# Desarrollo e implementación de un TPV para pagos de importe reducido

# Development and implementation of a POS for reduced amount payments


TRABAJO DE FIN DE GRADO

Daniel Jorquera Cabañes

Dirigido por: Inmaculada Pardines Lence y Marcos Sanchez-Élez Martín

Colaborador externo: Carlos Cossío Gómez

Grado en Ingeniería Informática

Facultad de Informática

Universidad Complutense de Madrid

2020

# Desarrollo e implementación de un TPV para pagos de importe reducido

# Development and implementation of a POS for reduced amount payments

Memoria de Trabajo de Fin de Grado

Daniel Jorquera Cabañes

Dirigido por: Inmaculada Pardines Lence y Marcos Sánchez-Élez Martín

Colaborador externo: Carlos Cossío Gómez

Grado en Ingeniería Informática

Facultad de Informática

Universidad Complutense de Madrid

2020

# Agradecimientos

En primer lugar, me gustaría agradecer a los directores de mi Trabajo de Fin de Grado, Inmaculada Pardines Lence y Marcos Sánchez-Élez Martín, todo el tiempo que me han dedicado y que ha hecho posible que este trabajo salga adelante, aconsejándome en la toma de decisiones y ayudándome a superar los inconvenientes que han ido surgiendo a lo largo de la realización de esta memoria.

A Carlos, mi tutor en Redsýs, por darme la oportunidad de formar parte de su equipo en el Departamento de Innovación y por confiar en un principiante para el desarrollo de este atractivo proyecto.

A mis padres y a mi hermana por confiar en mí desde el principio, sin cuyo apoyo y paciencia no me hubiera resultado posible alcanzar la meta que estoy a punto de conseguir. Gracias por hacerme creer que era posible.

# Resumen

Actualmente, gran parte de los pagos que efectuamos en nuestro día a día los realizamos mediante tarjetas bancarias. Es por ello por lo que es necesario mejorar y renovar constantemente los métodos de pago actuales para fomentar el uso de estas tecnologías por parte de toda la población, manteniendo un bajo coste tanto para los comercios que adquieren los TPVs como para los usuarios finales.

Una cantidad importante de estos pagos son pagos reducidos, muchas veces de un importe inferior a 20 €, como ocurre cuando compramos el periódico, cuando compramos un billete de metro o cuando tomamos un café. Por esta razón, en este Trabajo de Fin de Grado se propone el desarrollo e implementación de un terminal bancario para pagos reducidos, en este caso, de menos de 20 €. La elección de este importe concreto se debe a que los pagos inferiores al mismo no requieren de la inserción del código PIN y se pueden realizar mediante tecnología contactless, lo que hace que el terminal no necesite teclado ni lector de tarjetas con contactos y permita un proceso de compra mucho más ágil, rápido e higiénico, factor importante en la situación de pandemia actual.

**Palabras clave**: TPV, Bajo coste, EMV, Tarjetas bancarias, Contactless, Pagos reducidos, Código PIN, APDU, Mastercard, Microcontrolador.

# Abstract

Nowadays, a large part of the payments we carry out on daily basis are completed using credit cards. This is the reason why it is necessary to constantly improve and update current payment methods to encourage the use of these technologies by the entire population, keeping a low cost for both the stores that acquire the POS and end users.

A significant number of these payments are reduced amount payments, often less than € 20; for example, when we buy the newspaper, when we buy a metro ticket or when we order a coffee. This is the reason why this Final Degree Project proposes the development and implementation of POS only for reduced payments, in this case, less than € 20. The choice of this specific amount is due to the fact that payments of an amount lower than € 20 do not require the insertion of the PIN Code and can be done using contactless technology, what implies that the terminal does not need to have a keyboard or a contact chip card reader and allows a purchase process much more agile, fast and hygienic, an important factor in the current pandemic situation.

**Palabras clave**: POS, Low cost, EMV, Credit card, Contactless, Reduced amount payments, PIN Code, APDU, Mastercard, Microcontroller.

La propiedad intelectual del trabajo descrito en esta memoria pertenece a Redsýs Servicios de Procesamiento S.L.

# Índice general

# Capítulo 1

# 1. Introducción

## 1.1. Motivación

Hoy en día estamos acostumbrados a poder pagar cualquier tipo de servicio o producto con nuestra tarjeta de crédito, ahorrándonos la tediosa tarea de llevar dinero en metálico siempre que salimos de casa. Pero no siempre ha sido así.

En la primera mitad del siglo XX comenzó a surgir la idea en ciertos negocios de la creación de tarjetas de crédito para la compra de productos o servicios, pero exclusivamente en dicho negocio. Por ejemplo, compañías propietarias de gasolineras sacaron al público una tarjeta que permitía la compra de gasolina solo en las gasolineras pertenecientes a dicha compañía [1].

No fue hasta el año 1949 cuando surgió la primera tarjeta de crédito que se podía utilizar en varios establecimientos. Una serie de casualidades provocaron que un popular banquero americano que había cenado con unos compañeros se viese en la obligación de pedirle al restaurante que le fiase el importe de dicha cena, ya que se había olvidado el dinero en casa. Este incidente derivó en la creación de la primera tarjeta de crédito moderna, la *Diners Club* [2]. Con esta tarjeta el cliente podía consumir en una serie de restaurantes que estaban adheridos al proyecto, sin la necesidad de pagar en metálico.

Gracias al desarrollo tecnológico ocurrido en la segunda mitad del siglo XX y a la aparición de empresas como Mastercard o VISA, las tarjetas de crédito han ido evolucionando hasta ser tal y como hoy en día las conocemos, superando los 1100 millones de transacciones anuales en España en 2017 [3].

Sin embargo, debido al gran volumen de dinero que mueven estas transacciones en la sociedad, es importante que se realicen con unas altas condiciones de seguridad informática para evitar la posible intrusión de atacantes en las mismas, provocando pérdidas millonarias.

Estas condiciones de seguridad deben ser garantizadas en varios niveles. En primer lugar, en la programación segura de las aplicaciones, sin posibles brechas ni vulnerabilidades. En segundo lugar, en el cifrado de información específica contenida en la tarjeta, como pueden ser las pistas confidenciales, por ejemplo, el PAN (Primary Account Number). Y, en tercer lugar, en el cifrado de las comunicaciones, de tal manera

que ningún atacante que pueda estar escuchando dichas comunicaciones sea capaz de obtener ningún dato en claro.

Este Trabajo de Fin de Grado pretende implementar un nuevo terminal de punto de venta destinado a entornos de pagos con tarjetas bancarias de menos de 20€, sin dejar de lado la importancia de la seguridad informática en este ámbito, tanto desde el punto de vista de las infraestructuras, como de los algoritmos de cifrado y de los protocolos de comunicación utilizados para garantizar la confidencialidad e integridad de los datos transmitidos y autenticidad de las identidades de quienes transmiten dichos datos.

Como punto de partida para la elaboración de este trabajo, he utilizado el proyecto que estoy desarrollando en la empresa Redsýs Servicios de Procesamiento S.L, en cuyo departamento de innovación me encuentro actualmente realizando las prácticas externas.

El hecho de pertenecer a este departamento me ha permitido darme cuenta de la importancia que la innovación tiene en una empresa tecnológica, más si cabe en una que se dedica al negocio de los medios de pago, cuyo meteórico crecimiento en los últimos 70 años se basa en una constante innovación de las tarjetas de crédito. Sin embargo, la evolución de las tecnologías también ha provocado un aumento de los posibles riesgos que el uso de estas tarjetas conlleva. Este es otro punto clave en el que se basa la innovación, el constante cambio de determinados protocolos o algoritmos con el objetivo de prevenir dichos riesgos e "ir por delante" de los posibles atacantes.

Estratégicamente, la innovación tiene una importancia crucial, ya que permite a las empresas sacar al mercado productos y servicios con una seguridad más avanzada y adelantándose a la competencia, con los consecuentes beneficios económicos y de prestigio que ello conlleva.

## 1.2. Objetivos del trabajo

Este trabajo se ha realizado gracias a la colaboración con la empresa Redsýs Servicios de Procesamiento S.L., que me ha asignado este proyecto en el cual yo soy el único participante.

El objetivo de este proyecto es implementar el software de un TPV (Terminal Punto de Venta) o datáfono diseñado exclusivamente para pagos inferiores a 20 €. Esto es, en líneas generales, un TPV sin impresora, sin teclado y sin otros componentes presentes en los TPV tradicionales, lo que permitirá reducir los costes que el comercio o negocio tendrá que invertir en un TPV, desde unos 200 € que cuesta uno tradicional a alrededor de 50 € que costaría este nuevo formato de TPV.

En cuanto a las posibles aplicaciones de este proyecto, podríamos destacar las máquinas de tabaco, máquinas vending o en los autobuses públicos. Utilizando este último caso como ejemplo, este nuevo modelo de TPV nos permitiría subir al autobús y, si no tenemos billete, pedirle al conductor un viaje para una persona y acercar nuestra tarjeta de crédito al TPV. De esta manera, al no tener que introducir PIN, ni que el conductor

tenga que escribir explícitamente el importe a pagar, ni tener que esperar a que salga la copia del banco, ahorramos también un tiempo considerable.

Al comienzo de este proyecto se me entregó una placa base STM32 (Figura 1.1) basada en ARM y con una pantalla de pequeñas dimensiones, cuya única funcionalidad era detectar, cuando se acercase una tarjeta con chip, si era una tarjeta bancaria o no. En el caso de que el resultado fuese exitoso, mostraba un 'tick' verde, mientras que, si acercabas una tarjeta no bancaria, mostraba un 'cross' rojo [4].


Figura 1.1: Placa ST25R3911B-EMVCo

Esta placa es la base en torno a la cual se ha desarrollado este proyecto, que se ha programado en lenguaje C. En su implementación se distinguen las siguientes fases:

**-Fase 1:** Adquisición de los conocimientos necesarios para la programación de TPVs y de los protocolos de comunicación que utilizan los mismos para comunicarse con las tarjetas.

**-Fase 2:** Implementación de las funciones y métodos necesarios para que el TPV sea capaz de obtener todos los datos necesarios de la tarjeta mediante comandos APDU (Application Protocol Data Unit) [5], guardando en memoria la etiqueta, longitud y valor de cada uno de dichos datos.

**-Fase 3:** Inserción de las bibliotecas criptográficas necesarias para cifrar con los algoritmos de cifrado correspondientes las pistas que el protocolo dicta que tienen que ser cifradas.

**-Fase 4:** Generación de un JSON con los datos de la tarjeta y de la transacción estipulados en los protocolos, para después poder enviar dicho JSON al centro autorizador de pagos y que este pueda aceptar o rechazar el pago solicitado (se necesita un módulo de red para poder realizar una conexión con el exterior).

**-Fase 5:** Integración del proyecto realizado con un sistema operativo, que permita modularizarlo y ejecutarlo de forma ordenada mediante diferentes tareas (tasks).

**-Fase 6:** Instalación de un módulo de red que permita al TPV conectarse con el centro autorizador e implementación de los protocolos de comunicaciones correspondientes que permitan la conexión segura entre las partes, en este caso TLS 1.2.

***El objetivo de este Trabajo de Fin de Grado es implementar este sistema teniendo en cuenta la necesidad de mantener unas exigentes condiciones de seguridad en la programación, a la hora de cifrar las pistas que el TPV obtiene de la tarjeta y que son de especial relevancia y en la utilización del protocolo criptográfico TLS 1.2.***

## 1.3. Organización de la Memoria

La memoria de este trabajo consta de 5 capítulos, incluyendo el introductorio, que siguen el mismo orden que se ha seguido en la implementación del proyecto original. A continuación, se incluye una breve descripción de dichos capítulos.

En el capítulo 2 se realiza un estudio de los TPV que podemos encontrar actualmente en el mercado, así como su funcionamiento, el tipo de hardware que utilizan o el mecanismo de lectura de las tarjetas.

En el capítulo 3 se describe la implementación completa del proyecto llevada a cabo hasta el día de la entrega de este trabajo. Se explica la ejecución de cada uno de los cinco comandos base que componen toda transacción, analizando el flujo que se sigue y mostrando como ejemplo la ejecución del software con una tarjeta real.

En el capítulo 4 se lleva a cabo un análisis de los mecanismos de seguridad que se utilizan actualmente en infraestructuras críticas, como pueden ser las bancarias, y el porqué de su utilización.

En el capítulo 5 se exponen las principales conclusiones derivadas del trabajo realizado, así como posibles ideas de trabajo futuro para mejorar el proyecto propuesto.

# Capítulo 2

# 2. Sistemas de Terminal Punto de Venta (TPV)

## 2.1. Introducción

Un Terminal Punto de Venta (en adelante TPV) es un dispositivo que podemos encontrar en establecimientos comerciales y sirve para gestionar procesos de venta. Se componen de una parte software y de una parte hardware. Actualmente podemos encontrar cinco tipos de TPV:

- Fijo: Debe estar constantemente conectado a la red telefónica mediante un conector RJ-45.
- Virtual: Pensado para el comercio electrónico, caracterizado por su flexibilidad y por su gran seguridad.
- Móvil: Posee una tarjeta SIM en su interior que nos permite mover el terminal por el local comercial sin la necesidad de que esté conectado físicamente mediante cableado.
- PC: Utilizado por empresas que tienen varios puntos de venta y cuyo objetivo es centralizar los cobros.
- Wi-Fi: La única diferencia entre este tipo de terminal y el TPV móvil es que este utiliza la red Wi-Fi del negocio para conectarse a la red y agilizar así los cobros a los clientes del comercio.

En este trabajo me voy a centrar en el tipo fijo, ya que el TPV a implementar en el proyecto de la empresa es de este tipo. Los TPVs fijos y móviles, gracias a sus distintos componentes hardware, nos ofrecen 3 formas distintas de leer una tarjeta, que pueden coexistir en el mismo TPV. Son las siguientes:

- Banda magnética: Actualmente todas las tarjetas bancarias poseen una banda magnética en la parte posterior, en la cual es posible codificar información gracias a impulsos electromagnéticos. Cuando queremos realizar una transacción, pasamos la tarjeta por el lector de banda magnética del TPV, permitiendo que el terminal obtenga la información de dicha tarjeta. Este método está prácticamente en desuso, ya que sus condiciones de seguridad dejan mucho que desear. En primer lugar, la banda magnética de las tarjetas puede ser borrada si se acerca a un dispositivo que genera un campo electromagnético, con la consecuente pérdida de datos. Aunque las tarjetas bancarias están dotadas de una

alta coercitividad (pueden soportar la presencia de un campo magnético muy fuerte) y esto pasa con poca frecuencia, se han dado algunos casos [6]. El principal problema del uso de este método es la facilidad de clonación de estas tarjetas, con el inmenso riesgo para el cliente que esto supone.

- Con contactos: Consiste en la inserción de la tarjeta en la ranura del TPV (lector de tarjeta chip), de tal manera que este es capaz de leer la información contenida en el chip de la tarjeta. Es actualmente la opción más segura ya que en los pagos que se realizan con esta tecnología se puede pedir la autenticación del usuario mediante el código PIN y no es posible replicar el chip de la tarjeta.
- *Contactless*: Se basa en la tecnología NFC, gracias a la cual el TPV se comunica con la tarjeta mediante inducción en un campo magnético. De esta manera, se acerca la tarjeta bancaria a una parte del TPV debidamente identificado con el símbolo de la tecnología "*contactless*" y se establece la comunicación entre ambos. Esta será la tecnología con la que contará el proyecto de manera exclusiva, ya que el TPV no tendrá teclado ni ranura para la lectura con contactos ni con banda magnética.

Entre las funcionalidades de un TPV encontramos:

- Realizar, mediante uno de los métodos de interacción entre la tarjeta y el TPV anteriormente mencionados, operaciones de venta.
- Llevar la contabilización del inventario de productor
- Impresión de tickets y facturas
- Gestión de devoluciones

## 2.2. Funcionamiento

El primer paso para poder implementar este sistema es comprender perfectamente el flujo que sigue una transacción bancaria *contactless* y cómo funciona la comunicación entre el TPV y la tarjeta.

Una característica esencial para poder entender el conjunto de comandos que permite la conexión tarjeta-terminal es que todos los datos, incluidos estos comandos, siguen un esquema de codificación TLV [7] (*Tag-Length-Value*), en el cual el *tag* indica qué dato se va a representar, el *length* muestra el tamaño de dicho dato y el *value* contiene su valor.

Cabe destacar que el proceso que sigue una transacción es muy complejo y largo, pero este proyecto está centrado en los cinco primeros pasos, que son la base de una transacción, toda transacción envía siempre como mínimo dichos cinco comandos. Durante estas cinco fases se obtiene toda la información de la tarjeta y su titular, para poder después, en los siguientes pasos, manejar esta información, por ejemplo, comprobando la validez de esa tarjeta comunicándose con el centro autorizador, procesando restricciones o utilizando los métodos de verificación del titular.

Las fases serán las siguientes:

- Activación del campo de radiofrecuencia
- Selección de fichero, mediante el comando "Select PPSE"
- Selección de la aplicación, mediante el comando "Select AID"
- Procesamiento de la aplicación, mediante el comando "Get Processing Options"
- Lectura de registros, mediante el comando "Read Record"
- Envío de información relacionada con la transacción en un criptograma al ICC, mediante el comando "Generate Application Cryptogram"

Para comenzar, el terminal activa el campo de radiofrecuencia y se habilita el protocolo terminal-tarjeta, en lo que se conoce como paso 0.

Posteriormente, y ya como parte de la comunicación terminal-tarjeta, se realiza la selección del fichero que contiene las aplicaciones compatibles (comúnmente conocido como "Select PPSE Command"), paso mediante el cual la tarjeta proporciona al terminal la lista de aplicaciones que soporta. Por lo tanto, en este paso se utiliza el comando de selección de fichero para abrir el que contiene todos los identificadores de las aplicaciones soportadas, y en la respuesta se mostrará la lista con los identificadores de dichas aplicaciones y su nivel de prioridad.

Para continuar, se realiza la selección de aplicación (comúnmente conocida como "Select Application IDentifier"), paso mediante el cual el terminal selecciona la aplicación de pago elegida para ejecutar la transacción. En el caso de que no pueda seleccionar ninguna y si la tarjeta lo permite, el terminal le pide al titular que pase a interfaz con contactos. Por consiguiente, en este paso se utiliza el comando de selección de fichero para elegir siempre la aplicación de Mastercard, tal y como se decidió al comienzo del proyecto. En la respuesta, el terminal incluye datos como la etiqueta de la aplicación (en este caso," MASTERCARD") o datos discrecionales del emisor. Sin embargo, el campo de la respuesta más importante es el PDOL, un campo de datos en el que se incluyen las etiquetas de los datos que será necesario que se inserten en el siguiente paso de la transacción para que esta se pueda completar satisfactoriamente, como, por ejemplo, las capacidades adicionales del terminal (ATC). Para conocer a que hace referencia cada una de estas etiquetas se utiliza un listado que contiene todas las etiquetas y sus descripciones, proporcionado por EFTLAB [8].

Después, se inicia el procesamiento de la aplicación (comúnmente conocido como "Get Processing Options Command"), mediante el cual la tarjeta proporciona las capacidades de la aplicación y la localización de los datos. En este caso, se introducen los bytes correspondientes a este comando y se inserta la información que se quiere asignar a los datos cuyas etiquetas estaban referenciadas en el PDOL referido en el paso anterior. En la respuesta de este comando se encuentra uno de los datos más importantes, el Application File Locator (AFL), que es un conjunto de bytes que especifican qué registros han de ser leídos para obtener la información de la tarjeta.

Posteriormente, se ejecuta la lectura de datos (también conocida como "Read Record Command"), paso durante el cual el terminal lee los datos requeridos de la aplicación y los almacena. Este paso es uno de los más importantes, ya que se realiza la lectura de cada uno de los registros indicados en el AFL devuelto por el anterior comando. En

estos registros podemos encontrar infinidad de datos, como puede ser el Primary Account Number (PAN), la fecha de expiración de la tarjeta, el código de la moneda utilizado por la tarjeta o el código del país. Sin embargo, uno de los campos de datos más relevantes a leer en esta fase es el CDOL1, muy similar al PDOL mencionado en la fase de selección de aplicación. En este caso, también encontramos una serie de etiquetas, con sus respectivas longitudes, de los datos que deberán ser introducidos en el comando que se ejecutará a continuación. Cabe destacar que es posible que haya un segundo CDOL (CDOL2). Para saber a qué hace referencia cada una de las etiquetas del CDOL se utiliza la herramienta referenciada anteriormente [8].

Para terminar, se describe la última fase en la que se profundiza en este proyecto, el paso conocido como "First Generate Application Cryptogram", durante el cual se envía información relacionada con la transacción al Integrated Circuit Card (ICC), es decir, el chip de la tarjeta, que genera y devuelve un criptograma. En este paso incluimos los bytes asignados a este comando y posteriormente vamos introduciendo los datos asociados a cada una de las etiquetas presentes en el CDOL1 devuelto en la repuesta del anterior comando. En este CDOL se incluyen datos muy relevantes para la transacción que se está realizando, como el importe, el tipo de transacción, el país donde se está llevando a cabo la transacción o la hora a la que se está realizando. En el caso de que exista un CDOL2 el procesamiento será el mismo. En la respuesta a este comando se puede encontrar el Application Transaction Counter (ATC) que refleja el número de transacciones realizadas con dicha tarjeta y el Criptograma de Aplicación.

Es necesario almacenar en memoria toda la información que se recopila en estos cinco comandos, para poder enviarla al centro autorizador y poder completar la transacción.

# Capítulo 3

# 3. Implementación

## 3.1. Introducción

EMV es un estándar de interoperabilidad entre tarjetas con circuito integrado (ICC, es decir, con chip) y dispositivos TPV con soporte de circuito integrado, para la autenticación de pagos mediante tarjetas de crédito y débito. Sus siglas se refieren a Europay, Mastercard y Visa y su objetivo es facilitar la interoperabilidad mundial y la aceptación de transacciones seguras mediante la gestión y evolución de los procesos EMV. Este estándar está compuesto por una serie de comandos que permiten interactuar con la tarjeta, permitiendo el envío de un comando a la tarjeta, el procesamiento de la tarjeta y el envío de una respuesta [9]. Además, define el flujo (Figura 3.1) que deben seguir las transacciones si quieren cumplir con el estándar.

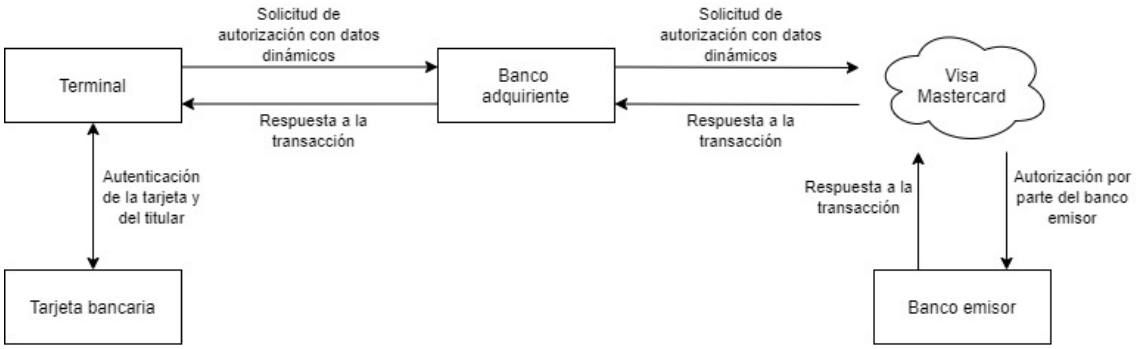


Figura 3.1: Flujo que sigue una transacción EMV

EMV tiene tres niveles de certificación:

- Nivel 1: El terminal físico, la lógica y la transmisión de los pagos son testeados.
- Nivel 2: El software desarrollado para facilitar la transmisión de la información de pago es comprobado.
- Nivel 3: Cada tipo de tarjeta es testeada con la solución de procesamiento completa, es decir, los componentes del nivel 1 y 2 más la aplicación de pagos.

Al proyecto objeto de este TFG se le ha llamado ChipPOS y en su implementación se buscará desarrollar un terminal listo para certificarlo y sacarlo al mercado, por lo que la intención final es obtener una certificación EMVCo nivel 3 [10].

Para la implementación del proyecto ChipPOS se decidió utilizar una placa ST25R3911B [4], fabricada por STMicroelectronics y compuesta por un microcontrolador STM32L476 con 1 Mbyte de memoria flash. Se eligió esta placa por varios motivos. En primer lugar y como ventaja principal, es que viene incorporado el nivel 1 del firmware EMVCo certificado, lo cual nos ahorra la necesidad de implementar ese software y lo que es más complicado, de certificarlo. También como ventaja principal, vienen incluidas en este microcontrolador unas librerías pre-certificadas para PCI DSS (Payment Card Industry Data Security Standard) [11]. Por otra parte, esta placa se basa en un lector de alta frecuencia y de alto rendimiento, que además incluye la tecnología NFC, necesaria para poder implementar un TPV que va a leer únicamente tarjetas *contactless*. Para continuar, es una placa de pequeño tamaño y asequible económicamente, ya que uno de los objetivos de la implementación del proyecto es reducir costes y espacio. Otra característica relevante de la placa es que tiene dos puertos, uno de alimentación para conectarlo con el PC y otro para realizar el *debugging* mediante ST-LINK/V2.

Por lo tanto, a la hora de desarrollar el software, se partió del proyecto entregado por ST que incluía la certificación EMVCo nivel 1, siendo nuestro objetivo la implementación del nivel 2 de tal manera que también pudiese ser certificada. El entorno de programación utilizado será Keil µVision 5 [12].

## 3.2. Punto de partida y herramientas

En el proyecto proporcionado por ST se distinguen claramente tres partes:

- El software de la aplicación, donde se encontraba todo lo relacionado con el código correspondiente a la certificación EMVCo nivel 1. Adicionalmente, incluye el main y los ficheros de configuración de la placa.
- Por otra parte, el middleware, donde se incluía, por ejemplo, la librería JPEG para poder mostrar imágenes en el display si fuese necesario.
- Por último, incorporaba una serie de drivers con ficheros que configuraban los leds, el timer, el buzzer…

La idea consiste en utilizar el software de aplicación incluido en la placa, concretamente la carpeta EMV para implementar el Kernel del TPV para poder optar a la certificación

del software (nivel 2 de EMVCo). Después nos centraremos en optimizarlo para conseguir la certificación nivel 3 de EMVCo. Para esto, se modifica el archivo emv_interop.c. Todos los cambios realizados que se detallan en este capítulo se ceñirán exclusivamente a este fichero.

Inicialmente, en este archivo tan solo se encontraba un método con la siguiente operativa:

- En primer lugar, mostraba el comando "Select PPSE" en el log y lo ejecutaba, guardando la respuesta en un buffer y mostrándola en el log, y devolviendo la longitud de dicha respuesta.
- En segundo y último lugar, comprobaba que la longitud de la respuesta era distinta de 2, en cuyo caso significaría que ha habido un error. En el caso de que no haya habido error (es decir, la tarjeta acercada a la placa es una tarjeta bancaria, por lo que se puede obtener un PPSE), mostraría un mensaje de éxito en el log y un símbolo verde con un "tick" en el display de la placa. En caso contrario (la tarjeta acercada es cualquier tarjeta no bancaria que tenga un chip), mostraría una 'X' en rojo y un mensaje de error en el log.

Como herramientas de ayuda para el desarrollo se han utilizado las siguientes:

- Lector de tarjetas *contactless* y con contactos
- Programa WinExpert
- Decodificador TLV online [13]

Para simular el funcionamiento de la comunicación entre una tarjeta inteligente y el terminal, es decir, poder visualizar las respuestas correctas que este debe devolver a cada comando y otra información que facilitase la implementación, se utiliza un lector de tarjetas tanto *contactless* como con contactos proporcionado por la empresa y un programa llamado WinExpert. Este programa permite abrir un escenario en el que se escriben comandos y se ejecutan sobre la tarjeta insertada en el lector. De esta forma, se puede saber si el comando que se está ejecutando está bien creado, y si es así, se puede implementar la función destinada a ese comando en el proyecto desarrollado usando la misma lógica.

Para facilitar el proceso de desarrollo del proyecto, se usó un decodificador TLV online, en el que se inserta la respuesta que devuelve un comando y muestra en qué etiquetas y valores se divide esa respuesta, incluso indica qué significa cada etiqueta.

Combinando el WinExpert con este decodificador, de manera a la implementación del proyecto, se simula la ejecución de los comandos en el WinExpert, para después copiar la respuesta que nos devuelve y procesarla en el decodificador TLV, y poder así entender qué está devolviendo la tarjeta.

Una vez conocido el flujo que sigue una transacción EMV, cuáles son los bytes asociados a cada uno de los comandos que se necesitan ejecutar y cuáles son los posibles códigos de error que estos pueden devolver, se comienza la implementación. Durante la explicación de cada uno de los comandos describo brevemente la forma en la que se han implementado, pero sin dar nombres concretos de métodos y variables, y

solo haciendo un pequeño repaso, ya que el código de este proyecto está protegido y calificado como confidencial por la empresa.

Por otra parte, el protocolo Price (Protocolo Integrado de Conexión entre Entidades), desarrollado por Redsýs, es el protocolo a seguir para el intercambio de los mensajes que forman parte del ciclo de vida de una transacción realizada con tarjeta, desde la gestión de la autorización hasta la liquidación de la misma, incluso la gestión y seguimiento de incidencias posteriores. Existen dos apartados diferenciados en función del soporte elegido para el intercambio de mensajes:

- Price On-line: cubre todo el intercambio de información de forma interactiva, es decir, aquellos mensajes que la entidad envía en tiempo real.
- Price diferido: se centra en el intercambio de mensajes agrupados por lotes y que utilizan una transmisión electrónica de archivos.

A continuación, se describen las diferentes fases en las que ha consistido el proyecto, tal y como se describe en la introducción del Trabajo de Fin de Grado.

## 3.3. Fase 1. Estudio de la interacción entre la tarjeta y el terminal

Esta fase se ha dedicado a entender cómo funcionan las comunicaciones entre un terminal y una tarjeta y cuáles son los protocolos y estándares que se siguen y se cumplen durante la ejecución de una transacción. En esta sección solo voy a describir el contenido de dichos manuales para hacer notar los conocimientos previos que se necesitan para realizar el trabajo específico que se presenta en esta memoria.

Se han usado cuatro manuales:

- M/Chip Advance Card Application Specification (Mastercard)
- EMV Integrated Circuit Card Specifications for Payment Systems
- Manual de Usuario de Tarjeta de Familia Advantis: Infraestructura común
- PIN-Pad EMV Integrado. Protocolo de comunicación

Todos estos manuales están repletos de información acerca de las especificaciones y funcionamiento de los TPVs y la comunicación de estos con las tarjetas. Son privados, pero serán referenciados siempre que se haya obtenido información de ellos para completar este trabajo.

El tercero de los manuales mencionados está elaborado por Advantis [14]. Advantis es un sistema operativo de tarjetas chip EMV que cumple los estándares internacionales para esta tecnología y que proporciona la infraestructura necesaria para el procesamiento de una transacción financiera. Es un producto propiedad de Redsýs, empresa que ha proporcionado los cuatro manuales y la asesoría de un especialista en la implementación de sistemas como el que se va a desarrollar en este trabajo.

De la información contenida en estos manuales es necesario describir un poco más en detalle las características de los comandos APDU [5], compuestos por varios parámetros, todos ellos de longitud de un byte, excepto el campo datos.

Estructura general de un comando APDU [5]:

- Clase
- Instrucción
- P1
- P2
- P3
- Campo datos
- Longitud

En cada comando los dos primeros parámetros, clase e instrucción, tienen un valor fijo. Cada comando tiene una combinación de clase e instrucción propia y diferente a los demás comandos. Para saber cuál es la clase e instrucción de cada comando es necesario acudir a los manuales [15]. Además, cada comando puede tener hasta cuatro parámetros más, los tres primeros, P1, P2 y P3 no están presentes en todos los comandos y se pueden necesitar uno, dos o los tres. Suelen aludir a parámetros de referencia establecidos por tablas de bits, es decir, si el bit 5 del P1 está a uno, se activa una funcionalidad, pero si está a 0 no se activa. El cuarto parámetro que puede incluir un comando es el campo datos, que es el único campo que puede tener más de un byte de longitud y en cada comando significa una cosa, por ejemplo, en el de selección de fichero se refiere al identificador del fichero que se quiere seleccionar. Para terminar, al final de cada comando se pone siempre un byte a 00 de acuerdo con la definición del protocolo de comunicación, que en *contactless* requiere insertar ese 00 al final para indicarle a la tarjeta cuantos datos puede devolver, y en el caso de 00 es la cantidad máxima de datos, es decir 0xFF.

Otro de los detalles a comentar son los posibles códigos de respuesta, compuestos siempre por dos bytes. La respuesta a un comando puede ser satisfactoria, en cuyo caso muestra lo que devuelve la tarjeta ante ese comando, con un 90 00 añadido al final. Este es el código de respuesta ante un comando que se ha ejecutado de forma correcta, pero no siempre es así. El estándar EMV establece una serie de códigos de respuesta ante comandos que no se han ejecutado correctamente, como pueden ser el 67 00, que indica que la longitud del P3 no es correcta o el 68 00, que indica que el parámetro “clase” no es válido. En el caso de que la tarjeta devuelva un código de error, no devolverá ningún dato adicional, por lo que la longitud de la respuesta en estos casos será de dos bytes.

Para poder ejecutar las pruebas de la implementación del proyecto realizado se ha utilizado una tarjeta real, de OpenBank, pero caducada, proporcionada por la empresa. Sin embargo, durante el desarrollo del software también se han utilizado otras tres tarjetas con diferentes configuraciones cada una, para comprobar el correcto funcionamiento del proyecto con todas ellas.

## 3.4. Fase 2. Implementación de las funciones y métodos mediante comandos APDU

Como se ha mencionado anteriormente, para poder llevar a cabo la comunicación entre el terminal y la tarjeta se utilizarán comandos APDU (Application Protocol Data Unit) [5], que serán generados por el programa en función de las características específicas de la tarjeta, del terminal y de la transacción.

La Figura 3.2 representa los comandos que se ejecutan durante el flujo de una transacción.

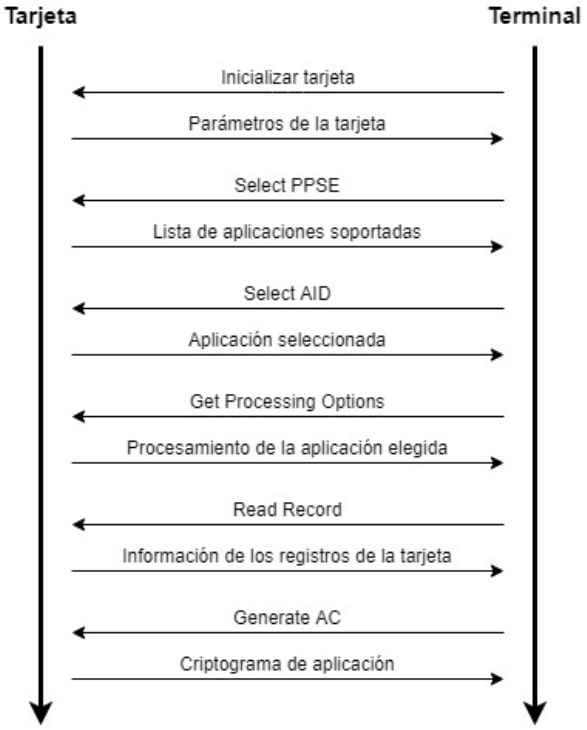


Figura 3.2: Conjunto de comandos que componen el flujo de una transacción

### 3.4.1. Comando “Select PPSE”

El desarrollo de este comando se realiza modificando el fichero emv_interop. En este fichero, proporcionado por ST, se realizarán todos los cambios necesarios para implementar el proyecto. Para empezar, se decidió crear una función para cada uno de los cinco comandos, de tal forma que la función principal de emv_interop solamente realizase llamadas a estas funciones y comprobase que la longitud de las respuestas era distinta de dos. Estas funciones devolverán por parámetro la longitud de la respuesta a los comandos, de tal manera que la función principal pueda comprobar si ha habido un error o no.

También se crearon dos funciones de respuesta, una de éxito con el "tick" verde y sonido de éxito y otra de error con la "X" y el sonido de error, de tal manera que en la función principal de emv_interop.c simplemente se llamase a una de estas funciones en base a la longitud de la respuesta.

De esta manera, se creó una función encargada de la ejecución del comando "Select PPSE". Los bytes correspondientes a este comando ya venían en el fichero emv_interop asociados a una variable local, como guía para que el desarrollador sepa cómo ejecutar los demás comandos. Para este comando, los valores de los parámetros son los indicados en la Tabla 3.1.

| CAMPO | VALOR | DESCRIPCIÓN |
|---|---|---|
| Clase | 00 | N/A |
| Instrucción | A4 | N/A |
| P1 | 04 | Significa que el parámetro 3 (P3) servirá para elegir el descriptor de fichero por su nombre |
| P2 | 00 | Significa que obtiene la primera o única ocurrencia de este fichero |
| P3 | 0E | Se corresponde con la longitud del campo datos |
| Campo datos | 32 50 41 59 2E 53 59 53 2E 44 44 46 30 31 | Contiene el identificador del fichero que estamos buscando, en este caso, una serie de bytes correspondientes al fichero que muestra todas las aplicaciones que soporta esta tarjeta |
| Final del comando | 00 | N/A |

Tabla 3.1: Estructura del comando "Select PPSE"

En el caso de la tarjeta de prueba utilizada, el comando sería:

```
static const uint8_t emvSelectppseApdu[] = { 0x00, 0xA4, 0x04, 0x00, 0x0E, 0x32, 0x50, 0x41, 0x59, 0x2E,
0x53, 0x59, 0x53, 0x2E, 0x44, 0x44, 0x46, 0x30, 0x31, 0x00 };
```

Una vez tenemos definido el comando a ejecutar, podemos implementar el método correspondiente a este comando. En primer lugar, se muestra el comando en el log y se ejecuta. Si la respuesta es distinta de 2, se muestra en el log y se guarda en una variable dinámica local, haciendo uso de malloc() y de memcpy(). Un procedimiento muy similar a este se utilizará en el resto de los comandos.

En cuanto a la respuesta del lector a este comando (Figura 3.3 y Figura 3.4), en primer lugar, nos devuelve el identificador del fichero que buscábamos y, en segundo lugar, cada una de las aplicaciones que soporta. En este caso solamente aparece una, con el ID de aplicación de Mastercard, su nombre "MASTERCARD" y la prioridad de la misma.

```
0x200038C8: 6F 2F 84 0E 32 50 41 59 2E
0x200038D1: 53 59 53 2E 44 44 46 30 31
0x200038DA: A5 1D BF 0C 1A 61 18 4F 07
0x200038E3: A0 00 00 00 04 10 10 50 0A
0x200038EC: 4D 41 53 54 45 52 43 41 52
0x200038F5: 44 87 01 01 90 00 00 60 F9
0x200038FE: 00 00 00 00 00 00 00 F0 49
0x20003907: 00 20 40 70 01 20 A0 33 00
```

Figura 3.3: Respuesta de la tarjeta al comando “Select PPSE”

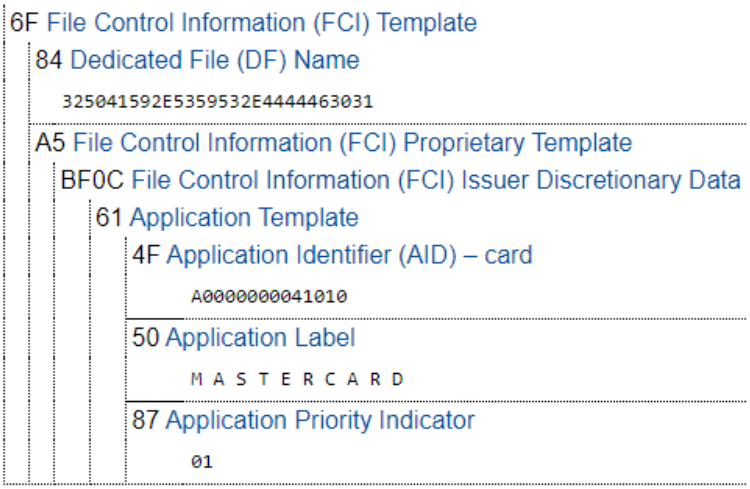


Figura 3.4: Respuesta al comando “Select PPSE” procesada por el decodificador TLV

### 3.4.2. Comando “Select AID”

Continuamos con el segundo comando a ejecutar, según dicta el estándar EMV, consistente en elegir el Application IDentifier referido a la aplicación de Mastercard. Por lo tanto, se utiliza el mismo comando de selección de fichero que en el caso anterior, siguiendo lo definido en la Tabla 3.2.

| CAMPO | VALOR | DESCRIPCIÓN |
|---|---|---|
| Clase | 00 | N/A |
| Instrucción | A4 | N/A |
| P1 | 04 | Significa que el parámetro 3 (P3) servirá para elegir el descriptor de fichero por su nombre |
| P2 | 00 | Significa que obtiene la primera o única ocurrencia de este fichero |
| P3 | 07 | Se corresponde con la longitud del campo datos |
| Campo datos | A0 00 00 00 04 10 10 | Contiene el identificador del fichero que estamos buscando, en este caso, el valor correspondiente a la aplicación de Mastercard |
| Final del comando | 00 | N/A |

Tabla 3.2: Estructura del comando “Select AID”

En el caso de la tarjeta de prueba utilizada, el comando sería:

```
static const uint8_t emvSelectAidApdu[] = { 0x00, 0xA4, 0x04, 0x00, 0x07, 0xA0, 0x00, 0x00, 0x00, 0x04,
0x10, 0x10, 0x00 };
```

Por otra parte, en la función destinada a la ejecución de este comando, se sigue un procedimiento similar al caso anterior. Se muestra el comando en el log y se ejecuta. En caso de que no devuelva un código de error, se muestra la respuesta en el log y se almacena en memoria.

En la respuesta a este comando, el lector devuelve el identificador de la aplicación elegida y su etiqueta “MASTERCARD”, además de algún otro dato poco relevante, como se muestra en la Figura 3.5 y en la Figura 3.6.

```
0x20003918: 6F 1F 84 07 A0 00 00 00 04
0x20003921: 10 10 A5 14 50 0A 4D 41 53
0x2000392A: 54 45 52 43 41 52 44 BF 0C
0x20003933: 05 9F 4D 02 0B 0A 90 00 00
0x2000393C: 20 F9 00 00 00 00 00 00 00
0x20003945: 00 00 00 00 00 00 00 00 00
0x2000394E: 00 00 00 00 00 00 00 00 00
0x20003957: 00 00 00 00 00 00 00 00 00
```

Figura 3.5: Respuesta de la tarjeta al comando “Select AID”

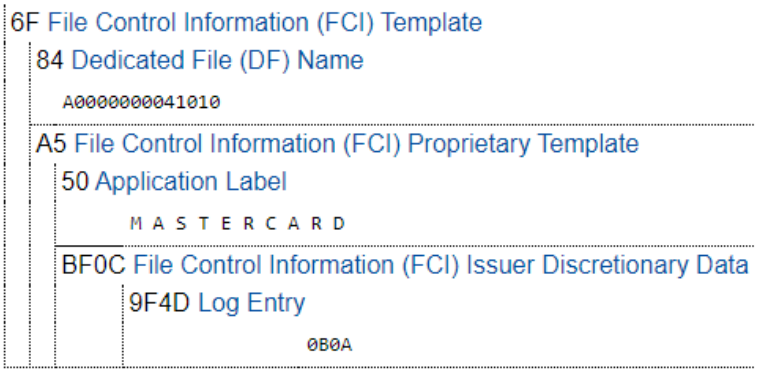


Figura 3.6: Respuesta al comando "Select AID" procesada por el decodificador TLV

Cabe destacar que es posible que este comando devuelva otro dato muy importante, introducido por la etiqueta 9F38. Esta etiqueta se refiere al PDOL (Processing Options Data Object List), una lista de las etiquetas de los datos que el terminal tiene que mandar a la tarjeta, como puede ser las capacidades adicionales del terminal (etiqueta 9F40), y sus longitudes. Es decir, además de todo lo que ha devuelto la tarjeta de prueba, puede devolver un campo similar al descrito en la Tabla 3.3.

| **TAG** | 9F38 | | Etiqueta de introducción del PDOL |
|---|---|---|---|
| **LENGTH** | 06 | | Longitud del PDOL |
| **VALUE** | TAG 1 | 9F40 | Etiqueta a la que el terminal debe dar valor |
| | LENGTH 1 | 05 | Longitud del valor 9F40 |
| | TAG 2 | 9F5C | Etiqueta a la que el terminal debe dar valor |
| | LENGTH 2 | 08 | Longitud del valor 9F5C |
| | … | … | … |
| | TAG N | Etiqueta N | Etiqueta a la que el terminal debe dar valor |
| | LENGTH N | Longitud N | Longitud del valor N |

Tabla 3.3: Estructura de un posible PDOL de una tarjeta

En particular, la tarjeta de prueba utilizada no contiene este campo.

### 3.4.3. Comando “Get Processing Options”

A partir de este comando ya se empiezan a realizar acciones propias de la aplicación de Mastercard, más concretamente se busca obtener los identificadores de los ficheros donde se almacenan los datos de la tarjeta.

Cabe destacar que a partir de ahora los comandos que se van a ejecutar son dinámicos. Es decir, en función de la tarjeta bancaria que se use, el software generará un comando u otro. Esto es debido a que, según el tipo de tarjeta, puede ser necesario incluir el campo referido al PDOL o no.

Consultando la parte del manual de EMV correspondiente a este comando [15], se puede definir el comando como se muestra en la Tabla 3.4.

| CAMPO | VALOR | DESCRIPCIÓN |
|---|---|---|
| Clase | 80 | N/A |
| Instrucción | A8 | N/A |
| P1 | 00 | N/A |
| P2 | 00 | N/A |
| P3 | 02 | Se corresponde con la longitud del campo datos |
| Campo datos | 83 00 | Se encuentra, en primer lugar, la longitud del campo datos. Este campo es siempre introducido por la etiqueta 83 y contiene los datos referidos a las etiquetas del PDOL. En el caso de que la tarjeta no tenga PDOL, se pone el campo longitud a 00 |
| Final del comando | 00 | N/A |

Tabla 3.4: Estructura del comando “Get Processing Options”

En el caso de la tarjeta de prueba utilizada y teniendo en cuenta que este comando es dinámico, el comando sería:

0x80, 0xA8, 0x00, 0x00, 0x02, 0x83, 0x00, 0x00

El ejemplo del apartado anterior, de una tarjeta con PDOL, se implementaría como se muestra en la Tabla 3.5.

| CAMPO | VALOR | DESCRIPCIÓN |
|---|---|---|
| Clase | 80 | N/A |
| Instrucción | A8 | N/A |
| P1 | 00 | N/A |
| P2 | 00 | N/A |
| P3 | 0F | Se corresponde con la longitud del campo datos |
| Campo datos | 83 0D 11 11 11 11 11 00 00 00 00 00 00 00 00 | Es necesario incluir después de la etiqueta 83, la suma de la longitud de los datos del PDOL, en este caso, 05 + 08 = 0D y posteriormente los datos referidos a esas etiquetas. Suponemos que los valores que queremos asignar a la etiqueta 9F 40 son todo unos y a la etiqueta 9F 5C son todo ceros |
| Final del comando | 00 | N/A |

Tabla 3.5: Estructura del comando "Get Processing Options" en el caso de que la tarjeta de prueba tuviese PDOL

Por lo tanto, el comando sería:

0x80, 0xA8, 0x00, 0x00, 0x0F, 0x83, 0x0D, 0x11, 0x11, 0x11, 0x11, 0x11, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

En cuanto a la implementación de las funciones relativas a este comando, se ha creado una función principal encargada de ejecutarlo, similar a la de los otros comandos, y dos funciones auxiliares.

- Función auxiliar 1: esta función rellena la cabecera del comando, es decir, la clase, instrucción, P1 y P2. Esta cabecera será siempre fija. También añade la longitud y el campo datos en función de lo devuelto por la otra función auxiliar.
- Función auxiliar 2: esta función genera el PDOL en función de los datos devueltos del comando "Select AID", en caso de que sea necesario, y se lo devuelve a la función auxiliar 1.
- Función principal: la función encargada de ejecutar el comando llama a la función auxiliar 1, que genera la cabecera y añade el 00 que se debe introducir al final de todos los comandos. Posteriormente, muestra el comando en el log, lo ejecuta y libera la memoria utilizada para su escritura. Comprueba que la longitud de la respuesta es distinta de dos y si ha sido así, muestra la respuesta en el log y la guarda en memoria.

En la respuesta a este comando encontramos uno de los datos más importantes del flujo EMV, el Application File Locator (AFL), que indica la localización de los ficheros elementales. La información proporcionada por esta etiqueta se lee en conjuntos de 4 bytes, como se muestra en la Tabla 3.6.

| BYTE | DESCRIPCIÓN |
|---|---|
| 1º | Los 5 bits más significativos indican el SFI (Short File Identifier) y los 3 bits menos significativos siempre se establecen a 0 |
| 2º | Indica el primer o único registro que se debe leer para ese SFI |
| 3º | Indica el último registro que se debe leer para ese SFI. Este número siempre debe ser igual o mayor al segundo byte. En caso de que sea mayor, se deberán leer todos los registros entre el primero y el último, ambos inclusive. En caso de que el último registro sea igual al primero, solo se leerá el registro marcado por el segundo byte |
| 4º | Indica el número de registros involucrados en la autenticación de datos offline, empezando desde el segundo byte. Este byte puede tomar un valor entre cero y el valor del tercer byte, menos el segundo byte más uno ([0, (B3-B2)+1]). Este dato no será relevante para el desarrollo de este proyecto |

Tabla 3.6: Estructura del AFL

La respuesta devuelta es la que se muestra en la Figura 3.7 y en la Figura 3.8.

```
0x20003990: 77 16 82 02 19 80 94 10 08 01 01
0x2000399B: 00 08 02 02 01 10 02 04 00 18 02
0x200039A6: 02 00 90 00 00 00 B0 F8 00 00 00
```

Figura 3.7: Respuesta de la tarjeta al comando “Get Processing Options”


Figura 3.8: Respuesta al comando “Get Processing Options” procesada por el decodificador TLV

De esta manera, en el caso de la tarjeta probada, hay que dividir el AFL en 4 partes. En la primera de ellas, tendremos que diseccionar el primer byte para saber cuál es el SFI.

08 = 0000 1000 → SFI = 01

Por lo tanto, a la hora de leer estos registros, se tiene que leer el registro número uno del fichero cuyo SFI sea el 01. El AFL será necesario para ejecutar el próximo comando.

### 3.4.4. Comando “Read Record”

Este comando, al igual que el anterior, también será dinámico, en función de la respuesta del comando anterior. Será necesario ejecutarlo tantas veces como registros de diferentes SFIs nos haya revelado la respuesta del comando “Get Processing Options”.

Por lo tanto, y observando el AFL proporcionado por la respuesta anterior, en el caso de la tarjeta probada será necesario hacer varias lecturas:

- El registro 01 del fichero cuyo SFI es 01
- El registro 02 del fichero cuyo SFI es 01
- Los registros 02, 03 y 04 del fichero cuyo SFI es 02
- El registro 02 del fichero cuyo SFI es 03

De esta manera, observando el manual de EMV utilizado [15], se define el comando como se muestra en la Tabla 3.7.

| CAMPO | VALOR | DESCRIPCIÓN |
|---|---|---|
| Clase | 00 | N/A |
| Instrucción | B2 | N/A |
| P1 | 01 | Toma el valor del registro que se quiere leer |
| P2 | 0C | Byte cuyos 5 bits más significativos se refieren al SFI del fichero buscado, mientras que los 3 bits menos significativos se establecen siempre a 100 |
| Final del comando | 00 | N/A |

Tabla 3.7: Estructura del comando “Read Record”

Un ejemplo de dos de las seis lecturas que se tienen que realizar son:

0x00, 0xB2, 0x01, 0x0C, 0x00

0x00, 0xB2, 0x03, 0x14, 0x00

A la hora de implementar el funcionamiento de este comando, se creó la función encargada de ejecutarlo y dos funciones auxiliares.

- Función principal: se encarga de recorrer la respuesta al comando “Get Processing Options”, que se ejecuta justo antes que este comando, y dentro de esta, busca el AFL y lo recorre en conjuntos de cuatro bytes guardando en variables el SFI y el límite inferior y superior del intervalo de registros a leer. Posteriormente, llama a la función auxiliar 1, pasándole estas variables por parámetro.

- Función auxiliar 1: genera el comando, recibiendo por parámetro el límite inferior y el SFI.
- Función auxiliar 2: recorre todos los registros comprendidos en el intervalo y genera, para cada registro, el comando llamando a la función auxiliar 2. A continuación, muestra el comando en el log, lo ejecuta y libera la memoria ocupada por el mismo. Comprueba si la longitud de la respuesta es distinta de dos, y si es así, muestra la respuesta en el log y la almacena en memoria.

La función auxiliar 1 realiza el mismo proceso para todos los registros comprendidos en el intervalo dentro del SFI pasado por parámetro y la función principal realiza este procedimiento para cada conjunto de cuatro bytes del AFL; en el caso de la tarjeta con la que se ha probado, cuatro veces.

Acerca de las respuestas (Figura 3.9 y Figura 3.10), en el caso del comando "Read Record" se tiene una por cada vez que se ha tenido que ejecutar; en el caso de la tarjeta de prueba, seis veces. En estas respuestas se pueden encontrar datos de la tarjeta, muchos de ellos necesarios para poder realizar las transacciones y otros meramente informativos. Tomando como ejemplo el caso en el que se encuentra más información relevante, concretamente la lectura del registro 02 del fichero con SFI 01, se observa información acerca del número de tarjeta (PAN), la fecha de expiración, el código del país emisor de esta o códigos de acción del emisor.

También se puede observar uno de los datos más relevantes que una tarjeta puede devolver, el CDOL1 (Card Risk Management Data Object List 1), que es una lista de los objetos, con su etiqueta y longitud, que se le pasarán al ICC para la generación del primer comando "First Generate AC", el próximo que se ejecutará. El funcionamiento del CDOL1 es muy similar al del PDOL explicado en el comando "Select AID". De este modo, seguido de la etiqueta 8C, se encuentra la longitud del CDOL1, 21 en este caso, y una serie de etiquetas y longitudes de todos los objetos que es necesario que aporte el terminal mediante el próximo comando para que la transacción se pueda llevar a cabo correctamente.

```
0x20003B48: 70 81 81 5F 25 03 13 09 01 5F 24 03 18
0x20003B55: 10 31 9F 07 02 FF 00 5A 08 54 89 13 01
0x20003B62: 46 85 56 26 5F 34 01 00 8E 0E 00 00 00
0x20003B6F: 00 00 00 00 00 42 03 1E 03 1F 03 9F 0D
0x20003B7C: 05 B4 50 84 00 00 9F 0E 05 00 00 00 00
0x20003B89: 00 9F 0F 05 B4 70 84 80 00 8C 21 9F 02
0x20003B96: 06 9F 03 06 9F 1A 02 95 05 5F 2A 02 9A
0x20003BA3: 03 9C 01 9F 37 04 9F 35 01 9F 45 02 9F
0x20003BB0: 4C 08 9F 34 03 8D 0C 91 0A 8A 02 95 05
0x20003BBD: 9F 37 04 9F 4C 08 5F 28 02 07 24 9F 4A
```

Figura 3.9: Una de las respuestas de la tarjeta al comando "Read Record"

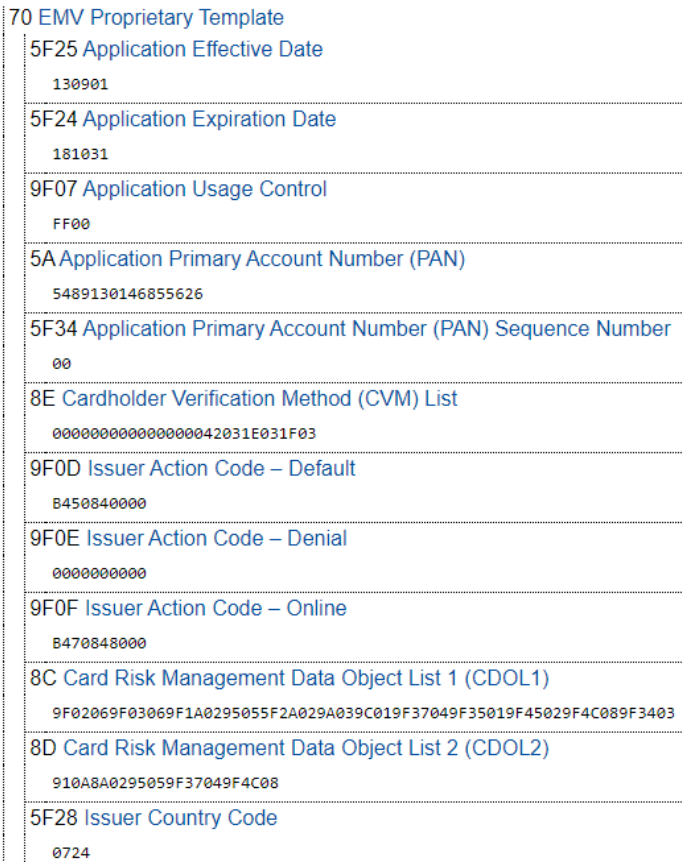


Figura 3.10: Una de las respuestas al comando "Read Record" procesada por el decodificador TLV

### 3.4.5. Comando "First Generate AC"

El "First Generate AC" es el último de los cinco comandos que conforman el proceso base de comunicación entre el TPV y la tarjeta (ICC), como se indicó en la Figura 3.2. Mediante el mismo, el terminal envía un criptograma con información relacionada con la transacción al ICC, el cual genera y devuelve otro criptograma de aplicación (Application Cryptogram). El terminal, para saber qué información debe mandar al ICC, se basa en el CDOL1 enviado en la respuesta al comando anterior (Read Record). Los criptogramas que se intercambian entre el terminal y el ICC pueden ser de tres tipos, como se muestra en la Tabla 3.8.

| | |
|---|---|
| AAC (Application Authentication Cryptogram) | Transacción declinada |
| ARQC (Authorisation Request Cryptogram) | Autorización online solicitada |
| TC (Transaction Certificate) | Transacción aprobada |

Tabla 3.8: Tipos de criptogramas

Para comenzar el proceso, el terminal realiza un análisis de riesgo basándose en la información obtenida en las respuestas a los comandos anteriores, sobre todo en base a los campos Issuer Action Codes (IAC) y Terminal Action Codes (TAC), proporcionados en la respuesta al comando "Read Record", descrito en la subsección 3.4.4. En función del resultado del análisis, el terminal envía a la tarjeta un tipo de criptograma u otro. Posteriormente, la tarjeta (es decir, el ICC) con los datos aportados por el terminal, realiza otro análisis de riesgo. Como en el caso del terminal, la tarjeta, en función del resultado del análisis, responderá con un tipo de criptograma u otro. El análisis de riesgo realizado por el terminal y el realizado por la tarjeta son diferentes por lo que el criptograma devuelto por el ICC puede diferir del enviado por el terminal. Hay diversos escenarios, según el resultado del análisis de riesgo realizado por el terminal:

- El terminal envía un AAC: El único caso en el que se enviaría un AAC sería en el que el servicio que se ha solicitado no está permitido para ese terminal.
- Cuando el terminal envía un TC hay que estudiar 3 escenarios asociados a la transacción:
    - Que el análisis de riesgo realizado por la tarjeta determine que el riesgo offline es aceptable. En este caso, se envía un TC al terminal y se acepta la transacción de manera offline.
    - Que el análisis de riesgo realizado por la tarjeta determine que el riesgo offline no es aceptable, se deniega la transacción y se envía un AAC al terminal.
    - Que, dándose las condiciones del punto anterior, es decir, que el riesgo offline no sea aceptable, se decida consultar al emisor (en este caso Redsýs), si autoriza realizar la transacción online.
      En este caso, el proceso sería el siguiente: la tarjeta devuelve un ARQC al terminal, determinando que la transacción debe continuar de manera online. En ese momento, el terminal se conecta con el emisor, que realiza un análisis de riesgo y decide si aceptar o rechazar la transacción. Hay dos posibles escenarios:
        - En el caso de que la acepte, será necesario ejecutar un segundo comando "First Generate AC" sobre la tarjeta, esta vez utilizando el CDOL2, presentado por la etiqueta 8D en la respuesta al comando "Read Record". Este comando se genera exactamente de la misma manera que el primer "First Generate AC" pero dándole valor a las etiquetas del CDOL2 en vez de las del CDOL1 y el tipo de criptograma enviado será ARQC. Por lo tanto, el terminal envía este criptograma al ICC, que vuelve a realizar un análisis de riesgo con todos los datos enviados en las dos comunicaciones y decide si devolver un TC, autorizando la transacción de forma online o denegándola, enviando un AAC.
        - En el caso de que el emisor deniegue la transacción al conectarse con él después de la primera comunicación terminal-tarjeta, el terminal enviará un AAC a la tarjeta, y esta también devolverá un AAC al terminal, dando por denegada la transacción.
- El terminal envía un ARQC: Esto se produciría si el terminal, como primera opción, decide ejecutar la transacción de manera online, ante lo que la tarjeta

solo puede responder con AAC en caso de que no lo soporte o con otro ARQC, siguiendo el proceso explicado en el punto anterior.

Todo lo explicado hasta este punto se resume en la Figura 3.11, donde se puede observar el flujo de decisión y los comandos utilizados.

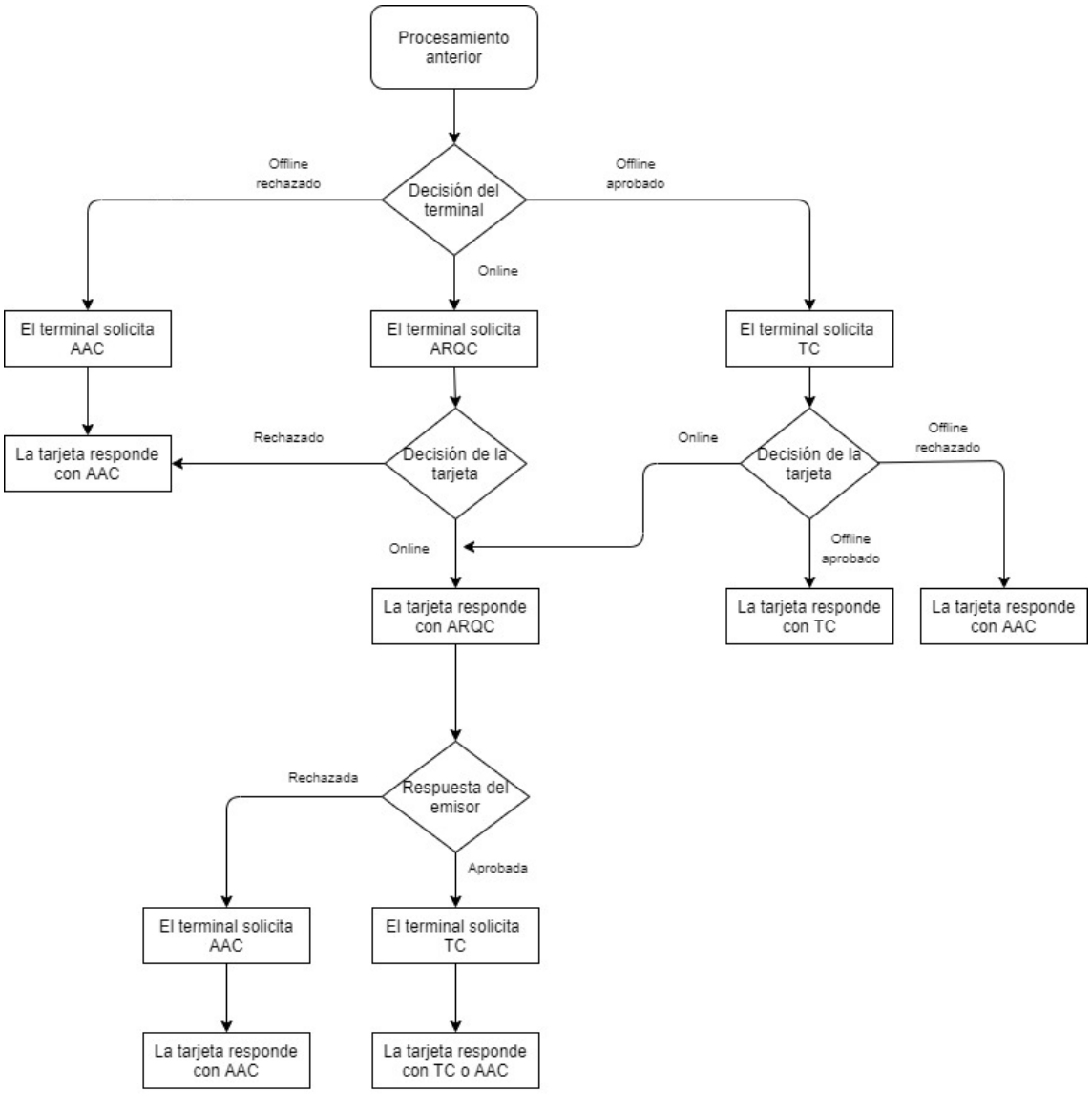


Figura 3.11: Diagrama de flujo de la generación del criptograma

Existe otro término importante en este comando, la CDA (Combined Data Authentication), que es en una firma dinámica generada por el ICC y calculada con diferentes claves aleatorias, combinadas con diversos datos del "Application Cryptogram". El terminal debe comprobar que esta firma es correcta utilizando los mismos datos que el ICC para poder continuar con la transacción. Es el propio terminal el que decide si se realiza la transacción con CDA, aunque para ello es necesario que

tanto la tarjeta como el terminal soporten esta tecnología, ya que es relativamente nueva. Sus predecesoras son el SDA (Static Data Authentication) que utiliza un número estático para realizar la firma y el DDA (Dinamic Data Authentication) que usa solamente un número aleatorio para firmar. Es aplicable tanto al primer como al segundo criptograma.

Hasta aquí se ha explicado el funcionamiento general del comando "First Generate AC". A continuación, se explicará el funcionamiento del mismo para el caso particular de las tarjetas Mastercard en la placa del proyecto.

Si se observa el manual de EMV [15], se forma el comando como se muestra en la Tabla 3.9.

| CAMPO | VALOR | DESCRIPCIÓN |
|---|---|---|
| Clase | 80 | N/A |
| Instrucción | AE | N/A |
| P1 | 40 | Será variable, ya que se corresponde con el tipo de criptograma que se quiere solicitar al ICC y si la transacción requiere firma o no. En el caso de que se desee un criptograma AAC, los dos bits más significativos se ponen a 0, en el caso de un TC se ponen a 01 y en el caso de ARQC se ponen a 10. En cuanto a la firma, si el terminal requiere CDA, es necesario poner el bit 5 a 1, y si no es necesario, a 0. El resto de los bits (6, 4, 3, 2 y 1) se pondrán a cero. En las pruebas de este proyecto se solicitarán criptogramas de tipo TC (transacción autorizada) y ninguna de las transacciones requerirá firma, ya que estamos en un entorno de desarrollo, por lo que el valor de este parámetro en las pruebas siempre será 40 |
| P2 | 00 | N/A |
| P3 | 2B | Se corresponde con la longitud que ocuparán todos los datos que el CDOL1 requiere que sean insertados por el terminal para completar la transacción |
| Campo datos | 00 00 00 00 00 01 00 00 00 00 00 00 07 24 00 00 00 00 00 09 78 16 08 11 00 11 11 11 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | En este campo se tienen que recorrer todas las etiquetas y longitudes que marca el CDOL1 e insertar un valor de ese tamaño para las mismas. En el caso de la tarjeta de prueba, la primera etiqueta (9F 02) se corresponde con el importe. Este valor tiene una longitud de seis bytes, de los cuales los cinco primeros se corresponden con los euros y el último con los céntimos en el caso de que la transacción se realice con esta moneda. El terminal necesita darle un valor a este objeto, por lo que al estar en un entorno de pruebas se le da el valor 00 00 00 00 00 01, que equivale a un céntimo de euro. Así sucesivamente habrá que |

| CAMPO | VALOR | DESCRIPCIÓN |
|---|---|---|
| | | darle valor a las etiquetas 9F 03, 9F 1A, 95, 5F 2A, 9A, 9C…, correspondientes, por ejemplo, al código del país del terminal, a la moneda en la que se realizará la transacción, a la fecha de la transacción o al tipo de terminal. |
| Final del comando | 00 | N/A |

Tabla 3.9: Estructura del comando "First Generate AC"

De modo que, para la tarjeta de pruebas, el comando sería algo similar a:

0x80, 0xAE, 0x40, 0x00, 0x2B, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x24, 0x00, 0x00, 0x00, 0x00, 0x00, 0x09, 0x78, 0x16, 0x08, 0x11, 0x00, 0x11, 0x11, 0x11, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

Cabe destacar que Mastercard soporta dos tipos de CDOL. Uno de ellos es el que tiene la misma forma y estructura que el usado en la tarjeta de prueba y el otro, añade al final del CDOL1 dos etiquetas más, 9F 21, de longitud 03, y 9F 7C, de longitud 14. Es decir, los objetos a los que es necesario dar valor en este comando son los mismos para la mayor parte de las tarjetas (Mastercard permite la extensión del CDOL1 en situaciones excepcionales, como puede ser el caso de unos pocos emisores de tarjetas). En este proyecto y con carácter provisional, se contemplarán el caso más común, y otro, que aparece en algunas tarjetas, cuyo CDOL1 está compuesto por dos etiquetas más.

A la hora de crear este comando, ha sido necesario implementar la función encargada de la ejecución principal del mismo y cinco funciones auxiliares. La primera de ellas genera la cabecera del comando, es decir, la clase, la instrucción, el parámetro 1, el parámetro 2 y la longitud, que obtendrá llamando a la segunda función auxiliar y a la que pasa por parámetro la etiqueta 8C, encargada de calcular este valor sumando todas las longitudes de los objetos del CDOL1. Al contemplar solo dos posibles tipos de CDOL1, se ha creado una serie de variables locales asignadas a los objetos requeridos en los dos CDOL1 que están compuestas por la etiqueta y por el valor que se le ha querido dar a ese objeto. Quedará de la siguiente forma:

```
static const uint8_t amount [] = {0x9F, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01}
```

De esta manera, la tercera función auxiliar se encarga de generar los datos requeridos para completar la transacción, concatenándolos mediante una llamada a la cuarta función por cada uno de los objetos a los que es necesario dar valor. Por lo tanto, cuando se llama a esta cuarta función se le pasa por parámetro el puntero al array donde

se está almacenando el CDOL1, la próxima posición del array disponible, la variable a concatenar y su tamaño. Al final de la tercera función contemplo el caso del CDOL que tiene dos etiquetas más. Por otra parte, la cuarta función auxiliar, simplemente concatena los valores al CDOL1 pasado por parámetro y actualiza los contadores. Por último, uno de los valores a rellenar (etiqueta 9F 37) es un número aleatorio que se utilizará para generar el criptograma de la tarjeta, para lo cual se ha creado una quinta función auxiliar generadora de números aleatorios que ejecuta un comando sobre la tarjeta para que devuelva un número aleatorio de una longitud de cuatro bytes. Este comando es 00 84 00 00 00. Para terminar, la función principal de este comando sigue el patrón de las anteriores funciones principales. Primero crea la cabecera, genera el CDOL1 y le añade el 00 al final de todos los comandos, para después mostrarlo en el log, ejecutarlo y liberar la memoria dinámica ocupada por este. Si la longitud de la respuesta es mayor de dos, la muestra en el log y la almacena en memoria.

En cuanto a la respuesta, como se muestra en la Figura 3.12 y en la Figura 3.13, la tarjeta devuelve el tipo de criptograma (ARQC, en este caso, al ser 80), el contador de transacciones realizadas con esa tarjeta, el criptograma de aplicación devuelto e información del emisor.

```
0x200040D0: 77 29 9F 27 01 80 9F 36 02 05 A8 9F 26 08
0x200040DE: A9 20 51 59 FE 3F 81 95 9F 10 12 02 10 A0
0x200040EC: 00 13 22 02 01 11 11 00 00 00 00 00 00 00
0x200040FA: FF 90 00 00 00 00 00 BB AD E4 28 00 00 00
```

Figura 3.12: Respuesta de la tarjeta al comando “First Generate AC”


Figura 3.13: Respuesta al comando “First Generate AC” procesada por el decodificador TLV

Por lo tanto, al devolver la tarjeta un criptograma ARQC será necesario ejecutar un último comando, “Second Generate AC”, pasando el CDOL2. A pesar de que como se ha explicado anteriormente, para esto es necesario conectarse con el emisor para saber si este acepta o rechaza la transacción tras el análisis de riesgo, en este proyecto se tiene que implementar únicamente la opción de que el emisor responda autorizando la transacción. Para ello, se seguirán unos pasos muy similares a los seguidos para la

implementación del comando “First Generate AC”. Consultando el manual de EMV [15], se construye el comando como se muestra en la Tabla 3.10.

| CAMPO | VALOR | DESCRIPCIÓN |
|---|---|---|
| Clase | 80 | N/A |
| Instrucción | AE | N/A |
| P1 | 40 | Su valor solo puede ser 40 (criptograma de tipo TC) o 00 (criptograma de tipo AAC). Este último valor solo se enviaría en caso de que el emisor hubiese denegado la transacción, opción que no se implementará en este proyecto |
| P2 | 00 | N/A |
| P3 | 1D | Longitud total que van a ocupar los valores asignados a las etiquetas del CDOL2, que se encuentran en una de las respuestas al comando “Read Record”, con etiqueta 8D |
| Campo datos | 00 00 00 00 00 00 00 00 00 00 11 11 00 00 00 00 00 57 30 76 90 11 11 11 11 11 11 11 11 | Valores asignados a las etiquetas de la misma forma que en el comando “First Generate AC”. Cabe destacar que hay algunas etiquetas del CDOL2 que ya se les ha dado valor en el CDOL1, por lo que los valores de estas etiquetas en el “Second Generate AC” deberán ser iguales que en el “First Generate AC”. Al resto de valores requeridos en el CDOL2 se les da un valor aleatorio |
| Final del comando | 00 | N/A |

Tabla 3.10: Estructura del comando “Second Generate AC”

Este comando sería algo parecido a:

0x80, 0xAE, 0x40, 0x00, 0x1D, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x11, 0x11, 0x00, 0x00, 0x00, 0x00, 0x00, 0x57, 0x30, 0x76, 0x90, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x00

A la hora de implementarlo, simplemente se ha creado un nuevo método muy similar a la función principal que ejecuta el comando “First Generate AC”, además de una función auxiliar. Este método auxiliar va insertando los datos de los objetos referenciados en el CDOL2 con el mismo mecanismo que en el primer Generate AC, usando variables locales que se han creado con la etiqueta y valor de estos objetos y ayudándose de otro de los métodos creados para el comando “First Generate AC”, que concatena todos estos datos y los devuelve por parámetro. Por otra parte, la función principal llama a otra de las funciones auxiliares implementadas para el comando anterior para generar la cabecera, pero esta vez pasando por parámetro la etiqueta 8D,

para que sea capaz de calcular la longitud que ocuparán todos los datos del CDOL2 a los que se les va a dar valor. Posteriormente, llama a la única función auxiliar creada específicamente para este comando, que devuelve por parámetro el comando completo, al que se le añade el 00 al final. Se muestra en el log, se ejecuta, y se libera la memoria utilizada por el mismo. Si la respuesta es mayor que dos, se muestra la respuesta en el log y se almacena. Por último, en la función principal del comando "First Generate AC", después del almacenamiento en memoria, se comprueba que el valor de la etiqueta 9F 27 es 80, y si es así, se llama a la función principal del comando "Second Generate AC".

En cuanto a la respuesta, como se muestra en la Figura 3.14 y en la Figura 3.15, este comando devuelve los mismos datos que el "First Generate AC". Sin embargo, en este caso se puede observar un detalle muy interesante. En el caso de la primera imagen, que es la respuesta producida por el comando ejecutado en WinExpert con el lector de tarjetas proporcionado por la empresa, se observa que la tarjeta devuelve un criptograma AAC (00), es decir, transacción denegada, ya que tras realizar el análisis de riesgo teniendo en cuenta la información proporcionada por el terminal así lo ha determinado. Por el contrario, en la segunda imagen, que es la respuesta devuelta a la ejecución del comando en la placa de mi proyecto, se observa que la tarjeta devuelve un criptograma de tipo ARQC (80). Por lo tanto, la tarjeta en su segundo análisis de riesgo considera que la placa es apta para realizar la transacción, pero con el lector de tarjetas utilizado como herramienta para el desarrollo, considera que no se cumplen las condiciones necesarias.

```
0x20004250: 77 29 9F 27 01 80 9F 36 02 05 C1 9F 26 08
0x2000425E: 38 A2 09 F5 8C 70 BE 24 9F 10 12 02 10 A0
0x2000426C: 00 13 22 06 01 11 11 00 00 00 00 00 00 00
0x2000427A: FF 90 00 F1 C6 85 E3 2D 18 30 08 00 00 00
```

Figura 3.14: Respuesta de la tarjeta al comando "Second Generate AC"


Figura 3.15: Respuesta al comando "Second Generate AC" procesada por el decodificador TLV

## 3.5. Fase 3. Inserción de las bibliotecas criptográficas

Llegados a este punto, es necesario cumplir con los requisitos de cifrado establecidos por el estándar PCI DSS (Payment Card Industry Data Security Standard) [11]. El estándar establece tres diferentes escenarios de cumplimiento:

- **Escenario 1:** El comercio trabaja con cifrado de pistas y no tendrá ninguna información relativa a la tarjeta que se procesa. La aplicación de pago del comercio solo tendrá acceso al BIN de la tarjeta
- **Escenario 2:** El comercio trabaja con cifrado de pistas, pero por necesidades de su negocio tiene acceso al PAN de la tarjeta. Ninguna otra información de la tarjeta estará disponible para el comercio, solo el PAN.
- **Escenario 3:** El comercio no trabaja con cifrado de pistas y, por lo tanto, la aplicación de pago del comercio tiene acceso a la información de las pistas en claro.

En el caso de este proyecto, se contemplará exclusivamente el escenario 1, de modo que, en un trabajo futuro se podrían contemplar los escenarios 2 y 3.

De esta forma, el protocolo Price, que define la comunicación entre el terminal y el centro autorizador, establece que para cumplir con las especificaciones del escenario 1 se tiene que cifrar tanto el PAN como la Pista 2. Esta es una pista que también se encuentra en la respuesta del comando "Read Record", como se muestra en la Figura 3.16 (en el caso de la tarjeta de prueba se encontraba en otro registro diferente).

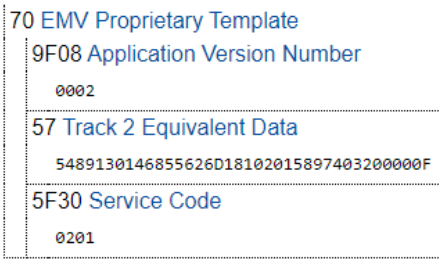


Figura 3.16: Respuesta de la tarjeta al comando "Read Record" en la que aparece la Pista 2

Este valor de la Pista 2, está compuesto por el PAN, un separador hexadecimal 'D', la fecha de caducidad de la tarjeta, el código de servicio y datos discrecionales definidos por los sistemas de pago.

Actualmente, el estándar bancario obliga a cifrar estas dos pistas mediante el algoritmo de cifrado simétrico TDES o 3DES con cifrado de bloque CBC. Este algoritmo se basa en el algoritmo de cifrado DES, pero hace un triple cifrado, es decir, el algoritmo DES se aplica tres veces con tres claves distintas, lo que hace que el algoritmo 3DES sea más seguro. Sin embargo, según el NIST (National Institute of Standards and Technology), está previsto que la seguridad del algoritmo de cifrado TDES se vea comprometida antes del 2030 [16], por lo que poco a poco las entidades bancarias y los procesadores de pago están migrando al algoritmo de cifrado AES, ya que este último tiene una

velocidad hasta seis veces mayor y, sobre todo, no está previsto que su seguridad se vea comprometida hasta después de 2030. Este proceso de migrado es lento y más aún en un entorno tan hermético como el bancario, por lo que en este proyecto se utilizará TDES para cifrar las pistas.

Para realizar el cifrado se ha utilizado la biblioteca criptográfica proporcionada por ST junto con la placa. En archivos de esta biblioteca relacionados con el algoritmo de cifrado TDES, se encontraban diversas funciones de cifrado y descifrado. Entre todas estas funciones, se encuentra el método de cifrado TDES con cifrado de bloque CBC, que tiene siete parámetros de entrada y devuelve si ha habido error en el cifrado o no. El primero de estos parámetros es el texto de entrada, que en este caso será el PAN o la Pista 2 y el segundo es la longitud de este texto. Para poder definir el tercero, la empresa ha proporcionado una serie de claves de 128 bits con un identificador del 0 al 9 cada una. De esta manera, según dicta el protocolo Price, el undécimo dígito del PAN (sin cifrar) determinará cuál será la clave a utilizar en el cifrado de las pistas, que se pasará como tercer parámetro a la función que cifra. El cuarto parámetro se corresponde con el vector de inicialización, se calcula en base a un número aleatorio distinto al asociado con la etiqueta 9F 37. El quinto parámetro es la longitud de dicho vector de inicialización, que será de tres bytes. El sexto parámetro se utiliza para devolver el texto ya cifrado, junto con el séptimo parámetro que indica la longitud del texto cifrado.

No todo lo explicado en el párrafo anterior está implementado para la placa sobre la que se está desarrollando este trabajo, por lo que he tenido que desarrollar una serie de funciones y objetos, que paso a describir a continuación, para que se pueda implementar el escenario 1 sobre la placa.

Llegados a este punto, se pone de manifiesto la necesidad de almacenar todos los objetos, con sus etiquetas, longitudes y valores, revelados en las repuestas a los comandos. Para ello, he creado un struct tTLV con tres punteros de tipo byte, tag, length y value, para después crear un buffer de tTLV llamado TLVList y un contador de este. De esta manera, se ha tenido que crear un método que se llamará en la función principal de cada uno de los comandos, después del almacenaje en memoria de la respuesta a estos comandos. Esta función se encargará de recorrer la respuesta y almacenar en el TLVList todos los objetos que en esta se encuentren, junto con su etiqueta, longitud y valor.

Además de la adaptación de la biblioteca criptográfica, para esta fase he tenido que crear una función que devuelve el undécimo dígito del PAN y un método donde se encuentran las claves proporcionadas por la empresa que recibe por parámetro el identificador de clave y devuelve, mediante un *switch*, la clave correspondiente.

Por otra parte, se ha tenido que crear la función que genera el vector de inicialización, que recibe por parámetro un número aleatorio de tres bytes, y los coloca en las tres primeras posiciones del vector. En las siguientes tres posiciones inserta los valores resultantes de hacer una XOR entre los tres bytes del vector aleatorio y tres bytes a FF y, para terminar, establece los dos últimos bytes a 00.

Por último, se ha creado el método donde se centraliza el cifrado, que recibe por parámetro un número aleatorio. Esta función calcula cuál es el undécimo dígito usando

el método auxiliar y busca en el TLVList los objetos con etiqueta 5A y 57 (PAN y Pista 2). Una vez encontrados, llama a la función de la biblioteca criptográfica y pasa como parámetros el valor de ese objeto, su longitud, la clave correspondiente al undécimo dígito, la respuesta de la función que calcula el vector de inicialización pasándole el número aleatorio por parámetro, su longitud, un puntero a un buffer vacío donde se insertará la respuesta y un cero (longitud actual de la respuesta). De esta forma, en la función principal de emv_interop.c, simplemente se tiene que añadir después de las llamadas a las funciones de todos los comandos, una llamada a la función de generación de números aleatorios y una llamada a esta función donde se centraliza todo el cifrado, pasando por parámetro el número aleatorio devuelto.

## 3.6. Fase 4. Generación y envío de un JSON al centro autorizador

Llegados a este punto, ya se tienen todos los datos necesarios para la ejecución de una transacción, pero es necesario crear el JSON (JavaScript Object Notation) para el intercambio de datos con el centro autorizador. No todos los datos obtenidos en la ejecución de los cinco comandos base de una transacción, es decir, de la fase 2 serán necesarios para la creación del mensaje. Para ello existen dos posibilidades, mandar el JSON con PUP al Gateway o mandarlo directo al Paystore. En el caso de este proyecto, se utilizará la segunda vía para realizar la conexión con el centro autorizador.

De esta manera y siguiendo el protocolo Price, establecido para estas comunicaciones, se debe crear un mensaje de petición online que enviará el PIN-PAD EMV integrado conteniendo el ARQC. Este mensaje debe estar compuesto por una serie de datos fijos para todas las transacciones desde el terminal como pueden ser datos del terminal, modelo, fabricante…, entre los cuales se tienen que insertar los datos concretos de la transacción que se quiere realizar. Estos datos se introducirán en el apartado "requestData", que está dividido en otros cinco subapartados.

- economicData: Está compuesto por dos valores, "amount", que en el caso de la transacción de prueba tiene el valor 00 00 00 00 00 01, y "currency", en este caso 978, valor atribuido a la moneda Euro.
- "encryptedData": Está compuesto por tres valores, "encrypted", que siempre estará a true, "rndNumber" que será un número aleatorio de tres bytes generado por el programa que se utilizará para poder cifrar las pistas y "keyId", número necesario también para cifrar pistas y que se corresponde con el undécimo dígito del PAN de la tarjeta leída.
- "cardData": Compuesto por un único valor, "cardBin", correspondiente al BIN (Bank Identification Number) de la tarjeta leída, es decir, los 6 primeros números del PAN.
- "numSec": Simple contador de secuencia.
- "emvData": En este subapartado se genera un buffer llamado "tlvData" compuesto por los datos obtenidos en la fase 2 que el centro autorizador requiere para completar la transacción. El orden es indiferente y se insertan de la siguiente manera:

```
“tlvData”:[
                    {
                    “tag”:”5F34”,
                    “value”:”00”
                    },
                    {
                    “tag”:”4F”,
                    “value”:”A0000000041010”
                    }
                    …
]
```

De esta forma se van concatenando todos los valores que establece el protocolo Price dentro de este buffer. Además, el subapartado “emvData” también tiene otro valor llamado “tlv” en el que se deben introducir todos los datos insertados en “tlvData” pero de manera consecutiva y siguiendo el formato TLV, de tal manera que quedaría algo similar a lo siguiente:

```
“tlv”:”5F3401004F07A0000000041010…”
```

En este campo se tienen que añadir exactamente los mismos objetos que los insertados en el valor “tlvData”, aunque no es necesario que sea en el mismo orden.

Tanto en el campo “tlvData” como en “tlv”, el PAN y la Pista 2 se deben insertar ya cifrados. De esta forma, la parte del JSON relativa a la tarjeta y a la transacción concreta que se quiere realizar quedaría de la manera que se expone en la Figura 3.17 y en la Figura 3.18.

```
"requestData":{
    "opData":{
        "economicData":{
            "amount":"0000000001",
            "currency":"978"
        },
        "encryptedData":{
            "encrypted":true,
            "rndNumber":"876534",
            "keyId":"8"
        },
        "cardData":{
            "cardBin":"548913"
        },
        "numSec":"1",
        "emvData":{
            "tlvData":[
                {
                "tag":"5F34",
                "value":"00"
                },
                {
                "tag":"4F",
                "value":"A0000000041010"
                },
                {
                "tag":"5A",
                "value":"9B2B3229652CA0E278FAA18E30925D73"
                },
                {
                "tag":"5F25",
                "value":"130901"
                },
                {
                "tag":"57",
                "value":"5D7CB068AABBEF6437A2DC1F9FE55FE71A7ADB49EFD185DB"
```


Figura 3.17: Estructura del del JSON

...


Figura 3.18: Estructura del JSON

Para que el software desarrollado crease este JSON se han tenido que realizar algunas modificaciones en el proyecto.

En primer lugar, se creó un puntero de bytes donde se almacenaría la información de “tlv”, un puntero de tipo char para almacenar la información de “tlvData” y otro puntero de tipo char para almacenar todo el mensaje.

De esta manera, se ha creado un método que se encarga de llamar a una función auxiliar pasando como parámetro cada una de las etiquetas que se insertarán en los campos “tlv” y “tlvData”. Este método auxiliar busca en TLVList un objeto cuya etiqueta sea la pasada por parámetro y cuando la encuentra, la inserta tanto en el buffer de “tlv” como en el buffer de “tlvData”, incluyendo en este último caso los corchetes, comas y comillas estipulados en el protocolo Price.

Por otra parte, todos los datos requeridos por el protocolo Price relacionados con la tarjeta se pueden encontrar en las respuestas a los comandos, menos dos, el primero es el número de intentos de PIN restantes, cuya etiqueta es 9F 17. Para poder obtener este dato es necesario ejecutar un comando adicional, llamado “Get Data”, que “pedirá” a la tarjeta el valor de una etiqueta concreta y esta se lo devolverá si lo tiene disponible. Para ejecutar este comando, y observando el manual de EMV [15], se define la clase a 80 y la instrucción a CA, siendo 9F 17 la etiqueta buscada, que se divide en dos partes, P1 (9F) y P2 (17). Para terminar, se añade el 00 al final del comando.

En cuanto a la implementación del comando, se ha creado una función que será llamada antes de la que ejecuta el comando “First Generate AC” y que recibe por parámetro la

longitud (como todas las funciones principales de los cinco comandos base) y la etiqueta que se está buscando. Esta función simplemente crea un comando con la clase, instrucción y parámetros definidos, lo muestra en el log, lo ejecuta y libera la memoria ocupada por el comando. Si la longitud de la respuesta es mayor que dos, se muestra en el log, se almacena en memoria y se llama a la función que repasa esta respuesta guardando todos los objetos, con su etiqueta, longitud y valor, que se encuentren en la misma.

En cuanto a la respuesta, simplemente se devuelve la etiqueta buscada con su valor, como se muestra en la Figura 3.19 y en la Figura 3.20.

```
0x200040B8: 9F 17 01 03 90 00 E9 9C 73 5F 90 7F 98 F1
0x200040C6: 00 00 00 00 00 00 FD BD F5 6F 13 7E D1 1E
0x200040D4: 28 F0 8D F7 C3 3F DA DB 54 FB 46 AD AF 68
```

Figura 3.19: Respuesta de la tarjeta al comando “Get Data”

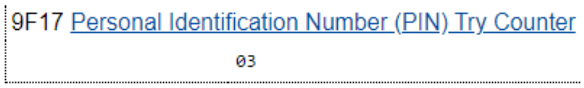


Figura 3.20: Respuesta al comando “Get Data” procesada por el decodificador TLV

El segundo dato que no se puede obtener de las respuestas de los comandos, es el que se corresponde con la etiqueta 9F 33, que está relacionado con las capacidades del terminal y tiene longitud 03. Es el propio terminal el que tiene que insertar aquí sus capacidades, por lo que, de momento, y siempre en entorno de prueba y de acuerdo con la empresa, se le dará valor 00 a estos tres bytes. En cuanto a la implementación, simplemente se crea una función que inserta un nuevo valor en el siguiente hueco disponible del TLVList, añadiendo la etiqueta 9F 33, la longitud 03 y el valor 00 00 00 y se invoca después de las llamadas a las funciones principales de todos los comandos.

También ha sido necesario crear algún método auxiliar para facilitar el flujo de información entre las funciones y aumentar la higiene del código, como, por ejemplo, una función que busca en el TLVList y devuelve el valor de una etiqueta pasada por parámetro.

Para continuar, se ha creado la función encargada de generar el mensaje completo, un buffer de tipo *char* en el que se insertan los valores fijos que utilizará este terminal en el periodo de pruebas para conectarse con el centro autorizador, siempre cumpliendo exhaustivamente con la sintaxis definida en el protocolo. Después, se llama varias veces a la función que devuelve un valor pasando una etiqueta por parámetro para poder insertar en el mensaje valores como el importe, el número aleatorio o la divisa. Posteriormente, se inserta el array donde se han almacenado los datos de “tlvData” y después los de “tlv”, lo que añade más información fija del terminal.

Por último, en la función principal de emv_interop, se incluye una llamada al método que va insertando las distintas etiquetas y valores en “tlv” y “tlvData” y una llamada al

método que genera el mensaje completo. Para terminar, se libera la memoria utilizada por los punteros de los buffers “tlv” y “tlvData”.

Con todo lo desarrollado hasta esta fase del proyecto, es posible leer una tarjeta que se conecta con el terminal mediante *contactless*, sacar de esta tarjeta la información necesaria para la ejecución de una transacción y crear un mensaje de acuerdo con el protocolo Price para que pueda ser enviado al centro autorizador, por lo que solo faltaría el envío de este mensaje.

Al no tener la placa un puerto para la conexión de red, de momento se enviará el JSON a una URL proporcionada por la empresa para comprobar su validez. Para esto se ha utilizado Postman [17], una herramienta que permite el envío de peticiones ‘HTTP request’ sin necesidad de desarrollar un cliente.

De este modo, se pondrá un punto de parada en el proyecto justo después de la generación del mensaje, de tal manera que, al acercar una tarjeta al terminal, se detenga en el punto de parada y se pueda copiar el mensaje generado. Después, se enviará la petición a una URL, que devolverá un mensaje de respuesta con los datos de la transacción ficticia y si ha sido aceptada o no, y en este último caso, el código de error.

En el caso de esta tarjeta, el centro autorizador no acepta la transacción, como se observa en el apartado “operationResult”: “N” de la Figura 3.21, con código de error 190, que significa “Denegación del emisor sin especificar motivo”. En el contexto de la crisis sanitaria provocada por la enfermedad Covid-19, a la hora de redactar este documento ha sido imposible disponer de una tarjeta de prueba proporcionada por la empresa que, al realizar el proceso provocase que el centro autorizador devolviese una respuesta exitosa.

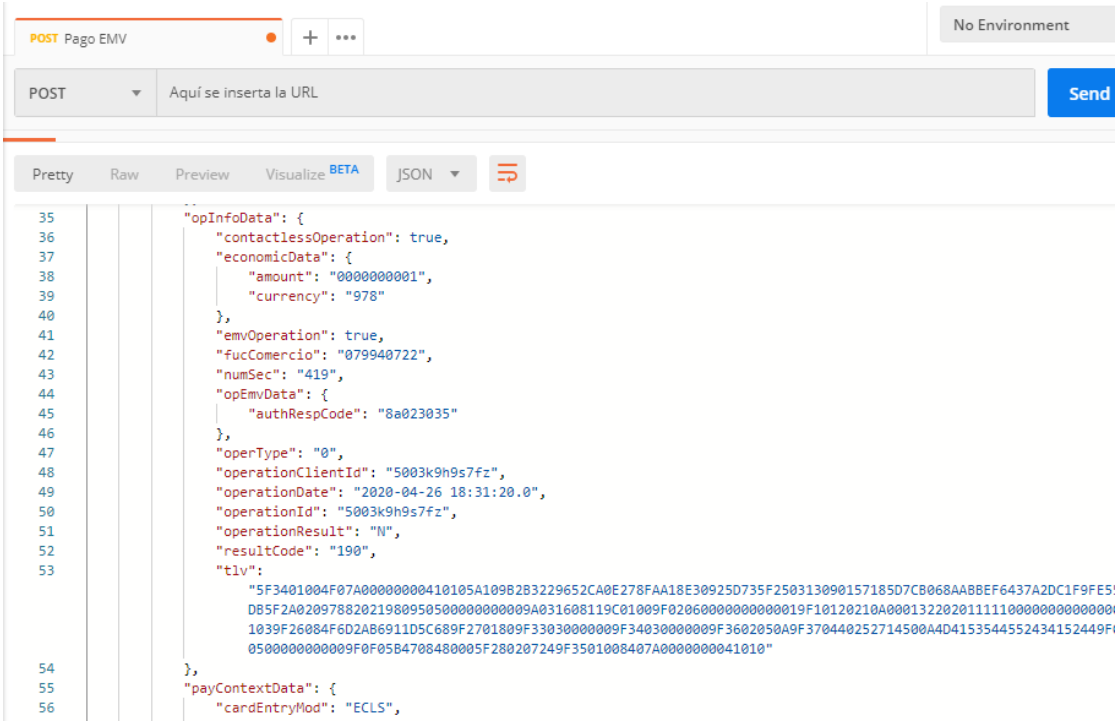


Figura 3.21: Respuesta del centro autorizador a la tarjeta de prueba

En cualquier caso, antes de que el Gobierno decretase el estado de alarma y la recomendación de establecer teletrabajo, se probaron otras tarjetas que sí provocaban que el centro autorizador devolviese un resultado satisfactorio. Como medida de precaución, se guardaron los mensajes generados por esas tarjetas y si se envían con Postman [17] devuelven una respuesta exitosa, como demuestra la presencia del campo "cardInfoData" y la "S" en "operationResult" en la Figura 3.22.

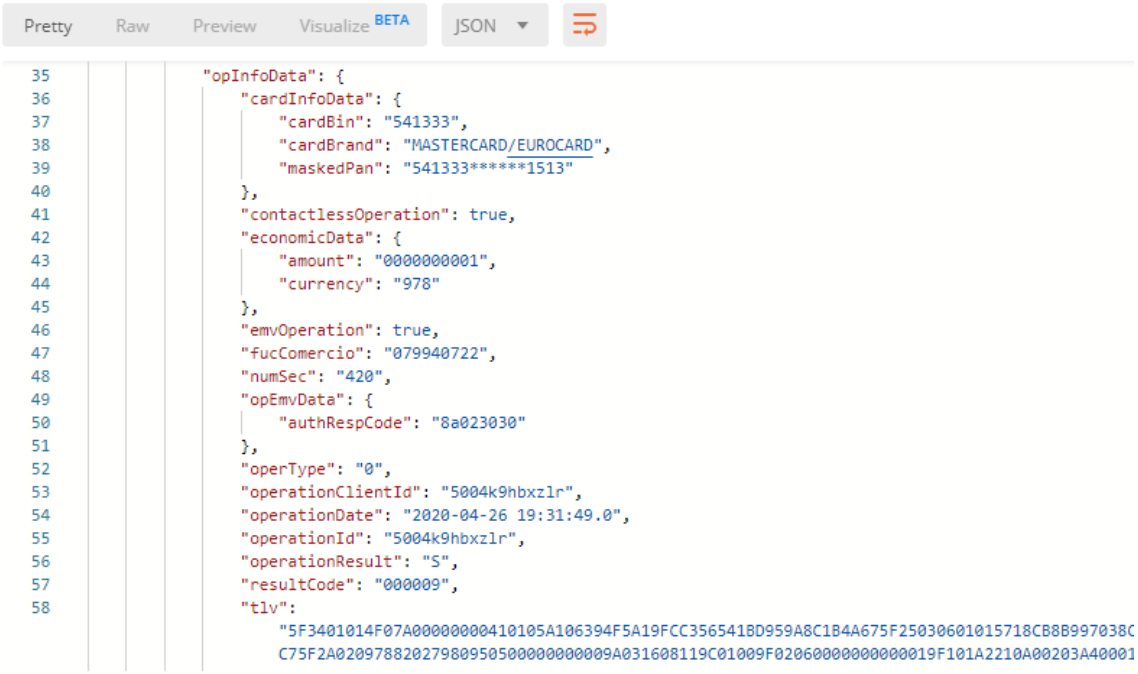


Figura 3.22: Respuesta del centro autorizador al mensaje generado por otra tarjeta de prueba

## 3.7. Fase 5. Integración del proyecto con un sistema operativo

En la planificación inicial del proyecto, estaba previsto que la siguiente fase fuese la instalación de un módulo de red en la placa, pero posteriormente, por decisión de diseño, se ha optado por adelantar la fase de la integración del proyecto con un sistema operativo y posponer la fase de la instalación de un módulo de red y el establecimiento de comunicaciones seguras. Por lo tanto, el proyecto continúa con la integración con un sistema operativo. Se plantearon dos opciones, una máquina virtual Java ME Embedded o un sistema operativo de la plataforma FreeRTOS [18]. La obtención de la MV Java resultó imposible, por lo que se decidió que se integraría el proyecto con un FreeRTOS.

Un sistema operativo de tiempo real o RTOS (Real Time Operative System) es un tipo concreto de sistema desarrollado para ejecutar aplicaciones que se caracterizan por tener unos requisitos específicos de ejecución y sobre todo de tiempo de respuesta. Se decidió utilizar un RTOS de software libre muy conocido para microcontroladores llamado

FreeRTOS. Este sistema operativo está implementado en forma de demos para varias arquitecturas, como Intel, Infineon, IBM… y también para ST, aunque no para todos los microcontroladores. Para ayudar a desarrollar esta fase del proyecto se estudió la información que aparece en la página web de FreeRTOS, donde hay un fichero zip con toda la información, que contiene una demo para el microcontrolador STM32L4 Discovery, muy similar al utilizado en este proyecto, que es el STM32L4. Este chip STM32L4 Discovery está integrado en la placa 32L476GDISCOVERY, que es una placa bastante distinta a la utilizada en este proyecto, ya que incorpora un joystick, un mayor número de leds, un LCD de 24 segmentos… Por lo tanto, se tendrá que adaptar esta demo a la placa que se está utilizando en el proyecto, esencialmente ajustando pines e interrupciones, para después poder integrarla con el software desarrollado en las fases 2, 3 y 4. Por lo tanto, a partir de ahora se trabajará sobre esta demo en vez de sobre el proyecto central.

Para ello, se han ido eliminando las configuraciones de pines que se encuentran en el main.h de la demo, buscando todas las referencias en el proyecto a esos pines para eliminarlas también y compilando y ejecutando después de cada cambio para comprobar que sigue funcionando. Una vez eliminada la configuración de pines definidos para STM32L4 Discovery, se añade en su lugar la configuración de pines del STM32L4 y se adapta a la demo a partir del esquemático proporcionado con el microcontrolador. En el momento en el que se ha escrito esta memoria del Trabajo de Fin de Grado, el proyecto ChipPOS se encontraba en esta fase y no ha podido completarse en su totalidad. Faltaría acabar de adaptar la configuración de pines al microcontrolador utilizado en este proyecto (STM32L4), para poder así comenzar a crear tasks que permitan la ejecución de tareas de forma planificada y eficiente, asignando niveles de prioridad y gestionando los recursos hardware de la placa adecuadamente.

## 3.8. Fase 6. Instalación de un módulo de red

La siguiente fase consistirá en la instalación del módulo de red WIZ610io [19] fabricado por Wiznet .Este módulo incluye, además de un conector RJ-45, un chip W6100 que proporciona ya la implementación de las tres primeras capas de red según el modelo OSI. Soporta protocolos TCP/IP como TPC, UDP, IPv6, IPv4, ICMPv6, ICMPv4… además de incluir 100Base-TX Ethernet PHY y un controlador MAC Ethernet, haciéndolo ideal para un proyecto como este. Este módulo de red se conectará a la placa mediante la interfaz SPI.

# Capítulo 4

# 4. Seguridad en las transacciones con tarjeta bancaria

## 4.1 Introducción

El uso de las tarjetas bancarias está tan ampliamente implantado en la sociedad actual gracias a los robustos mecanismos de seguridad implementados en ellas. Las personas pueden pagar tranquilamente con su tarjeta de crédito con la confianza de que no se les clonará la tarjeta ni se obtendrán sus datos para efectuar operaciones ajenas a su conocimiento. Es por ello que las tecnologías que rigen la seguridad de las transacciones bancarias deben estar constantemente renovándose y actualizándose, ya que, de la misma manera que ellas lo hacen, los atacantes continúan buscando maneras de quebrantar esta seguridad.

## 4.2 Seguridad en la comunicación tarjeta-emisor

Debido al riesgo que entrañaba la facilidad de duplicar una tarjeta de crédito en la tecnología de banda magnética, hace unos años las compañías bancarias tomaron la decisión de aumentar la seguridad de las tarjetas y comenzaron a implantar tecnologías basadas en tarjetas inteligentes. La mayor novedad fue la adición a las tarjetas de un chip, de tal forma que cada tarjeta se convierte en un pequeño ordenador capaz de, entre otras cosas, almacenar información, como, por ejemplo, las claves necesarias para garantizar la seguridad.

Es en el estándar EMV, mencionado anteriormente en esta memoria, donde se comienzan a implantar las tarjetas con chip, que utilizaban cifrado de clave privada. Estas tarjetas compartían una clave secreta con su banco emisor, el cual utilizaba esta clave secreta para generar una MAC en cada mensaje intercambiado entre ambos, verificando así su autenticidad. Sin embargo, para que el banco adquiriente pudiese comprobar dicha autenticidad era necesario que estuviese en línea, haciendo imposible garantizar la seguridad en transacciones offline.

Es unos años después cuando se comienza a aplicar la criptografía de clave pública en el ámbito bancario, lo que mejoraba considerablemente todo el proceso, permitiendo que la firma pudiese ser verificada por cualquier entidad que tuviera la clave pública del

emisor, pero nadie más que el emisor podía generar la firma usando su clave privada. Para que este método criptográfico funcione, es necesario que todos los terminales tengan una copia de las claves públicas de los emisores para poder verificar la autenticidad de la clave pública de la tarjeta [20].

La autenticidad es el proceso de confirmación de que algo o alguien es quien dice ser, y en el ámbito bancario es uno de los aspectos de seguridad fundamentales que deben ser garantizados para poder realizar una transacción bancaria de forma segura. Esta autenticidad debe garantizarse en dos puntos:

- Autenticidad de la tarjeta: Actualmente todas las tarjetas bancarias tienen un par de claves pública-privada único y diferente al del resto de tarjetas, proporcionado por el banco emisor. La clave pública de la tarjeta se firma digitalmente por el emisor para que el terminal pueda verificar su autenticidad.
  De esta manera, cuando se inserta una tarjeta en el terminal, la tarjeta envía el certificado digital de la tarjeta firmado digitalmente por el emisor al terminal, para que este pueda comprobar, utilizando la clave pública del emisor, que el certificado enviado es auténtico, y obteniendo de este modo la clave pública de la tarjeta.
  Una vez llegado a este punto, es necesario realizar una comprobación adicional, es necesario verificar si la tarjeta tiene una clave pública o no, ya que podría haber sido robada de otras tarjetas.
  Para ello, el terminal genera un desafío y se lo envía a la tarjeta para que lo firme con su clave privada y lo devuelva firmado al terminal. Por último, el terminal utiliza la clave pública de la tarjeta, obtenida en el paso anterior, para verificar la autenticidad del desafío firmado devuelto por la tarjeta.
  El funcionamiento general del proceso de verificación de la autenticidad de la tarjeta se explica esquemáticamente en la Figura 4.1.

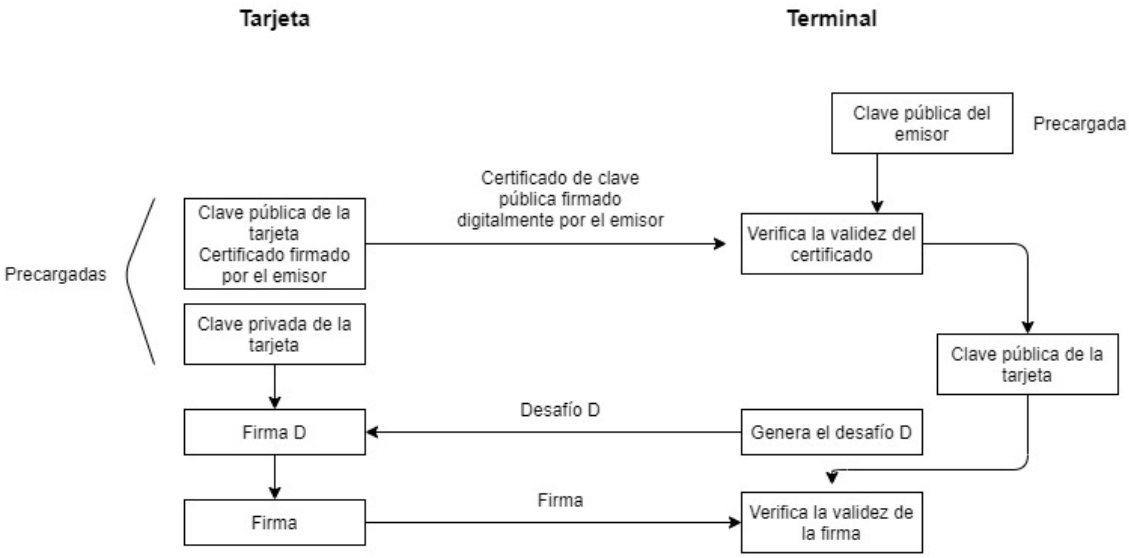


Figura 4.1: Proceso de autenticación de una tarjeta

- Autenticidad de la transacción: Cuando se realiza una transacción, el banco emisor necesita verificar la autenticidad de la misma. Para ello, cuando el terminal ya tiene toda la información relativa a la transacción, se la envía a la tarjeta para que esta la firme con su clave privada y se la devuelva al terminal. De este mecanismo se pueden beneficiar tanto el comercio propietario del terminal como el banco emisor, ya que en este punto el terminal puede comprobar que esa firma devuelta por la tarjeta es auténtica utilizando la clave pública de la tarjeta, mientras que, para el emisor, una firma válida junto con la información de una transacción significa que el dueño de la tarjeta ha aceptado realizarla.
  El proceso de verificación de la autenticidad de la transacción se explica brevemente en la Figura 4.2.

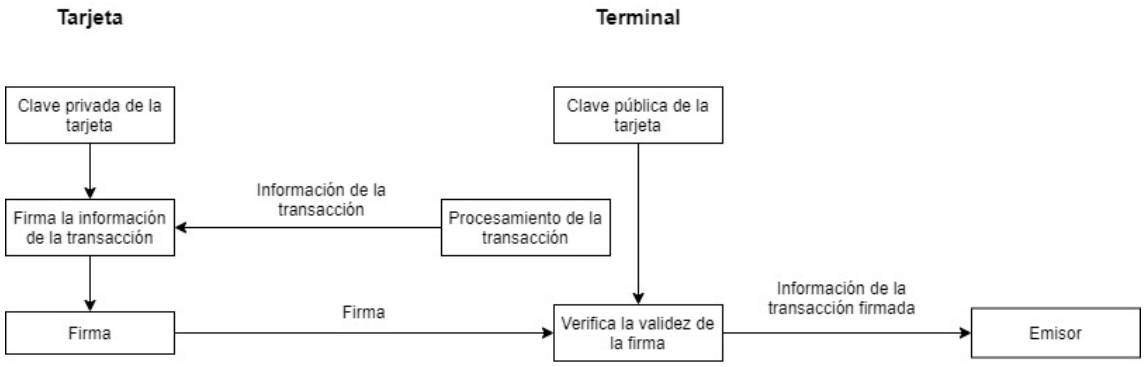


Figura 4.2: Proceso de autenticación de una transacción

## 4.3 Seguridad en las comunicaciones

Hasta hace unos años, el protocolo de cifrado de las comunicaciones utilizado en las transacciones bancarias era el SSL (Secure Sockets Layer), el cual con el tiempo ha evolucionado y actualmente se denomina TLS (Transport Layer Security), es decir, TLS es una versión más reciente de SSL, que corrige algunas vulnerabilidades de seguridad detectadas en las versiones anteriores del protocolo SSL. Ambos han sido completamente compatibles hasta la versión 1.3 de TLS (no acepta SSL) y su objetivo es proporcionar confidencialidad a la transacción bancaria que se quiere realizar. Actualmente, en las transacciones bancarias se utiliza principalmente la versión 1.2 del protocolo TLS.

Tanto SSL como TLS se basan en el uso de los certificados digitales para el establecimiento de comunicaciones seguras a través de Internet. Estos certificados digitales son documentos digitales únicos que garantizan la vinculación entre una persona o entidad con su clave pública. De esta forma el protocolo TLS funciona de la siguiente manera: cuando un terminal tiene toda la información de la transacción lista para enviar, se conecta con el emisor para solicitar que se identifique. En respuesta a

esta acción, el emisor responde presentando una firma digital y su certificado digital X509 para identificarse como sitio seguro. Posteriormente, el terminal realiza una serie de comprobaciones para verificar que ese certificado enviado por el emisor es válido, entre las que se encuentran:

- Integridad del certificado: verifica que el certificado es auténtico y no ha sido modificado, para ello verifica la firma digital incluida en él usando la clave pública de la Autoridad de Certificación (AC) que lo ha firmado.
- Vigencia del certificado: revisa el periodo de validez y si el certificado ha sido revocado.
- Verifica el emisor del certificado, haciendo uso de certificados raíz almacenados en el terminal y que contienen las claves públicas de las ACs de confianza [21]. El certificado solo será válido si está firmado por una AC en la que el terminal confía.

Para terminar, si, tras estas comprobaciones, el terminal determina que el certificado es válido, se establece la conexión segura entre dicho terminal y el emisor. El funcionamiento del protocolo SSL/TLS se describe esquemáticamente en la Figura 4.3.

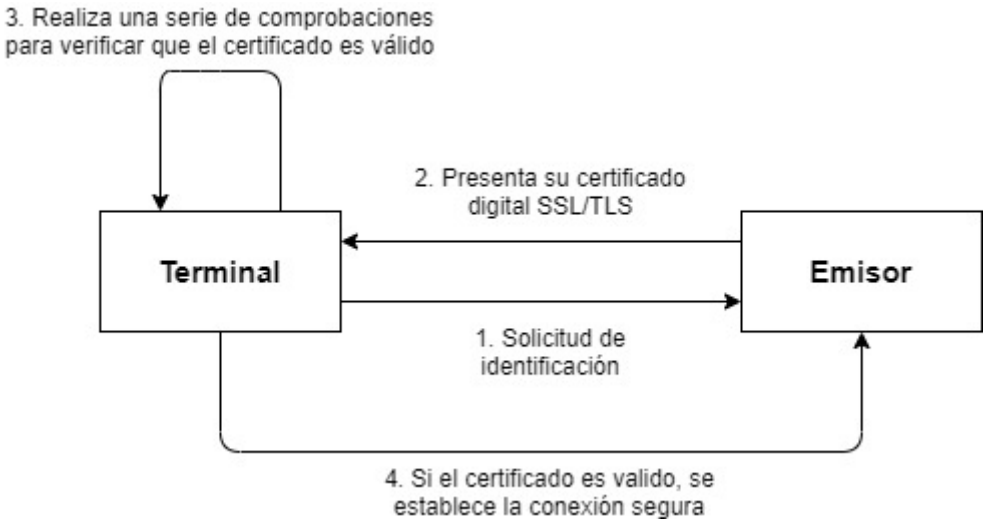


Figura 4.3: Esquema general del funcionamiento del protocolo SSL/TLS

# Capítulo 5

# 5. Conclusiones y trabajo futuro

## 5.1 Conclusiones

El objetivo final del proyecto ChipPOS es implementar un TPV para pagos de importe reducido listo para su integración en los sistemas destino. Debido a su ambicioso alcance, la implantación del proyecto completo requiere un largo plazo de desarrollo, que excede el alcance de este Trabajo de Fin de Grado. Por esta razón, en esta memoria solo se abordan las fases descritas en el capítulo 3. A partir de la última de estas fases se abordarán las tareas descritas en el apartado 5.2 hasta completar definitivamente el proyecto.

Durante el desarrollo del mismo se ha podido observar el funcionamiento de las comunicaciones entre una tarjeta bancaria, un terminal bancario y el procesador de pagos o centro autorizador, en el transcurso de una transacción. Se han descrito detalladamente todos los pasos que sigue una transacción bancaria, incluyendo toda la información que el terminal obtiene de la tarjeta y la que aporta el propio terminal para poder completar la transacción. También se ha detallado cómo se genera el mensaje que el terminal envía al procesador de pagos para que este evalúe con toda la información aportada si acepta o rechaza la transacción.

El alcance de este Trabajo de Fin de Grado cubre la parte del proyecto ChipPOS correspondiente a la siguiente operativa: el terminal lee una tarjeta bancaria mediante tecnología *contactless*, y, siguiendo el flujo EMV, obtiene toda la información de la tarjeta necesaria para la ejecución de una transacción. En este punto y tras una serie de intercambios de información entre el terminal y la tarjeta, el terminal genera un mensaje de tipo JSON en el que incluye todos los datos de la tarjeta, del terminal y de la transacción requeridos por el protocolo Price. Este mensaje será enviado al centro autorizador, que comprobará la validez de esa información y dará una respuesta al terminal. Este último paso no se ha implementado aún y se describe en el apartado 3.8.

Este proyecto se ha desarrollado a lo largo de 900 horas de trabajo y ha requerido conocimientos de programación en C, trabajo con microcontroladores, criptografía, sistemas operativos, comunicaciones y uso de JSON.

La elaboración del proyecto descrito en esta memoria del Trabajo de Fin de Grado no habría sido posible sin los valiosos conocimientos obtenidos a largo de la carrera. Por

mencionar solo algunos de estos conocimientos, lo aprendido en las asignaturas de "Fundamentos de programación" y "Tecnología de la programación" me ha permitido desarrollar el software de manera mucho más ágil, tanto por los conocimientos de programar en C como por la lógica de programación obtenida. También he utilizado conocimientos obtenidos en asignaturas de gestión de proyectos, como puede ser "Ingeniería del software" o lo aprendido en "Sistemas operativos", muy útil para la última fase, la integración del software con un sistema operativo en tiempo real. Por último, lo aprendido en la asignatura "Redes y seguridad" sin duda me ha ayudado en el aspecto criptográfico del proyecto, permitiéndome desarrollarlo de forma más ágil y precisa.

## 5.2 Trabajo futuro

A pesar de todo el trabajo realizado en este proyecto, será necesario realizar una serie de tareas futuras más allá de lo incluido en esta memoria para que el proyecto se pueda completar. Este trabajo futuro consistirá en:

- Completar la integración del proyecto con un FreeRTOS.
- Instalación de un módulo de red sobre la placa, como se describe en el punto 3.8.
- Adaptación del software implementado y descrito en esta memoria de tal forma que sea capaz de procesar una transacción con cualquier tipo de tarjeta bancaria, es decir, no solo Mastercard, sino también Visa, American Express…
- Creación de un prototipo funcional sin certificar, rediseñando la placa para que sea todavía más eficiente, en coste y tamaño, eliminando la pantalla y manteniendo exclusivamente unos leds que se enciendan de un color u otro dependiendo de si la transacción ha sido aceptada o no. Este prototipo ya tendrá el soporte de comunicaciones Ethernet integrado.
- Crear un producto industrializable con certificaciones, es decir, una versión industrializable de ChipPOS, certificada por los organismos competentes, PCI y EMVCo pero también por las marcas Visa y Mastercard.

# Appendix A – Introduction

## Motivation

Nowadays, we are used to been able to pay any kind of service or product with our credit card, which let us avoid the tedious task of bringing cash money with us every time we leave our homes. But this has not always been this way.

During the first half of the 20th century, the idea of paying with a credit card in order to buy products or services appeared in some business, but they were exclusive for those business. An example would be gas stations, who provided their clients a card that could be used to buy gas only in the gas stations which belonged to that specific company [1].

It was not until 1949 when the idea of the first credit card that could be used in different shops was born. Several coincidences led a popular American banker who have had dinner with his colleagues to ask the restaurant to sell on credit because he had not brought money with him. This incident provoked the creation of the first modern credit card, known as the Dinners Club [2]. With it, the customer could pay in different restaurants that formed part of the project without the need of using cash money.

Thanks to the technological development that took place in the second half of the 20th century and to the appearance of companies such as Mastercard or VISA, credit cards have evolved and turned into what we now know, exceeding 1100 million of annual transactions in Spain during 2017 [3].

However, due to the huge volume of money moved by these transactions in our society, is extremely important that these movements are carried out with high computer security in order to avoid the possible intrusion of attackers, that could provoke millionaire losses.

These safety conditions must be guaranteed in several levels. Firstly, apps must be safely programmed, without any possible vulnerability or breach. Secondly, in the encoding of specific information carried by the card, such as confidential tracks, for example, the PAN (Primary Account Number). Finally, and thirdly, these security conditions must be guaranteed in the encoding of the communications, in such a way that an attacker that has access to the mentioned communication is not able to extract data from it.

This end-of-degree project aims to implement a new POS terminal for payment environments whose amount is below € 20, without setting aside the importance of computer security in this area, both from the point of view of the infrastructures and the

point of view of the encryption algorithms and communication protocols used for ensure the confidentiality and integrity of the transmitted data and the authenticity of those who transmit the data.

As a starting point in order to carry out this project, I used the project that I am developing for the company Redsýs Servicios de Procesamiento S.L., in whose development and innovation department I am accomplishing my external internship

The fact of belonging to this department has led me realize the importance of the role that innovation plays in a technological company, even more if possible, in one that is in charge of payment methods, whose exponential growth in the last 70 years is related to a constant innovation of credit cards. Nevertheless, technologies' evolution has also increased the risks that the use of credit cards could involve. This is another key issue in which innovation is based, the constant change of certain protocols and services with a more developed and advanced protocols and algorithms with the aim of preventing the mentioned risks and been one step forward than the attacker.

Strategically, innovation is a crucial factor, because it allows companies to develop services and products with higher security which might result in overtaking competence companies and in an increase in economic benefits and prestige.

## Aims of the project

This statement has been carried out with the collaboration of the company Redsýs Servicios de Procesamiento S.L., who gave me this project, in which I am the only participant.

The goal of the project, called ChipPOS, is the implementation of the software of a POS (Point of Sale) developed exclusively for payments whose amount is below € 20. This would be, a terminal without printer, keyboard and other components that the traditional POS have, which would reduce the cost that the business would have to invest in the POS from € 200 that is the average price of a traditional terminal to around € 50 that would be the price of these new format.

Regarding the possible application of this project, cigarette machine, vending machine or public buses would be examples or where to implement it. Taking the last example, this new model of POS would let us take the bus, and in the case that we did not have a ticket, we could ask the driver for one and place our card close to the POS. In that way, the amount of time that we save without needing to write the amount, or waiting until the ticket is printed, among other things, is noteworthy.

At the beginning of this project, a high-performance HF reader (Figure 1.1) based in ARM and with a small dimension screen whose only functionality was to be able to detect if the card with chip that was been placed close to it was a bank card or not. In the case that it was, it showed a green "tick" in the screen, but if the placed card was not a bank card it showed a red "cross" [4].


Figure 1.1: ST25R3911B high-performance HF reader

This high-performance HF reader is the basis around which the whole project has been developed. It has been programmed in language C and we can distinguish 6 phases of its implementation.

-**Phase 1**: Acquisition of the needed knowledge in order to program POS and the communication protocols that these terminals use in order to establish communication with cards.

-**Phase 2**: Implementation of the needed functions and methods needed for the POS to be able to obtain the data it needs from the card by APDU commands (Application Protocol Data Unit) [5] saving in its memory the label, length and value of each of these data.

-**Phase 3**: Insertion of needed cryptographic libraries to encode with the pertinent encoding algorithms the tracks that the protocol requires to be encoded.

-**Phase 4**: Creation of a JSON with the data from the card and the transaction mentioned in the protocols in order to send that JSON to the authorizing payment center so that the chosen payment can be accepted or rejected. (A network module is needed in order to create a connection with the exterior).

-**Phase 5**: Integration of the developed project with an operative system that allows to modularize and execute it in an ordered way by means of different tasks.

-**Phase 6**: Installation of a network module that allows the POS to connect with the authorizing payment center and implementation of protocols of the correspondent communications needed to create a safe connection between both parts, in this case TLS 1.2.

***The main goal of this end-of-degree project is implementing this system taking into account the need of maintaining the demanding security conditions in programming when the time to encode the tracks that the POS obtains from the card and that are specially relevant and in the use of the cryptographic protocol TLS 1.2.***

## Statement organization

The statement of this project is presented in 5 chapters, including the introduction, that follow the same order that has been followed in the implementation of the original project. a small description of each chapter is presented below.

In the chapter number 2 a study of the different POS that we can now find in the market is carried out. The functioning, type or hardware and the mechanism used to read the cards is also described.

In the chapter number 3 the whole implementation of the project is described, from the first day until the project is handled. the execution of the five base commands that make the transaction is explained, analyzing the flow it follows showing as an example the execution of the software with a real card.

In the chapter number 4, an analysis of the security mechanisms that are now used in critical infrastructures (as bank ones), both in the phase of authentication and in the phase of encrypting communications, is carried out and explained why they are used.

In the chapter number 5 the conclusions obtained from the project are exposed, as well as further ideas and recommendations for future development of the project.

# Appendix B – Conclusions and future work

## Conclusions

The final aim of the ChipPOS project is implementing a POS for payments of reduced amount, ready for its integration in the destination's system. Due to its ambitious scope, the implementation of this project would require more time than the one available for this end-of-degree project. For this reason, in this statement, only the phases described in the chapter 3 are explained. From the last of these phases, the tasks described in the section 5.2 are tackled until the project is finished.

During the development of it, the functioning of the communications between a bank card, a bank terminal and a payment processor during a transaction, have been observed. All the steps followed in a bank transaction have been explained with detail, including all the information that the terminal takes from the card and the information the terminal provides in order to complete the transaction. The process of building up the message that the terminal sends to the payment processor in order to decide if it rejects or accepts the transaction is also explained.

The scope of this end-of-degree project goes through the part of the ChipPOS project that corresponds to the following way of working: the terminal reads a bank card trough contactless technology, and following the EMV flow, gets all the information from the card needed for the transaction. At this point, and after several information exchanges between the terminal and the card, the terminal generates a JSON message that includes all the data required by Price protocol, for the terminal, the card and even the transaction. This message is posteriorly sent to the authorizing center, that will check the validity of that information and create an answer to the terminal. The last step has not been implemented yet and is explained in the section 3.8.

This project has been developed for 900 work hours and requires knowledge in C programming, working with microcontrollers, cryptography, operating systems, communications and the use of JSON.

The scope of the project described in this end-of-degree statement would not have been possible without the valuable knowledge achieved during the bachelor. Just to mention some of these relevant knowledges, the information learned in "Introduction to programming" and "Computer programming technology" have allowed me to develop in an easier way the software, not only for the ability of programming in C but also because of the programming logic obtained. Also, the knowledge obtained in courses related to project management such as "Software engineering" or "Operating systems", this one very useful in the last phase in which a software with a real time operative

system was integrated, had helped me to implement the project in a better way. Finally, everything learnt in “Networks and security” has led me create the cryptographical part of the project in a more precise way.

## Future work

Despite all the work that has been carried out on this project, it will be necessary to implement several future tasks beyond what is included in this statement for the project to be completed. The future work would be:

- Finishing the integration of the project with a FreeRTOS.
- Installing a network module on the high-performance HF reader, as described in the section 3.8.
- Adaptation of the implemented and described software in order to make it able to process a transaction with any kind of bank card, which means not only Mastercard, but also Visa, American Express...
- Creation of a functional uncertified prototype, redesigning the high-performance HF reader in order to make it even more efficient in both price and size by eliminating the screen and maintaining exclusively LEDs that would show one or other color depending if the transaction is accepted or rejected. This prototype would have the Ethernet communication module integrated.
- Create a certified product, which means, an industrialized version of ChipPOS, certified by the competent organization, PCI and EMVCo, but also by brands as Mastercard and VISA.

# Bibliografía


[1] CurioSfera Historia, “Historia de la Tarjeta de Crédito - Origen, Inventor y Evolución,” 2020. https://curiosfera-historia.com/historia-tarjeta-credito/ (accessed Jun. 16, 2020).

[2] Diners Club, “Diners Club,” 2020. https://www.dinersclub.pe/portal/conoce-nuestro-club/historia# (accessed Jun. 16, 2020).

[3] Statista, “Transferencias bancarias: número total 2010-2018 | Statista,” 2019. https://es.statista.com/estadisticas/528311/numero-total-de-transferencias-bancarias-espana/ (accessed Jun. 16, 2020).

[4] STMicroelectronics, “ST25R3911B-EMVCO - EMVCo reference design for the ST25R3911B high performance HF reader - STMicroelectronics,” 2020. https://www.st.com/en/evaluation-tools/st25r3911b-emvco.html (accessed Jun. 16, 2020).

[5] CardWerk, “ISO7816 part 4 section 5 APDU level data structures,” 2020. https://cardwerk.com/smart-card-standard-iso7816-4-section-5-basic-organizations/ (accessed Jun. 16, 2020).

[6] Ruy Alonso Rebolledo, “¿Cómo funciona la banda magnética de las tarjetas bancarias? | El Economista,” Jul. 03, 2017. https://www.eleconomista.com.mx/finanzaspersonales/Como-funciona-la-banda-magnetica-de-las-tarjetas-bancarias-20170703-0071.html (accessed Jun. 16, 2020).

[7] ISO/IEC, “ISO/IEC 8825-1:2015 Information technology — ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER),” 2015.

[8] EFTLab, “Complete list of EMV & NFC tags - EFTLab - Breakthrough Payment Technologies,” *2020*. https://www.eftlab.com/knowledge-base/145-emv-nfc-tags/ (accessed Jun. 16, 2020).

[9] EMVCo, “Overview - EMVCo,” 2020. https://www.emvco.com/about/overview/ (accessed Jun. 16, 2020).

[10] Ursula Librizzi, “EMV Certification Explained,” 2018. https://blog.payjunction.com/emv-terminal-certification-explained (accessed Jun. 16, 2020).

[11] PCI Security Standards Council, “Official PCI Security Standards Council Site - Verify PCI Compliance, Download Data Security and Credit Card Security Standards,” 2020. https://www.pcisecuritystandards.org/ (accessed Jun. 16, 2020).

[12] Keil, “µVision IDE,” 2020. http://www2.keil.com/mdk5/uvision/ (accessed Jun. 16, 2020).

[13] Emvlab, “TLV Utilities,” 2019. https://emvlab.org/tlvutils/ (accessed Jun. 16, 2020).

[14] Advantis, “Solución de emisión multiaplicación Contactless Advantis,” 2020. https://www.advantis.es/es/que-es-advantis/ (accessed Jun. 16, 2020).

[15] EMVCo, “EMV Integrated Circuit Card Specifications for Payment Systems. Book 3. Application Specification,” 2011.

[16] NIST, “Keylength - NIST Report on Cryptographic Key Length and Cryptoperiod (2020),” 2020. https://www.keylength.com/en/4/ (accessed Jun. 16, 2020).

[17] Postman, “Postman | The Collaboration Platform for API Development,” 2020. https://www.postman.com/ (accessed Jun. 16, 2020).

[18] FreeRTOS, “FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions,” 2020. https://www.freertos.org/ (accessed Jun. 16, 2020).

[19] Wiznet, “WIZ610io | WIZnet Co., Ltd.,” 2020. https://www.wiznet.io/product-item/wiz610io/ (accessed Jun. 16, 2020).

[20] Wenliang Du, “Computer and Internet Security: A Hands-on Approach,” 2019.

[21] Dante Odín Ramírez López and Carmina Cecilia Espinosa Madrigal, “El Cifrado Web (SSL/TLS) | Revista .Seguridad,” 2018. https://revista.seguridad.unam.mx/numero-10/el-cifrado-web-ssltls (accessed Jun. 17, 2020).