considerara la tarjeta virtual creada y el hueco para tarjeta virtual que se le ha asignado, ocupado.

Del mismo modo que se define un comando de creación de tarjetas virtuales, se debe implementar su opuesto, de forma que se habilite la liberación de la memoria de aquellas tarjetas que ya no estén en uso.

El objetivo de este comando es designar un hueco que este ocupado por una tarjeta virtual y desvincular la tarjeta virtual asociada después de realizar un reset de todos los sectores. La información de dicha tarjeta (el *manufacturer block*, los *Sector Trailer* y las claves de nivel de gestor y usuario) permanece en la memoria interna de la tarjeta pero al eliminar la asociación ya no puede accederse a ella. Al instalarse una nueva tarjeta virtual en dicho hueco la información existente se sobrescribiría.

4.5.2 Comandos del nivel de seguridad de usuario

En el nivel de usuario nos encontramos con la necesidad de verificar la identidad del usuario, además de garantizar la seguridad de los mensajes mediante cifrado. Para ello utilizaremos una encriptación DES con una clave de 8 bytes, clave que utilizaremos como un número PIN para identificar al usuario.

Por lo tanto el primer paso que se tiene que realizar durante este nivel de seguridad es validar el PIN. Esto lo haremos mediante un comando propio, el comando de verificación del PIN.

Para garantizar la seguridad de la clave y que esta no viaje en claro, el propio comando de verificación ira cifrado con la clave que se suministra en el campo de datos. De esta forma el

dominio de seguridad de la tarjeta virtual deberá descifrar el mensaje y comprobar que el PIN facilitado es el que él tiene almacenado.

Una vez que se devuelva la respuesta de este comando sin error, todos los comandos posteriores durante esa sesión serán cifrados con esa misma clave.

Puesto que disponemos de un número único de 8 bytes que nos sirve para identificar al usuario y cifrar los datos, se hace necesario que dicho número pueda cambiar su valor, a discreción del propio usuario y siempre que esté autenticado. Para ello diseñamos otro comando para modificar esta contraseña.

El comando de cambio de PIN requerirá mandar las dos claves, el PIN antiguo y el PIN nuevo, por motivos de seguridad. Una vez se reciba la respuesta de que el comando se ha procesado correctamente, todos los comandos y respuestas desde ese momento hasta que se vuelva a modificar el PIN irán cifrados con la nueva clave, incluso para las futuras sesiones del nivel de seguridad de usuario.

Ambos comandos se encuentran detallados en el anexo 7.1.3.3 - Comandos del nivel de seguridad de usuario.

5 DESARROLLO

Este capítulo está dedicado al desarrollo de la aplicación en sí con la que trataremos de cumplir nuestro objetivo utilizando todas las herramientas expuestas en los anteriores capítulos. En el iremos explicando paso a paso el camino que hemos seguido desde la toma de contacto hasta la finalización de la aplicación completa.

Para el desarrollo del proyecto emplearemos una tarjeta inteligente de interfaz dual, interfaz con contactos e interfaz sin contactos, donde contaremos con una implementación de GlobalPlatform y Java Card en la parte con contactos y una tarjeta MIFARE Classic de 4kB en la interfaz sin contactos. Nuestra tarjeta inteligente tiene las dimensiones de una tarjeta SIM y a pesar de tener interfaz sin contactos, no cuenta con antena incorporada, por lo que nos es imposible acceder a ella de otra forma que no sea a través de sus contactos.

Comentaremos también los problemas que nos han ido surgiendo al ir implementando el caso real y como dista del caso ideal, indicando en cada caso las soluciones que hemos escogido para solventarlo. Expondremos también las mejoras que hemos llevado a cabo y que no estaban recogidas en el caso ideal.

Comenzaremos con una primera toma de contacto donde nos familiarizamos con la tarjeta y las aplicaciones, como se crean y cargan las aplicaciones en la tarjeta y como se gestionan las APDUs en una aplicación. También utilizamos esta primera toma de contacto para desarrollar el código correspondiente al cálculo de la clave MF_PASSWORD así como las primeras lecturas y escrituras sobre la tarjeta MIFARE Classic.

Una vez conocemos el funcionamiento de las aplicaciones, la gestión de las APDUs y tenemos acceso a la tarjeta MIFARE Classic procedemos con la implementación de nuestros comandos derivados de MIFARE4Mobile y su gestión. De esta forma tendremos los comandos básicos para crear una tarjeta virtual y acceder a ella.

Una vez tenemos la estructura de la aplicación lista pasaremos a la seguridad con la implantación de los tres niveles de seguridad. Activaremos los canales seguros (de forma automática en el nivel de emisor y manual en el nivel de gestor) y la verificación del código PIN

del usuario. En este punto toda la información con la que trabajemos será considerada cifrada de cara al intercambio de APDUs.

Por último añadiremos los últimos detalles a la aplicación y comentaremos los resultados de las pruebas de forma general. Los workbenchs utilizados para las pruebas se pueden consultar en el capitulo 7.2-Workbenchs.

5.1 APROXIMACIÓN A MIFARE CLASSIC DESDE JAVACARD

Para completar los objetivos planteados por el proyecto comenzaremos con una primera toma de contacto con la tarjeta inteligente que consistirá en la creación de una aplicación sencilla que nos permita recibir una APDU y responderla. Este primer paso nos servirá para aprender la estructura básica que debe tener una aplicación, como compilarla y obtener los archivos a cargar e instalar en la tarjeta en la tarjeta. Para la gestión de todo este procedimiento se emplea el programa GPShell.

Para visualizar mejor esta toma de contacto utilizaremos el código Java Card, Figura 5-1, y el sript de GPShell, Figura 5-2, presentados a continuación. Solo destacaremos la estructura básica de la aplicación y los comandos más básicos de la comunicación con la tarjeta desde GPShell.

```
   public class VMF extends Applet {

     /* constructores */
     MemoryAccess oMemAccess;
     static DESKey deskey; //clave DES para el calculo de la MF_PASSWORD
     static Cipher cipherCBC; //cifrador para el calculo de la MF_PASSWORD

     public static void install(byte[] bArray, short bOffset, byte bLength) {
            new VMF(bArray, bOffset, bLength);
     }
     protected VMF(byte[] bArray, short bOffset, byte bLength) {
            deskey=        (DESKey)        KeyBuilder.buildKey(KeyBuilder.TYPE_DES,
KeyBuilder.LENGTH_DES3_2KEY, false);
            cipherCBC= Cipher.getInstance(Cipher.ALG_DES_CBC_ISO9797_M2, false);
            oMemAccess=Memory.getMemoryAccessInstance(Memory.MEMORY_TYPE_MIFARE,
null, (short) 0);
            register();
     }
   public void process (APDU apdu) {
   if (buffer[ISO7816.OFFSET_CLA]!=CLA_code){
                  ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
            }
   switch (buffer[ISO7816.OFFSET_INS]) {
                  case INS_read:       read(apdu);
                                       break;
```

```
                    case INS_write:       write(apdu);
                                          break;
                    default:
        ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
                }
    }
    private void read(APDU apdu) {
                byte[] buffer= apdu.getBuffer();
                short byteRead=e destino
    (short) 0, // offset en el buf apdu.setIncomingAndReceive();
                oMemAccess.readData(
    bufferLectura, // buffer d
    fer de destino
    arrayPWD, // array que contiene la MF_PASSWORD
    (short)0, // offset en el array  de password
    (short) arrayPWD.length, // longitud en bytes de la clave
    (short) sector, // sector objetivo
    bloque); // bloque objetivo
                Util.arrayCopy(bufferLectura, (short) 0, buffer, (short) 0, (short)
16);
                apdu.setOutgoingAndSend((short) 0, (short) 16);
        }
    private void write(APDU apdu) {
                byte[] buffer= apdu.getBuffer();
                short byteRead= apdu.setIncomingAndReceive();
                oMemAccess.writeData(
    datosAEscribir, // array con los datos a escribir
    (short) 0, // offset en el arry de datos
    (short) datosAEscribir.length, // longitud en bytes de los datos
    arrayPWD, // array que contiene la MF_PASSWORD
    (short)0, // offset en el array  de password
    (short) arrayPWD.length, // longitud en bytes de la clave
    (short) sector, // sector objetivo
    bloque); // bloque objetivo
        }
    }
```

Figura 5-1 – Código Java Card

```
mode_211
enable_trace
establish_context
card_connect -readerNumber 1

// selección de nuestra aplicación
select -AID 01020304050600
// lectura sector 0 bloque 1
send_APDU -sc 0 -APDU 84100001080B54570745FE3AE70F
// escritura sector 0 bloque 1
send_APDU -sc 0 -APDU 84100001180B54570745FE3AE7 0102030405060708091011121314151 6
send_APDU -sc 0 -APDU 84100001080B54570745FE3AE70F //lectura sector 0 bloque 1
```

Figura 5-2 – Script GPShell

La primera aplicación desarrollada es un applet básico que ante cualquier comando APDU que se le envía responde de la misma forma. Esta sencilla aplicación permite un acercamiento a la tecnología de tarjeta inteligente JavaCard, siendo los procesos similares a los que se deberán realizar para soluciones más complejas como la que se desarrollará en este proyecto.

El primer paso a la hora de introducir nuevos comandos APDU es actualizar un método denominado , para que pueda reconocer los nuevos comandos y los derive a los métodos correspondientes. También se encargara de no realizar ninguna acción con el comando APDU que selecciona la aplicación y que por tanto no tiene que procesar. Se encarga además de detectar los comandos cuyos campos CLA y/o INS no estén recogidos por la aplicación, como comandos no comandos no soportados, descartándolos y devolviendo el error correspondiente.

El siguiente paso es la creación de dichos métodos que realizaran las funciones indicadas por los comandos. Con carácter general todos los métodos comienzan con la revisión de la cabecera del comando APDU, de esta forma, si hubiera algún error en esta, podríamos cortar el flujo del programa ahorrando tiempo y recursos. Comprobados los parámetros se descargan los datos de la APDU y se procesan de acuerdo al método elegido.

Una vez nos hemos familiarizado con los procesos de instalación y selección de una aplicación y sabemos cómo comunicarnos con ella, el siguiente objetivo es desarrollar una solución que permita acceder a la memoria asignada a la tarjeta MIFARE Classic y que por defecto se encuentra accesible a través del interfaz sin contactos. Puesto que se trata de una memoria no directamente accesible desde la parte Java Card de la tarjeta, se considera un segmento de memoria externo. En este sentido, la API Java Card incluye una interfaz denominada , incluida dentro del paquete , que permite el acceso a esa memoria externa.

Los parámetros necesarios para el uso de son:

- Tener instanciado un acceso a un subsistema de memoria. JavaCard que nos permite designar esta instanciación para dos subtipos: memorias MIFARE donde asume su estructura y memorias definidas manualmente sin estructura solo espacio de almacenamiento.
- Un array objetivo donde o bien se vaya a depositar los datos leídos o bien se depositen los que van a escribirse.
- Una clave de autenticación para poder ejecutar la lectura o escritura.

Uno de estos parámetros, la clave de autenticación, es, en nuestro caso, una contraseña de 8 bytes, denominada MF_PASSWORD, que autoriza las operaciones de lectura o escritura sobre los bloques del sector especificado.

El cálculo de la clave MF_PASSWORD, tal y como esta descrita en la especificación AN02105, requiere de conocer la clave A y B del sector, por lo que dota de acceso tanto en escritura como en lectura a los bloques de dicho sector. .

Debido a la carga computacional que exige el cálculo de esta función, se ha optado por, una vez generada, almacenar en la propia aplicación una tabla con todas la MF_PASSWORD de los sectores. De esta forma con una simple comparación se validará el proceso, siempre y cuando no se modifiquen las claves A y B de un sector específico.

La tabla de MF_PASSWORD tiene un tamaño de 320 bytes (40 sectores a 8 bytes cada clave) almacenados en forma de un array unidimensional referenciado mediante offset, tal como se refleja en la Tabla 5-1.

Tabla 5-1 - Array LibreriaClaves

MF_PASSWORD (sector 0)	MF_PASSWORD (sector 1)	……	MF_PASSWORD (sector 39)

Habiendo validado la correcta generación de la clave MF_PASSWORD, se procede a continuación a comprobar el comportamiento de los métodos y de . Para ello se diseñó una aplicación que permitiera realizar operaciones de lectura y escritura. Para evitar realizar operaciones incorrectas que pudieran afectar al comportamiento de la tarjeta (bloqueo memoria, etc.) se realizaron inicialmente las pruebas con operaciones únicamente de lectura para, posteriormente, lanzar la escritura en bloques específicos. Por motivos de seguridad, siempre evitamos escribir en bloques sensible de la tarjeta MIFARE Classic, en nuestro caso, los *Sector Trailer*. En esta fase de prueba obtuvimos los siguientes resultados:

- El bloque 0 del sector 0, como era de esperar y siguiendo la especificación de MIFARE Classic, es únicamente accesible en modo lectura y no puede modificarse.
- No se puede realizar la lectura de los bloques *Sector Trailer*. El documento AN02105 ya indica que el acceso siguiendo esta metodología de operación permite escribir pero no leer los *Sector Trailer*, aspecto que se ha corroborado mediante la prueba. Al intentar realizar la prueba de lectura sobre el bloque de *Sector Trailer* mediante el método de no obtenemos ningún dato como respuesta y el propio método nos indica que los datos devueltos en el array de lectura son 0, es decir, no puede leer nada en ese bloque.
- Al no poder realizar lecturas satisfactorias del *Sector Trailer* nos abstuvimos de intentar una escritura en ellos. Por un lado no podríamos leer nuevamente el *Sector Trailer* para verificarlo y por otro, un error podría bloquear un sector entero o modificar las claves KeyA y KeyB de manera irreversible. Evitamos también intentar una escritura modificando solo las condiciones de acceso ya que los accesos mediante el uso de la clave MF_PASSWORD las ignoran y por tanto, sin poder leer el *Sector Trailer*, no podríamos comprobar si realmente se han modificado. De cualquier manera asumimos que se podía realizar una escritura en los *Sector Trailer* ya que en las especificaciones así viene reflejado y por tanto en la aplicación no se prohíbe la escritura de dichos *Sector Trailer*.
- El acceso de lectura y escritura mediante estos métodos debe realizarse por bloque individual para cada llamada al método, es decir, no se pueden realizar operaciones sobre varios bloques consecutivos. Para validar este comportamiento se trató de leer 32 bytes consecutivos (2 bloques enteros). El resultado obtenido fueron los 16 bytes del bloque seleccionado seguido de 16 bytes de información que no correspondían con el bloque inmediatamente siguiente sino que eran 4 bytes (del byte 8 al 11) del bloque siguiente y 12 bytes de origen desconocido. Por tanto no era posible leer 2 o más bloques consecutivos con este método. En el caso de la

escritura, tras ver los resultados obtenidos en la lectura, no se hicieron las pruebas de intentar escribir más allá del tamaño del bloque objetivo por precaución a que se pudiera escribir en una zona sensible.

Resultado de este trabajo inicial, se ha evaluado el funcionamiento y limitaciones de los métodos básicos de Java Card que utilizaremos así como los de GPShell. Cabe destacar la imposibilidad desde Java Card de leer los *Sector Trailer* y como accediendo con los métodos de la interfaz MemoryAccess no se tienen en cuenta las restricciones en el acceso impuestas por las condiciones de acceso.

5.1.1 GESTIÓN DE LOS *Sector Trailer*

El no poder leer los *Sector Trailer* nos impide conocer las condiciones de acceso y las claves A y B de cualquier sector, y por tanto poder limitar el grado de acceso a los diferentes bloques.

Para solucionar este problema ampliaremos las dimensiones del array Tabla 5-2. Aprovechando que vamos a ampliar el array para incluir las claves MF_PASSWORD daremos un paso más e incluiremos también las claves KeyA y KeyB de cada sector, pasando a un array que contiene las claves KeyA, KeyB y MF_PASSWORD, es decir, 20 bytes por cada sector y 40 sectores en total, ampliando el array desde los 320 bytes iniciales a los 800 bytes actuales.

El hecho de incluir también las claves KeyA y KeyB de cada sector nos sirve para más adelante, cuando queramos simular mejor el comportamiento de las tarjetas virtuales MIFARE Classic utilizando para los comandos de lectura y escritura las claves KeyA y KeyB, en lugar de la clave MF_PASSWORD (que solo se utilizara internamente). Además de esta forma seremos capaces de utilizar las condiciones de acceso de forma más efectiva, ya que sus restricciones vienen determinadas por la clave facilitada.

Tabla 5-2 - Array LibreriaClaves expandido

KeyA	KeyB	MF_PASSWORD	KeyA	KeyB	MF_PASSWORD	.....	KeyA	KeyB	MF_PASSWORD
Sector 0			Sector 1				Sector 39		

El principal problema que se observa de la aproximación empleada es la falta de sincronización entre la memoria de la aplicación desarrollada y la memoria MIFARE Classic. Cualquier actualización del *Sector Trailer* a través del interfaz sin contacto no se verá reflejado en la aplicación y por tanto se trabajaría con datos erróneos. Así, un cambio de clave, invalidaría la MF_PASSWORD calculada y por tanto supondría un error constante de autenticación.

Puesto que durante el desarrollo del proyecto no es posible realizar pruebas accediendo a la tarjeta MIFARE por la parte sin contactos, hemos planteado tres opciones para solventar este problema meramente teóricas:

- Incluir un bit de dirty en una posición fija dentro del sector cuyo valor sea siempre 0 y, cuando se acceda por la parte sin contactos y se modifique el *Sector Trailer*, se

ponga a 1. Para hacerlo general a todos los sectores no podría ser ni en el bloque 0, este bloque en el sector 0 es el Manufacturer Block y no puede ser alterado; ni en el bloque 3, correspondiente al *Sector Trailer* en los sectores del 0 al 31 (ambos inclusive); ni en un bloque superior a 3, puesto solo los sectores entre el 32 y el 39 (ambos inclusive) disponen de más de cuatro bloques. El problema de este método es que solo avisa de una modificación y simplemente prohibiría el uso de ese sector necesitándose una actualización manual del array para volver a funcionar correctamente.

- Actualizar de forma automática el array . Para ello únicamente se podría actualizar el contenido de los *Sector Trailer* en terminales que o bien puedan acceder también a la parte con contactos y de ese modo a la aplicación (algo absurdo porque si intentamos acceder a través de la parte sin contactos es precisamente para no tener que hacer uno de estos) o bien que estén conectados a una red central que permita salvar el nuevo estado de los *Sector Trailer* y actualizarlo cuando se acceda mediante la parte con contactos. Este método requiere que tanto el dispositivo externo sin contactos y con contactos dispongan de unas características especiales.
- La última opción sería disponer de un software que pudiera acceder a ambas partes. Si partimos del uso de la tarjeta inteligente en un dispositivo móvil podremos acceder a la tarjeta por la parte sin contactos utilizando, por ejemplo, la tecnología NFC y al mismo tiempo al estar la tarjeta dentro del dispositivo este tendría acceso a los contactos de la misma. En esta situación podríamos disponer de un software que controle el flujo de información que se envía y recibe mediante NFC y cuando detecte una escritura en un *Sector Trailer* acceda a la aplicación y actualice el array .

El problema de que se ignoren las condiciones de acceso siempre supone una limitación para las funcionalidades que ofrece la tarjeta MIFARE. Si se facilita la clave MF_PASSWORD esta se usa tanto para lectura como la escritura por lo que, en caso de necesitarse, no se podría bloquear el acceso parcial a un bloque (por ejemplo para un usuario) sin bloquearlo para todos los usuarios.

Para solventar este problema desarrollaremos en el código un sistema para habilitar las funciones de las condiciones de acceso que expondremos más adelante.

5.2 Desarrollo de la implementación MIFARE4Mobile para MIFARE Classic

Una vez analizadas las diferentes formas de acceso a la memoria MIFARE Classic desde un applet Java Card e identificar las limitaciones del procedimientos, se procede a realizar la implementación de la emulación de los comandos básicos MIFARE4Mobile para tarjetas del tipo MIFARE Classic.

En esta fase incluimos los comandos modificados de M4M. Además, se incluyen dos comandos de gestión que fueron eliminados en la versión final del programa al no ser necesarios. La estructura y detalle de estos comandos se encuentra recogida no obtante en el anexo 7.1.1-Comandos descatalogados Estos comandos serían:

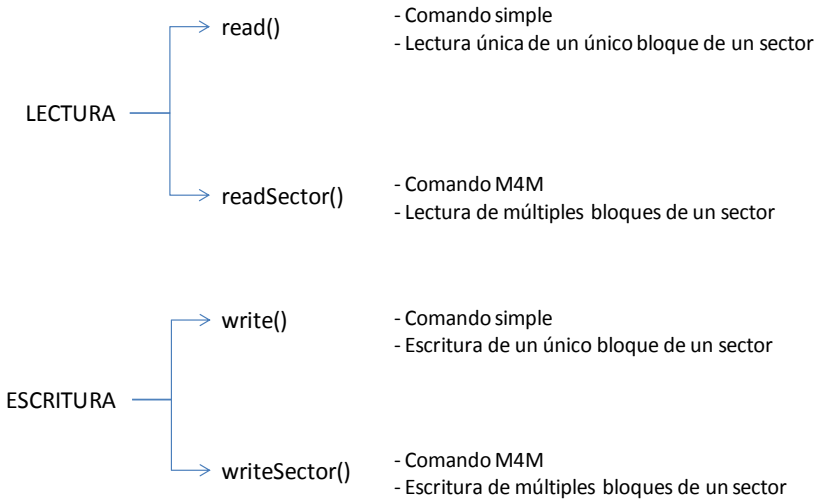
- Comando : comando para la actualización del array . El campo P1 del APDU especifica el sector cuyas claves KeyA y KeyB se van a actualizar y en el campo de datos aparecen estas de forma consecutiva. A la recepción de este comando, y una vez comprobado que los valores recibidos estaban dentro del rango esperado, se actualiza el array y se calcula la nueva clave MF_PASSWORD.
- Comando : comando para comprobar que se gestionaba de forma correcta el array . El campo P1 del APDU hace referencia al sector objetivo y el campo P2 a la clave objetivo. Devuelve la clave solicitada del sector solicitado.

El método recoge la APDU comando enviada y, a partir de la información facilitada por esta, actualiza el array del sector especificado en el parámetro P1 con las claves KeyA y KeyB, facilitadas en el campo de datos. Una vez copiadas llama al método que calcula la nueva clave MF_PASSWORD de ese sector y la actualiza.

El método recibe como parámetro la APDU comando y devuelve, del sector especificado por el campo P1, la clave solicitada por el campo P2 (KeyA, KeyB o MF_PASSWORD) que este almacenada en el array . Este método solo se utilizada durante esta fase del proyecto para comprobar que se actualiza correctamente el array .

Por último nos encontramos con los comandos de lectura, escritura y reset de los bloques. En el apartado anterior ya habíamos creado métodos para la lectura y escritura básicos para comprobar las limitaciones de la interfaz que utilizaremos como base para implementar los métodos definidos por M4M.

Existirán dos métodos distintos tanto para lectura como para escritura, un método general y simple y un segundo método que emula el comportamiento del comando M4M. El método simple consistirá en un método para lectura y para escritura, que recibirá un comando APDU que designará como objetivo un bloque en particular de un sector específico, es decir, solo



hará una lectura o escritura de un bloque único. En el segundo caso el método estará basado en los comandos M4M, y realizara una lectura o escritura, mediante los métodos y respectivamente, sobre múltiples bloques de un sector especifico, es decir, con un mismo comando se podrá leer o escribir varios bloques de un mismo sector. En la Figura 5-3 aparecen de forma más simplificada los comandos mencionados.

Figura 5-3 - Comandos de lectura y escritura

El método es el más particular de los tres ya que requiere el conocimiento del estado inicial y su ejecución es una variante del comando , con la particularidad de que los datos que se van a escribir son fijos, ceros para los bloques de datos y el valor inicial de creación para el *Sector Trailer*. Además se reescribirán todos los bloques del sector, con la única excepción del bloque 0 del sector 0 que es el *Manufacturer Block* y no puede modificarse. Con el fin de ahorrar código, el método llamara interno al método ya que en una sola llamada podemos escribir todo el sector.

La inclusión del método nos obliga a generar otro array parecido al array , que denominaremos , donde se almacenen todos los *sectorTrailer* originales con lo que se genera una tarjeta virtual nueva con la única diferencia que se incluya también el campo de las condiciones de acceso y no aparezca el de MF_PASSWORD. La estructura de este array sería la indicada en la Tabla 5-3.

Tabla 5-3 - Array OriginalSectorTrailer

Key A	AC	Key B	Key A	AC	Key B	.....	Key A	AC	Key B
Sector 0			Sector 1				Sector 39		

5.3 DESARROLLO DE LA SEGURIDAD EN EL ACCESO

En el estado actual ya tenemos una aplicación que nos permite comunicarnos con la tarjeta MIFARE Classic y acceder a su contenido. Ahora tenemos que fijar nuestra atención en la seguridad.

De momento la única medida de seguridad de la que disponemos es la necesidad de permisos de administrador de la tarjeta para poder instalar y desinstalar la aplicación y la clave MF_PASSWORD para poder ejecutar los comandos de lectura y escritura, todo esto lamentablemente no es suficiente para considerar nuestro sistema seguro.

Tomaremos varias medidas de seguridad para garantizar la integridad y confidencialidad de las claves, que hasta el momento se transmiten en claro. Para el caso de los comando de acceso a datos de la memoria MIFARE Classic, la clave MF_PASSWORD permite acceder a cualquier bloque de cualquier sector sin tener en cuenta las restricciones de lectura y/o escritura impuestas por las condiciones de acceso pudiéndose incluso modificar las claves de los *Sector Trailer*. Por otra parte, tendremos que crear comandos para la gestión de las tarjetas virtuales, tanto para su creación como su posterior gestión. Hasta este momento, cualquier acceso a la aplicación se hacía con las claves de canal seguro del nivel de seguridad de emisor y por tanto, cualquier acceso es considerado administrador y tiene acceso total a todos los comandos.

Para ello llevaremos a cabo cuatro modificaciones importantes:

- Activaremos las restricciones impuestas por las condiciones de acceso.
- Generaremos un nivel de seguridad para el administrador de la tarjeta física, de tal forma que solo él pueda instalar y desinstalar aplicaciones y crear y eliminar las tarjetas virtuales.
- Generaremos un nivel de seguridad para el administrador de la tarjeta virtual, siendo este el encargado de manejar las claves de la tarjeta inteligente y los datos más sensibles de la misma, como los *Sector Trailer*.
- Estableceremos tres tipos de canales seguros: canal seguro para el administrador de la tarjeta inteligente, canal seguro para el administrador de la tarjeta virtual y canal seguro para el usuario final.

5.3.1 ACTIVACIÓN DE LAS CONDICIONES DE ACCESO

Como hemos comentado anteriormente, al intentar acceder a una memoria en una interfaz externa necesitamos utilizar una clave denominada MF_PASSWORD. Uno de los principales inconvenientes en seguridad al utilizar este método de acceso sobre una memoria MIFARE Classic es que se pasan por alto las condiciones de acceso. Si bien en determinadas aplicaciones este hecho podría no suponer un problema en la mayoría de los casos es un grave fallo de seguridad ya que una vez obtenida la clave MF_PASSWORD de un sector cualquiera puede alterar los datos de dicho sector y, de alterar el *Sector Trailer*, podría modificar las claves y/o las condiciones de acceso pudiendo evitar que cualquier otro usuario o administrador tuviera acceso a dicho sector.

A pesar de los problemas que presenta carecer de esta medida de seguridad, MIFARE4Mobile no cuenta con ella y por tanto, como intentamos emular su funcionamiento, el uso de esta medida dependerá de si el administrador de la tarjeta física la activa o no. Esta activación se lleva a cabo mediante un parámetro que se pasa durante la instalación de la aplicación y es irreversible, teniendo que reinstalar nuevamente la aplicación para cambiarlo (eliminándose todos los datos de la aplicación y las tarjetas virtuales).

Incluir la comprobación de las condiciones de acceso supone además la necesidad de que la aplicación pueda tener acceso a las mismas y, dado que originalmente las condiciones de acceso de cada sector se encuentran en el correspondiente *Sector Trailer* de dicho sector, necesitamos almacenarlas dentro de la aplicación junto con el resto del *Sector Trailer*. Para ello adaptaremos nuevamente el array para que incluya las condiciones de acceso del sector. El array quedara como muestra la Tabla 5-4.

Tabla 5-4 – Array LibreriaClavesCompleto

Key A	AC	Key B	MF_PASSWORD	Key A	AC	Key B	MF_PASSWORD	.....	Key A	AC	Key B	MF_PASSWORD
Sector 0				Sector 1					Sector 39			

Otro de los cambios que supone la inclusión de las condiciones de acceso es que el usuario final ya no tiene que acceder con la clave MF_PASSWORD, sino con alguna de las dos claves KeyA o KeyB. Por tanto, la clave que debe de pasarse en los comandos ya no será la clave MF_PASSWORD de 8 bytes sino una de las dos claves KeyA o KeyB de 6 bytes. La aplicación se encargara internamente de saber si está habilitada la comprobación de las condiciones de acceso y saber que clave recibe.

Los comandos de lectura, escritura y reset, que hasta ahora podían acceder a cualquier bloque de cualquier sector, ven limitada su actuación. En los tres casos se debe pasar una doble validación, identificación de la clave suministrada y comprobación las condiciones de acceso.

El primer paso consiste en leer la clave suministrada y compararla con las claves KeyA y KeyB del sector objetivo que tenemos almacenadas en el array . El segundo paso consiste en verificar, con dicha clave, los bits de las condiciones de acceso. Hay que tener en cuenta que el bloque 0 del sector 0 siempre estará bloqueado contra escritura y que dado el carácter de esta aplicación ambas claves, KeyA y KeyB, siempre serán leídas como ceros aunque las condiciones de acceso permitan su lectura.

Aparece también un nuevo tipo de error durante la ejecución de la aplicación cuando no pueden comprobarse las condiciones de acceso debido a que el array esté incompleto. Al necesitarse de este almacenamiento interno para verificar los datos contenidos en los *Sector Trailer*, puede darse el caso de que un determinado sector aun no tenga actualizado su segmento dentro del array una vez instalada la aplicación y este se halle vacío. En esta situación no se pueden comprobar ni las claves ni las condiciones de acceso hasta que se ejecute un comando `setKey` para dicho sector. La aplicación lanzaría un error NOT_AUTHORIZED, el mismo que si la clave facilitada fuera errónea.

Esta situación particular la eliminaremos en la versión final cuando se haga obligatorio pasar toda la información de configuración al instalar una tarjeta virtual nueva.

5.3.2 NIVEL DE SEGURIDAD DEL ADMINISTRADOR DE LA TARJETA FÍSICA

El administrador de la tarjeta física, al que llamaremos nivel de seguridad de emisor, suele ser el fabricante de la tarjeta en sí o cualquier grupo en el que este haya delegado el control de la misma. Este nivel de seguridad tiene control sobre toda la tarjeta física, en especial sobre el dominio de seguridad del emisor, dominio de seguridad por defecto.

El acceso como administrador de la tarjeta física otorga ciertos privilegios como la carga e instalación de aplicaciones y, en nuestro proyecto concreto, la creación y eliminación de tarjetas virtuales, donde se asegurara de inicializar las mismas, cargar el dominio de seguridad específico para cada tarjeta virtual y la gestión que estos procedimientos requieran. El administrador de la tarjeta física, aunque es responsable de crear las tarjetas virtuales, no tiene acceso directo a la gestión de las mismas ni a los datos que estas contengan, ya que la información contenida pertenece a los administradores de la tarjeta virtual.

Toda la información manejada en este nivel de seguridad se considera delicada y por tanto confidencial, ya que incluirá la carga e instalación de las aplicaciones y todos los datos iniciales de las tarjetas virtuales necesarias para su creación. Para proteger estos datos se utilizara un canal seguro del tipo SCP02 con la opción de implementación i=15. Hay que destacar que utilizando esta opción se cifran únicamente los comandos APDU y las respuestas APDU se transmiten en claro. Si bien puede parecer que la seguridad podría no ser suficiente en este nivel de seguridad, las únicas respuestas que contienen datos existen solo durante la fase de apertura de canal seguro y son respuestas que se pueden consultar fácilmente en la especificación de GlobalPlatform, el resto del tiempo únicamente los comandos APDU contendrán datos de instalación.

Añadiremos un array llamado VC que nos servirá para indicar las tarjetas virtuales instaladas, cada byte indicara una tarjeta virtual numerada según su posición en el array, si toma el valor uno significara que la tarjeta virtual está instalada y si es cero que no hay tarjeta virtual.

Como el nivel de seguridad de emisor es el encargado de la creación y eliminación de las tarjetas virtuales es el único que tendrá acceso a dos nuevos comandos creados para tal fin, los comandos y , encargados precisamente de la creación y eliminación de las tarjetas virtuales.

El comando está formado por tres parejas de APDUs comando-respuesta, la primera contiene el *manufacturer block* (el bloque 0 del sector 0) y las 3 claves necesarias para crear los canales seguros del gestor. Las dos parejas restantes contienen 8 *Sector Trailer* cada pareja para completar los 16 sectores. Este comando primero verifica que existen huecos disponibles para instalar una nueva tarjeta y luego asigna el primer hueco disponible. Una vez se recibe el último comando se calculan internamente todas las MF_PASSWORD de los sector tráiler proporcionadas.

El comando solo necesita un comando que especifica el hueco de tarjeta virtual que se quiere desinstalar y la aplicación elimina todas las conexiones referentes a dicha tarjeta virtual y, si dicha tarjeta estaba seleccionada, resetea todos los sectores de la tarjeta física. De esta forma aunque internamente parte de los datos (el *manufacturer block*, los *Sector Trailer* y las 3 claves de canal seguro) siguen presentes, no se puede tener acceso a ellos y cuando se vuelva a instalar una tarjeta virtual en el mismo hueco estos datos se sobrescribirán.

Ambos comandos vienen detallados en el capítulo 7.1.3.2-Comandos del nivel de seguridad de emisor.

Puesto que contamos con 4 tarjetas virtuales de 15 sectores al llegar a este punto, necesitamos actualizar nuevamente los arrays libreriaClaves y originalSectorTrailer ya que en lugar de basarnos en los 40 sectores que tiene la tarjeta MIFARE Classic pasamos a los 64 sectores, 16 sectores por cada una de las 4 tarjetas virtuales. De esta forma los nuevos y ya definitivos arrays quedan tal y como aparecen en la Tabla 5-5 y la Tabla 5-6.

Tabla 5-5 - Array libreriaClaves

Key A	AC	Key B	MF_PASSWORD	Key A	AC	Key B	MF_PASSWORD	…….	Key A	AC	Key B	MF_PASSWORD
Sector 0 (tarjeta virtual 0)				Sector 1 (tarjeta virtual 0)					Sector 15 (tarjeta virtual 3)			

Tabla 5-6 - Array originalSectorTrailer

Key A	AC	Key B	Key A	AC	Key B	……	Key A	AC	Key B
Sector 0 (tarjeta virtual 0)			Sector 1 (tarjeta virtual 0)				Sector 15 (tarjeta virtual 3)		

5.3.3 NIVEL DE SEGURIDAD DEL ADMINISTRADOR DE LA TARJETA VIRTUAL

Una vez creada una tarjeta virtual nueva surge la figura del administrador de la misma, que se asignará a una persona o grupo a la que dicha tarjeta virtual pertenece, pudiendo ser el propio administrador de la tarjeta física o alguien al que este haya concedido ese espacio de memoria. Este administrador se encarga de gestionar los datos y la comunicación entre la tarjeta virtual y el usuario final. Para este administrador de la tarjeta virtual es necesario crear

un nivel de seguridad para este administrador al que denominaremos nivel de seguridad de gestor.

El nivel de gestor posee la capacidad de alterar todos los bloques dentro de una tarjeta virtual, salvo los bloques que originalmente sean inmodificables. Además es el encargado de restringir el acceso a los datos según las especificaciones impuestas (en nuestro caso las condiciones de acceso) y será el único que tenga acceso al comando .

Puesto que el administrador de la tarjeta virtual puede no tener acceso directo a la misma, será en este nivel de seguridad donde utilizaremos la gestión remota de MIFARE4Mobile, descrita en el apartado 3.4.2-Gestión remota de MIFARE4Mobile, de este modo podrá actualizar los datos de forma remota con seguridad extremo-extremo y de forma más eficiente que con comandos simples.

En su creación cada tarjeta virtual se genera un dominio de seguridad propio que cumpla los requisitos requeridos por el administrador de la tarjeta virtual. La conexión cifrada mediante canales seguros se hará con este dominio de seguridad y no con el del emisor lo que permite que el administrador de la tarjeta física no necesite conocer las claves utilizadas por este nuevo dominio de seguridad. Esto también implica que en la generación de este nuevo dominio de seguridad se requerirá un nuevo juego de claves para poder establecer el canal seguro, las claves que se mandaban a la aplicación durante la instalación de la tarjeta virtual.

Al contrario que en el caso del nivel de seguridad del emisor, la información sensible se encuentra también en las respuestas APDU. En este caso, no nos sirve el mismo nivel de seguridad que nos proporciona el canal seguro utilizado para la comunicación en el nivel de seguridad del emisor lo que nos obliga a implementar el cifrado también las respuestas.

La creación de un dominio de seguridad totalmente nuevo esta fuera del objetivo de este proyecto por lo que, como solución, hemos diseñado una aplicación, que cargaremos junto con la que tenemos ya diseñada, que hará las veces de un dominio de seguridad. Será la encargada de generar las claves de sesión, establecer un canal seguro, descifrar los datos recibidos a través de los comandos APDU y cifrar los que se van a enviar a través de las respuestas APDU.

5.3.4 NIVEL DE SEGURIDAD DE USUARIO FINAL

En el nivel de seguridad más bajo nos encontramos con el usuario final. Este nivel de seguridad es el que menos privilegios tiene y su única finalidad será permitir la lectura y escritura, siempre respetando las limitaciones impuestas por el administrador de la tarjeta inteligente. Si fuera necesario se le podría negar el acceso a la lectura, escritura o ambas a un determinado bloque o sector, por ejemplo a los *Sector Trailer* evitando así que puedan modificar las condiciones de acceso.

Puesto que en este nivel de seguridad se puede leer y escribir datos en la tarjeta, la información transmitida tanto en los comandos como en las respuestas APDU se consideran información sensible y por tanto es necesario protegerla, pero dado que el acceso a dicha

información podrá bloquearse desde el nivel de seguridad de gestor, no se va a exigir tanta seguridad como en los casos anteriores (recordemos que el protocolo de canal seguro SCP02 consta de 3 claves de 8 bytes y encriptación 3DES). En este caso nos conformaremos con una encriptación más sencilla de tipo DES con una clave de 8 bytes que ambos, usuario y aplicación, conocerán. La clave de 8 bytes que utilizaremos, que denominaremos PIN (*Personal Identification Number* o Número de Identificación Personal), será elegida por el mismo usuario y podrá modificarlo a su criterio.

En este nivel de seguridad tendremos únicamente 4 comandos: dos comandos de PIN, uno para identificar al usuario e iniciar el cifrado de los datos y el segundo para actualizar el propio PIN; y los comandos de lectura, , y escritura, , todos ellos detallados en el anexo 7.1-Comandos APDU.

Utilizaremos el dominio de seguridad que hemos creado, el que utilizamos para establecer y mantener un canal seguro en el nivel de seguridad gestor, para el cifrado en el nivel de seguridad de usuario. Para ello el dominio de seguridad deberá distinguir el nivel de seguridad del usuario que este accediendo y tendremos que ampliarlo para que pueda realizar un cifrado de tipo DES con una clave de 8 bytes para el nivel de seguridad de usuario

5.4 DESARROLLO DEL PROGRAMA COMPLETO Y RESULTADOS

Una vez definidos e implementados los distintos niveles de seguridad únicamente nos resta comprobar el funcionamiento global de la aplicación y esperar que responda a todas nuestras especificaciones.

Puesto que no vamos a tener una única tarjeta virtual y cada tarjeta virtual tiene un juego de claves distinto dentro del dominio de seguridad que hemos desarrollado tiene que haber una diferenciación entre las distintas tarjetas por lo que nos obliga a crear dos nuevos comandos, los comandos de selección y deselección de tarjeta virtual. Estos comandos determinarán la tarjeta virtual que se encuentra activa y por tanto la que se encuentra cargada en la memoria MIFARE para su acceso sin contactos.

Si no hay una tarjeta virtual seleccionada solo se podrá acceder al nivel de seguridad de emisor, por el contrario, si hay una tarjeta seleccionada, solo se podrá acceder, también previa apertura de los canales de seguridad correspondientes, a los niveles de seguridad de emisor y de usuario. De esta forma el dominio de seguridad de emisor no podrá acceder a la tarjeta virtual y tanto el dominio de gestor como el de usuario solo accederán a la tarjeta activa.

Al seleccionar una tarjeta virtual tendremos que autenticarnos como gestor o usuario mediante los comandos de apertura de canal seguro, la aplicación lo diferenciara por el código INS de la APDU.

El diagrama de flujo de cómo se seleccionan los distintos niveles de seguridad se expone en la Figura 5-4.

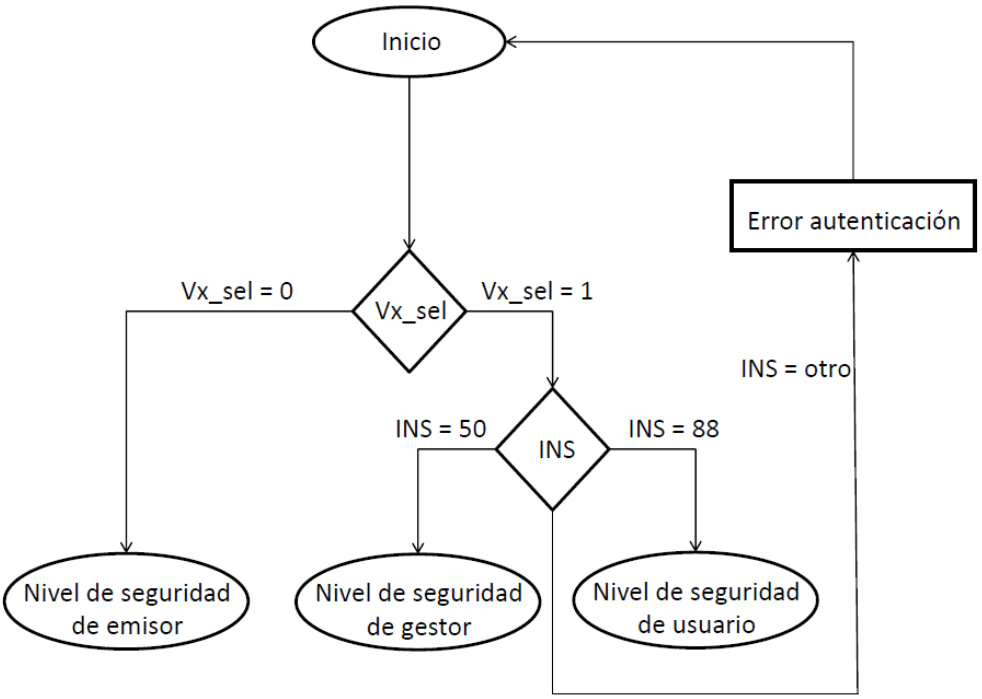


Figura 5-4 - Interacciones entre la aplicación y los dominios de seguridad

Una vez asignado un nivel de seguridad hay que comprobar que únicamente se pueden ejecutar los comandos permitidos para dicho nivel y que las limitaciones impuestas a dichos comandos se cumplen. Para ello nos conectaremos como cada nivel de seguridad y ejecutaremos un benchmark diseñado para comprobar cada uno de los aspectos de dicho nivel. Los benchmarks utilizados para este proyecto se encuentran adosados en el anexo 7.2-Workbenchs.

CONCLUSIONES Y LÍNEAS FUTURAS

6

Una vez finalizada la aplicación y realizadas las pruebas de comportamiento plantearemos las conclusiones finales que arroja nuestro proyecto. Con ellas analizaremos el cumplimiento de los requisitos que planteábamos en los objetivos y, en los casos en los que no se haya podido alcanzar un punto del objetivo, los motivos y/o soluciones que hemos tomado para solventarlo. También incluiremos aquellas posibles mejoras que se pueden implementar y que por falta de tiempo, de conocimientos, de habilidad o por que la tecnología actual no lo permite, no hemos podido implementar.

También abordaremos los proyectos más significativos que se están implementando ahora mismo, en el mismo campo en el que se desarrolla este proyecto y los avances que se espera traiga el futuro, como nuevas tecnologías y dispositivos.

Por último especularemos sobre el impacto que los avances en este campo puedan llegar a suponer en nuestra vida diaria.

6.1 CONCLUSIONES

Tras la finalización del programa completo y las pruebas realizadas para validar su funcionalidad (recogidas en el capítulo 7.2-Workbenchs) hemos podido comprobar que cumplimos con el objetivo que nos habíamos planteado al principio del proyecto. Partiendo de una tarjeta inteligente genérica, hemos desarrollado una aplicación que nos permite simular tarjetas virtuales, de tal forma que con solo una tarjeta física disponga de varias tarjetas virtuales independientes.

Para empezar hemos implementado una solución de acceso mediante contactos a un tipo de tarjeta, en nuestro caso MIFARE Classic, que no es estándar y cuyo método de acceso típico es sin contactos. Durante este acceso por contactos a través de Java Card, hemos simulado el comportamiento del acceso a la tarjeta de manera similar a como se hace de forma habitual,

utilizando los mismos parámetros pero dotando a la comunicación de un nivel de seguridad mayor.

De cara a la creación y gestión de las tarjetas virtuales, hemos dividido el acceso a la aplicación en 3 niveles diferenciados de seguridad, cada uno de los cuales posee una seguridad única para dicho nivel y cuenta con unas herramientas, privilegios y comandos específicos. Cada uno de los niveles de seguridad es independiente de los otros dos, si bien entre ellos se complementan, son auto excluyentes, es decir, solo uno de ellos puede estar activo en un mismo momento.

Los comandos de cada nivel de seguridad responden únicamente ante su nivel de seguridad asociado y solamente cuando este está activo, impidiendo que puedan ejecutarse por usuarios no autorizados.

También se destaca que los niveles de seguridad que hemos establecido y comprobado en nuestro proyecto no solo cumplen los estándares habituales en este tipo de casos, sino que la seguridad global con la que cuenta nuestra aplicación en comparación con las que hemos utilizado como base y referencia es superior. Desde el primer contacto con la aplicación se abre un canal seguro que protege los datos que suministramos y recibimos de la tarjeta y lo hacemos tal y como se indica en las especificaciones, facilitando la migración futura a otros sistemas de nuestra aplicación. Comparándolo con las aplicaciones de referencia, nuestra aplicación se encarga del cifrado los datos no solo en las APDUs comando sino también en las respuestas, incluso en el nivel de seguridad de usuario final, donde utilizamos un sistema de encriptación probado denominado DES en lugar de la encriptación CRYPTO1 utilizada por la tarjeta MIFARE Classic.

En cuanto a la simulación de tarjetas virtuales trabajamos sobre una tarjeta de tipo MIFARE Classic que se encuentra en la parte sin contactos de nuestra tarjeta física por lo que las tarjetas virtuales se irán cargando sobre su memoria. Como las tarjetas virtuales que generamos son de solo 1kB solo utilizaremos los primeros 16 sectores de la memoria MIFARE.

El objetivo del proyecto por tanto se cumple dentro de nuestro entorno controlado y deja abierta la puerta para un sistema más avanzado, con creación de múltiples tarjetas con estructuras distintas y gestión de las mismas para un proyecto futuro, sirviendo este como base para realizarlo.

Por otra parte como hemos tenido que desarrollar un método de acceso por contactos en una tarjeta que normalmente se accede sin contactos. De esta forma hemos establecido una forma de actualización de las distintas tarjetas virtuales mediante la interfaz con contactos al tiempo que se respeta el acceso por la de sin contactos.

6.2 LÍNEAS FUTURAS

La meta final de nuestro proyecto era sentar ciertas bases prácticas para el desarrollo de tarjetas inteligentes que integren múltiples tarjetas virtuales cumpliendo los requisitos de funcionalidad y seguridad que se esperan de sistemas de este tipo.

Este proyecto no es pionero ni único en su campo, pero si corrige en ciertos puntos los que ya están establecidos abriendo las puertas a un desarrollo más avanzado sobre el mismo. A partir de este proyecto se puede desarrollar una aplicación más compleja con una gestión más avanzada de las tarjetas virtuales, que permita la instalación de distintos tarjetas virtuales siguiendo el modelo de MIFARE4Mobile (nuestra aplicación trabaja únicamente con tarjetas MIFARE Classic de 1kB). Asimismo se puede desarrollar una interfaz de comunicación sin contactos, bien en una tarjeta que tenga este tipo de tecnología disponible físicamente (tarjetas inteligente que tengan tanto la interfaz con contactos como la interfaz sin contactos) o bien a través de algún dispositivo que cuente con ella (tal y como se propone en MIFARE4Mobile con un dispositivo móvil).

También hemos utilizado una aplicación que permite el acceso remoto en una tarjeta como la MIFARE Classic que, a priori, no dispone de una seguridad suficiente para permitir este tipo de acceso. De esta forma sería posible que, a través de un dispositivo que incorporara dichas tarjetas como un teléfono móvil, permitieran un acceso remoto a las tarjetas para poder actualizarlas.

Alejándonos a un desarrollo más complejo que el de nuestro proyecto encontramos distintas empresas, especificaciones y productos que también se encaminan hacia lo que hemos querido conseguir con nuestro proyecto.

Por un lado tenemos la especificación de MIFARE4Mobile que está muy enfocada en la creación y gestión de tarjetas virtuales en elementos seguros (por ejemplo tarjetas SIM) utilizados en dispositivos móviles (smartphones, tablets, etc). Uniendo las tarjetas virtuales con estos dispositivos se pueden crear aplicaciones en los propios dispositivos que gestionen cada una de las tarjetas por separado, teniendo acceso al elemento seguro, sin que estas aplicaciones se instalen directamente en este, por ejemplo, una tienda de ropa podría crear una aplicación móvil que ofreciera promociones y noticias a un socio, el socio en este caso interactuaría con esta aplicación, y no necesitaría tener una tarjeta física en su cartera que le reconociese su condición de socio, esta se encontraría instalada como una tarjeta virtual en su dispositivo. El principal inconveniente de MIFARE4Mobile es que solo trabaja con tarjetas de la familia MIFARE y está pensado para utilizarse a través de un dispositivo móvil.

También podemos encontrar en el mercado actual tarjetas inteligentes físicas pensadas para soportar multiaplicación, es decir, su estructura de memoria está pensada para recoger múltiples aplicaciones de proveedores distintos, diferenciando y protegiendo cada aplicación para ese proveedor de tal forma que cada aplicación es independiente de las demás y con una seguridad única y especifica. El objetivo de estas tarjetas inteligentes es que sirvan para todo tipo de servicios, desde el control de accesos dentro de una compañía hasta los abonos del transporte público. Esta idea encaja, a grandes rasgos, con la motivación y objetivos de este

proyecto pero se basa más en múltiples aplicaciones específicas dentro de la tarjeta que en las tarjetas virtuales que nosotros perseguimos.

En definitiva en el futuro se tiende a desarrollar sistemas que integren más y más componentes por lo que tener multitud de tarjetas distintas en nuestra mano va a ser una de tantas cosas que poco a poco van a ir desapareciendo a medida que vayan popularizándose las tarjetas inteligentes multiaplicación y las tarjetas inteligentes con múltiples tarjetas virtuales. Ahora mismo es impensable un escenario en el que todas las tarjetas puedan acabar integrándose en una sola debido a que deben alcanzarse aún múltiples acuerdos comerciales y estatales, hay que recordar que el DNI-electrónico que utilizamos actualmente es una tarjeta inteligente pero dadas las medidas de seguridad asociadas a su expedición y uso no es un elemento que vaya a ser probable que comparta espacio físico con otro tipo de tarjetas, lo mismo pasaría con las tarjetas bancarias.

ANEXOS 7

7.1 COMANDOS APDU

Como ya hemos comentado en nuestra aplicación hemos especificados varios comandos, la mayoría basados en los comandos ya establecidos por MIFARE4Mobile. En este anexo describiremos en detalle todos los comandos que hemos utilizado durante el proyecto dado que son estos mismos comandos los que se utilizan para comunicarse con la aplicación.

En un primer grupo tenemos los comandos descatalogados. Este grupo recoge todos los comandos que hemos creado durante el proyecto, sobretodo como comandos de comprobación y que han ido retirándose de la aplicación a medida que íbamos avanzando en las distintas fases.

El segundo recoge todos los comandos de entrada y salida de datos, es decir, los de lectura y escritura, todos ellos como modificaciones de los comandos de M4M. En este grupo podemos encontrar dos divisiones, lo comandos únicos y los comandos de gestión remota.

Por último tenemos los comandos que siguen presentes en la versión final de la aplicación y hemos tenido que especificar expresamente para nuestro proyecto.

7.1.1 COMANDOS DESCATALOGADOS

Los comandos descatalogados aparecen sobre todo durante las primeras fases del desarrollo ya que sirven para realizar labores de comprobación de resultados.

En esta categoría recogemos solo dos comandos, los comandos y el .

7.1.1.1 COMANDO SETKEY

El comando sirve para actualizar el array con las claves KeyA y KeyB de un sector particular. Su función, en conjunto con el comando , era comprobar por un lado que el array se actualizaba correctamente y que el cálculo de la clave MF_Password se realizaba correctamente.

La estructura del comando es la que se muestra en la Tabla 7-1.

Tabla 7-1 - Estructura del comando

CLA	INS	P1	P2	Lc	DATOS
00	00	Sector	XX	0C	Claves KeyA y KeyB

Parámetros:

- CLA= “0x00”: código CLA que utilizábamos al principio del desarrollo de la aplicación.
- INS= “0x00”: valor que corresponde a la instrucción en nuestra aplicación.
- P1: representa el sector cuyas claves vamos a actualizar.
- P2: no se utiliza en este comando por lo que su valor puede ser cualquiera.
- Lc: tamaño, en bytes, del campo datos. Siempre tendrá el valor “0x0C” correspondiente a 12 bytes.
- Datos: claves KeyA y KeyB del sector objetivo a actualizar.

La respuesta al comando consiste en la palabra de estado. Si el comando se ha procesado correctamente se devuelve el código “0x9000” establecido en ISO7816 en su parte 4, en caso contrario se devuelve un código de error.

Códigos de error:

- WRONG_LENGTH= “0x6700”: indica que no se han proporcionado exactamente 12 bytes en el campo de datos que corresponden al tamaño de las dos claves, KeyA y KeyB.
- WRONG_P1P2= “0x6B00”: indica que el valor del campo P1 es erróneo, es decir, apunta a un sector que se encuentra fuera de rango tomando un valor superior a 39.

7.1.1.2 COMANDO GETKEY

El comando sirve para extraer del array cualquiera de las claves almacenadas de un sector específico. Su función junto con el comando era comprobar la correcta actualización del array .

La estructura del comando es la que se muestra en la Tabla 7-2.

Tabla 7-2 - Estructura del comando

CLA	INS	P1	P2	Lc	Le
00	30	sector	clave	00	variable

Parámetros:

- CLA= "0x00": código CLA que utilizábamos al principio del desarrollo de la aplicación.
- INS= "0x30": valor que corresponde a la instrucción en nuestra aplicación.
- P1: representa el sector cuya clave vamos a leer.
- P2: indica la clave a extraer. Puede tomar los siguientes valores: "0x01" para la KeyA, "0x02" para la KeyB, "0x03" para la MF_PASSWORD y "0x04" para las condiciones de acceso.
- Lc= "0x00": siempre tendrá el valor "0x00" ya que no cuenta con campo de datos.
- Le= campo de valor variable según la clave requerida. Puede tomar los valores= "0x06" cuando se pida la KeyA o KeyB, "0x08" para la clave MF_PASSWORD y "0x04" para las condiciones de acceso.

La respuesta al comando consiste en la clave solicitada más la palabra de estado. Si el comando se ha procesado correctamente la palabra de estado toma el valor "0x9000" establecido en la parte 4 de la norma ISO7816, en caso contrario solo se devuelve la palabra de estado con un código de error.

Códigos de error:

- WRONG_DATA= "0x6A80": indica que la clave que se ha requerido en el campo P2 no es ninguna de las cuatro establecidas, es decir, el parámetro P2 toma un valor superior a 4.
- WRONG_P1P2= "0x6B00": indica que el valor del campo P1 es erróneo, es decir, apunta a un sector que se encuentra fuera de rango tomando un valor superior a 39.

7.1.2 COMANDOS DE ENTRADA Y SALIDA DE DATOS

En el caso de los comandos de entrada y salida de datos encontraremos tres clases de comandos: comandos de lectura, comandos de escritura y comando de reset. De todas las clases de comandos tendremos dos versiones del comando; una versión para el nivel de seguridad de usuario, que consiste en comando y respuesta APDU estándar y una versión para

el nivel de seguridad de gestor, que consistirá en un comando y respuesta encapsulados en un comando de gestión remota.

7.1.2.1 COMANDOS DE ENTRADA Y SALIDA DE DATOS ESTÁNDAR

Como ya hemos comentado los comandos de entrada y salida de datos estándar son los que utilizaremos para el nivel de seguridad del usuario. Se trata de comandos únicos que irán cifrados y permiten acceder a múltiples bloques de un mismo sector con un único comando.

7.1.2.1.1 Comando

El comando se utiliza para leer uno o varios bloques no necesariamente consecutivos de un sector especificado.

En caso de que entre esta lectura se incluya el *Sector Trailer* se devolverá, para esta, los 16 bytes con los bytes del 0 al 5 y del 10 al 15, que son los correspondientes a las claves KeyA y KeyB respectivamente, puestos a cero.

Las condiciones de acceso para los bloques de datos y las del bloque de *Sector Trailer* serán ignoradas por esta función y se accederá con una clave. En nuestro caso, como ya comentamos, solo se ignoraran si el administrador al instalar la aplicación así lo dispusiera en cuyo caso se facilitara la clave MF_PASSWORD. Si las condiciones de acceso no se ignoraran habría de facilitarse la clave KeyA o KeyB.

Los bloques que se van a leer vendrán determinados por la combinación del número de sector a leer y un BlockBitMap que determinará los bloques dentro de dicho sector. Los bloques definidos por el BlockBitMap no podrán exceder los límites del sector (no se podrá leer un bloque 4 en un sector que solo tenga bloques del 0 al 3) y por tanto para sectores de 4 bloques el valor del BlockBitMap siempre será igual o inferior a 0x000F y si se detecta que es mayor se devolverá un error de parámetro incorrecto. La Figura 7-1 muestra un ejemplo de codificación de un BlockBitMap para un sector de 16 bloques.

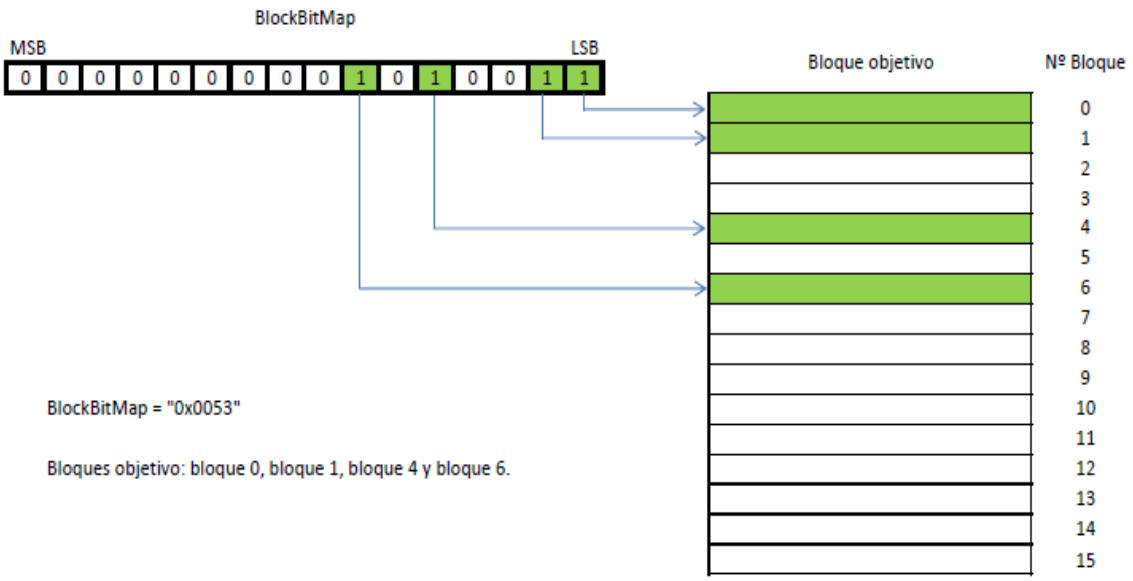


Figura 7-1 - Ejemplo de la codificación de un BlockBitMap para un sector de 16 bloques

Al igual que en caso del comando hemos adaptado el comando de M4M a un comando APDU que es el que utilizaremos con la estructura mostrada en la Tabla 7-3.

Tabla 7-3 - Comando

CLA	INS	P1	P2	Lc	DATOS	Le
84	10	Sector	XX	0A o 08	BlockBitMap (2 bytes), MF_PASSWORD (8 bytes) o KeyA/KeyB (6 bytes)	Variable

Parámetros:

- CLA = “0x84”: código CLA para los comandos que van cifrados.
- INS = “0x10”: valor que corresponde a la instrucción en nuestra aplicación.
- P1: representa el sector que vamos a leer.
- P2: no se utiliza en este comando por lo que su valor puede ser cualquiera.
- Lc: tamaño, en bytes, del campo datos. Puesto que los datos son fijos en tamaño este parámetro también es fijo en valor y puede tomar los valores “0x0A” cuando se proporciona la clave MF_PASSWORD o “0x08” cuando se proporciona una de las claves KeyA o KeyB.
- Datos: se compone de dos partes consecutivas. Los primeros dos bytes están designados al BlockBitMap que nos facilita el poder, con un solo comando, leer varios bloques. Los siguientes 8 o 6 bytes corresponden a la clave correspondiente al sector que queremos leer y será o bien la clave MF_PASSWORD o bien una de las dos claves KeyA o KeyB.
- Le: tamaño, en bytes, de los datos de respuesta esperados. Los valores de este parámetro serán múltiplos de 16, tamaño en bytes de un bloque.

La respuesta al comando se divide en dos partes, la primera los datos leídos, en orden consecutivo, de los bloques que solicitaba el comando y después de estos, la palabra de estado. Al igual que con el comando si el comando se ha procesado correctamente se devuelve el código de estado “0x9000”. En caso contrario se devuelve un código de error.

Códigos de error:

- PARAM_ERROR = “0xD4”: este error se puede dar en tres casos: o bien se ha especificado un sector fuera de rango, o el BlockBitMap toma el valor “0x0000” o el BlockBitMap, refiriéndose a un sector de cuatro bloques, toma un valor superior a “0x000F”. En nuestro caso utilizaremos el código correspondiente según la norma 7816 para el error PARAM_ERROR con valor “0x63D4”.
- NOT_AUTHORIZED = “0xE9”: la contraseña facilitada no corresponde con la clave MF_PASSWORD del sector generada por la tarjeta. Como en los casos anteriores el código de error es el correspondiente según la norma 7816 para el error NOT_AUTHORIZED con valor “0x63E9”.

7.1.2.1.2 Comando

El comando se utiliza para escribir uno o varios bloques no necesariamente consecutivos de un sector especificado.

La estructura del comando es muy parecida a la del comando salvo por la finalidad y por los datos pasados por el comando. Al igual que en el comando las condiciones de acceso del *Sector Trailer* no son tenidas en cuenta por este comando sino que se ejecutara siempre que se facilite la clave MF_PASSWORD correcta en el caso de que la aplicación no tenga en cuenta las condiciones de acceso o una de las claves KeyA o KeyB en el caso de que se tengan en cuenta.

Además, los bloques que van a escribirse se determinan, igual que en el comando , mediante la combinación del sector y un BlockBitMap de igual configuración que en el caso del comando .

La escritura de los bloques es una escritura no atómica, es decir, si durante la escritura hay una interrupción como una desconexión o una pérdida de corriente algunos bloques pueden haber sido actualizados pero no necesariamente haberlo sido todos.

Al igual que en los casos anteriores hemos adaptado el comando de M4M a la estructura final que vemos en la Tabla 7-4.

Tabla 7-4 - Comando

CLA	INS	P1	P2	Lc	DATOS
84	20	Sector	XX	Variable	BlockBitMap (2 bytes), MF_PASSWORD (8 bytes) o KeyA/KeyB (6 bytes), datos a escribir

Parámetros:

- CLA = “0x84”: código CLA para los comandos que van cifrados.
- INS = “0x20”: valor que corresponde a la instrucción en nuestra aplicación.
- P1: representa el sector que vamos a escribir.
- P2: no se utiliza en este comando por lo que su valor puede ser cualquiera.
- Lc: tamaño, en bytes, del campo datos. Únicamente tiene como datos fijos el BlockBitMap y la clave MF_PASSWORD y el resto de datos a escribir será un múltiplo de 16 bytes según el número de bloques a escribir, el valor de este campo es variable.
- Datos: se compone de tres partes consecutivas. Los primeros dos bytes están designados al BlockBitMap que nos facilita el poder, con un solo comando, leer varios bloques. Los siguientes bytes corresponden, bien a la clave MF_PASSWORD o bien a una de las claves KeyA o KeyB, correspondientes al sector que queremos leer. Por último están los datos que queremos escribir, 16 bytes por sector a escribir, que estarán ordenados en orden ascendente empezando por el de menor valor.

La respuesta al comando consiste en la palabra de estado. Al igual que con los comandos anteriores si el comando se ha procesado correctamente se devuelve el código de estado “0x9000”. En caso contrario se devuelve un código de error.

Códigos de error:

- PARAM_ERROR = “0x63D4”: este error se puede dar en cinco casos: si se ha especificado un sector fuera de rango, si el BlockBitMap toma el valor “0x0000”, si el BlockBitMap, refiriéndose a un sector de cuatro bloques, toma un valor superior a “0x000F”; si se intenta escribir el bloque 0 del sector 0 (Manufacturer Block) o si el número de bloques a escribir en el campo de datos (teniendo en cuenta que cada 16 bytes sería un bloque) no coincide con los bloques designados por el BlockBitMap. En nuestro caso utilizaremos el código correspondiente según la norma ISO7816 para el error PARAM_ERROR con valor “0x63D4”.
- NOT_AUTHORIZED = “0x63E9”: la contraseña facilitada no corresponde con la clave MF_PASSWORD del sector generada por la tarjeta. Como en los casos anteriores el código de error es el indicado en la norma 7816 para el error NOT_AUTHORIZED.

7.1.2.2 COMANDOS DE ENTRADA Y SALIDA DE GESTIÓN REMOTA

En nuestro proyecto todos los comandos de gestión remota están destinados al nivel de seguridad del gestor. La principal diferencia entre los comandos que hemos denominado estándar y estos comandos es que, en estos últimos el comando se encuentra codificado en formato TLV y encapsulado dentro de un comando de gestión remota.

Asimismo todos los comandos que se envíen y reciban de esta forma se consideraran siempre cifrados.

7.1.2.2.1 Comando

El comando se utiliza para resetear una lista de sectores de una tarjeta del tipo MIFARE Classic sin la necesidad de facilitar la clave MF_PASSWORD de ninguno de los sectores especificados.

Los sectores afectados por este comando volverán al estado inicial de creación de la tarjeta virtual, es decir, los bloques de datos estarán compuestos por ceros y los *Sector Trailer* tomaran el valor con el que los inicializaron. La única excepción es el bloque 0 del sector 0, el Manufacturer Block, que no sufre ninguna modificación.

La configuración del TLV del comando es:

- Tag = "0x08": código correspondiente al comando .
- Length: depende del número de sectores que vayan a resetearse tomando un valor mínimo de 1 y máximo de 40 (para una tarjeta MIFARE Classic de 4K que cuenta con 40 sectores, en el caso de las de 1K el máximo sería 16).
- Value: lista de los sectores a resetear, concatenados e identificados mediante el número de sector ocupando 1 byte cada uno.

El TLV respuesta es:

- Tag = "0x08": el código de la respuesta debe ser el mismo que el del comando.
- Length = "0x02": valor fijo para esta respuesta.
- Value: únicamente se responde con la palabra de estado.

La palabra de estado siempre tomara el valor "0x9000" en el caso de que la recepción y ejecución del comando haya sido satisfactoria. En caso de que se produzca un error tomará uno de los siguientes valores:

- INCORRECT_PARAM = "0x6A86": el valor del campo length en el comando es 0 o superior a 16 en las tarjetas de 1K o 40 en las tarjetas de 4K.
- PARAM_ERROR = "0x63D4": al menos uno de los números de sector es erróneo, es decir, toman un valor superior a 15 en tarjetas de 1K o a 39 en las tarjetas de 4K.

7.1.2.2.2 Comando

El comando se utiliza para leer múltiples bloques no necesariamente consecutivos de un sector especificado. Si, en la petición de lectura, se requiere leer el *Sector Trailer* este comando devolverá los 16 bytes correspondientes a ese bloque pero los bytes correspondientes a las claves KeyA y KeyB se mostraran siempre como ceros aunque las condiciones de acceso permitan su lectura.

Las condiciones de acceso para la lectura tanto del *Sector Trailer* como de los bloques de datos son ignoradas por este comando, únicamente requiriéndose facilitar la clave MF_PASSWORD, derivada de las claves KeyA y KeyB, de dicho sector.

La configuración del TLV del comando es:

- Tag = “0x07”: código correspondiente al comando .
- Length = “0x11”: tamaño fijo del campo Value para este comando.
- Value: aparecen concatenados el número de sector objetivo (1 byte), el BlockBitMap con la secuencia codificada de bloques a leer (2 bytes) y por último la clave MF_PASSWORD del sector.

El TLV respuesta es:

- Tag = “0x07”: el código de la respuesta debe ser el mismo que el del comando.
- Length: valor variable según el número de bloques leídos. A 16 bytes por bloque, el valor de este campo debe ser múltiplo de 16 más 2 bytes de la palabra de estado.
- Value: aparecen concatenados dos partes, los primeros dos bytes que corresponden a la palabra de estado y el resto grupos de 16 bytes, un grupo por cada bloque leído, con los datos leídos.

La palabra de estado siempre tomara el valor “0x9000” en el caso de que la recepción y ejecución del comando haya sido satisfactoria. En caso de que se produzca un error tomará uno de los siguientes valores:

- INCORRECT_PARAM = “0x6A86”: cuando alguno de los parámetros en el comando no aparece, el tamaño de uno de estos parámetros no es correcto o el BlockBitMap hace referencia a más de 14 bloques al mismo tiempo.
- NOT_ENOUGH_MEM_SPACE = “0x6A84”: no hay espacio suficiente en la respuesta del STORE DATA para almacenar todos los datos pedidos.
- PARAM_ERROR = “0x63D4”: el número de sector especificado es superior a 15 en tarjetas de 1K o a 39 en tarjetas de 4K y por tanto esta fuera de rango o si el BlockBitMap excede el número de bloques soportado del sector especificado, por ejemplo, si toma un valor superior a “0x000F” en un sector de 4 bloques.
- NOT_AUTHORIZED = “0x63E9”: la clave MF_PASSWORD es errónea.

7.1.2.2.3 Comando

El comando se utiliza para escribir múltiples bloques no necesariamente consecutivos de un sector especificado.

Las condiciones de acceso para la escritura tanto del *Sector Trailer* como de los bloques de datos son ignoradas por este comando, únicamente requiriéndose facilitar la clave MF_PASSWORD, derivada de las claves KeyA y KeyB, de dicho sector.

La escritura de los bloques no es atómica, lo que significa que si el comando Update MIFARE Classic Sector se interrumpe por una pérdida de energía o por que la tarjeta se resetea, algunos de los bloques podrían haber sido modificados pero no todos.

La configuración del TLV del comando es:

- Tag = “0x06”: código correspondiente al comando .
- Length: valor variable del campo Value. El tamaño depende del número de bloques que vayan a escribirse, entre 1 como mínimo y 14 como máximo, a razón de 16 bytes cada bloque. Además se cuentan también los 11 bytes fijos del campo datos.
- Value: aparecen concatenados el número de sector objetivo (1 byte), el BlockBitMap con la secuencia codificada de bloques a escribir (2 bytes), la clave MF_PASSWORD del sector y por último los grupos de 16 bytes correspondientes a cada bloque que quiera escribirse, un grupo por cada bloque y ordenados en orden que en el que vayan a escribirse, empezando por el bloque con el número más bajo.

El TLV respuesta es:

- Tag = “0x06”: el código de la respuesta debe ser el mismo que el del comando.
- Length = “0x02”: valor fijo para la respuesta de este comando.
- Value: contiene la palabra de estado.

La palabra de estado siempre tomara el valor “0x9000” en el caso de que la recepción y ejecución del comando haya sido satisfactoria. En caso de que se produzca un error tomará uno de los siguientes valores:

- INCORRECT_PARAM = “0x6A86”: cuando alguno de los parámetros en el comando no aparece, el tamaño de uno de estos parámetros no es correcto o el BlockBitMap hace referencia a más de 14 bloques al mismo tiempo.
- PARAM_ERROR = “0x63D4”: el número de sector especificado es superior a 15 en tarjetas de 1K o a 39 en tarjetas de 4K y por tanto esta fuera de rango o si el BlockBitMap excede el número de bloques soportado del sector especificado, por ejemplo, si toma un valor superior a “0x000F” en un sector de 4 bloques o si uno de los bloques designados es el bloque 0 del sector 0.
- NOT_AUTHORIZED = “0x63E9”: la clave MF_PASSWORD es errónea.

7.1.3 COMANDOS PROPIOS

En los comandos propios recogemos todos los comandos que hemos tenido que crea específicamente para nuestra aplicación, sin basarnos en ningún otro modelo y que aparecen en la versión final de la misma.

Los comandos desarrollados para el nivel de seguridad del emisor, es decir, los comandos de gestión de tarjetas virtuales parten de la idea de los comandos de MIFARE4Mobile para este caso pero dada su funcionalidad y estructura no se puede hablar de comandos modificados a partir de los ya existentes sino comandos totalmente nuevos y propios.

En cuanto a los comandos del nivel de seguridad de usuario puesto que hemos impuesto unas medidas de seguridad propias la gestión de las claves para el cifrado y la verificación del usuario también son originales de nuestro proyecto.

7.1.3.1 COMANDOS DE SELECCIÓN DE TARJETA VIRTUAL

Para poder seleccionar la tarjeta virtual activa primero tendremos que seleccionarla, de esta forma la indicaremos al nivel de seguridad que se identifique después con que datos tiene que trabajar. La selección y des-selección de las tarjetas virtuales la haremos mediante dos comandos que hemos desarrollado llamados y .

El comando selección es un comando simple que identifica la tarjeta que vamos a activar. Está limitado a seleccionar uno de los cuatro huecos disponibles para tarjetas inteligentes y es el primer paso que tenemos que dar para poder autenticarnos en el nivel de seguridad de gestor o usuario.

La estructura del comando aparece representada en la Tabla 7-5.

Tabla 7-5 - Comando

CLA	INS	P1	P2
84	AA	xx	00

Parámetros:

- CLA= “0x84”: código CLA para los comandos que van cifrados.
- INS= “0xAA”: código para la instrucción .
- P1= desde “0x00” hasta “0x03”: valor que nos indica que a que segmento corresponden los datos dentro de la instalación. Cada instalación consta de 3 segmentos.
- P2: ponemos como valor base el cero aunque como no se utilizan durante la interpretación del comando pueden tomar cualquier valor.
- No precisa del campo DATOS

Su respuesta constara solo de la palabra de estado.

Palabra de estado:

- “0x9000”: código que indica que el comando se ha procesado correctamente.
- “0x6A80”: código de error que indica que se ha enviado la petición de selección de activación para una tarjeta por encima del hueco 3.

El comando es un comando muy sencillo que desvincula la tarjeta activa. De esta forma permite que se pueda iniciar sesión en el nivel de seguridad de emisor y que pueda seleccionarse una nueva tarjeta virtual. Su estructura que aparece en la Tabla 7-6.

Tabla 7-6 - Comando

CLA	INS	P1	P2
84	BB	00	00

Parámetros:

- CLA= “0x84”: código CLA para los comandos que van cifrados.
- INS= “0xBB”: código para la instrucción .
- P1 y P2: ponemos como valor base el cero aunque como no se utilizan durante la interpretación del comando pueden tomar cualquier valor.
- No precisa del campo DATOS

Su respuesta constara solo de la palabra de estado.

Palabra de estado:

- “0x9000”: código que indica que el comando se ha procesado correctamente.
- “0x6A80”: código de error que indica que se ha enviado la petición de desselección sin que hubiera una tarjeta activa.

7.1.3.2 COMANDOS DEL NIVEL DE SEGURIDAD DE EMISOR

En primer lugar, se define un comando de creación e instalación de una tarjeta virtual, denominado comando . Para este proyecto, se reserva el espacio equivalente a 10 posibles tarjetas virtuales.

Los espacios de memoria se asignan en orden ascendente, seleccionando el primer hueco disponible. En caso de no tener espacio libre se reportará el consiguiente error.

Para la creación de la tarjeta virtual necesitaremos suministrar los bloques *Sector Trailer* de cada uno de los sectores de cara a los comando que resetean uno o varios sectores, restaurando el valor del sector tráiler al que se facilita durante la instalación.

Como los datos correspondientes a los 40 sectores ocuparían más de una APDU cada comando de creación e instalación de tarjeta virtual se compondrá de cuatro APDUs comando, con sus respectivas respuestas, cada uno de ellos con la información de 10 sectores hasta completar los 40. Una vez se reciba la última respuesta se considerara la tarjeta virtual creada y el hueco para tarjeta virtual que se le ha asignado, ocupado.

La estructura del comando es la que se muestra en la Tabla 7-7.

Tabla 7-7 - Comando

CLA	INS	P1	P2	Lc	DATOS
84	00	xx	00	variable	variable

Parámetros:

- CLA= “0x84”: código CLA para los comandos que van cifrados.
- INS= “0x00”: código para la instrucción .
- P1= desde “0x00” hasta “0x02”: valor que nos indica que a que segmento corresponden los datos dentro de la instalación. Cada instalación consta de 3 segmentos.
- P2: ponemos como valor base el cero aunque como no se utilizan durante la interpretación del comando pueden tomar cualquier valor.
- Lc= variable: el primer segmento tomara el valor “0x40” y los dos siguiente el valor “0x80”.
- DATOS: el primer segmento contendrá el *Manufacturer Block* de la nueva tarjeta virtual y las 3 claves para crear un canal seguro. Los dos segmentos restantes contendrán cada uno los *Sector Trailer* de 8 segmentos ordenados de forma consecutiva comenzando por el del sector 0.

Dependiendo del segmento que hayamos enviado se recibirán dos respuestas distintas. En todos los casos se devolverá una palabra de estado indicando si se ha procesado correctamente el comando o, por el contrario, ha surgido algún error. En el último de los 3 comandos se devolverá además un byte indicando el hueco del array de tarjetas virtuales que ocupa la tarjeta virtual recién creada. La respuesta tomará por tanto los siguientes valores:

- VCx = “0xXX”: (solo en la respuesta del comando del último segmento) valor de 1 byte que nos indica que hueco ocupa la tarjeta virtual recién instalada.

Palabra de estado:

- “0x9000”: código que indica que el comando se ha procesado correctamente.
- “0x6A80”: código de error que indica que se ha enviado un comando que hace referencia a un segmento por encima de los cuatro que se necesitan, es decir, el valor de P1 suministrado es superior a 3.
- “0x6700”: código de error que indica que la longitud del campo de datos es errónea.
- “0x6A84”: código de error que indica que todos los huecos para almacenar las tarjetas virtuales se encuentran ocupados y no se puede crear ninguna tarjeta virtual nueva.

Del mismo modo que se define un comando de creación de tarjetas virtuales, se debe implementar su opuesto, de forma que se habilite la liberación de la memoria de aquellas tarjetas que ya no estén en uso. Este comando lo llamaremos comando .

La estructura de este comando se encuentra reflejada en la Tabla 7-8.

Tabla 7-8 - Comando

CLA	INS	P1	P2	Lc	DATOS
84	01	00	00	01	Número del hueco a eliminar (1 byte)

Parámetros:

- CLA = “0x84”: código CLA para los comandos que van cifrados.
- INS = “0x01”: código para la instrucción .
- P1 y P2:= ponemos como valor base el cero aunque como no se utilizan durante la interpretación del comando pueden tomar cualquier valor.
- Lc = “0x01”: siempre tomara el mismo valor ya que el campo de datos siempre contendrá un solo byte.
- DATOS: valor numérico de la posición del hueco donde reside la tarjeta virtual que queremos eliminar.

En la APDU respuesta obtenemos solamente la palabra de estado dependiendo de cómo se haya procesado el comando:

- “0x9000”: código que indica que el comando se ha procesado correctamente.
- “0x6A80”: código de error que indica que se está haciendo referencia a un número de hueco más allá del rango que hemos establecido, es decir, que se está intentando desinstalar una tarjeta virtual que se encuentra en un número de hueco por encima del 10, este inclusive.
- “0x6984”: código de error que indica que se está intentando eliminar una tarjeta virtual de un hueco que actualmente se encuentra disponible y por tanto no hay tarjeta que desinstalar.

7.1.3.3 Comandos del nivel de seguridad de usuario

El primer paso que se tiene que realizar durante este nivel de seguridad es validar el PIN y por tanto se requiere de un comando nuevo.

La estructura del comando es la que se muestra en la Tabla 7-9.

Tabla 7-9 - Comando

CLA	INS	P1	P2	Lc	DATOS
80	88	00	00	08	PIN (8 bytes)

Parámetros:

- CLA = “0x80”: valor de CLA usado también en el comando INITIALIZE UPDATE, el primero de los dos comandos usados para la apertura de un canal seguro.

- INS = “0x88”: código para la instrucción .
- P1 y P2 = “0x00”: ponemos como valor base el cero aunque como no se utilizan durante la interpretación del comando pueden tomar cualquier valor.
- Lc = “0x08”: siempre tomara el mismo valor ya que el campo de datos únicamente contendrá el PIN de 8 bytes.
- DATOS: valor de 8 bytes que constituyen el número de identificación personal (PIN).

La respuesta a este comando consiste en la palabra de estado donde nos indica el correcto procesamiento del comando o, en su defecto, los errores que se han generado. En este caso la palabra de estado solo tomara dos posibles valores:

- “0x9000”: indica que el comando se ha procesado correctamente y que tras el procesado de este comando todos los mensajes serán encriptados.
- “0x63E9”: indica que el número PIN que se ha facilitado no se corresponde con el que se encuentra almacenado en la aplicación.

Puesto que disponemos de un número único de 8 bytes que nos sirve para identificar al usuario y cifrar los datos, se hace necesario que dicho número pueda cambiar su valor, a discreción del propio usuario y siempre una vez este esté autenticado. Para ello diseñamos otro comando para modificar esta contraseña. Por motivos de seguridad se volverá a pedir y evaluar el PIN actual antes de realizar el cambio.

La estructura de este comando es la que se muestra en la Tabla 7-10.

Tabla 7-10 - Comando

CLA	INS	P1	P2	Lc	DATOS
84	80	00	00	10	PIN actual (8 bytes), PIN nuevo (8 bytes)

Parámetros:

- CLA = “0x84”: Valor de CLA para los comandos que van cifrados.
- INS = “0x80”: Código para la instrucción de .
- P1 y P2 = “0x00”: ponemos como valor base el cero aunque como no se utilizan durante la interpretación del comando pueden tomar cualquier valor.
- Lc = “0x10”: siempre tomara el mismo valor ya que el campo de datos contendrá dos PINs y por tanto será de 16 bytes.
- DATOS: campo formado por la concatenación del PIN actual y el PIN nuevo.

La respuesta a este comando consiste en la palabra de estado donde nos indica el correcto procesamiento del comando o, en su defecto, los errores que se han generado. En este caso la palabra de estado solo tomara dos posibles valores:

- “0x9000”: indica que el comando se ha procesado correctamente y que, a partir de la recepción de esta respuesta los mensajes irán encriptados con el nuevo PIN.

- “0x63E9”: indica que el número PIN que se ha facilitado como el PIN actual no se corresponde con el que se encuentra almacenado en la aplicación.

7.2 WORKBENCHS

En el diseño de los *workbenchs* para realizar las pruebas de la aplicación que hemos diseñado se destacan 3 casos distintos, uno por cada nivel de seguridad: nivel de seguridad de emisor, nivel de seguridad de gestor y nivel de seguridad de usuario.

Para facilitar la lectura de las capturas nos guiaremos por un código de colores según el tipo de comando que hemos utilizado, de la siguiente forma:

- (Magenta): Comandos de apertura de canal seguro y selección de tarjeta virtual.
- (Azul): Comandos específicos del nivel de seguridad de emisor.
- (Rojo): Comandos específicos del nivel de seguridad de gestor (comandos de *Remote Access*).
- (Naranja): Comandos específicos del nivel de seguridad de usuario.
- (Verde): Comandos de lectura.
- (Amarillo): Comandos de escritura.
- (Gris): Comandos no especificados utilizados para ver un error.

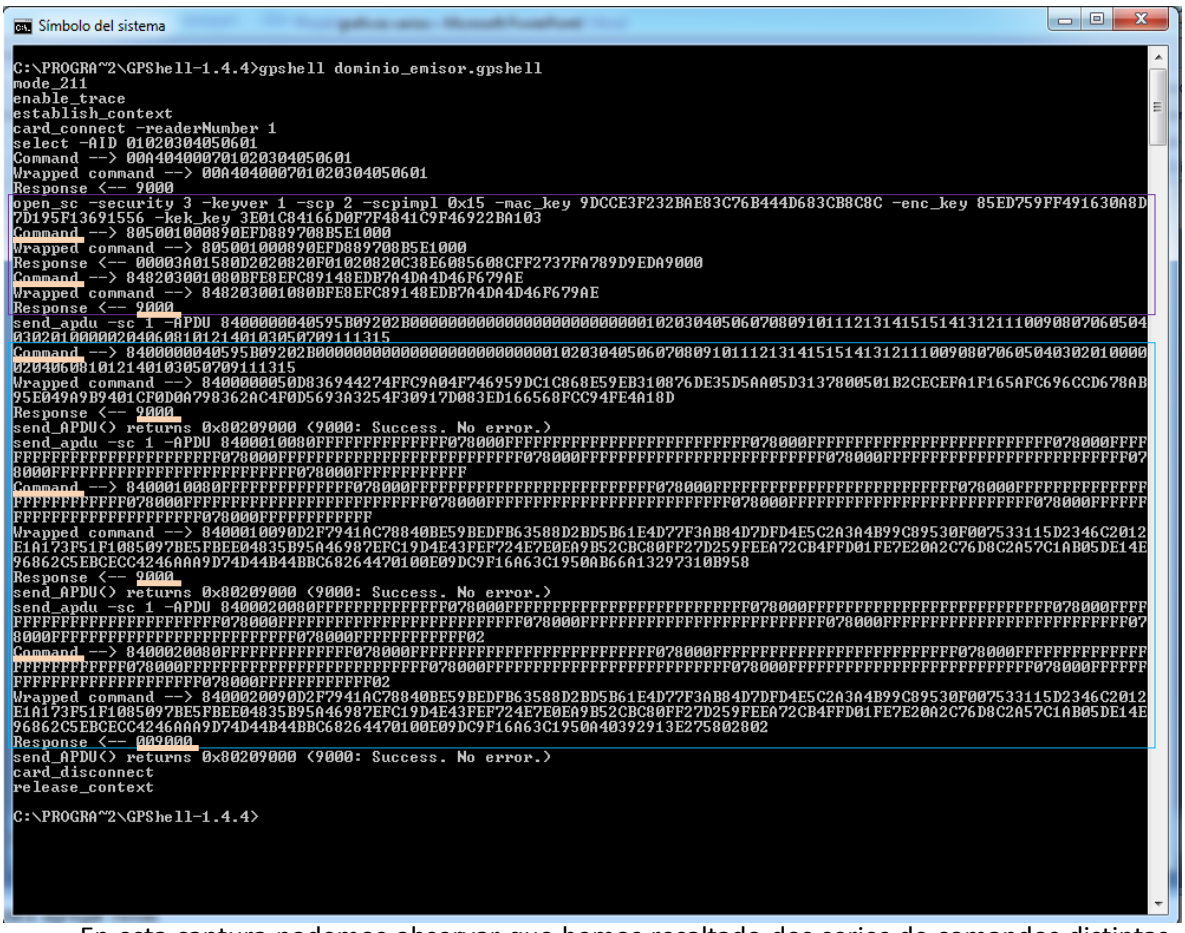
Cabe recordar que en el nivel de seguridad de gestor todos los comandos se envían como comandos de acceso remoto, excepto los dos comandos de la apertura del canal seguro.

7.2.1 NIVEL DE SEGURIDAD DE EMISOR

El nivel de seguridad de emisor se encarga de crear, instalar y borrar las tarjetas virtuales. La creación e instalación se hacen al mismo tiempo y para la creación de las tarjetas virtuales siempre se utilizara el primer hueco disponible, empezando desde el número cero en adelante.

Se realizaran tres pruebas para comprobar su correcto funcionamiento: una instalación aislada, una desinstalación y posterior instalación y cinco instalaciones seguidas para superar el límite máximo de huecos (4 es el máximo número de tarjetas virtuales que podemos tener instaladas).

7.2.1.1 Creación e instalación de una tarjeta virtual



En esta captura podemos observar que hemos resaltado dos series de comandos distintas. La primera serie se corresponde a la apertura de un canal seguro, un procedimiento que, en este nivel de seguridad, siempre tiene que realizarse antes de intentar cualquier otra acción. La segunda serie pertenece a los tres comandos propios de una creación e instalación de una tarjeta virtual tal y como se especifica en 4.5.1-Comandos del nivel de seguridad de emisor

En la apertura de canal lógico hay que fijarse como el parámetro P1 toma el valor “0x01” indicando que la apertura del canal seguro pertenece al nivel de seguridad de emisor. La negociación se realiza de forma automática entre GPShell y el dominio de seguridad de emisor de la tarjeta.

En el segundo grupo, los tres comandos pertenecientes a la creación e instalación de la tarjeta virtual, se puede apreciar cómo se van enviando cada segmento de los *Sector Trailer*s fijándose en como el parámetro P1 toma valores desde el “0x00” al “0x03” ambos inclusive. Los *Sector Trailer* que se cargan son los que ya se encuentran dentro de la tarjeta MIFARE Classic y en todos los casos corresponde con la configuración de transporte. La respuesta final que obtenemos, tras el tercer y último comando del grupo, es la que nos indica que hueco ocupa la tarjeta virtual recién creada, en este caso ocupa el hueco número “0x00”, es decir, el primero de los huecos. En este caso no había aún ninguna tarjeta virtual instalada por lo que este resultado era el esperado.

7.2.1.2 ELIMINACIÓN Y POSTERIOR CREACIÓN DE UNA TARJETA VIRTUAL

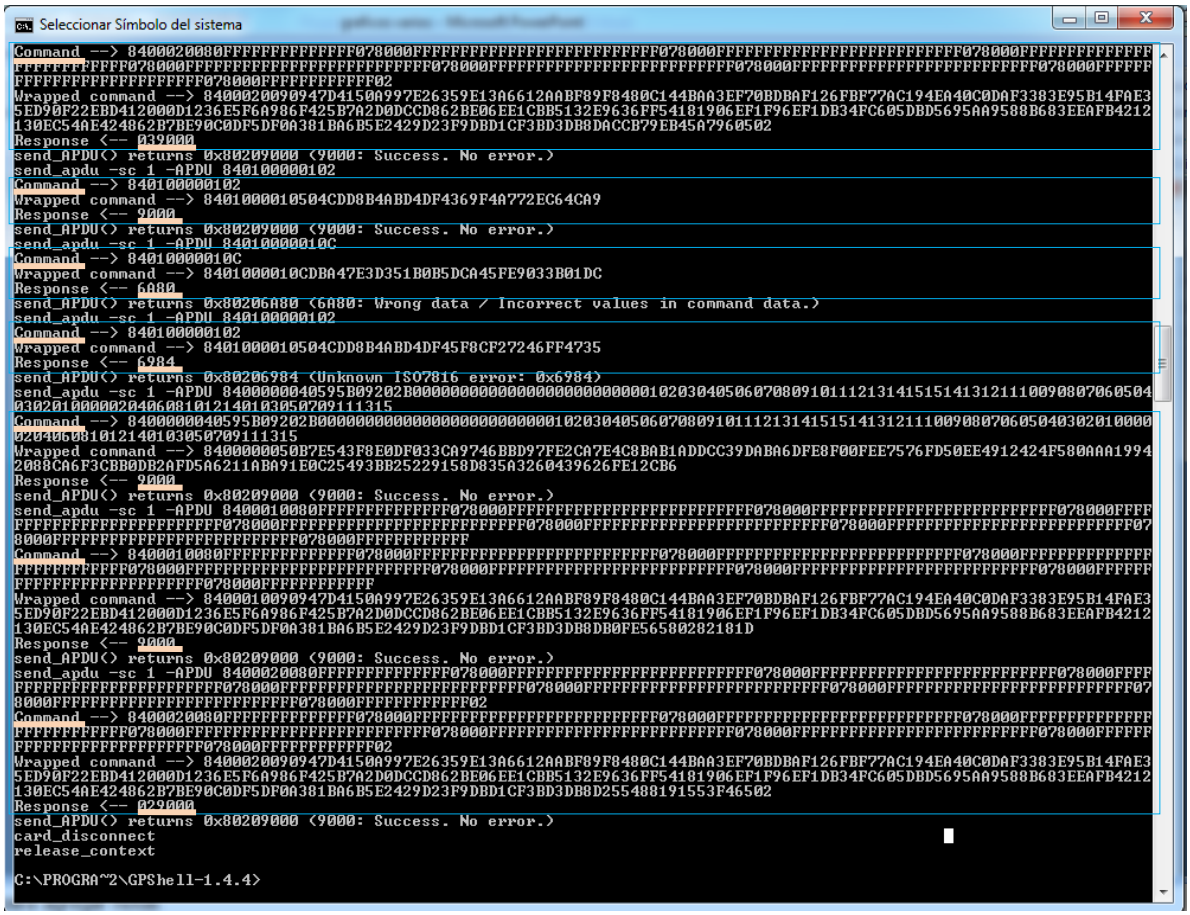


Figura 7-3 - Captura de eliminación y posterior creación de una tarjeta virtual

En este caso podemos observar cinco grupos distintos de comandos. El primero de los grupos es la respuesta al último de los tres comandos de creación de una tarjeta virtual, los tres siguientes son tres comandos de desinstalación de una tarjeta virtual con distintos parámetros y el quinto es el grupo de los tres comandos de creación de una tarjeta virtual.

El orden en el que aparece es para reflejar que realmente el hueco que vamos a desinstalar efectivamente se libera cuando se recibe el comando. Para ello hemos creado anteriormente varias tarjetas virtuales. Como se puede observar, el primer cuadro resaltado pertenece a la instalación de una tarjeta virtual, en concreto la tarjeta virtual que se almacenara en el hueco número 3. Con esta referencia y dado el orden de creación e instalación de las tarjetas virtuales sabemos que, si se ha instalado en el hueco 3 es porque los huecos del 0 al 2, ambos inclusive, estaban completos.

Los tres siguientes cuadros son varios intentos de desinstalación con diferente número de hueco objetivo y todas siguiendo la estructura anteriormente explicada 4.5.1-Comandos del nivel de seguridad de emisor. Como se puede observar es en el campo de datos donde encontramos la diferencia entre una y otra y los diferentes comandos nos servirán para probar, no solo que una tarjeta virtual se desinstala correctamente sino también para

demostrar que la aplicación responde con los errores oportunos cuando los datos introducidos no son válidos.

El primero de los comandos de desinstalación tiene como objetivo el hueco número 2. Como ya hemos comentado como a la última tarjeta virtual instalada se le ha asignado el hueco número 3 estamos seguros que el hueco número 2 está ocupado por una tarjeta virtual y por tanto este comando debería de eliminarla. Obtenemos como respuesta un "0x9000" que nos indica que el procesamiento del comando ha sido correcto pero no nos asegura que haya funcionado, para ello más adelante haremos una instalación que debería de crearse en el hueco que ahora ha quedado libre.

El segundo de los comandos actúa sobre el hueco número 12. Como ya hemos comentado anteriormente nuestro proyecto se ha diseñado con espacio para 4 tarjetas virtuales y por tanto un hueco superior al número 3 estaría fuera de límites. Obtenemos como respuesta "0x6A80" que nos indica que el dato que hemos suministrado en el campo de datos del comando no es válido, en este caso por encontrarse fuera de límites.

El último de los comandos de desinstalación vuelve a solicitar una desinstalación sobre el hueco número 2. En el primero de los comandos de desinstalación ya se liberó dicho hueco y por tanto ahora estaría pidiendo una desinstalación en un hueco vacío. La respuesta "0x6984" obtenida nos indica que el dato proporcionado no es válido indicándonos que el hueco sobre el que queremos realizar la desinstalación en este caso ya está vacío.

Por último, tenemos el grupo de tres comandos para la instalación de una nueva tarjeta virtual. Si nos fijamos en la respuesta al último comando aparece que la instalación de la nueva tarjeta virtual se ha realizado sobre el hueco número 2 tal y como estaba previsto. Así demostramos que el comando de desinstalación ha funcionado correctamente.

7.2.1.3 Error al superar el límite máximo de huecos

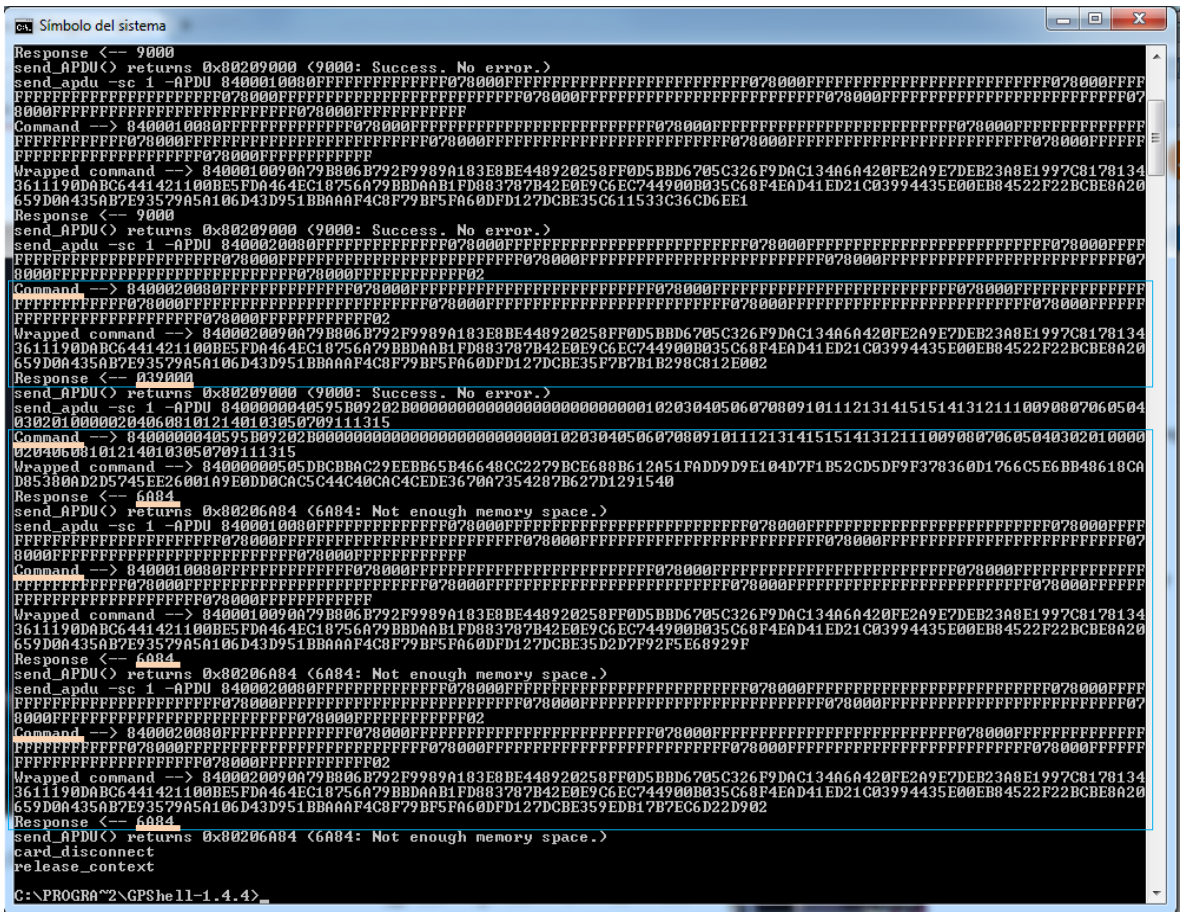


Figura 7-4 - Captura de intento de instalación sin huecos libres

En la captura podemos apreciar dos conjuntos de comandos resaltados. Ambos conjuntos están formados por comandos de instalación de una tarjeta virtual. En este caso queremos comprobar que sucede si intentamos crear una nueva tarjeta virtual cuando ya tenemos todos los huecos ocupados y por tanto el primer paso que hemos tomado ha sido rellenar todos esos huecos.

El primer grupo de comandos que vemos en la captura corresponde al último de los rellenos que hemos efectuado. Como podemos observar la respuesta al último comando de instalación, con valor “0x039000” nos dice que la tarjeta virtual ocupa la posición número tres y se ha creado satisfactoriamente. Debemos recordar que para nuestro proyecto solo contamos con cuatro huecos para instalar tarjetas virtuales y que empiezan a numerarse por el hueco cero. Por tanto con esta instalación queda confirmado que el último hueco se ha instalado y todos los demás huecos están ocupados.

Una vez tenemos los huecos ocupados lanzamos nuevamente los comandos de instalación para comprobar si efectivamente la aplicación no nos permite seguir instalando una vez esté completa. En nuestro caso y dado que lo hacemos con un programa externo lanzamos los tres comandos seguidos, en la práctica, con una aplicación propia, cada respuesta debe de ser analizada antes de lanzar otro comando por lo que únicamente se enviaría el primer comando

de instalación y se comprobaría que la memoria está llena. Para todos los comandos recibimos la misma respuesta “0x6A84” que nos indica que no se dispone de espacio suficiente en memoria lo que significa que todos los huecos disponibles para tarjetas virtuales están ocupados.

7.2.2 NIVEL DE SEGURIDAD DE GESTOR

El nivel de seguridad de gestor es el encargado de gestionar la tarjeta virtual creada, no puede modificar el tamaño o posición de ninguna tarjeta virtual, ya que es competencia exclusiva del nivel de seguridad de emisor, pero si puede modificar su contenido. En este nivel se podrán controlar todos los aspectos de una tarjeta virtual, pudiendo un mismo gestor controlar varias tarjetas virtuales siempre y cuando disponga de las claves correspondientes.

Exceptuando los comandos de apertura del canal seguro y selección y des-selección de tarjeta virtual, todos los comandos y respuestas se transmitirán en formato de acceso remoto 4.4.2-Comandos de MIFARE4Mobile.

Para las pruebas en todos los casos habrá que abrir un canal seguro para que pueda haber comunicación en este nivel. La apertura, negociación y establecimiento del canal seguro en este caso se hace desde el programa GPSHELL para los comandos. Las respuestas al establecimiento del canal seguro no corren a cargo del dominio de seguridad principal, como era el caso del nivel de seguridad de emisor, sino del dominio de seguridad que hemos creado nosotros. Al no realizarse de forma automática hay un especial interés en que se gestione correctamente en las pruebas para comprobar que está bien programado y actua conforme a la especificación de GlobalPlatform.

Como ya hemos comentado durante la especificación y el desarrollo no solo los comandos sino también las respuestas estarán cifradas. Como este caso no está recogido por los protocolos de GlobalPlatform, GPSHELL no está diseñado para revertir el cifrado en las respuestas y será trabajo de la aplicación que se comunique con la tarjeta implementar esta función.

Para poder observar los resultados vamos a modificar las respuestas APDU para que, en lugar de enviar la respuesta cifrada esta se pase primero por el dominio de seguridad, que le retira el cifrado y se envía ya en claro. De esta forma no solo podremos observar que las respuestas obtenidas son las que esperábamos sino que además comprobaremos que el cifrado de la respuesta se hace de forma correcta. No enviamos la parte cifrada y la que va en claro a la vez debido a la limitación de espacio del buffer APDU, que podríamos llegar a llenar rápidamente con varios comandos simultáneos en un único comando de acceso remoto.

Dada la capacidad que tenemos con los comandos de acceso remoto podemos realizar multitud de pruebas simultáneas utilizando una sola APDU y de una misma vez. Las pruebas se recogerán en dos capturas distintas. Como la cantidad de información generada puede ser difícil de seguir vamos a descomponer cada comando y respuesta recibida para facilitar la lectura y comprensión de los resultados.

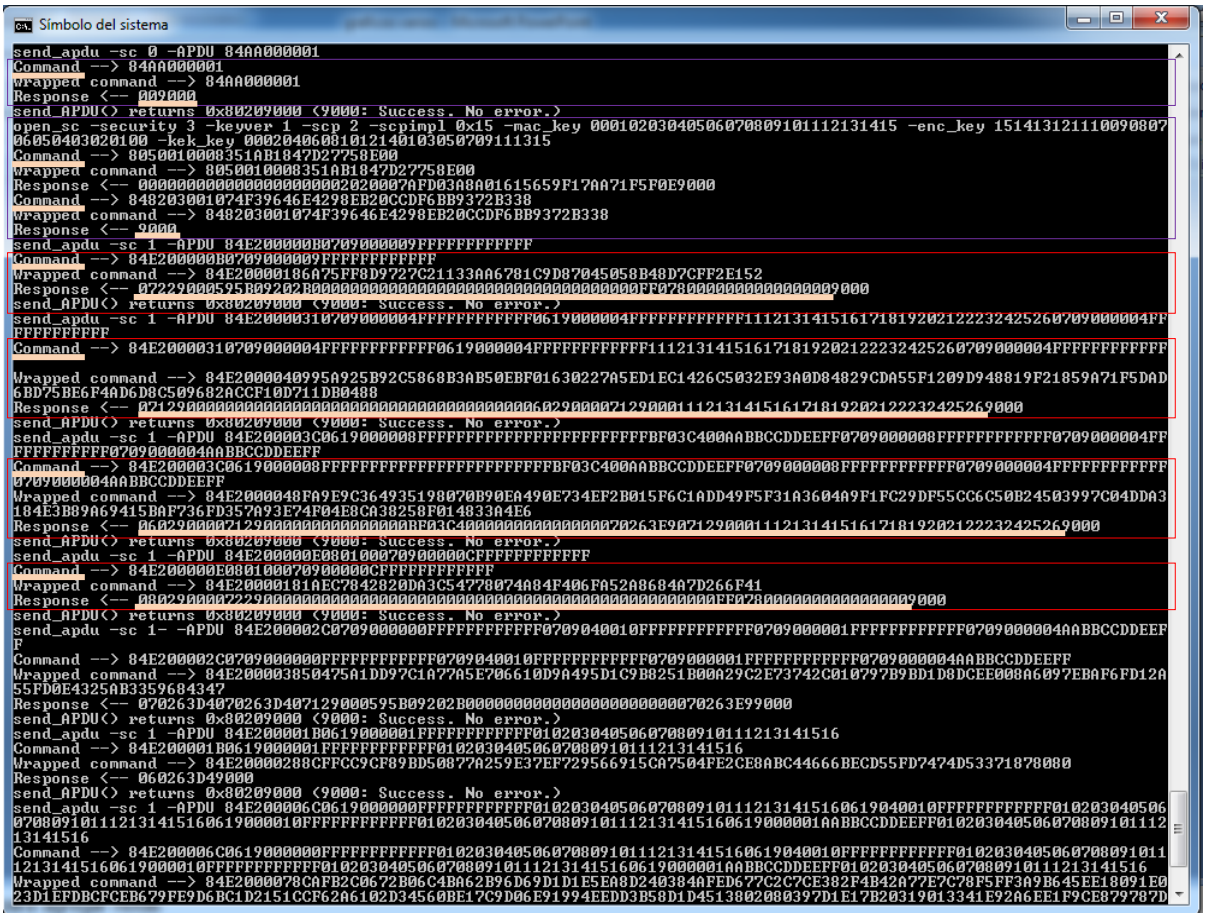


Figura 7-5 - Captura de las pruebas realizadas en el nivel de seguridad de gestor (1)

Observando la primera captura podemos contar seis grupos de mensajes APDU distintos que descomponemos[10][11] a continuación:

El primer grupo representa la selección de tarjeta virtual. Este comando es necesario en los niveles de seguridad de gestor y usuario y debe ser el primero, ya que es el que le indica al dominio de seguridad la tarjeta sobre la que se va a operar y las claves asociadas al canal seguro. En la respuesta aparece como confirmación la tarjeta virtual seleccionada y la palabra de estado de procesado correcto. Para las pruebas que vamos a realizar la tarjeta virtual designada es la que ocupa el espacio 0.

El segundo grupo representa la apertura del canal seguro que se utilizara durante el resto de la sesión. Al igual que en el caso del nivel de seguridad de emisor consiste en dos comandos que identifican al lector y a la tarjeta y aseguran la creación de unas claves de sesión. Sin embargo, a diferencia del nivel de seguridad de emisor, las respuestas de la tarjeta las gestiona un dominio de seguridad propio diseñado por nosotros y que haría referencia al dominio de seguridad que se instalaría junto con la tarjeta virtual al crearse esta.

[10] Todos los mensajes APDU descompuestos se muestran en formato hexadecimal

[11] Todos los comandos de acceso remoto comparten la cabecera únicamente variando el campo Lc y por tanto no los incluimos en la descomposición de los comandos.

Para el resto de grupos descompondremos primero la pareja de APDUs y tras cada pareja haremos un breve análisis de los datos enviados y obtenidos con el fin de facilitar al máximo la comprensión de dichos APDUs. Las parejas de APDUs serían las siguientes:

- Comando: 84E200000B0709000009FFFFFFFFFFFF
    - Comando acceso remoto: 0709000009FFFFFFFFFFFF
        - Etiqueta: 07 (comando de lectura)
        - Longitud: 09 (9 bytes)
        - Campo de datos:
            - 00: sector objetivo
            - 0009: Block Bit Map, bloques 0 y *Sector Trailer*
            - FFFFFFFFFFFF: clave de acceso al bloque
- Respuesta: 07229000595B09202B00000000000000000000000000000000FF0780000000000000009000
    - Respuesta de acceso remoto: 07229000595B09202B00000000000000000000000000000000FF0780000000000000000
        - Etiqueta: 07 (comando de lectura)
        - Longitud: 12 (18 bytes)
        - Campo de datos:
            - 9000: palabra de estado (procesado correcto)
            - 595B09202B00000000000000000000: datos leídos bloque 0
            - 000000000000FF078000000000000000: datos leídos bloque *Sector Trailer*

Realizamos una lectura de los bloques 0 (*Manufacturer Block*) y *Sector Trailer* del sector 0. Es un método simple de comprobar que se lee ya que el *Manufacturer Block* es el único bloque que sabemos a ciencia cierta que siempre tiene que tener un valor distinto de cero 3.3.1.3.1-Manufacturer Block y que su información es inalterable. Por otro lado también comprobamos como, tal y como especifica MIFARE4Mobile 4.4.2.1-Comando readSector, solo devuelve en claro las condiciones de acceso mientras pone el valor de las claves a cero. En la respuesta comprobamos que el comando se ha procesado correctamente y obtenemos la información de dichos bloques.

- Comando: 84E20000310709000004FFFFFFFFFFFF0619000004FFFFFFFFFFFF1112131415161718192021222324252607090000004FFFFFFFFFFFF
    - Comando acceso remoto: 0709000004FFFFFFFFFFFF

        - Etiqueta: 07 (comando de lectura)
        - Longitud: 09 (9 bytes)
        - Campo de datos:
            - 00: sector objetivo
            - 0004: Block Bit Map, bloque 2
            - FFFFFFFFFFFF: clave de acceso al bloque
    - Comando acceso remoto: 0619000004FFFFFFFFFFF1112131415161718192021222324252 6
        - Etiqueta: 06 (comando de escritura)
        - Longitud: 19 (25 bytes)
        - Campo de datos:
            - 00: sector objetivo
            - 0004: Block Bit Map, bloque 2
            - FFFFFFFFFFFF: clave de acceso al bloque
            - 1112131415161718192021222324252 6: datos a escribir
    - Comando acceso remoto: 0709000004FFFFFFFFFFF
        - Etiqueta: 07 (comando de lectura)
        - Longitud: 09 (9 bytes)
        - Campo de datos:
            - 00: sector objetivo
            - 0004: Block Bit Map, bloque 2
            - FFFFFFFFFFFF: clave de acceso al bloque
- Respuesta: 0712900000000000000000000000000000000000602900007129000111213141516171819202122232425269000
    - Respuesta de acceso remoto: 071290000000000000000000000000000000000000
        - Etiqueta: 07 (comando de lectura)
        - Longitud: 12 (18 bytes)
        - Campo de datos:
            - 9000: palabra de estado (procesado correcto)
            - 00000000000000000000000000000000: datos leídos bloque 2.

- Respuesta acceso remoto: 06029000
  - Etiqueta: 06 (comando de escritura)
  - Longitud: 02 (2 bytes)
  - Campo de datos:
    - 9000: palabra de estado (procesado correcto)
- Respuesta de acceso remoto: 0712900011121314151617181920212223242526
  - Etiqueta: 07 (comando de lectura)
  - Longitud: 12 (18 bytes)
  - Campo de datos:
    - 9000: palabra de estado (procesado correcto)
    - 111213141516171819020212223242526: datos leídos bloque 2.

Esta prueba consiste en varios comandos concatenados dentro de un mismo comando de acceso remoto. Por un lado nos sirve para comprobar que los comandos, a pesar de ir en un mismo comando APDU, se procesar individualmente y que el comando de escritura realmente escribe. Para ello el primero de los comandos es una lectura sobre el bloque 2 del sector 0, el cual se comprueba que está vacío.

Para afianzar este resultado los dos comandos siguientes son una escritura y una lectura sobre el mismo bloque. Con la primera cambiamos los datos y con la segunda volvemos a leerlos. Tal y como obtenemos en las respuestas ambos comandos se procesan correctamente y comprobamos que los datos del bloque 2 efectivamente se han modificado.

- Comando: 84E200003C0619000008FFFFFFFFFFFFFFFFFFFFFFFFBF03C400AABBCCDDEEFF0709000008FFFFFFFFFFFF0709000004FFFFFFFFFFFF0709000004AABBCCDDEEFF
  - Comando acceso remoto: 0619000008FFFFFFFFFFFFFFFFFFFFFFFBF03C400AABBCCDDEEFF
    - Etiqueta: 06 (comando de escritura)
    - Longitud: 19 (25 bytes)
    - Campo de datos:
      - 00: sector objetivo
      - 0008: Block Bit Map, bloque *Sector Trailer*
      - FFFFFFFFFFFF: clave de acceso al bloque
      - FFFFFFFFFFFFBF03C400AABBCCDDEEFF: datos a escribir
  - Comando acceso remoto: 0709000008FFFFFFFFFFFF
    - Etiqueta: 07 (comando de lectura)

        - Longitud: 09 (9 bytes)
        - Campo de datos:
            - 00: sector objetivo
            - 0008: Block Bit Map, bloque *Sector Trailer*
            - FFFFFFFFFFFF: clave de acceso al bloque
    - Comando acceso remoto: 0709000004FFFFFFFFFFF
        - Etiqueta: 07 (comando de lectura)
        - Longitud: 09 (9 bytes)
        - Campo de datos:
            - 00: sector objetivo
            - 0004: Block Bit Map, bloque 2
            - FFFFFFFFFFFF: clave de acceso al bloque
    - Comando acceso remoto: 0709000004AABBCCDDEEFF
        - Etiqueta: 07 (comando de lectura)
        - Longitud: 09 (9 bytes)
        - Campo de datos:
            - 00: sector objetivo
            - 0004: Block Bit Map, bloque 2
            - AABBCCDDEEFF: clave de acceso al bloque
- Respuesta: 060290000712900000000000000BF03C4000000000000000070263E907129000111213141516171819202122232425269000
    - Respuesta acceso remoto: 06029000
        - Etiqueta: 06 (comando de escritura)
        - Longitud: 02 (2 bytes)
        - Campo de datos:
            - 9000: palabra de estado (procesado correcto)
    - Respuesta de acceso remoto: 071290000000000000000BF03C400000000000000
        - Etiqueta: 07 (comando de lectura)
        - Longitud: 12 (18 bytes)
        - Campo de datos:
            - 9000: palabra de estado (procesado correcto)

- 000000000000BF03C400000000000000: datos leídos bloque 2.

  - Respuesta de acceso remoto: 070263E9
    - Etiqueta: 07 (comando de lectura)
    - Longitud: 02 (2 bytes)
    - Campo de datos:
      - 63E9: palabra de estado (no autorizado, clave no válida)
  - Respuesta de acceso remoto: 071290001112131415161718192021222324252 6
    - Etiqueta: 07 (comando de lectura)
    - Longitud: 12 (18 bytes)
    - Campo de datos:
      - 9000: palabra de estado (procesado correcto)
      - 111213141516171819202122232425 26: datos leídos bloque 2.

Esta prueba nos sirve para comprobar cómo las restricciones impuestas por las condiciones de acceso se cumplen y como alterándolas podemos modificar el acceso a los bloques. Para ello lo primero que hacemos es modificar las condiciones de acceso para un bloque particular, en este caso el bloque 2 del sector 0. El primer comando es una escritura del *Sector Trailer* del sector 0 que modifica la clave Key B y las condiciones de acceso de tal forma que la lecturas y escritura del bloque 2 únicamente pueden realizarse con la clave Key B y puesto que en la configuración inicial la Key A y la Key B son iguales también tenemos que modificar la Key B.

Una vez hemos modificado el *Sector Trailer* comprobamos que el cambio ha surtido efecto y mandamos un comando para que lea el contenido de dicho bloque. En la respuesta podemos comprobar que las condiciones de acceso efectivamente se han modificado y, aunque las claves estén puestas a cero en la respuesta, podemos asumir que la clave Key B se ha modificado igualmente, aunque solo tenemos una forma de asegurarnos.

Procedemos a leer el bloque 2 del sector 0, primero con la clave antigua (que sería la clave Key A en este momento) y luego con la nueva clave Key B. En la respuesta podemos comprobar cómo para el primer comando de lectura obtenemos un error de acceso no autorizado y sin embargo en el segundo podemos leer el contenido sin problemas. De esta forma demostramos que efectivamente las condiciones de acceso se tienen en cuenta a la hora de leer y escribir (la escritura la suponemos ya que en la comprobación de las condiciones de acceso el código de lectura y escritura son iguales).

- Comando: 84E200000E080100070900000CFFFFFFFFFFFF
  - Comando acceso remoto: 080100
    - Etiqueta: 08 (comando de reset)
    - Longitud: 01 (1 bytes)

        - Campo de datos:
            - 00: sector objetivo
    - Comando acceso remoto: 070900000CFFFFFFFFFFFF
        - Etiqueta: 07 (comando de lectura)
        - Longitud: 09 (9 bytes)
        - Campo de datos:
            - 00: sector objetivo
            - 000C: Block Bit Map, bloques 2 y *Sector Trailer*
            - FFFFFFFFFFFF: clave de acceso al bloque
- Respuesta: 080290000722900000000000000000000000000000000000000000000000000000FF07800000000000000009000
    - Respuesta de acceso remoto: 08029000
        - Etiqueta: 08 (comando de reset)
        - Longitud: 02 (2 bytes)
        - Campo de datos:
            - 9000: palabra de estado (procesado correcto)
    - Respuesta de acceso remoto: 0722900000000000000000000000000000000000000000000000FF078000000000000000
        - Etiqueta: 07 (comando de lectura)
        - Longitud: 22 (34 bytes)
        - Campo de datos:
            - 9000: palabra de estado (procesado correcto)
            - 00000000000000000000000000000000: datos leídos bloque 2.
            - 000000000000FF078000000000000000: datos leídos *Sector Trailer*

Una vez comprobado que se evalúan correctamente las condiciones de acceso nos interesa volver al *Sector Trailer* original para continuar con las pruebas. Si bien podríamos volver a escribirlo, es una buena ocasión para comprobar el comando de reset que, no solo devolvería el valor del *Sector Trailer* al valor original (aquel con el que se creó la tarjeta virtual) sino que además pondría el resto de bloques a cero. El primer comando de esta prueba es por tanto el comando de reset del sector 0 seguido de un comando de lectura de los bloques 1 y *Sector Trailer* para comprobar que efectivamente se han reseteado. En la respuesta podemos

observar como el comando de reset se procesa correctamente y cuando hacemos la lectura el bloque 2 se encuentra vacío y el *Sector Trailer* ha recuperado su configuración de transporte.

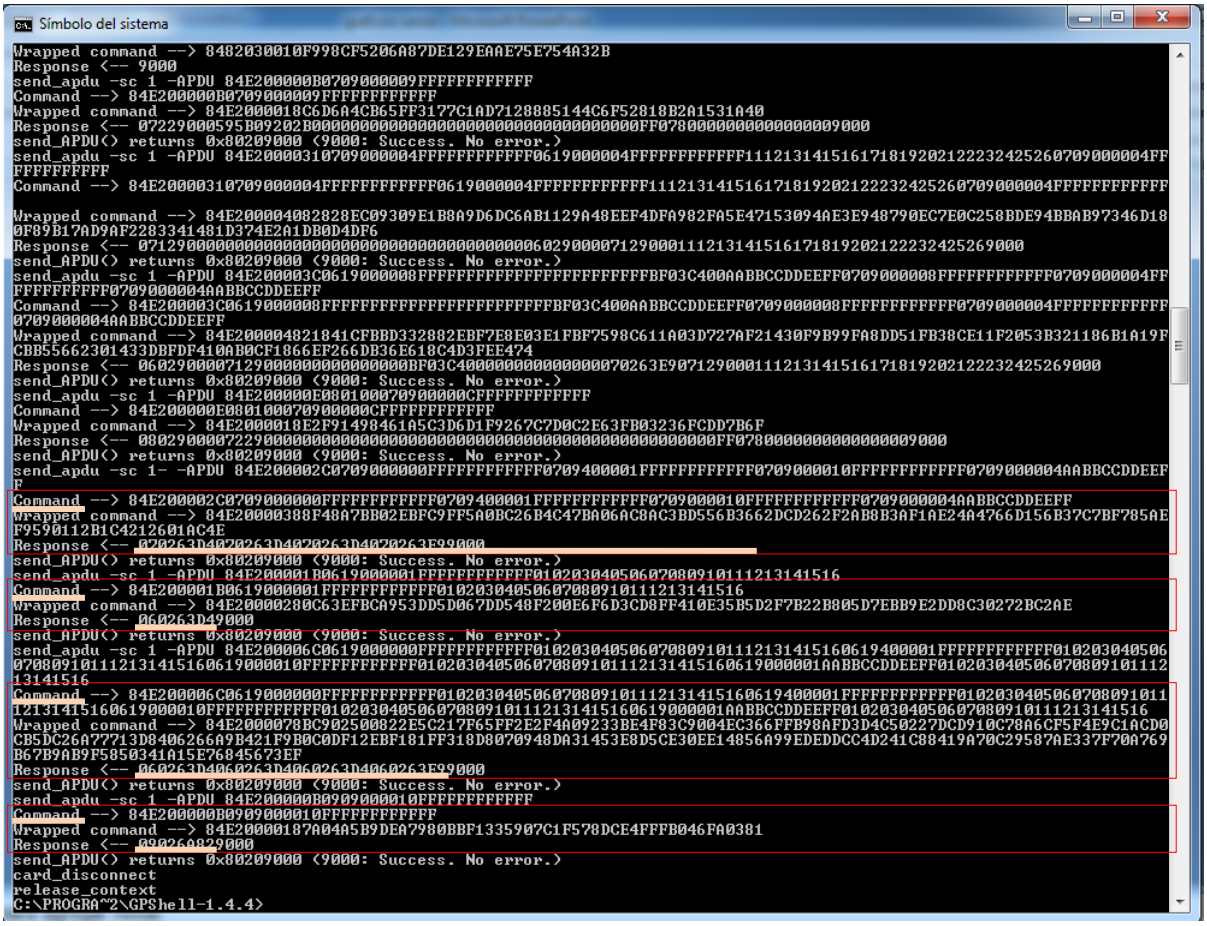


Figura 7-6 - Captura de las pruebas realizadas en el nivel de seguridad de gestor (2)

Lo último que nos queda por probar serían los errores que tiene que devolver la aplicación cuando hay algún error en los parámetros o en los datos que suministramos. Dichos errores son los que aparecen resaltados en la y al igual que en el caso anterior procederemos a desarrollarlos.

- Comando: 84E200002C0709000000FFFFFFFFFFFF0709400001FFFFFFFFFFFF0709000010FFFFFFFFFFFF0709000004AABBCCDDEEFF
    - Comando acceso remoto: 0709000000FFFFFFFFFFFF
        - Etiqueta: 07 (comando de lectura)
        - Longitud: 09 (9 bytes)
        - Campo de datos:
            - 00: sector objetivo
            - 0000: Block Bit Map, sin objetivo
            - FFFFFFFFFFFF: clave de acceso al bloque

- Comando acceso remoto: 0709400001FFFFFFFFFFF
  - Etiqueta: 07 (comando de lectura)
  - Longitud: 09 (9 bytes)
  - Campo de datos:
    - 40: sector objetivo
    - 0001: Block Bit Map, bloque 0
    - FFFFFFFFFFFF: clave de acceso al bloque
- Comando acceso remoto: 0709000010FFFFFFFFFFF
  - Etiqueta: 07 (comando de lectura)
  - Longitud: 09 (9 bytes)
  - Campo de datos:
    - 00: sector objetivo
    - 0010: Block Bit Map, bloque 16
    - FFFFFFFFFFFF: clave de acceso al bloque
- Comando acceso remoto: 0709000004AABBCCDDEEFF
  - Etiqueta: 07 (comando de lectura)
  - Longitud: 09 (9 bytes)
  - Campo de datos:
    - 00: sector objetivo
    - 0004: Block Bit Map, bloque 0
    - AABBCCDDEEFF: clave de acceso al bloque

- Respuesta: 070263D40702263D4070263D4070263E99000
  - Respuesta acceso remoto: 070263D4
    - Etiqueta: 07 (comando de lectura)
    - Longitud: 02 (2 bytes)
    - Campo de datos:
      - 63D4: palabra de estado (error en los parámetros)
  - Respuesta acceso remoto: 070263D4
    - Etiqueta: 07 (comando de lectura)
    - Longitud: 02 (2 bytes)
    - Campo de datos:
      - 63D4: palabra de estado (error en los parámetros)

- Respuesta acceso remoto: 070263D4
    - Etiqueta: 07 (comando de lectura)
    - Longitud: 02 (2 bytes)
    - Campo de datos:
        - 63D4: palabra de estado (error en los parámetros)
- Respuesta de acceso remoto: 070263E9
    - Etiqueta: 07 (comando de lectura)
    - Longitud: 02 (2 bytes)
    - Campo de datos:
        - 63E9: palabra de estado (error en la clave proporcionada)

En este primer APDU comprobamos todos los errores que nos podemos encontrar en accesos a lecturas. En orden tenemos: lectura especificar bloque objetivo (el blockBitMap está a cero), lectura de un sector fuera de rango, lectura de un bloque fuera de rango y lectura con clave errónea. Comprobamos que obtenemos los valores que corresponden según especificación 4.4.2.1-Comando readSector, esto es error en los parámetros en los tres primeros y error de contraseña en el último.

- Comando: 84E200001B0619000001FFFFFFFFFFF0102030405060708090101112131415 16
    - Comando acceso remoto: 0619000001FFFFFFFFFFF0102030405060708090101112131415 16
        - Etiqueta: 06 (comando de escritura)
        - Longitud: 19 (25 bytes)
        - Campo de datos:
            - 00: sector objetivo
            - 0001: Block Bit Map, bloque 0
            - FFFFFFFFFFFF: clave de acceso al bloque
            - 0102030405060708090101112131415 16: datos a escribir
- Respuesta: 060263D49000
    - Respuesta acceso remoto: 060263D4
        - Etiqueta: 06 (comando de escritura)
        - Longitud: 02 (2 bytes)
        - Campo de datos:
            - 63D4: palabra de estado (error en los parámetros)

Este comando intenta una escritura en el bloque 0 del sector 0. Sabemos que este bloque, el *Manufacturer Block*, es especial ya que se encuentra siempre protegido contra escritura. El error devuelto es el especificado para este caso, un error en los parámetros.

- Comando:
  84E200006C0619000000FFFFFFFFFFFF0102030405060708091011121314151606190400010FFFFFFFFFFFF0102030405060708091011121314151606190000010FFFFFFFFFFF0102030405060708091011121314151606190000001AABBCCDDEEFF0102030405060708091011121314151606190000001AABBCCDDEEFF01020304050607080910111213141516
  - Comando acceso remoto:
    06C0619000000FFFFFFFFFFFF01020304050607080910111213141516
    - Etiqueta: 06 (comando de escritura)
    - Longitud: 19 (25 bytes)
    - Campo de datos:
      - 00: sector objetivo
      - 0000: Block Bit Map, sin objetivo
      - FFFFFFFFFFFF: clave de acceso al bloque
      - 01020304050607080910111213141516: datos a escribir
  - Comando acceso remoto:
    0619400001FFFFFFFFFFFF01020304050607080910111213141516
    - Etiqueta: 06 (comando de escritura)
    - Longitud: 19 (25 bytes)
    - Campo de datos:
      - 40: sector objetivo
      - 0001: Block Bit Map, bloque 0
      - FFFFFFFFFFFF: clave de acceso al bloque
      - 01020304050607080910111213141516: datos a escribir
  - Comando acceso remoto:
    0619000010FFFFFFFFFFFF01020304050607080910111213141516
    - Etiqueta: 06 (comando de escritura)
    - Longitud: 19 (25 bytes)
    - Campo de datos:
      - 00: sector objetivo
      - 0010: Block Bit Map, bloque 16
      - FFFFFFFFFFFF: clave de acceso al bloque
      - 01020304050607080910111213141516: datos a escribir

- Comando acceso remoto: 0619000001AABBCCDDEEFF0102030405060708091011121314151 6
    - Etiqueta: 06 (comando de escritura)
    - Longitud: 19 (25 bytes)
    - Campo de datos:
        - 00: sector objetivo
        - 0001: Block Bit Map, bloque 0
        - AABBCCDDEEFF: clave de acceso al bloque
        - 0102030405060708091011121314151 6: datos a escribir

- Respuesta: 060263D4060263D4060263D4060263E99000
    - Respuesta acceso remoto: 060263D4
        - Etiqueta: 06 (comando de escritura)
        - Longitud: 02 (2 bytes)
        - Campo de datos:
            - 63D4: palabra de estado (error en los parámetros)
    - Respuesta acceso remoto: 060263D4
        - Etiqueta: 06 (comando de escritura)
        - Longitud: 02 (2 bytes)
        - Campo de datos:
            - 63D4: palabra de estado (error en los parámetros)
    - Respuesta acceso remoto: 060263D4
        - Etiqueta: 06 (comando de escritura)
        - Longitud: 02 (2 bytes)
        - Campo de datos:
            - 63D4: palabra de estado (error en los parámetros)
    - Respuesta acceso remoto: 060263E9
        - Etiqueta: 06 (comando de escritura)
        - Longitud: 02 (2 bytes)
        - Campo de datos:
            - 63E9: palabra de estado (error en la clave proporcionada)

En el caso de los comando para escritura vemos que el resultado es el mismo que para los casos de lectura y coincide con las especificaciones 4.4.2.2-Comando writeSector. En los tres primeros casos (sin bloque objetivo y fuera de rango en el campo sector y en el campo bloque)

obtenemos el error de parámetro erróneo mientras que, en el caso de la proporcionar una clave errónea nos aparece el error de contraseña no valida.

- Comando: 84E200000B0909000010FFFFFFFFFFF
    - Comando acceso remoto: 0909000010FFFFFFFFFFF
        - Etiqueta: 09 (comando no asignado)
        - Longitud: 09 (9 bytes)
        - Campo de datos:
            - 000010FFFFFFFFFFF: datos de relleno
- Respuesta: 09026A829000
    - Respuesta acceso remoto: 09026A82
        - Etiqueta: 09 (comando de escritura)
        - Longitud: 02 (2 bytes)
        - Campo de datos:
            - 6A82: palabra de estado (comando no encontrado)

Por último intentamos enviar, en un comando de acceso remoto, un comando que no hemos especificado. Obtenemos el error de comando no encontrado indicando que el comando no es válido.

7.2.3 NIVEL DE SEGURIDAD DE USUARIO

El nivel de seguridad de usuario es el último de los tres niveles de seguridad y el que menos seguridad tiene. Aun así, tiene más seguridad que la que nos ofrecen MIFARE Classic y MIFARE4Mobile. Todas las comunicaciones en este nivel de seguridad se inician mediante una pseudo-apertura de canal seguro, en realidad se trata de que la entidad externa tiene que enviar, en un comando APDU encriptado, el PIN personal del usuario. Esta encriptación se realiza utilizando codificación DES, usando el propio PIN como clave. De esta forma la tarjeta al quitar el cifrado con la clave podrá compararla con la que tiene almacenada para verificar la identidad del usuario.

Para el caso del nivel de seguridad de usuario una sola captura será suficiente para todas las pruebas que tenemos que realizar.

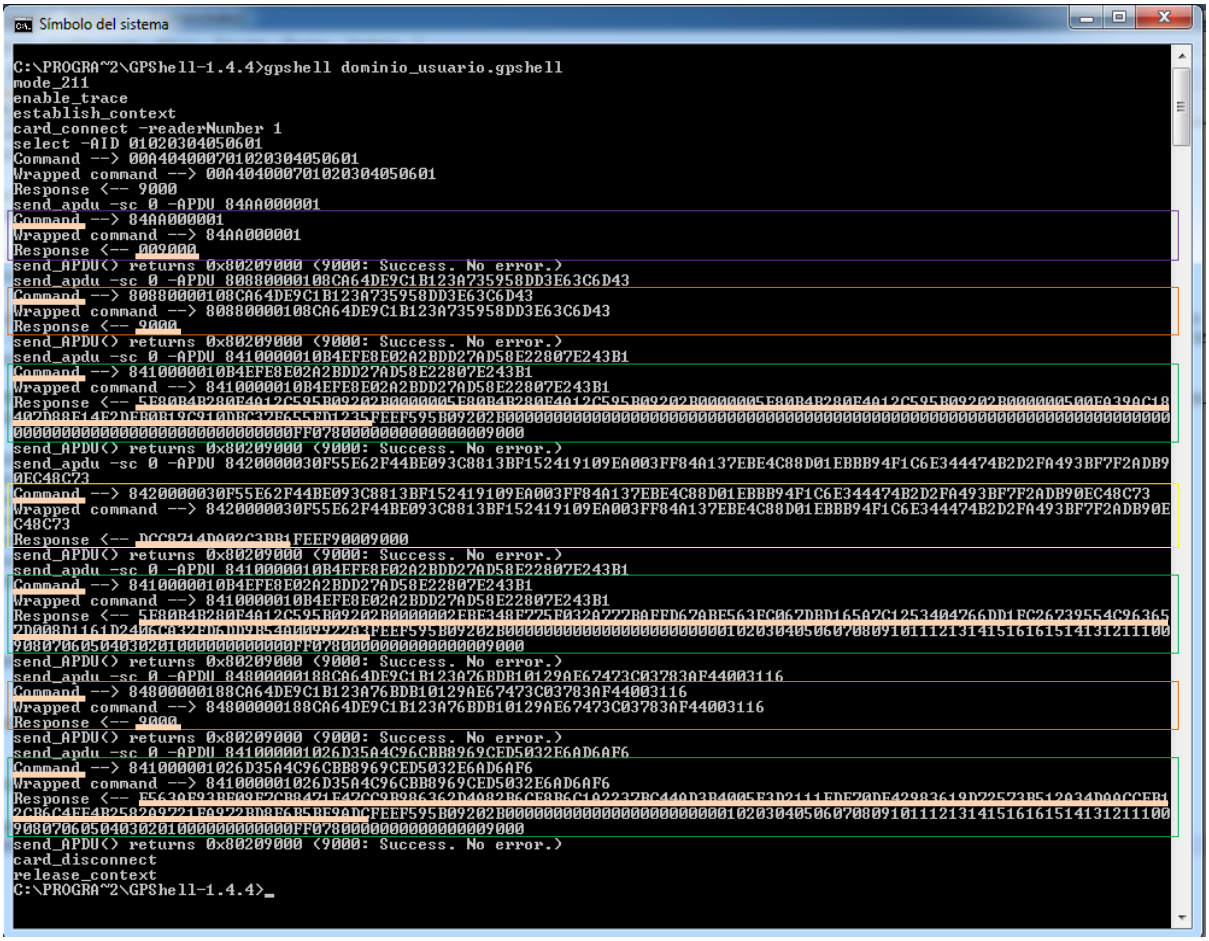


Figura 7-7 - Captura de las pruebas realizadas en el nivel de seguridad de usuario

Debido a que el sistema de encriptación utilizado no es el que tiene predefinido GPShell, este no puede calcular automáticamente los mensajes APDU como hacía en los otros dos niveles de seguridad anteriores y por tanto tenemos que pasarle los comandos APDU cifrados directamente. Estos comandos los hemos conseguido enviando los comandos en claro con anterioridad y haciendo que la propia tarjeta nos devuelva los comandos ya cifrados. Para las respuestas, la APDU respuesta irá divida en dos partes, la primera parte, cifrada y la segunda, en claro, habiendo obtenido esta última internamente a través del dominio de seguridad y partiendo de la respuesta cifrada. La transición entre estas dos partes viene dada por dos bytes seguidos de valor “0xFEEF”.

Las dos únicas parejas de APDUs que no irán cifradas por completo serán las de selección de tarjeta virtual (va todo en claro) y las de verificación de usuario (solo se cifra el comando). Estos dos comandos son los dos primeros que aparecen en la captura respectivamente.

Para facilitar el seguimiento y comprensión de las siguientes parejas de APDUs, utilizadas durante estas pruebas, descompondremos[12] cada uno de los comandos añadiendo una pequeña descripción después.

[12] Todos los mensajes APDU descompuestos se muestran en formato hexadecimal

- Comando: 80880000108CA64DE9C1B123A735958DD3E63C6D43
    - Comando sin cifrar: 8088000008000000000000000
        - CLA: 80
        - INS: 88 (comando de comprobación de PIN)
        - P1: 00
        - P2: 00
        - Lc: 08 (8 bytes)
        - Campo de datos:
            - 0000000000000000: PIN
- Respuesta: 9000
    - Palabra de estado: 9000 (procesado correcto)

Una vez seleccionada una tarjeta virtual, este comando es el primero que tiene que realizarse en el nivel de seguridad de usuario. Es el comando de comprobación de PIN que, por un lado, verifica de cara a la aplicación la identidad del usuario y por otro, comprueba que la encriptación se realiza correctamente, debido a que este PIN es el que se utiliza como valor de encriptación.

Obtenemos como resultado la palabra de estado de procesado correcto. A partir de este momento nos encontramos en el nivel de seguridad de usuario y toda la comunicación será cifrada.

- Comando: 8410000010B4EFE8E02A2BDD27AD58E22807E243B1
    - Comando sin cifrar: 8410000008000FFFFFFFFFFFF
        - CLA: 84
        - INS: 10 (comando de lectura)
        - P1: 00 (sector 0)
        - P2: 00
        - Lc: 08 (8 bytes)
        - Campo de datos:
            - 000F: Block Bit Map, bloques 0, 1, 2 y *Sector Trailer*
            - FFFFFFFFFFFF: clave de acceso al bloque
- Respuesta: 595B09202B00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000FF07800000000000000009000
    - 595B09202B00000000000000000000: datos bloque 0, sector 0
    - 00000000000000000000000000000000: datos bloque 1, sector 0
    - 00000000000000000000000000000000: datos bloque 2, sector 0

- 000000000000FF078000000000000000: datos *Sector Trailer*, sector 0
- 9000: palabra de estado: (procesado correcto)

Continuamos con una lectura múltiple de todos los bloques de un mismo sector, en este caso del sector 0. En la respuesta aparecen, de forma consecutiva y ordenada, los datos de todos los bloques del sector 0 además de la palabra de estado. Se puede observar que por seguridad, las claves KeyA y KeyB aparecen puestas a cero.

- Comando: 8420000030F55E62F44BE093C8813BF152419109EA003FF84A137EBE4C88D01EBBB94F1C6E344474B2D2FA493BF7F2ADB90EC48C73
  - Comando sin cifrar: 84200000280006FFFFFFFFFFFF0102030405060708091011121314151616151413121110090807060504030201
    - CLA: 84
    - INS: 20 (comando de escritura)
    - P1: 00 (sector 0)
    - P2: 00
    - Lc: 28 (40 bytes)
    - Campo de datos:
      - 0006: bloques Block Bit Map, 1 y 2
      - FFFFFFFFFFFF: clave de acceso al bloque
      - 0102030405060708091011121314151 6: datos a escribir, bloque 1
      - 16151413121110090807060504030201: datos a escribir, bloque 2
- Respuesta: 9000
  - 9000: palabra de estado: (procesado correcto)
- Comando: 8410000010B4EFE8E02A2BDD27AD58E22807E243B1
  - Comando sin cifrar: 8410000008000FFFFFFFFFFFF
    - CLA: 84
    - INS: 10 (comando de lectura)
    - P1: 00 (sector 0)
    - P2: 00
    - Lc: 08 (8 bytes)
    - Campo de datos:
      - 000F: Block Bit Map, bloques 0, 1, 2 y *Sector Trailer*

- FFFFFFFFFFFF: clave de acceso al bloque

- Respuesta:
  595B09202B00000000000000000000000102030405060708091011121314151616151413121110090807060504030201000000000000FF07800000000000000009000
  - 595B09202B00000000000000000000: datos bloque 0, sector 0
  - 0102030405060708091011121314151 6: datos bloque 1, sector 0
  - 1615141312111009080706050403020 1: datos bloque 2, sector 0
  - 000000000000FF078000000000000000: datos *Sector Trailer*, sector 0
  - 9000: palabra de estado: (procesado correcto)

Volvemos a realizar la prueba de verificación de la escritura en dos pasos, primero escritura y luego lectura para comprobar, solo que esta vez realizaremos una escritura múltiple sobre los bloques 1 y 2 utilizando solo un comando.

Como en la prueba anterior tenemos un comando de escritura que esta vez tiene como objetivos tanto al bloque 1 como al bloque 2 y cómo los datos a escribir se encuentran concatenados empezando por el bloque menor.

El segundo comando es una escritura sobre todo el sector donde podemos comprobar en la respuesta cómo efectivamente ambos bloques han actualizado su valor.

- Comando:
  84800000188CA64DE9C1B123A76BDB10129AE67473C03783AF44003116
  - Comando sin cifrar: 848000001000000000000000000010111211213141516 17
    - CLA: 84
    - INS: 80 (comando de cambio de PIN)
    - P1: 00
    - P2: 00
    - Lc: 10 (16 bytes)
    - Campo de datos:
      - 0000000000000000: PIN actual
      - 1011121314151617: PIN nuevo

- Respuesta: 9000
  - 9000: palabra de estado: (procesado correcto)

En esta prueba realizamos un cambio de PIN, lo que significa que tanto la comprobación de PIN al iniciar una nueva sesión en el nivel de seguridad de usuario como todos los comandos a partir de este punto se verán afectados por dicho cambio.

En el comando se tiene que facilitar el PIN actual, como medida de seguridad, y a continuación el PIN nuevo.

En la respuesta obtenemos la palabra de estado de procesado correcto por lo que a partir de este momento la encriptación de todos los mensajes se realizara con el nuevo PIN.

- Comando: 841000001026D35A4C96CBB8969CED5032E6AD6AF6
    - Comando sin cifrar: 8410000008000FFFFFFFFFFFF
        - CLA: 84
        - INS: 10 (comando de lectura)
        - P1: 00 (sector 0)
        - P2: 00
        - Lc: 08 (8 bytes)
        - Campo de datos:
            - 000F: Block Bit Map, bloques 0, 1, 2 y *Sector Trailer*
            - FFFFFFFFFFFF: clave de acceso al bloque
- Respuesta: 595B09202B00000000000000000000000010203040506070809101112131415161615141312111009080706050403020100000000000FF07800000000000000009000
    - 595B09202B00000000000000000000000: datos bloque 0, sector 0
    - 01020304050607080910111213141516: datos bloque 1, sector 0
    - 16151413121110090807060504030201: datos bloque 2, sector 0
    - 000000000000FF078000000000000000: datos *Sector Trailer*, sector 0
    - 9000: palabra de estado: (procesado correcto)

Como última prueba realizamos una lectura de todos los bloques del sector 0 para comprobar que el cambio de PIN ha sido efectivo. A estas alturas ya hemos comprobado que el comando de lectura funciona correctamente y sabemos el resultado que esperamos obtener.

La diferencia entre este comando y el anterior comando de lectura que hemos realizado radica en el comando encriptado enviado. Si nos fijamos salvo la cabecera que no se cifrar el cuerpo del comando APDU es distinto en este caso que en el caso anterior debido a que hemos utilizado el nuevo PIN para cifrar.

Dado que el resultado que obtenemos es el mismo que habíamos obtenido con anterioridad podemos concluir que el comando ha sido procesado correctamente por la aplicación y por tanto el cambio de PIN efectivamente se ha realizado.

ACRÓNIMOS

AES	Estándar de encriptación avanzado (*Advance Encription Standard*)
AID	Identificador de la aplicación (*Application IDentifier*)
APDU	Unidad de datos del protocolo de aplicación (*Aplication Protocol Data Unit*)
CLA	Clase
C-MAC	Código de autenticación de mensaje del comando APDU (*Command Message Authentication Code*)
DEK	Clave de encriptación de datos (*Data Encription Key*)
DES	Estándar de encriptación de datos (*Data Encription Standard*)
EEPROM	Memoria de solo lectura programable mediante borrado eléctrico (*Electrically Erasable Programmable Read-Only Memory*)
ICV	Vector de inicialización (*Initial Chain Vector*)
INS	Instrucción
M4M	MIFARE for mobile
OPEN	Entorno de GlobalPlatform
PCD	Dispositivo de acoplamiento en proximidad o lector de tarjetas inteligentes sin contacto (*Proximity Coupling Device*)
R-MAC	Código de autenticación de mensaje de la respuesta APDU (*Response Message Authentication Code*)
SCP	Protocolo de canal seguro (*Secure Channel Protocol*).
SW	Palabra de estado (*Status Word*)
S-ENC	Clave de encriptación de canal seguro (*Secure channel ENCription key*)
S-MAC	Clave del código de autenticación de mensaje del canal seguro (*Secure channel Message Authentication Code key*)
VC	Tarjeta virtual (*Virtual Card*)

BIBLIOGRAFÍA

[1] Naciones Unidas, Resolución A/HCR/20/L.3 “Promoción, protección y disfrute de los derechos humanos en internet”, 29 de Junio de 2012.

[2] Pew Research Center, Global attitudes survey, Q70 y Q72, Primavera 2015.

[3] Eurosmart, Annual forecast of worldwide secure elements shipments, 2015.

[4] Edward Bellamy, Looking Backward, Houghton Mifflin, 1887.

[5] ISO7811, Identification cards – Recording technique.

[6] Jurgen Dethloff y Helmut Gröttrup, Identification System, Patente GB1317915, Junio 1969.

[7] Jurgen Dethloff y Helmut Gröttrup, Identifying card for use in an identification System, Patente GB1318850, Agosto 1969.

[8] Roland Moreno, Data Storage Systems, Patente FR2266222, Marzo 1974 – Octubre 1975.

[9] Jurgen Dethloff, Identification system safeguarded against misuse, Patente US4105156, Diciembre 1976.

[10] First Draft of a Report on the EDVAC, John Von Neumann, J. Presper Eckert y John Mauchly, 1945.

[11] ISO7810, Identification cards – Physical characteristics, tercera edición, Noviembre 2003.

[12] ISO7816, Identification cards – Integrated circuits card.

[13] ISO14443, Identification cards – Contactless integrated circuit cards – Proximity cards, tercera edición, Julio 2016.

[14] MULTOS Developer’s guide, MAO-DOC-TEC-005 v1.41, 2016.

[15] Sun Microsystems, Java Card Platform (API, JCRE y JCVM), versión 2.2.2, Marzo 2006.

[16] GlobalPlatform, Card Specification, versión 2.2, Marzo 2006.

[17] ISO15693, Identification cards – Contactless integrated circuit cards – Vicinity cards, segunda edición, Diciembre 2006.

[18] LEGIC Advant transponder chips, versión 01, 13 de Junio de 2002.

[19] NPX, Tap into MIFARE, Enero de 2015.

[20] MF0ICU1, MIFARE Ultralight contactless single-ticket IC, revisión 3.9, 23 Julio 2014.

[21] MF1S70yyX/V1, MIFARE Classic EV1 4k – Mainstream contactless smart cardIC for fast and easy solution development, revisión 3.1, 8 Septiembre 2014.

[22] MF1SPLUSx0y1, Mainstream contactless smart card for fas and easy solution development, revisión 3.2, 21 Febrero 2011.

[23] MF3ICDx21_41_81, MIFARE DESFire EV1 contactless multi-aplication IC, revisión 3.2, 9 Diciembre 2015.

[24] SmartMX2 P40 family P40C012/040/072, Secure smart card controller, revisión 3.0, 24 Abril 2015.

[25] Karsten Nohl, "Cryptanalysis of Crypto-1", Universidad de Virginia, 10 Marzo 2008.

[26] de Koning Gans, Gerhard; J.-H. Hoepman; F.D, Garcia, "A Practical Attack on the MIFARE Classic" . VIII Smart Card Research and Advanced Application Workshop (CARDIS 2008), LNCS, Springer, 15 Marzo 2008.

[27] Courtois, Nicolas T.; Karsten Nohl; Sean O'Neil, "Algebraic Attacks on the Crypto-1 Stream Cipher in MiFare Classic and Oyster Cards". Cryptology ePrint Archive, 14 Abril 2008.

[28] Garcia, Flavio D.; Gerhard de Koning Gans; Ruben Muijrers; Peter van Rossum, Roel Verdult; Ronny Wichers Schreur; Bart Jacobs, "Dismantling MIFARE Classic". XIII European Symposium on Research in Computer Security (ESORICS 2008), LNCS, Springer, 14 Octubre 2008.

[29] Garcia, Flavio D.; Peter van Rossum; Roel Verdult; Ronny Wichers Schreur. "Wirelessly Pickpocketing a Mifare Classic Card". XXX IEEE Symposium on Security and Privacy (S&P 2009), IEEE, 17 Marzo 2009

[30] M4M_Overview, MIFARE4Mobile - Overview, version 2.1, Julio 2013.

[31] M4M_Arch, MIFARE4Mobile - Architecture, version 2.1, Julio 2013.

[32] M4M_Classic_HI, MIFARE Classic Host Interface, versión 1, Junio 2013.

[33] M4M_RM_API, MIFARE4Mobile 2 - Remote Management API, version 2.1, Julio 2013.

[34] AN02105, Secure Access to MIFARE Memory on Dual Interface Smart Card ICs, version 1.2, Septiembre 2015.

[35] M4M_VC_Creation, MIFARE Virtual Card Creation, versión 1.0.1, Junio 2013.