UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada
Sistemas de información y Comunicaciones

Máster en
Tecnologías de Telecomunicación

Trabajo Fin de Máster

Aplicación móvil en Android para la comunicación con una tarjeta eSignus mediante el protocolo Bluetooth Low Energy

Autor: Francisco M. Santana Verona
Tutor: Fernando de la Puente Arrate
Fecha: Julio de 2012

IUMA
t +34 928 451 086 | iuma@iuma.ulpgc.es
f +34 928 451 083 | www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

Versión: 2.0 – 14 junio 2011

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada
Sistemas de información y Comunicaciones

Máster en
Tecnologías de Telecomunicación

Trabajo Fin de Máster

Aplicación móvil en Android para la comunicación con una tarjeta eSignus mediante el protocolo Bluetooth Low Energy

HOJA DE FIRMAS

Alumno/a:	Francisco M. Santana Verona	Fdo.:
Tutor/a:	Fernando de la Puente Arrate	Fdo.:

Fecha: Julio de 2012

IUMA
t +34 928 451 086
f +34 928 451 083
iuma@iuma.ulpgc.es
www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada
Sistemas de información y Comunicaciones

Máster en
Tecnologías de Telecomunicación

Trabajo Fin de Máster

Aplicación móvil en Android para la comunicación con una tarjeta eSignus mediante el protocolo Bluetooth Low Energy

HOJA DE EVALUACIÓN

Calificación: .....

Presidente	Francisco José Guerra	Fdo.:
Secretario	F. Javier del Pino	Fdo.:
Vocal	Gustavo Marrero	Fdo.:

Fecha: Julio de 2012

IUMA
t +34 928 451 086
f +34 928 451 083
iuma@iuma.ulpgc.es
www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

Versión: 2.0 – 14 junio 2011

Agradecimientos

En primer lugar me gustaría agradecer a mis padres por su apoyo incondicional tanto en los momentos más llevaderos como en los más difíciles de la realización de este trabajo. En segundo lugar, agradecer a mi tutor, Fernando, por la ayuda prestada y por ofrecerme la oportunidad de realizar este proyecto conjunto que ha supuesto para mí una vía enorme de conocimientos y oportunidades. También me gustaría agradecer a David por su paciencia y dedicación a la hora de explicarme los detalles de la tarjeta eSignus y del protocolo Bluetooth Low Energy. Por último, dar las gracias a la empresa INELCAN S.L. por cedernos una tarjeta eSignus de pruebas y el teléfono móvil para realizar la aplicación.

Sin todos ellos este trabajo fin de máster no hubiera sido posible. Sinceramente, gracias.

Índice general

Capítulo 1

Introducción

En la actualidad, el uso masivo de Internet ha traído consigo muchas ventajas, como la posibilidad de realizar compras electrónicamente o realizar trámites burocráticos; pero por otra parte nos ha dejado la necesidad de aumentar la seguridad en la autenticación online de cara a asegurar el intercambio de información que estos procesos suponen. Este es el problema principal que numerosas aplicaciones adolecen, desde un acceso a un servicio web hasta una transferencia bancaria. Gracias a la criptografía, y en especial a la criptografía de clave pública, se han conseguido grandes avances en este campo. Sin embargo, aunque los modelos teóricos funcionan bien la mayoría de las veces, en las implementaciones reales encontramos ligeras brechas que podrían ser usadas para romper la seguridad del sistema completo.

La autentificación de alguien en un servicio podría categorizarse según una palabra clave propia (PIN, contraseña, etc..), un dispositivo propio (tarjeta de identificación, teléfono móvil, etc..) o una característica única del individuo (huella digital, secuencia de ADN, identificación biométrica, etc.). Para asegurarnos la autenticidad deberíamos usar al menos dos de estos mecanismos en nuestras aplicaciones, cerciorándonos a su vez que estos no pueden ser obtenidos por los atacantes.

La criptografía de clave pública ofrece una de las formas más fiables de autentificación en la actualidad cuando se combina con la tecnología de tarjetas inteligentes. Pero incluso este método tiene sus propias debilidades. Una de ellas es el hecho de que el propio usuario no sabe realmente qué o quién controla la tarjeta: si es el dispositivo en sí mismo o es la interfaz en forma de ordenador, ATM, etc..

Aunando esfuerzos y tecnologías diversas, el IUMA en convenio con la FULP (Fundación Universitaria de Las Palmas) ha desarrollado para la empresa INELCAN S.L. un firmador electrónico llamado eSignus [5] teniendo en mente el concepto de "dispositivo de confianza" [9]. Para ello, le han dotado de una pantalla de tinta electrónica y un teclado que el usuario puede usar para interactuar con la tarjeta. Al introducir la pantalla y el teclado proporcionamos un nivel adicional de seguridad, ya que será posible mostrarle al usuario en la propia tarjeta los datos que serán firmados, y ser este el encargado de

aceptar o no la firma de esos datos.

Por otro lado, el uso de los teléfonos móviles es cada vez más habitual; incluso se esta empezando a utilizar como método de pago. Pero sin embargo, el usuario por lo general no es consciente de la inseguridad de su móvil. Las amenazas no solo incluyen a las propias de los navegadores web, sino también a las aplicaciones de terceros; en el caso de los teléfonos móviles tampoco está extendido el uso de software antivirus y cortafuegos, haciéndolos todavía más vulnerables a ataques. Es en este punto donde surge la realización de este trabajo fin de máster, donde podemos combinar la tecnología de la tarjeta eSignus y las aplicaciones móviles.

Por cuestiones de control del consumo, la tarjeta eSignus usa el estándar Bluetooth Low Energy para comunicarse. Si bien se espera que esta tecnología se implante rápidamente, por el momento los móviles que soportan esta tecnología [4] son el iPhone 4S (iOS), el Motorola Droid RAZR (Android) y el Samsung Galaxy S III (Android). Puesto que INELCAN S.L. ya está trabajando en una aplicación para el iPhone 4S bajo iOS, hemos decidido centrar este trabajo fin de máster en la aplicación para el sistema operativo Android.

Por tanto, nos planteamos la realización de una aplicación para dispositivos móviles Android que sea capaz de comunicarse con la tarjeta eSignus mediante Bluetooth Low Energy. El usuario de la aplicación será capaz de realizar tanto transferencias bancarias como cobros, todo ello firmando los datos de las operaciones en la tarjeta eSignus y usando el móvil como medio de conexión con el banco.

Para llevar a cabo el desarrollo de este trabajo fin de máster, utilizaremos recursos tanto software como hardware:

- Tarjeta eSignus: la empresa INELCAN S.L. nos proporcionó una tarjeta de pruebas especialmente diseñada para nuestras comunicaciones. Los cambios realizados respecto a la tarjeta comercial fue la eliminación de la petición de PIN al conectarnos (para realizar tests con mayor rapidez).

- Motorola Droid RAZR: smartphone de última generación cedido por la empresa INELCAN S.L. para el desarrollo de este trabajo fin de máster. Este dispositivo móvil funciona bajo el sistema operativo Android e incorpora la tecnología Bluetooth Low Energy de conexión inalámbrica.

- Eclipse + Android SDK: el framework de desarrollo de Android desarrollado por Google funciona como un plugin para el entorno de desarrollo Eclipse, así que usamos ambas herramientas como pilares de programación para nuestro proyecto.

- StarUML: software específico para la realización de diagramas en lenguaje de modelado UML.

- Javadoc y LyX: paquetes para la generación de la documentación, tanto a nivel de código fuente como del presente documento en formato LáTeX.
- API Bluetooth Low Energy de Motorola: API para la gestión de las comunicaciones Bluetooth Low Energy en Android.

Capítulo 2

Estado actual

2.1. Seguridad en las transacciones online

A la hora de realizar transacciones online, las empresas, entidades financieras y usuarios deben asegurarse que estos movimientos se están realizando en un entorno seguro de comunicaciones. Procesos como consultar los movimientos bancarios de una cuenta o ingresar dinero deberían ser lo más seguro posibles para evitar fraudes y suplantaciones de identidad. Es por ello que en los últimos años han surgido toda una serie de técnicas y tecnologías para asegurar que remotamente un usuario accede e interactúa de manera segura con su dinero.

2.1.1. Autenticación del usuario

En las transacciones online es importante comprobar que un usuario es quién dice ser. En la mayoría de los casos se utilizan claves de acceso que solo deben ser conocidas por propio usuario. Existen varios mecanismos relacionados con las claves de acceso:

- PIN (Personal Identification Number): normalmente suele consistir en un código alfanumérico que debe introducirse conjuntamente con un identificador de usuario para realizar el acceso satisfactoriamente. Este tipo de claves las pueden proporcionar directamente el comercio o entidad, o pueden ser creadas por el propio usuario. En el primer caso y por lo general, la clave combinará números, letras minúsculas y mayúsculas, y quizás algún carácter fuera del alfabeto como el guión o el punto; en el segundo caso, la clave la crea el propio usuario en base a algún criterio que le sea fácil de recordar (fecha de nacimiento, número de teléfono combinado con su nombre, etc..), disminuyendo considerablemente la seguridad de su clave y aumentando el riesgo de ataques por fuerza bruta y diccionarios.

- TAN (Transaction Authentication Number): este método es similar al PIN, solo que los códigos son generados previamente y distribuidos físicamente (mediante papel o

tarjetas), o generados justo antes de la transacción y distribuidos mediante medios digitales como SMS o correo electrónico. Existen variantes de esta técnica:

- mTAN: la entidad envía el código de autenticación mediante un SMS al móvil del cliente.
- iTAN: se le entrega al usuario una tabla con una serie de claves de acceso, y a la hora de autenticarlo se le pide la clave correspondiente a una cierta columna y fila de la misma.
- iTANplus: es una variante de iTAN, pero usando CAPTCHAs[1]. Este CAPTCHA puede identificar el código a introducir o como método para comprobar la identidad del usuario.

- OTP (One Time Password): se tratan de contraseñas de un solo uso enviadas al móvil del cliente, por correo electrónico o generadas por dispositivos.

Además, la gran mayoría de bancos implementan en sus webs otra técnica para aumentar la seguridad a la hora de introducir la clave de acceso: el teclado virtual. Para evitar que keyloggers (un keylogger es un programa que captura las pulsaciones de teclado de manera transparente al usuario) intercepten y almacenen la contraseña tecleada por el usuario, los bancos ponen a disposición de sus usuarios un teclado virtual en pantalla. Así, en vez de teclear la clave el usuario hace click en cada letra de la misma, evitando que este tipo de programas la capturen.

Por otro lado, estan los sistemas basados en firmas electrónicas; almacenan información personal que identifica al usuario unívocamente (en el caso de España, el DNI electrónico es un dispositivo de estas características, ya que guarda certificados digitales expedidos para el propietario por la FNMT). Este es, sin duda, el medio mas seguro de operar de los disponibles comercialmente. Sin embargo, prácticamente todos los sistemas basados en claves son poco robustos.

2.1.2. Seguridad durante la transacción

De nada nos sirve que el usuario se autentique de manera segura si las comunicaciones pueden ser interceptadas, leídas y modificadas. El famoso ataque del hombre en medio (man-in-the-middle, MitM) ha creado la necesidad de mejorar la seguridad también durante el intercambio de información online. Es por ello que aparecen los certificados digitales.

Un certificado digital es un archivo que identifica a un usuario, empresa u organismo. Este archivo contiene una serie de datos personales junto con una clave pública, y todo

[1] **C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part: prueba de Turing pública y automática para diferenciar máquinas y humanos.

ello está firmado por una autoridad certificadora que le aporta validez. En España la autoridad certificadora más conocida es la FNMT (Fábrica Nacional de Moneda y Timbre), encargada de crear, por ejemplo, los certificados incluidos en el DNI electrónico.

El uso de certificados digitales en Internet viene ligado con el protocolo HTTPS y SSL. SSL es un conjunto de protocolos criptográficos para las comunicaciones seguras en una red (normalmente Internet). Se utiliza conjuntamente con HTTPS para aumentar la seguridad de las conexiones y asegurar que los ataques del hombre en medio no descubran información sensible de los usuarios.

Cuando se realiza una conexión HTTPS, tanto el cliente como el servidor establecen qué tipo de cifrado y autenticación se va a usar en base a los algoritmos implementados y soportados por ambos. Una vez están establecidos estos parámetros, cliente y servidor intercambian certificados y negocian una clave secreta con la que se cifrarán los datos de la comunicación. Esta fase se denomina *handshake* porque es un acuerdo entre las partes antes de empezar a transmitir datos.

Con estos procesos se asegura que los datos enviados por una parte a la otra parte son efectivamente de quien dicen ser. Además, aunque sean escuchados por terceros no pueden ser descifrados al no conocer la clave secreta y si los datos son alterados en el transcurso de la comunicación, el destino será capaz de darse cuenta debido a la inclusión de los certificados digitales.

Ahora bien, prácticamente todos los bancos tienen estos certificados, pero no sus clientes. Esto se traduce en que el usuario puede saber que se conecta a la web del banco, pero este no puede estar seguro de la identidad del cliente.

2.2. Uso del móvil como método de pago (NFC)

Un medio de pago que se está empezando a implantar es el uso del teléfono móvil con NFC. La tecnología Near Field Communications (NFC) es un estándar para comunicaciones inalámbricas que permite que dos dispositivos equipados con chips NFC puedan transmitirse información entre sí de forma segura con solo acercarlos entre ellos. En la actualidad existen varias empresas que están apostando por el uso de NFC como método de pago en el móvil.

Google Wallet es un ejemplo de ello. Se trata de una aplicación que convierte un teléfono móvil Android en una cartera virtual, almacenando de forma segura las tarjetas de crédito y usándolas para realizar pagos mediante la tecnología NFC. Actualmente Google Wallet solo está disponible en Estados Unidos para pagos usando el móvil, aunque tienen una versión web disponible mediante Internet (sin usar NFC).

VISA también posee una tecnología similar denominada VISA PayWave, consistente en una serie de etiquetas RFID (no NFC) empotradas en una tarjeta de crédito tradicional; el comercio debe tener un lector RFID que recibirá los datos del cliente grabados en la

tarjeta cuando se aproximan entre sí. En el caso de VISA PayWave, no es necesario una firma para compras de pequeñas cantidades.

De cara a la eficiencia, el uso del móvil con NFC como sistema de pago reduciría las colas. Por ejemplo, uno de los lugares donde más impacto tendrían sería en el transporte público, ya que no se requeriría tickets ni dinero en metálico y el proceso se realizaría muy rápidamente. Por otro lado y de cara a la seguridad hay una serie de cuestiones a tener en cuenta:

- Robo de datos: el chip NFC no tiene acceso a los datos del teléfono; por tanto, un virus o troyano no podría acceder a la información de la transacción porque los elementos del sistema donde afecta el virus y donde se encuentran los datos de la transacción son elementos distintos.
- Robo del móvil: el problema es idéntico al robo de una tarjeta de crédito; por tanto, si el móvil es robado debemos avisar a la entidad para que bloqueen este tipo de pagos. Además, los pagos superiores a una cierta cantidad requieren de un PIN para ser efectivos.

A pesar de todo, al tratarse de un dispositivo programable siempre existen posibilidades de buscar formas de realizar ataques.

2.3. Uso del móvil como TPV

A la izquierda, iZettle. A la derecha, Square.

Actualmente es común el pago mediante tarjetas de crédito en los comercios, así que el uso del TPV clásico es algo más que habitual. Pero el uso de TPVs en los comercios lleva consigo una desventaja, y es la necesidad de tener una línea fija de teléfono. Por tanto, no sería posible usar un TPV en un comercio ambulante ya que no dispondríamos de línea de teléfono fija. Sin embargo, en los últimos años ha surgido como alternativa al

TPV el uso del móvil, donde el único requisito es tener conexión a Internet para validar los pagos. Los productos más conocidos para ello son Square[2], iZettle[3] y Paypal Here [4].

Estos tres productos siguen el mismo principio de funcionamiento: tenemos un dispositivo físico conectado al móvil (a la entrada de audio en el caso de Square y Paypal Here, y a la alimentación en el caso de iZettle) que actúa como lector de tarjetas de crédito y mediante una aplicación instalada en el móvil son capaces de gestionar los pagos que se realicen.

El problema que volvemos a encontrar es que nada impide a cualquiera abrir estos lectores de tarjeta y reprogramarlos para que alteren los datos, o realicen alguna tarea adicional (como por ejemplo, almacenar información). Además, estos dispositivos seguirían siendo totalmente funcionales ya que no existirían alteraciones físicas en los mismos; incluso seguiríamos sin saber qué es exactamente lo que se está firmando al no disponer de ningún mecanismo para averiguarlo (para el usuario es transparente, pero en este tipo de situaciones no debería serlo).

En el caso de iZettle y de Paypal Here no hemos encontrado información relacionada con usos fraudulentos (lo cuál no quiere decir que no existan), pero respecto a Square si que hemos hallado modificaciones para usar el dispositivo lector como clonador de tarjetas, así como una forma de hacerle creer a la aplicación que hay una tarjeta conectada.

El dispositivo externo de Square va conectado a la entrada de audio del móvil y lo que hace es convertir los datos de la banda magnética de la tarjeta en una serie de sonidos. Para que la aplicación piense que hay una tarjeta conectada, Adam Laurie mostró en la conferencia Black Hat del año 2011[5] convirtió una tarjeta de crédito robada en sonidos interpretables por la aplicación de Square y se los pasó como datos de entrada. Como resultado, la aplicación no fue capaz de distinguir entre los sonidos recién creados por el dispositivo lector y los proporcionados por Laurie. Por tanto, podríamos perfectamente ir guardando los sonidos de las tarjetas de crédito que fuéramos leyendo a lo largo de un día normal de trabajo, y usar estos sonidos para realizar pagos ficticios con la aplicación móvil de Square.

Pero el uso más fraudulento encontrado del lector de Square es como clonador de tarjetas. Al tener un lector un lector de tarjetas de crédito propio, nada impediría a alguien crear una aplicación falsa de Square. Esta aplicación iría recogiendo datos de la banda magnética de la tarjeta ayudándose del lector. Con la información recién adquirida podrían perfectamente realizarse clonaciones de las tarjetas, tal y como Douglas G. Bergeron afirma en una carta abierta en la web de su empresa[6].

Tenemos por tanto varios problemas con el uso de estos productos:

[2]https://squareup.com/
[3]http://www.izettle.com/
[4]https://www.paypal.com/webapps/mpp/credit-card-reader
[5]https://www.blackhat.com/html/bh-us-11/bh-us-11-archives.html#Laurie
[6]http://www.sq-skim.com/

- No sabemos qué estamos firmando: la aplicación puede estar modificando los datos que le envía a la tarjeta o el lector puede haber sido modificado sin nosotros tener constancia de ello.
- En el caso concreto de Square, una aplicación que aparente ser la original puede estar guardando muchísima información almacenada en la propia tarjeta de crédito.

Son sin duda productos muy útiles y una forma muy buena de acercar el comercio electrónico al móvil, pero por otro lado nos plantea el mismo inconveniente que habíamos hablado con anterioridad, y es la confianza en los datos que se están firmando.

2.4. Seguridad en Android e iOS

Cuando hablamos en términos de seguridad de las aplicaciones en Android e iOS, y obviando cuestiones relacionadas con la encriptación de los datos o la privacidad, encontramos dos formas de ataque principales: **la instalación de aplicaciones** y **las vulnerabilidades software**.

Ambos sistemas operativos poseen tiendas virtuales para descargar aplicaciones (ya sean de pago o gratuitas) e instalarlas directamente en el teléfono móvil. La tienda de Apple se llama Apple Store y destaca por un control absoluto de las aplicaciones que los desarrolladores incorporan a la misma. Para que una aplicación sea publicada en la tienda, Apple debe aprobarla previamente y firmarlas con una clave privada de Apple que certifica que cumplen con las restricciones de usabilidad y seguridad impuestas por la compañía.

Algo muy distinto ocurre con las aplicaciones en Android, ya que la publicación en la tienda oficial de Google, denominada Google Play (anteriormente conocida como Android Market), no requiere de ninguna revisión por parte de Google; de hecho, las aplicaciones deben estar firmadas, pero pueden ser auto-firmadas que nada impediría que aparecieran en la tienda. Además de la tienda oficial, existen decenas de tiendas no oficiales que también recogen aplicaciones para móviles con Android. Esto crea un agujero de seguridad ya que no nos es posible asegurar que las aplicaciones son legítimas. El modelo utilizado por Google para la gestión de su tienda es el *crowd sourcing*, donde una aplicación se considera "buena" si ha recibido buenas puntuaciones y comentarios. Si alguna aplicación recibe numerosas malas puntuaciones o comentarios negativos de sus usuarios, Google la retira inmediatamente de la tienda y la desinstala remotamente de los teléfonos.

Cuando una aplicación es instalada en iOS, el sistema operativo crea automáticamente un sandbox donde estará la aplicación y los datos de la misma. Un sandbox es un conjunto de controles y limitaciones que restringen el acceso de la aplicación a elementos del propio sistema, como los recursos de red, las preferencias del teléfono o los archivos del sistema operativo. Por tanto, una aplicación en iOS no es capaz de leer datos de otra aplicación

que esté funcionando simultáneamente, creando una barrera adicional de seguridad para evitar el intercambio de datos entre aplicaciones.

En el caso de Android, estos sandboxes son definidos por la propia aplicación en un fichero llamado AndroidManifest, donde se declaran los permisos, restricciones y servicios que tendrá. Cuando el usuario instala la aplicación en Android, el teléfono le pregunta al usuario si está de acuerdo en conceder los permisos solicitados a la misma. Los usuarios pueden rechazarlos (la aplicación no se instalaría) o aceptarlos (la aplicación se instala). La ventaja es que cada aplicación puede tener un sandbox personalizado, siendo innecesario proporcionarle permisos que no requiere. La desventaja es que este modelo fuerza al usuario a tomar decisiones sobre la seguridad; además, si el usuario ha descargado una aplicación es para instalarla y usarla, siendo obligados a aceptar los términos impuestos por ella.

En este sentido vemos como la política de Apple para la publicación de aplicaciones en su tienda es bastante mejor en términos de seguridad, ya que el filtrado es bastante más fino. También destaca el aislamiento entre aplicaciones en el sistema iOS, donde cada aplicación es independiente del resto. Sin embargo, Google ofrece más flexibilidad al programador ya que no revisa las aplicaciones directamente, sino que serán sus usuarios quienes decidan si la aplicación merece estar en la tienda o no. Además, en Android es posible comunicar aplicaciones entre sí ya que el sandbox es configurable pudiendo ser esto una ventaja si quisiéramos hacer trabajar más de una aplicación conjuntamente.

Si hablamos ahora en términos de **vulnerabilidades del propio software**, en los sistemas iOS se utilizan dos técnicas para la ejecución segura de código: protección de la ejecución de datos (Data Execution Protection, DEP) y la aleatorización del espacio de direcciones (Address Space Layout Randomization, ASLR).

La técnica DEP es una característica de seguridad relacionada con los sistemas operativos que marca zonas de memoria como no ejecutables, impidiendo la ejecución de código que pudiera usar técnicas de buffer overflow o similares. Por otro lado, la técnica ASLR vuelve aleatorias las localizaciones de memoria de los componentes del sistema, haciendo extremadamente difícil para un atacante ejecutar código propio, incluso habiendo encontrado una vulnerabilidad. Pese a que el uso de DEP y ASLR pudieran hacer bastante fuerte a iOS, en los últimos años ha aparecido un término cada vez más recurrente relacionado con los iPhone e iPads, el *jailbreak*. El *jailbreak* no es más que un proceso para deshabilitar la firma de código e instalar algo similar a un parche para quitar la protección de ejecución de datos DEP. De esta forma es posible instalar y ejecutar aplicaciones que no hayan sido firmadas por Apple. Además, también elimina el concepto original de sandbox, y numerosas aplicaciones funcionan a nivel de administrador, volviendo al sistema enormemente vulnerable a ataques.

El sistema Android no posee estas técnicas, haciéndolo completamente vulnerable a la ejecución de código propio. Sabemos que las aplicaciones Android se desarrollan en Java,

que es relativamente inmune a corrupciones en la memoria, pero nada impediría que los programadores se aprovecharan igualmente de esta brecha para ejecutar código ilegítimo.

Capítulo 3

Bluetooth Low Energy

Bluetooth es una tecnología ampliamente extendida en la actualidad que nos permite interconectar mediante un enlace inalámbrico dos dispositivos (comúnmente, teléfonos móviles). Recientemente el consorcio que se encarga de mantener el estándar ha elaborado una mejora sustancial en el protocolo, al que han denominado Bluetooth Low Energy versión 4.0.

Con la llegada de este nuevo estándar, Bluetooth viene apostando fuerte por el bajo consumo ya que es un problema que los estándares antiguos adolecían. Antes de Bluetooth LE, los dispositivos Bluetooth tenían el inconveniente de consumir demasiado debido a que el protocolo les obligaba a estar continuamente escuchando por conexiones entrantes. Con Bluetooth LE esta parte cambia drásticamente, teniendo un comportamiento similar al que tienen los protocolos homólogos Zigbee o 6LOWPAN.

En las secciones posteriores explicaremos aspectos claves para entender cómo funciona el protocolo Bluetooth Low Energy.

3.1. Procedimientos y modos de operación

El protocolo Bluetooth se basa en el modo de operación maestro/esclavo. La conexión con otros dispositivos se realiza en modo *piconet*, que no es más que una red formada por un dispositivo y todos los que estén en su rango. El protocolo Bluetooth LE incluye diversos procedimientos y modos de operación adicionales que se añaden a los del protocolo original, como son:

1. *Device filtering*: como no es necesario responder a las peticiones de todos los dispositivos de la red, este procedimiento controla el número de peticiones de conexión entrantes haciendo uso de una lista denominada "White List". Esta lista está situada en un dispositivo de filtro y en ella se recogen todos aquellos dispositivos que tienen "permiso" para realizar conexiones. De esta forma se consigue reducir el consumo ya que el Host del dispositivo no atiende las peticiones de todos los dispositivos que

intentan conectarse, sino solamente los que hayan pasado el filtro.

2. *Advertising*: procedimiento para establecer conexiones unidireccionales entre dispositivos; es usado cuando el dispositivo esta en modo *advertising* (ver apartado **Estados de la capa de enlace**).

3. *Scanning*: procedimiento para escuchar transmisiones broadcast unidireccionales; es usado cuando el dispositivo esta en modo *scanning* (ver apartado **Estados de la capa de enlace**).

4. *Discovering*: usando este procedimiento conjuntamente con el de scanning, la red Bluetooth LE puede descubrir nuevos dispositivos conectados.

3.2. Seguridad

Uno de los aspectos fundamentales en la seguridad de Bluetooth es el emparejamiento entre dispositivos. Esta técnica se denomina Emparejamiento Seguro Simple (en inglés, Secure Simple Pairing) y su objetivo principal es proporcionar un método para que el emparejamiento sea sencillo y a su vez seguro. Por tanto, esta técnica debe ser inmune a ataques del hombre en medio y a ataques de tipo *eavesdropping* pasivo, que no son más que escuchas en la comunicación pero sin ser parte activa de ella.

Bluetooth Low Energy usa cuatro modelos de asociación para el emparejamiento dependiendo de las características de los dispositivos involucrados:

- *Just Works*: se usa en situaciones donde uno de los dispositivos no tiene mecanismos de entrada y salida de datos para mostrar datos (por ejemplo, unos auriculares vía Bluetooth). Se usa el protocolo de comparación de números (Numeric Comparison), pero al usuario solo se le pregunta si acepta la conexión. No proporciona protección alguna contra los ataques del hombre en medio, pero si contra el *eavesdropping* pasivo.

- *Out of Band*: se usa cuando hay implementados mecanismos de fuera de banda para descubrir servicios, intercambiar y transferir claves en el proceso de emparejamiento. Debido al mecanismo de fuera de banda este modelo proporciona protección frente a los ataques del hombre en medio.

- *Passkey Entry*: se usa cuando uno de los dispositivos tiene implememententados mecanismos para la entrada de datos pero no para salida de los mismos, mientras que el dispositivo con el que se quiere conectar solo puede mostrar datos.

- *Numeric comparison*: se usa cuando ambos dispositivos tienen capacidad para entrar y mostrar datos (por ejemplo, teléfonos móviles o PCs).

El emparejamiento entre dispositivos Bluetooth está compuesto por las siguientes fases:

1. Intercambio de información: lo primero que intercambian ambos dispositivos antes de un emparejamiento son sus características de entrada y salida para poder determinar el modelo de asociación adecuado (si pueden mostrar datos, si tienen la posibilidad de introducirlos, etc.).

2. Intercambio de la claves públicas: cada dispositivo genera un par de claves (pública y privada) y se envían entre ellos únicamente las claves públicas recién generadas; también se crean claves usando Diffie-Hellman que evitará los ataques del hombre en medio.

3. Fase 1 de la autenticación: el protocolo en este momento dependerá del modelo de asociación; uno de los objetivos de esta fase es asegurar que no hay ataques del hombre en medio entre ambos dispositivos y para ello se realizan comprobaciones de checksums usando tanto el canal fuera de banda como la ayuda del usuario.

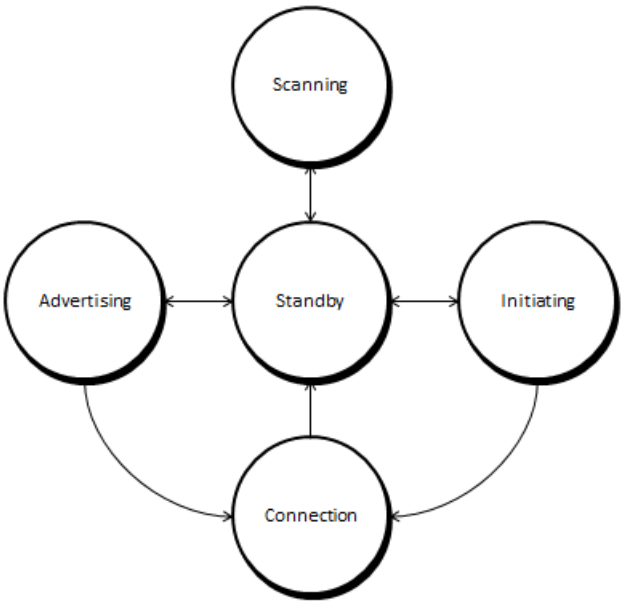
4. Fase 2 de la autenticación: los dispositivos completan el intercambio de las claves y se verifica su integridad.

5. Cálculo de la clave de enlace: se crean claves para establecer el enlace usando las direcciones Bluetooth, las claves anteriormente intercambiadas y la clave Diffie-Hellman construida en el paso 2.

6. Autenticación y encriptación LMP (Link Manager Protocol): se generan las claves de encriptación.

Una vez ha acabado el paso 6, cada dispositivo poseerá las claves que usarán para encriptar la información enviada entre ellos.

3.3. Capa de enlace



La capa de enlace en Bluetooth LE se define en términos de cinco estados:

1. Standby: no transmite ni recibe paquetes y se puede entrar en el desde cualquiera de los otros estados.

2. Advertising: transmite paquetes de *advertising* y escucha/responde a otros eventos generados por estos mismos paquetes; se entra desde el estado de *standby*.

3. Scanning: escucha los paquetes desde dispositivos que estén en estado *advertising*; se entra desde el estado de *standby*.

4. Initiating: escucha los paquetes de uno o varios dispositivos concretos y responde a estos paquetes para iniciar la conexión; se entra desde el estado de *standby*.

5. Connection: se puede entrar desde los estados *initiating* y *advertising*; en este estado se definen dos roles:

   a) Maestro: si se entra desde el estado de *initiating*. Se conectara a un dispositivo esclavo y definirá los tiempos de transmisión.

   b) Esclavo: si se entra desde el estado de *advertising*.

3.4. Attribute Protocol (ATT)

La ATT permite a un dispositivo que actúa como servidor la posibilidad de proporcionar una serie de características y valores asociados a los clientes. Estos atributos pueden ser descubiertos, leídos y escritos por un cliente, e indicados y notificados por el servidor.

Un atributo (o característica) es un valor discreto que tiene las siguientes propiedades asociadas:

- Tipo: definido por un identificador único universal (UUID); pueden ser creados por cualquiera y distribuidos públicamente.
- Manejador: usado para acceder al mismo en el servidor, así como a su valor; los manejadores son descubiertos por el cliente usando PDUs (Protocol Data Unit).
- Conjunto de permisos: lectura, escritura o ambos.

El ATT puede ser usado para acceder a información que requiera autorización y un enlace autenticado y encriptado para ello. Si se intenta acceder a un recurso y no se esta autorizado para ello, el servidor envía una notificación con un código de error. La especificación de Bluetooth no define los requisitos de acceso para los atributos, así que queda en manos de cada dispositivo cómo llevar a cabo este tratamiento.

3.5. Generic Attribute Profile (GATT)

La capa GATT define un framework usando la ATT donde se especifican los procedimientos y formatos de los servicios y sus características. Los procedimientos incluyen descubrimiento, lectura, escritura y notificación, así como configuraciones diversas.

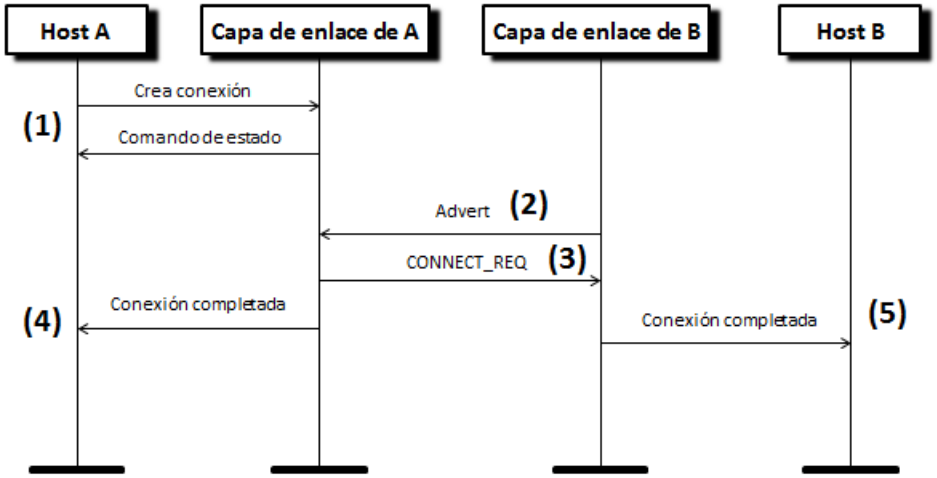
Se siguen manteniendo los roles clásicos de cliente y servidor. El cliente será el que inicie los comandos y haga peticiones al servidor, recibiendo así mismo las respuestas, indicaciones y notificaciones de este, mientras que el servidor aceptará los comandos solicitados y enviará a los clientes la respuesta con los valores de estos.

El perfil GATT usa el ATT para transportar datos en forma de comandos, peticiones, respuestas, indicaciones, notificaciones y confirmaciones entre dispositivos. Estos datos están contenidos en los PDUs del ATT.

3.6. Ejemplos

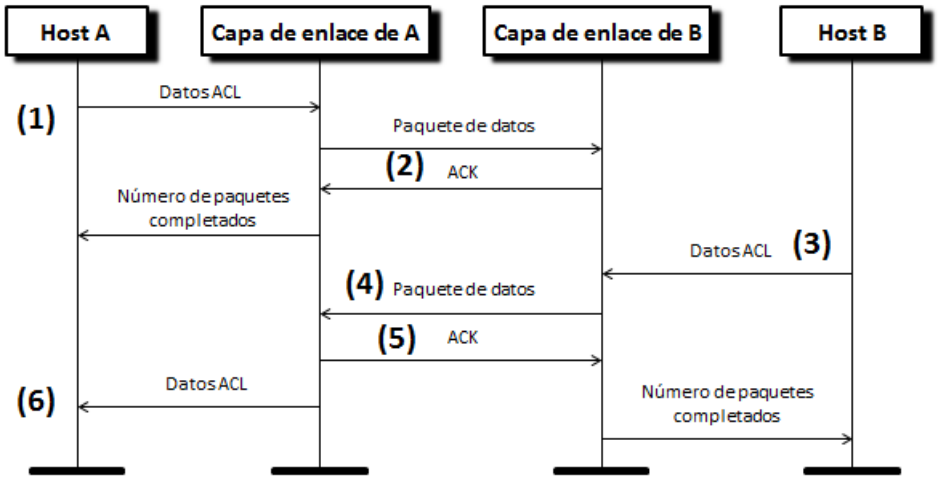
En esta sección veremos diagramas que recogen ejemplos de conexiones e intercambio de datos entre dispositivos bajo el protocolo Bluetooth LE.

3.6.1. Inicio de conexión



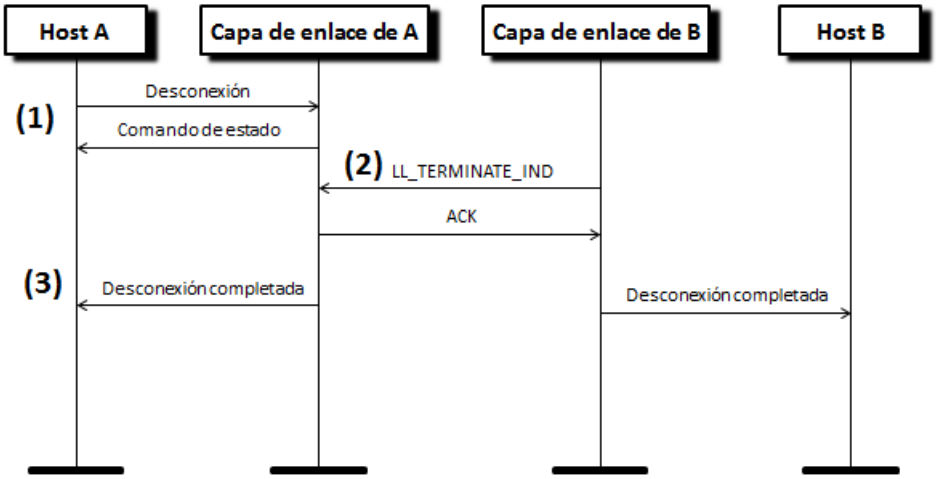
El dispositivo A desea iniciar una conexión con el dispositivo B. El host A crea la conexión informando a su capa de enlace (1). La capa de enlace del dispositivo B hace un advert para comprobar si hay algo para él (2). En este momento, la capa de enlace de A le envía la petición de conexión (3). Automáticamente la conexión se crea (4 y 5) y se completa la fase de inicio de conexión.

3.6.2. Envío de datos



El dispositivo A desea leer un dato del dispositivo B, así que le envía una petición de lectura (1). La capa de enlace del dispositivo B contesta que ha recibido la petición (2) y el dispositivo A vuelve a un estado de *standby*. Cuando el dispositivo B tenga listo el dato (3), se lo envía a la capa de enlace del dispositivo A (4); este le confirma la recepción del mismo (5) y se completa la lectura del dato una vez es enviado al host A desde su propia capa de enlace. (6)

3.6.3. Desconexión



Una vez que una conexión no es necesaria, cualquiera de los dispositivos implicados puede pedir la desconexión independientemente de si son esclavos o maestros. Es tan sencillo como hacer una petición de desconexión a la capa de enlace del otro dispositivo (1). El otro dispositivo envía el comando para la desconexión, y el primero le contesta con un ACK indicando que lo ha recibido correctamente (2). A partir de este momento la conexión se corta y acaba el proceso (3).

3.7. Bluetooth Low Energy en Android

Como el protocolo Bluetooth LE es bastante nuevo aún y no existe una gran variedad de smartphones bajo Android que lo implementen, por el momento el SDK de Android no lo soporta de manera nativa. Las alternativas no oficiales para el uso del protocolo son actualmente dos: Open Bluetooth Low Energy SDK for Android y la API de Motorola para Bluetooh LE.

Respecto a Open Bluetooth Low Energy SDK for Android[13], se trata de un intento libre de proporcionar unas librerías para el manejo y gestión del protocolo en Android. Su implementación y uso es bastante similar a la que incorpora originalmente Android en su SDK para el protocolo Bluetooth. En ella, se distinguen claramente los roles de cliente-servidor, y las clases que tenemos que crear para usar la API suelen ser subclases de las clases originales de la API. Por ejemplo, si necesitamos crear un cliente que realice conexiones con un servidor, la clase encargada heredaría todos los métodos originales de la API, y seríamos nosotros los encargados de personalizarla para nuestra aplicación. Las últimas actualizaciones de la API a la hora de realizar este trabajo datan de Mayo de 2012, pero la documentación y soporte oficial es bastante escaso al tratarse de una alternativa libre.

La API de Motorola [10], sin embargo, es algo más personalizada y tiene su propio

sistema para realizar las conexiones y escuchas. A la hora de trabajar con ella no requiere una jerarquía de clases tan definida como la API anterior, sino que podemos usarla conjuntamente con nuestro código sin necesidad de modificar, por ejemplo, la herencia de la clase actual. Esto nos da una ventaja respecto a la API anterior, puesto que las clases no se acoplan tanto y no hay dependencias fuertes. En nuestra opinión, tiene un soporte más firme ya que está siendo mejorada por los ingenieros de Motorola y su foro oficial [6] suele estar bastante concurrido con preguntas y reportes sobre su uso.

Capítulo 4

Tarjeta eSignus

Como ya hemos comentado, la tarjeta eSignus [8] permite al usuario firmar transacciones bancarias de forma segura. El funcionamiento básico de la misma es el siguiente:

1. La tarjeta pide en primer lugar el PIN de usuario. A continuación esta queda en un estado de *standby* esperando por la recepción de datos para firmar. De no recibirse nada en un periodo de tiempo, la tarjeta se apaga automáticamente.

2. Se le envían a la tarjeta los datos que se deseen firmar con el formato especificado en el apartado *Formato de tramas*.

3. La tarjeta muestra estos datos en pantalla y espera la confirmación del usuario para su firma. Esta confirmación no es más que pulsar en el botón de aceptar del teclado de la propia tarjeta.

4. Si el usuario aceptó, la tarjeta comienza a firmar los datos. Este proceso puede durar unos 2 o 3 segundos.

5. Una vez acabada la operación, la tarjeta envía los datos firmados al dispositivo que le envió los datos en el paso 2.

En los siguientes subapartados explicaremos el formato de las tramas a la hora de enviarle datos a la tarjeta, el envío y la recepción de datos.

4.1. Formato de tramas

La pantalla de la tarjeta tiene 11 líneas de 128 caracteres de ancho. Según el juego de caracteres, se definen tres formatos de letra con sus identificadores hexadecimales siguientes:

- Estándar (0x01): con 6 píxeles de ancho, caben 21 caracteres por línea.
- Negrita (0x02): con 8 píxeles de ancho, caben 16 caracteres por línea (usado para mostrar el titular de cuenta o el importe).
- Numérica (0x03): con 5 píxeles de ancho, caben 25 caracteres por línea (usado para mostrar números de cuenta).

Además, se definen tres formatos adicionales denominados "invertidos", donde el color de la letra es blanca y el fondo oscuro: estándar invertido (0x04), negrita invertida (0x05) y numérica invertida (0x06).

En base a los formatos especificados con anterioridad, el formato de la trama es muy sencillo y se comprenderá mejor con un ejemplo real. A continuación se muestra una trama típica de transferencia de dinero entre dos cuentas tal y como se vería en la pantalla de la tarjeta:

En la versión actual, las tramas no pueden tener huecos en blanco. Es decir, si en una línea estándar caben 21 caracteres y queremos escribir solamente "Hola" (sin comillas), tendremos que rellenar el resto con espacios en blanco. Por tanto, para cada línea que queramos imprimir debemos calcular el número de espacios necesarios para completarla según el formato elegido. Así mismo, si necesitáramos una línea en blanco deberemos rellenarla de tantos espacios en blanco como caracteres quepan según el formato. Se espera poder mejorar este procedimiento, pero para este proyecto es el que tomaremos como referencia.

Una vez tenemos creada la trama con el texto que queremos imprimir (con los espacios en blanco necesarios), debemos especificar el formato para los caracteres que contiene.

Para ello añadimos el carácter especial ~ (en hexadecimal, 7B) y a continuación en hexadecimal y por pares la longitud del texto y su formato. Así, a la trama ejemplo vista con anterioridad habría que añadirle al final lo siguiente:

7B	Especificación de inicio de formato ("~")
15 **01**	21 caracteres en formato estándar
19 **03**	25 caracteres en formato numérico
2A **01**	42 caracteres en formato estándar
30 **02**	48 caracteres en formato negrita
19 **06**	25 caracteres en formato numérico invertido
2A **01**	42 caracteres en formato estándar
10 **05**	16 caracteres en formato negrita invertida

Los espacios en blanco los incluimos como caracteres de un formato específico también. Para que la trama esté lista para su envío a la tarjeta, debemos añadir al final el carácter '#', usado en la transmisión para delimitar los datos de una operación dada.

4.2. Envío de datos a la tarjeta

Como en todo dispositivo que use Bluetooth Low Energy, existen una serie de características de escritura que deben usarse para enviar datos. En el caso de la tarjeta eSignus, la característica de escritura es la 0x16.

Por tanto, para enviarle un dato a la tarjeta debemos mandarle el mensaje que queramos para que se escriba en la característica 0x16 con el UUID de la tarjeta (definido con anterioridad).

4.3. Recepción de datos desde la tarjeta

La forma que tiene la tarjeta de enviarnos datos es mediante notificaciones. Las notificaciones que recibimos desde la tarjeta eSignus son básicamente de dos tipos: ACK y recepción de firma. Cumpliendo con el estándar Bluetooth Low Energy, la tarjeta divide la firma en paquetes de 20 bytes antes de su envío. Posteriormente va mandando en forma

de notificaciones cada uno de estos paquetes, de tal forma que la notificación que incluya el carácter '#' nos indicará que se trata del último paquete de la firma.

A su vez, la tarjeta nos envía notificaciones de tipo ACK (terminadas también en el carácter '#') cuando:

1. Ha recibido 4 tramas de datos desde el móvil: es decir, cada cuatro paquetes enviados desde este a la tarjeta nos devuelve un ACK#.

2. Ha recibido la trama de confirmación de recepción de firma: cuando el móvil recibe el último paquete de la firma, envía una trama a la tarjeta que incluye la palabra clave "Aceptada#".

Por tanto, observamos como el smartphone no necesita leer de una característica de la tarjeta (de hecho, esto podría suponer un riesgo para la seguridad), sino que es la tarjeta quien nos envía los datos en forma de notificaciones.

Capítulo 5

Análisis

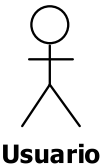
En esta primera fase definiremos cuales serán los usuarios de nuestro software y los Casos de Uso que lo componen. En la especificación del flujo de ejecución de cada caso de uso definiremos tres límites de ejecución de acciones:

1. Sistema: lo definimos como la aplicación en sí misma.
2. Banco: entidad financiera con la que se comunicará la aplicación.
3. Usuario: agente encargado de ejecutar las acciones sobre la aplicación.

Los diagramas serán leídos según la numeración establecida en los pasos, ejecutándolos en ese orden.

5.1. Tipos de usuarios

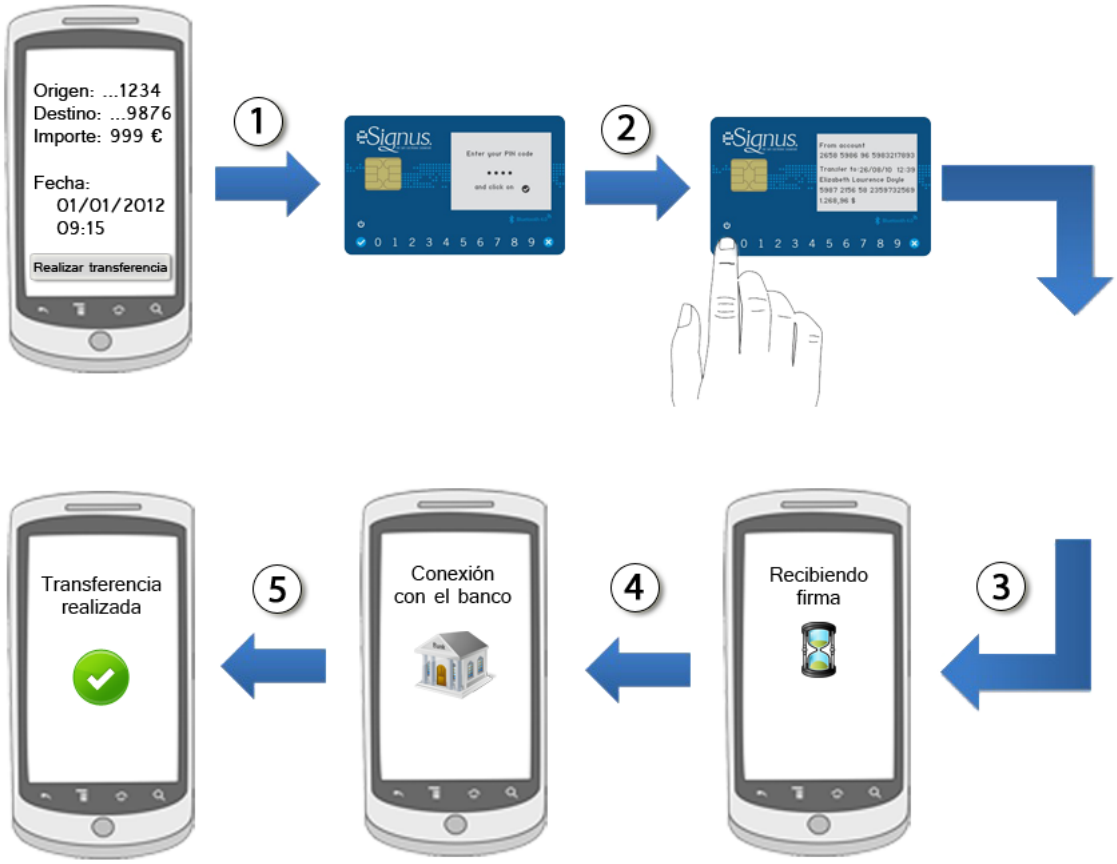
Al inicio de todo desarrollo software es necesario definir a los usuarios potenciales del sistema. En nuestro trabajo la jerarquía ha resultado muy sencilla, pues no es necesario tener permisos especiales para el mantenimiento de la aplicación, ni tampoco existen perfiles diferenciados. Por tanto, el conjunto de usuarios del sistema se agrupa en su totalidad en un único conjunto *Usuario*.



Jerarquía de usuarios

5.2. Transferencia

En un primer uso de la aplicación, el usuario podrá realizar una transferencia bancaria desde su smartphone Android usando como firmador la tarjeta eSignus. El usuario indicará los valores de la transferencia, enviará los datos para que los firme la tarjeta y enviará la firma al banco. En la imagen se muestra el esquema de funcionamiento de la Transferencia usando la tarjeta eSignus y nuestra aplicación.

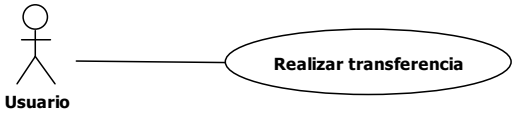


Pasos del proceso de Transferencia usando la tarjeta eSignus

El usuario deberá elegir la cuenta de origen desde la que quiere traspasar el dinero. Dado que en este trabajo no se ha implementado la conectividad con el banco (se ha simulado), permitimos la opción de añadir cuentas para su uso posterior en la aplicación. En un entorno real los números de cuenta de origen los proporcionaría el banco previa autentificación con el servicio que hubiese implantado. Posteriormente introducirá la cuenta de destino, así como el importe que desee transferir (como en las operaciones reales, el concepto de la transferencia es opcional). Una vez que la aplicación ya tiene los datos, construirá un paquete que será enviado a la tarjeta usando el protocolo Bluetooth Low Energy (1). Cuando el usuario decida aceptar la firma de los datos (2), la tarjeta empezará a enviar la firma resultante al teléfono (3). Los últimos pasos serán el envío de

la firma al banco, previa autenticación (4), la validación de este de la firma recibida y el envío posterior de su confirmación hacia nuestra aplicación para dar por finalizada la transferencia (5).

5.2.1. Realizar transferencia



Actores: Usuario
Precondiciones: ninguna.
Postcondiciones: Transferencia realizada.
Flujo básico:

1. Usuario pulsa el botón de "Transferencia" en la pantalla principal.
2. Se elige una cuenta de origen.
3. Se elige una cuenta de destino.
4. Se introduce el importe que se desea transferir.
5. Se confirma la transferencia (ver caso de uso "Confirmar transferencia").
6. El smartphone se conecta con la tarjeta (ver caso de uso "Conectar smartphone con tarjeta").
7. Se le envían los datos que se quieren firmar a la tarjeta (ver caso de uso "Enviar datos hacia la tarjeta").
8. Se espera la confirmación por parte del usuario de la firma en la tarjeta (ver caso de uso "Confirmar firma en la tarjeta").
9. Se reciben los datos firmados desde la tarjeta (ver caso de uso "Recibir datos de la tarjeta").
10. Una vez recibidos, se realiza una conexión con el banco (ver caso de uso "Realizar conexión con el banco") y se le envían los datos recién firmados (ver caso de uso "Enviar firma al banco").
11. Por último, el banco nos enviará la conformidad de la firma (ver caso de uso "Recibir confirmación del banco").

Flujos alternativos:

En el paso 2 del flujo básico, Usuario podrá elegir entre una de las cuentas de origen ya almacenadas o añadir una distinta y usarla en ese mismo momento (ver caso de uso "Añadir cuenta de origen"); ídem en el paso 3 (ver caso de uso "Añadir cuenta de destino").

Antes del paso 5, incorporamos un paso 4.1 donde *Usuario* introduciría el concepto de la operación (este paso es opcional).

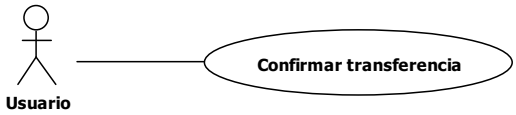
Si en cualquier momento entre los pasos 6 y 9 la conexión entre el smartphone y la tarjeta se interrumpe o se pierde, el proceso vuelve a ejecutarse desde el paso 6 del flujo básico.

Si en cualquier momento entre los pasos 10 y 11 la conexión entre el smartphone y el banco se interrumpe o se pierde, el proceso vuelve a ejecutarse desde el paso 10 del flujo básico.

Interfaz gráfica:

Se compone de las imágenes de interfaz de los casos de uso usados en los distintos pasos del flujo básico.

5.2.2. Confirmar transferencia

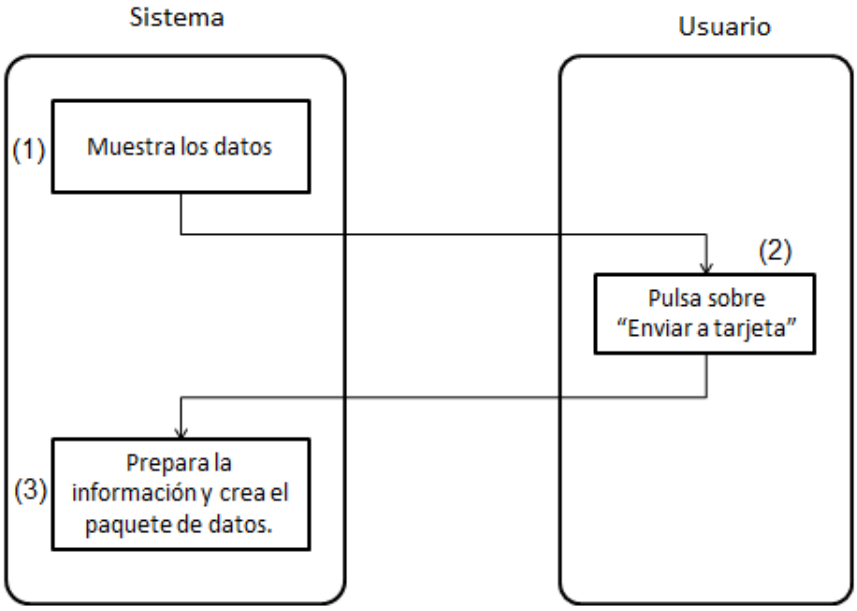


Actores: Usuario

Precondiciones: debe haber un proceso de transferencia en curso previo al envío de datos a la tarjeta.

Postcondiciones: el paquete de datos es enviado a la tarjeta para que siga el proceso de transferencia.

Flujo básico:



Interfaz gráfica:

El sistema muestra una pantalla similar a esta con los datos que serán firmados por la tarjeta.

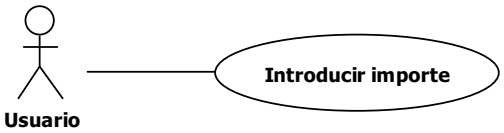
5.3. Datáfono

El segundo uso de la aplicación se basa en una transacción offline, como podría ser el uso de un datáfono. Este datáfono virtual estaría personalizado para el comercio con su nombre y demás información necesaria, ya que pertenece a él y es su medio para realizar cobros a usuarios con tarjeta eSignus.

Pasos del proceso de compra con datáfono usando la tarjeta eSignus

El usuario introducirá el importe de la operación y pulsará en el botón de confirmación. A continuación, los datos son enviados a la tarjeta (1). Si el usuario está conforme, firmará los datos (2) y el resultado será devuelto al teléfono (3). Por último, el teléfono realizará las operaciones oportunas para completar la transacción (4). Si se requiriese algún tipo de ticket o resguardo de la operación, podría realizarse una conexión con una impresora vía Bluetooth y mandarle a esta que imprima el ticket, o simplemente enviar un SMS o email de confirmación.

5.3.1. Introducir importe



Actores: Usuario
Precondiciones: ninguna.
Postcondiciones: ninguna.

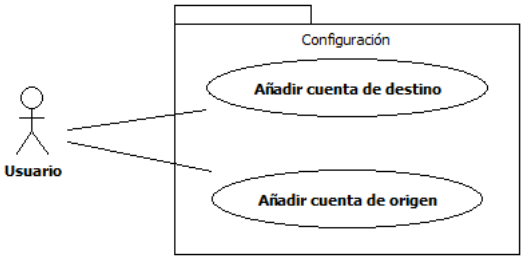
Flujo básico:

1. Usuario introduce el importe deseado usando la simulación de datáfono mostrado por la aplicación.

Interfaz gráfica:

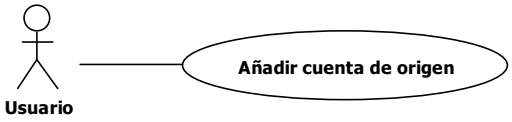
Interfaz principal donde el usuario introduce el importe de la operación.

5.4. Configuración



Como añadido adicional, el usuario podrá especificar previamente cuentas de origen (puesto que la conexión con el banco es simulada; recordemos que estos datos debería proporcionarlos el banco automáticamente) y de destino usadas habitualmente para que queden guardadas. De esta forma, no tendrá que introducirlas en cada nuevo movimiento.

5.4.1. Añadir cuenta de origen

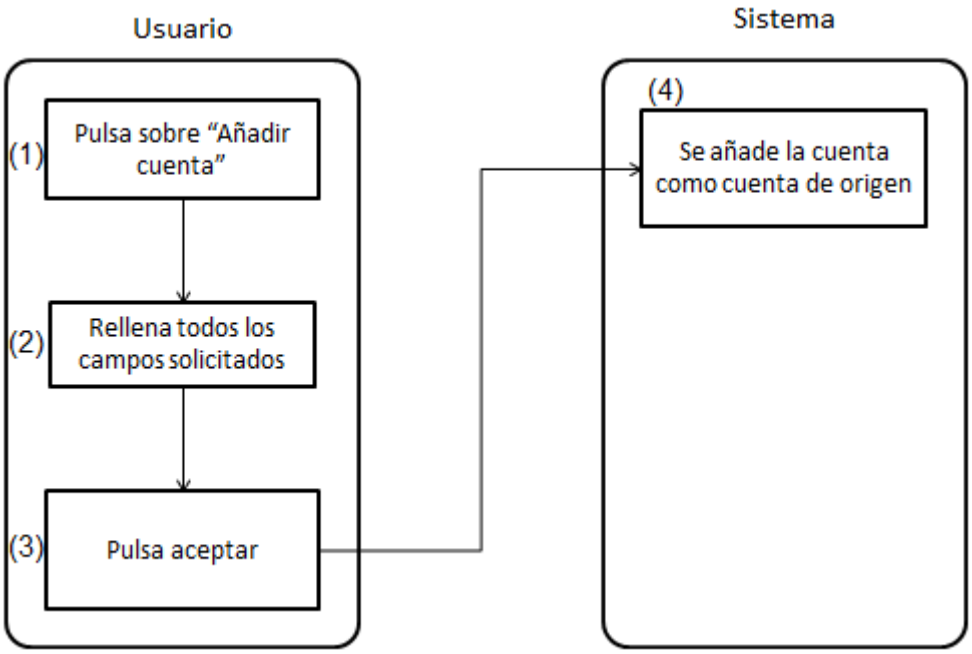


Actores: Usuario

Precondiciones: ninguna.

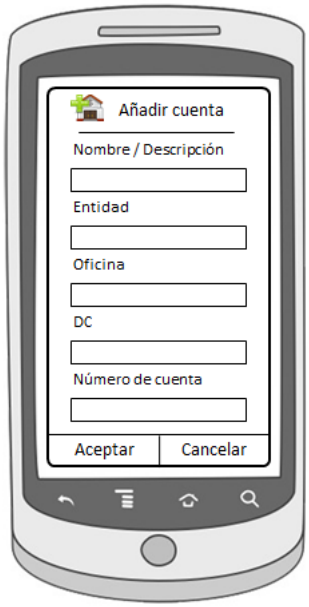
Postcondiciones: cuenta de origen añadida.

Flujo básico:



Interfaz gráfica:

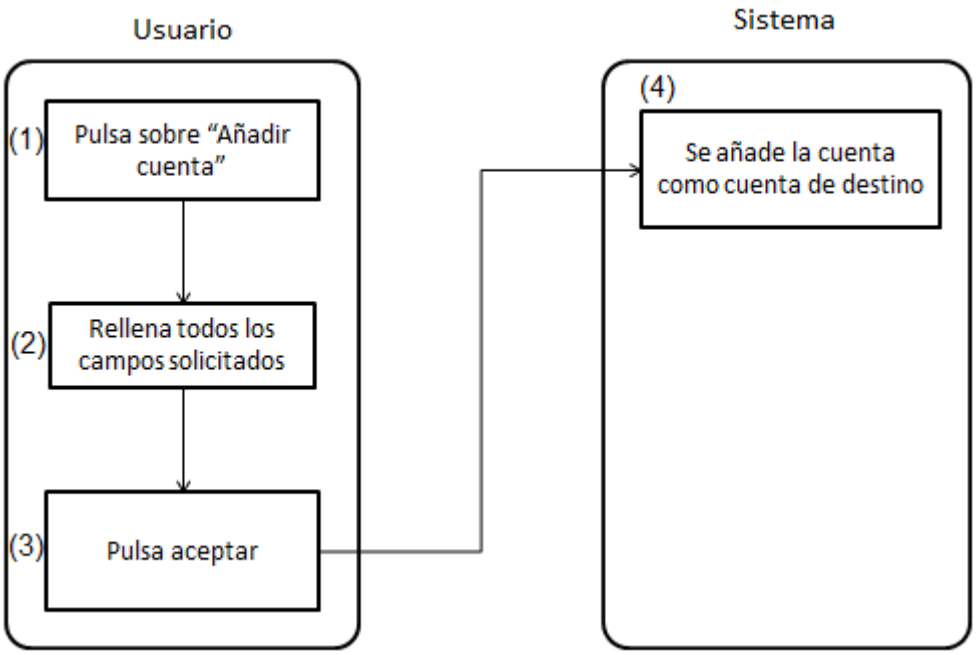
El usuario pulsa sobre el botón "Añadir cuenta"



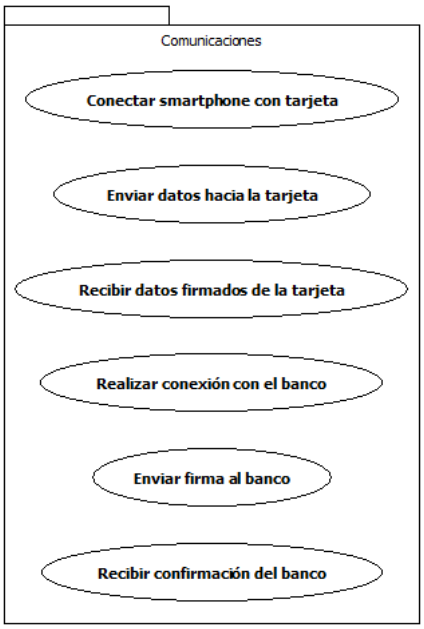
Se le muestra el formulario para que rellene los campos. Al pulsar en Aceptar, se añade la cuenta.

5.4.2. Añadir cuenta de destino

Actores: Usuario
Precondiciones: ninguna.
Postcondiciones: cuenta de destino añadida.
Flujo básico:

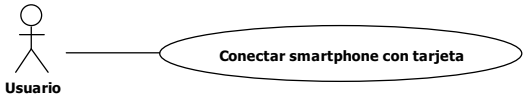


5.5. Comunicaciones



Internamente, el sistema realizará de manera transparente al usuario las comunicaciones con la tarjeta: conexión, envío de datos al banco y a la tarjeta y recepción de datos y confirmaciones. Se han identificado como Casos de Uso de la aplicación, pero sin actores que los ejecuten.

5.5.1. Conectar smartphone con tarjeta

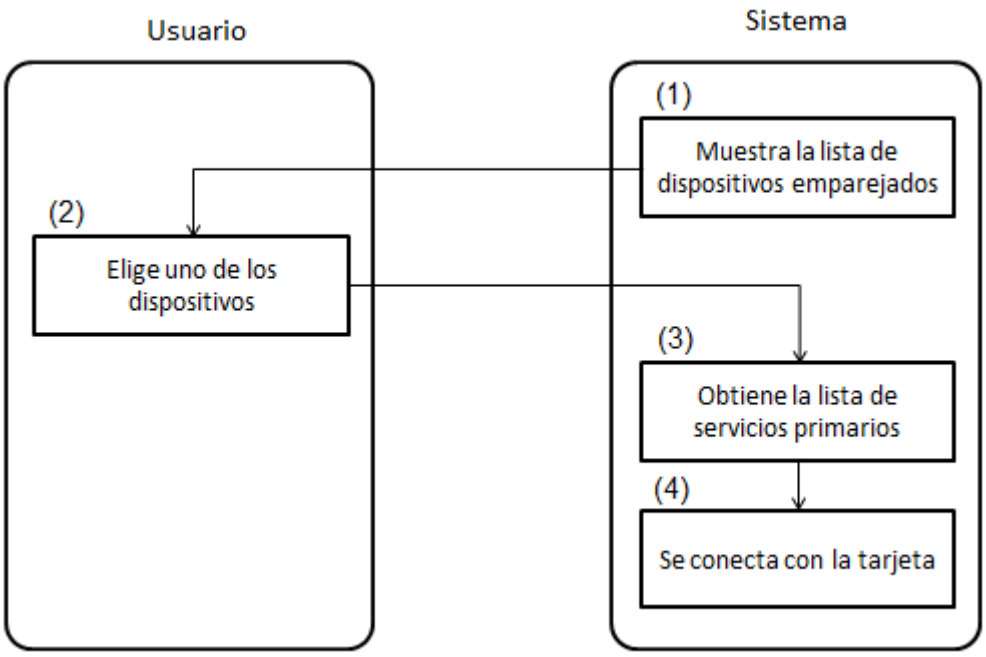


Actores: Usuario

Precondiciones: el smartphone debe tener el Bluetooth activado y la tarjeta debe haber sido emparejada previamente con el teléfono.

Postcondiciones: la tarjeta queda conectada al smartphone vía Bluetooth LE.

Flujo básico:



Flujos alternativos:

Si la tarjeta no ha sido emparejada con el smartphone previamente (usando las herramientas de emparejamiento del propio sistema operativo Android), esta no saldrá en la lista de dispositivos del paso 1 y por tanto no será posible la conexión.

Si en el paso 3 la tarjeta no devolviera la lista de servicios, puede ser que el UUID no sea el correcto. En este caso, el intento de conexión se interrumpe de vuelta al paso 1.

En el paso 4, el smartphone hace la petición de conexión con la tarjeta eSignus, pero la respuesta es recibida de manera asíncrona. Por tanto, esta conexión no es 100 % segura y la tarjeta podría denegarla en caso de que el UUID no permita conexiones o simplemente que el propio enlace se caiga.

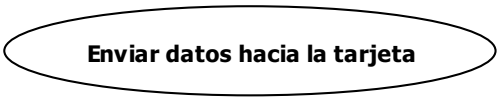
Interfaz gráfica:

El sistema muestra la lista de dispositivos Bluetooth emparejados con el smartphone.

La aplicación intenta realizar la conexión con el dispositivo indicado.

Se establece el enlace entre la tarjeta y el smartphone.

5.5.2. Enviar datos hacia la tarjeta

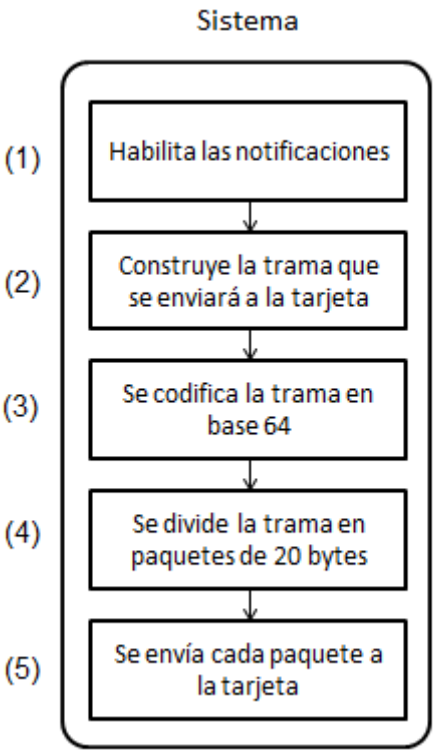


Actores: ninguno.

Precondiciones: debe existir una conexión entre la tarjeta eSignus y el smartphone según el caso de uso *Conectar smartphone con tarjeta*.

Postcondiciones: el sistema queda a la espera de la confirmación de la firma en la tarjeta.

Flujo básico:



Flujos alternativos:

Los sucesivos envíos del paso 5 podrían fallar si el enlace se cae. De ocurrir esto, el caso de uso se interrumpiría y pasaría a ejecutarse el caso de uso *Conectar smartphone con tarjeta.*

Interfaz gráfica:

La aplicación, tras habilitar las notificaciones y construir la trama, comienza a enviarle datos a la tarjeta.

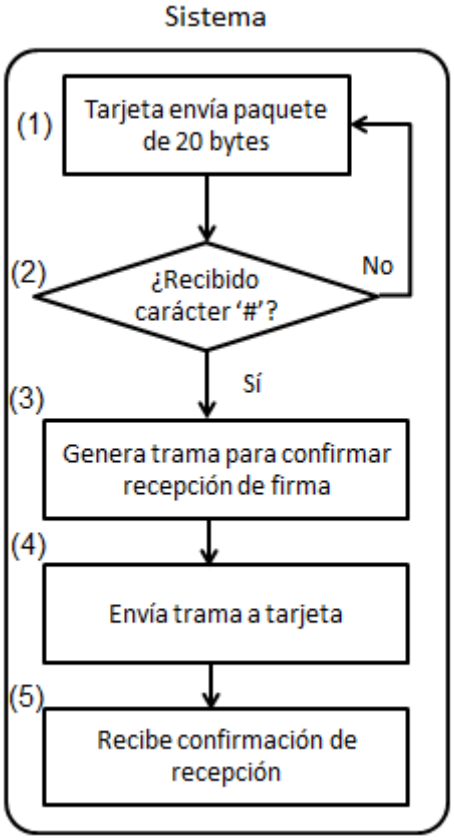
5.5.3. Recibir datos firmados de la tarjeta

Actores: ninguno.

Precondiciones: debe existir una conexión entre la tarjeta eSignus y el smartphone según el caso de uso *Conectar smartphone con tarjeta.*

Postcondiciones: ninguna.

Flujo básico:



Flujos alternativos:

En cualquiera de los pasos del flujo básico, la conectividad entre ambos dispositivos puede cesar. Si se diera la circunstancia, el caso de uso se interrumpiría y se ejecutaría el caso de uso *Conectar smartphone con tarjeta.*

Interfaz gráfica:

Una vez que la tarjeta firme los datos, el smartphone comenzará a recibirlos.

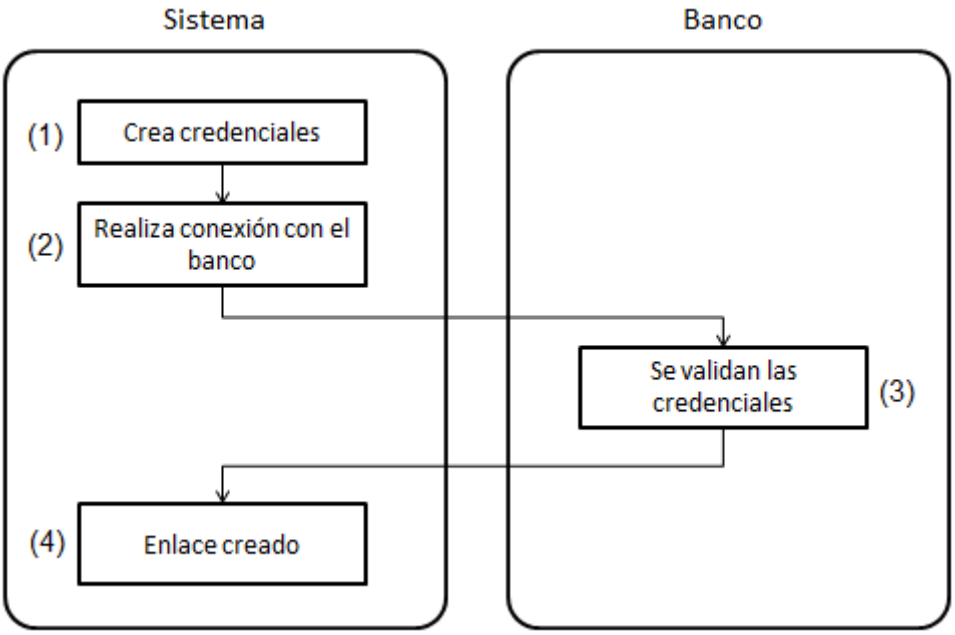
5.5.4. Realizar conexión con el banco

Actores: ninguno.

Precondiciones: el smartphone debe tener conectividad habilitada con el servicio del banco encargado de verificar la firma.

Postcondiciones: se crea una conexión entre el smartphone y el servicio del banco.

Flujo básico:



Interfaz gráfica:

El smartphone se conecta al banco para posteriormente enviarle la firma.

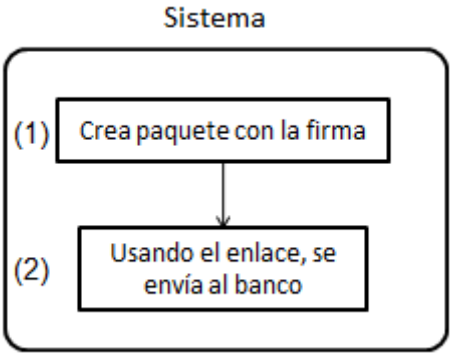
5.5.5. Enviar firma al banco

Actores: Usuario

Precondiciones: el sistema debe tener una firma recibida desde la tarjeta según el caso de uso *Recibir datos firmados de la tarjeta* y debe existir una conexión válida con el servicio del banco según el caso de uso *Realizar conexión con el banco*.

Postcondiciones: ninguna.

Flujo básico:



Flujos alternativos:

Como el protocolo del servicio del banco no está definido, los flujos alternativos surgirán a partir de excepciones o errores que pudieran darse en el flujo básico: formato inadecuado de envío, error en el envío de la firma, etc..

Interfaz gráfica:

La aplicación envía la firma al banco para que este la verifique.

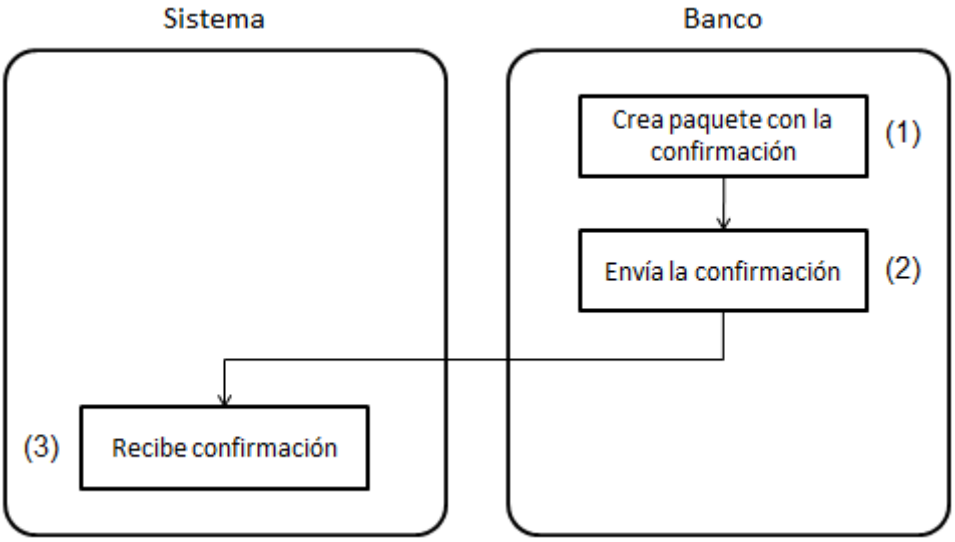
5.5.6. Recibir confirmación del banco

Actores: ninguno.

Precondiciones: debe existir una conexión válida con el servicio del banco según el caso de uso *Realizar conexión con el banco* y el sistema tuvo que enviar previamente una firma al banco según el caso de uso *Enviar firma al banco*.

Postcondiciones: el sistema completa el proceso de transferencia al ser este el último caso de uso que se ejecuta.

Flujo básico:



Flujos alternativos:

Como el protocolo del servicio del banco no está definido, los flujos alternativos surgirán a partir de excepciones o errores que pudieran darse en el flujo básico: la firma es valida, la firma no es válida, error en el envío de la confirmación, etc..

Interfaz gráfica:

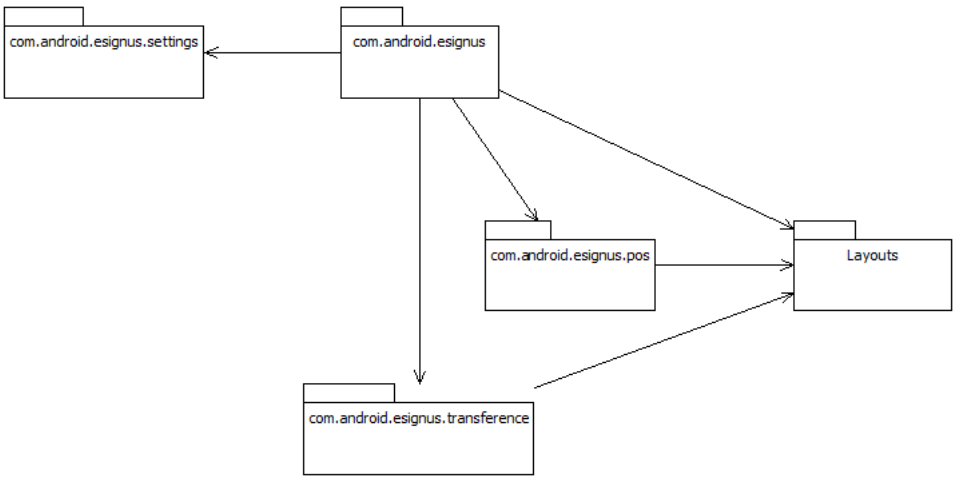
La aplicación espera por el banco para que este le confirme la validez de la firma.

Si la firma es correcta, el banco da su conformidad y el proceso termina satisfactoriamente.

Capítulo 6

Diseño

Nuestra aplicación se compone de dos subaplicaciones (transferencia y datáfono), así que cada una de ellas irá en un paquete independiente. A su vez, todos estos paquetes deben comunicarse con las vistas para mostrar elementos por pantalla. Android no nos permite jerarquizar las vistas en subpaquetes, así que todas ellas se incluyen en el paquete *Layouts*. Por último, existe un paquete de configuración de la aplicación para que esta pueda ser personalizada (almacenar cuentas de origen y cuentas de destino preestablecidas o de uso común).



Android sigue la convención típica de nombres de paquetes en Java, intentando que estos puedan leerse como si fueran directorios. Así, el paquete *com.android.esignus.settings* podría ser perfectamente la carpeta *com/android/esignus/settings*. De esta forma es fácil referenciar los paquetes y subaplicaciones cuando la aplicación va creciendo.

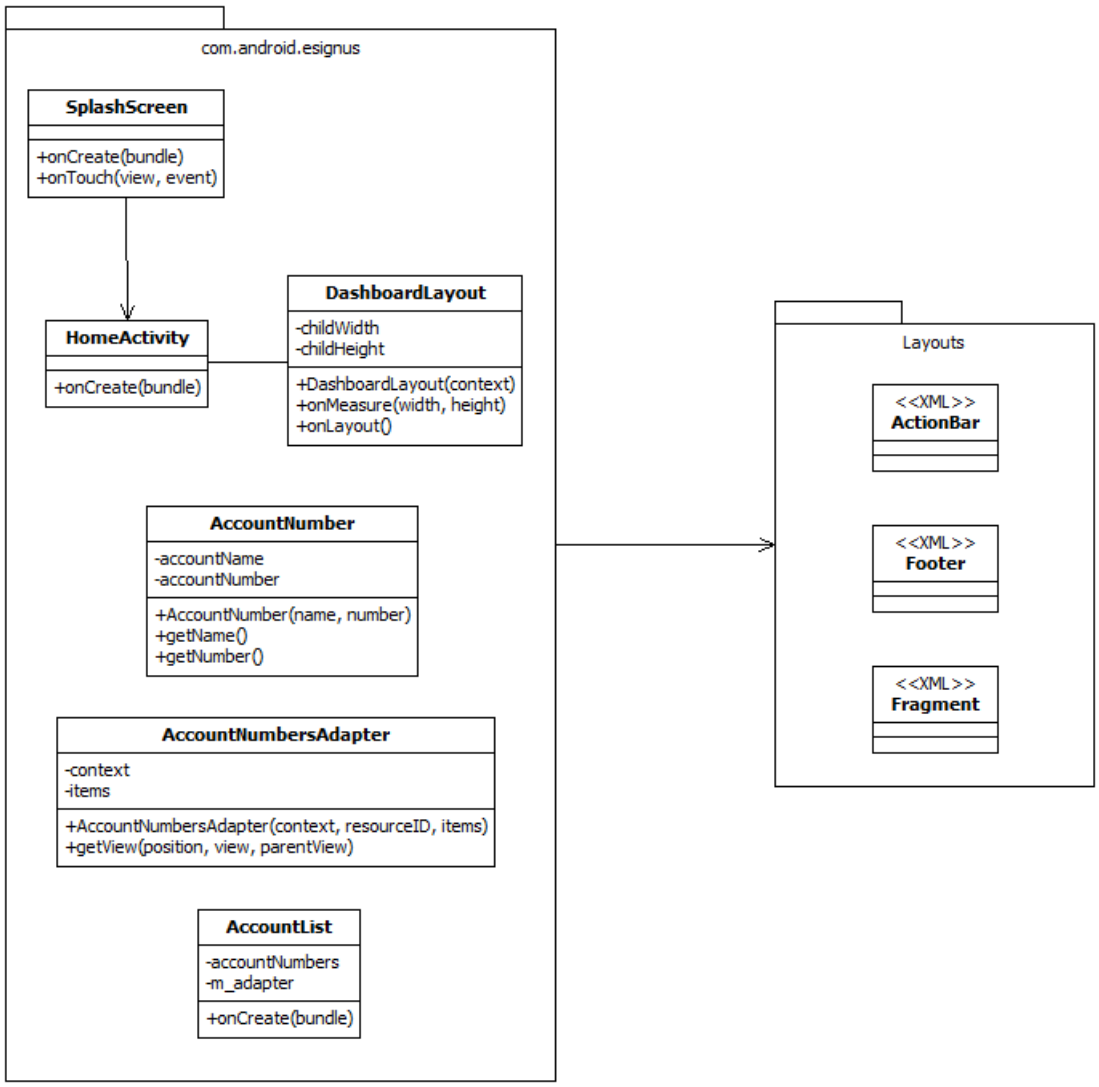
Teniendo en cuenta esto, hemos englobado nuestra aplicación bajo el nombre clave de paquete **com.android.esignus**. Este paquete contiene las clases encargadas de gestionar el acceso inicial a la aplicación:

- *SplashScreenActivity*: cuando iniciamos la aplicación, se nos muestra una pantalla

con el logo del producto eSignus durante un corto periodo de tiempo. Esta pantalla es descartable pulsando sobre la pantalla.

- *HomeActivity*: menú principal de la aplicación con accesos a las dos subaplicaciones (transferencia y datáfono) así como a la configuración.
- *DashboardLayout*: se encarga de dibujar y colocar en pantalla los distintos elementos de la vista inicial usando el patrón Dashboard.
- *AccountNumber* / *AccountNumberAdapter* / *AccountList*: clases encargadas de mostrar los números de cuenta y sus nombres; fueron creadas de cara a la interfaz gráfica de la aplicación.

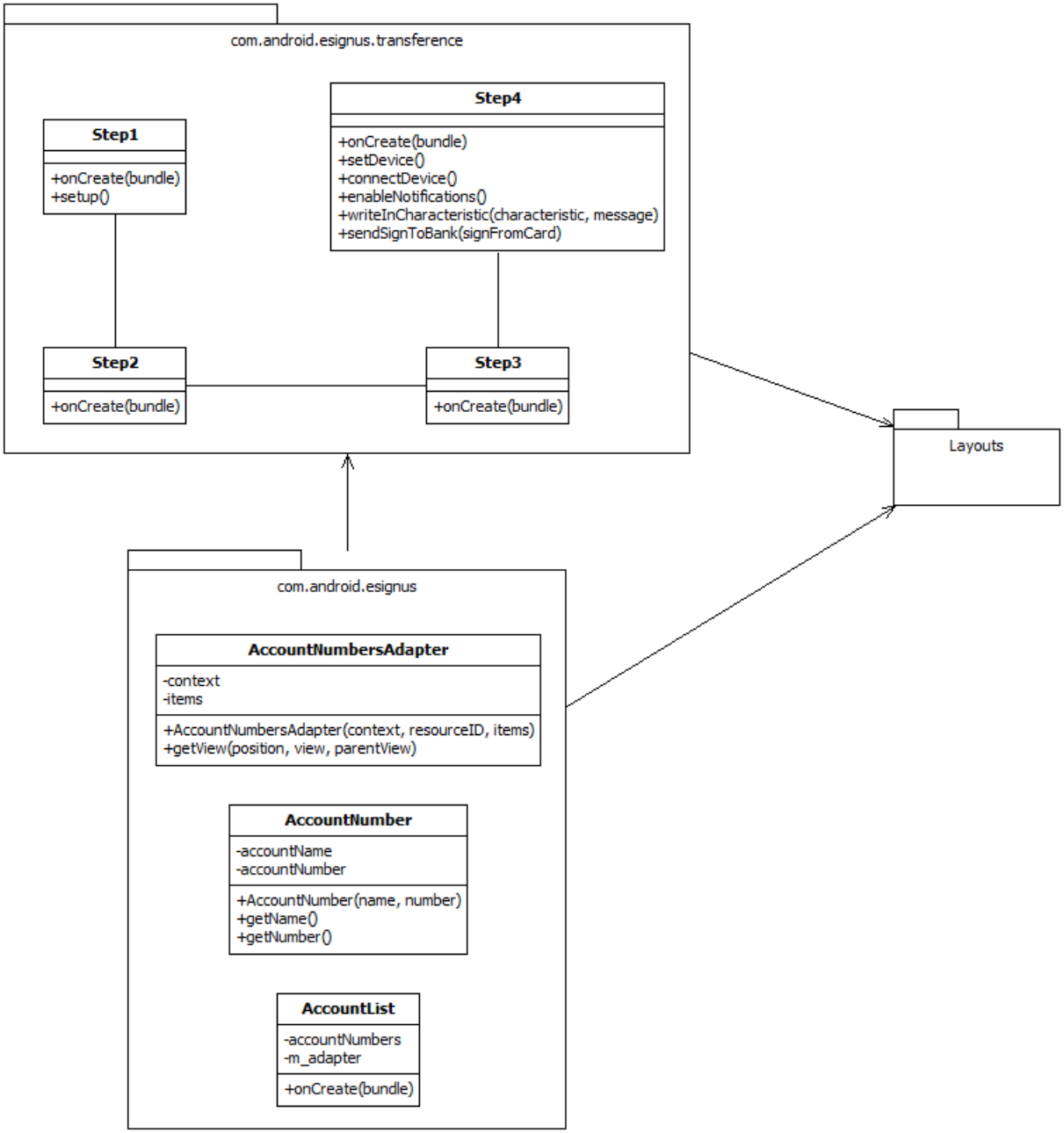
Este paquete debe comunicarse con las vistas correspondientes del paquete **Layouts**, como son las pertenecientes al menú principal (*Fragment*, *Footer* y *ActionBar*), la vista de la pantalla de splash (*Splash*) y las vistas para mostrar los números de cuenta (*Listview_header* y *Listview_row*).



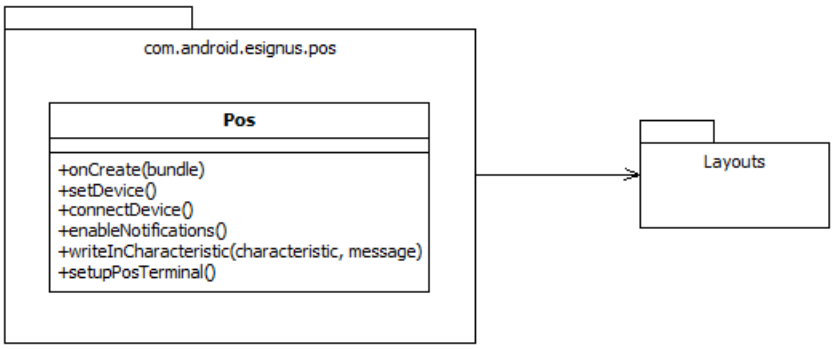
La aplicación de transferencia está recogida en el paquete **com.android.esignus.transference**. Como este proceso sigue un proceso de pasos, hemos visto conveniente llamar a cada clase según el paso en que se encuentre el proceso:

- *Step1*: muestra la vista de selección de la cuenta de origen desde la que se hará la transferencia; así mismo permite añadir cuentas de origen adicionales.
- *Step2*: crea otra vista donde se elige la cuenta de destino y se introduce el importe de la operación; opcionalmente se puede especificar el concepto de la transferencia.
- *Step3*: muestra una vista de información donde se muestran todos los datos recogidos en los pasos anteriores (cuentas de origen y destino, importe, concepto, fecha y hora de la transferencia).
- *Step4*: esta clase es la que se encarga de mostrar los dispositivos emparejados, crear la conexión con la tarjeta, enviarle los datos, recibir la firma y conectarse con el banco. Como la conexión con el banco no está implementada, hemos podido incluirla en esta clase; cuando se acuerde el protocolo, debería ir en otra clase aparte para un mejor mantenimiento de la aplicación.

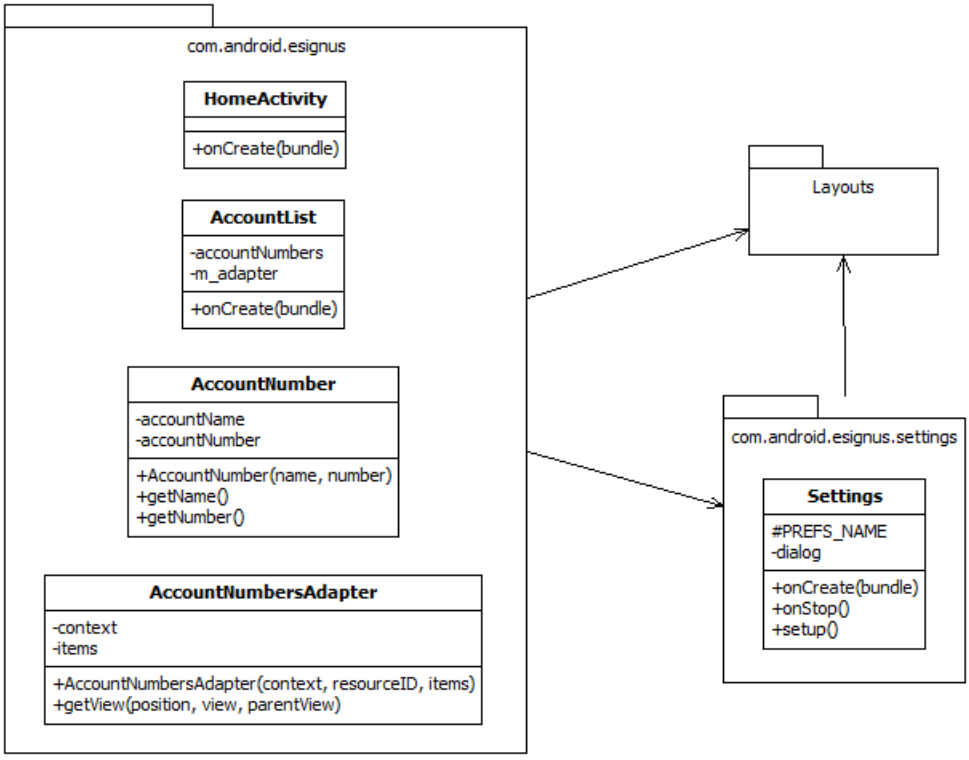
Este paquete también se comunica con el paquete **Layouts** para mostrar toda la información del proceso.



La aplicación del datáfono se incluye en el paquete **com.android.esignus.pos** y se compone de una única actividad denominada *Pos* donde el usuario especifica el importe, envía y recibe de la tarjeta y finaliza la operación. Podría decirse que es una clase parecida a la clase *Step4* del proceso de transferencia, pero incluyendo la gestión del datáfono en pantalla y quitando la conectividad con el banco.



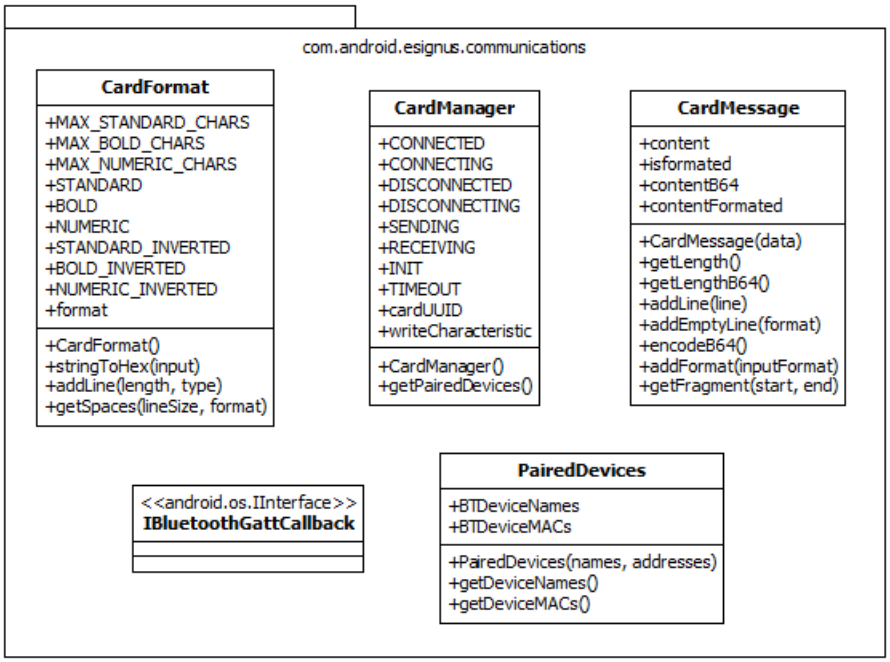
La configuración de la aplicación se engloba en el paquete **com.android.esignus.settings** y usa las funciones típicas que proporciona Android para la gestión de configuraciones. En nuestro caso, permitimos que el usuario guarde cuentas de origen y de destino comúnmente utilizadas para agilizar las transferencias. Esta configuración es accesible desde cualquier lugar de la aplicación de cara a completas las vistas que así lo requiriesen con la información almacenada.



El paquete de comunicaciones **com.android.esignus.communications** es usado tanto por la subaplicación de transferencia como por la del datáfono. Las clases que lo componen son:

- *CardMessage*: es el tipo de la trama que será enviada a la tarjeta; es creada en tiempo de ejecución en función de los datos proporcionados por el usuario.

- *CardManager*: contiene información sobre los dispositivos emparejados, el estado de la conexión y las características de la tarjeta (característica de escritura y UUID de conexión).
- *CardFormat*: implementa los formatos posibles de las tramas enviadas a la tarjeta; cuando creamos un formato, debemos añadírselo a la trama creada con la clase CardMessage para que la tarjeta la de por válida.
- *PairedDevices*: gestión de dispositivos emparejados (nombres y direcciones MAC).
- *IBluetoothGattCallback*: clase interna de la propia API de Motorola para los perfiles estándar.



Este paquete no requiere comunicación con las vistas ya que se tratan de clases de gestión, no de visualización de información.

Capítulo 7

Implementación

En este apartado explicaremos los aspectos más relevantes del desarrollo del proyecto.

7.1. Patrón DashBoard

La aplicación sigue uno de los patrones básicos que Google dio a conocer en su conferencia anual Google I/O del año 2010 [7] y que las aplicaciones móviles actuales Android implementan. Este patrón está centrado en la interfaz gráfica y su usabilidad de cara al usuario final, no es un patrón de diseño como Singleton, Builder, etc.., más orientados al desarrollo de código.

Ejemplos de aplicaciones que usan el patrón Dashboard, como son Facebook, Evernote o Twitter.

El patrón se llama *Dashboard*. Con este patrón lo que pretendemos conseguir es una pantalla inicial que permita al usuario acceder de manera rápida a las acciones principales

de la aplicación. Para ello, la pantalla principal consistirá en una serie de iconos con un subtítulo asociado, teniendo así una vista principal clara y sin demasiados elementos visuales que puedan confundir al usuario a la hora de usar la aplicación.

La implementación de este patrón resulta bastante simple siguiendo el tutorial de AndroidHive [1]. Básicamente lo que necesitamos es:

- Definir los estilos del dashboard (tamaño de textos bajo los iconos, color de fondo, etc..), la barra superior (imagen de fondo, alto, etc..) y la barra inferior (color de fondo, alto, etc..).
- Diseñar la vista (layout) de cada uno de estos elementos por separado.
- Diseñar una vista que incluya las vistas de los elementos anteriores.
- Crear una clase que gestione la disposición automática de elementos en el dashboard según su número.

Una vez hecho todo esto, tendremos que poner esa actividad en la arquitectura como actividad principal de nuestra aplicación.

7.2. Uso de la API Bluetooth Low Energy de Motorola

A fecha de la realización de este trabajo fin de máster, Android no soporta de manera nativa en su Android SDK el uso de Bluetooth Low Energy. Para este trabajo hemos tenido que usar la API que Motorola ha creado y de la que haremos un pequeño análisis en esta sección.

Todas las comunicaciones se realizan de manera asíncrona, recibiéndose notificaciones en llamadas de tipo *callback* al finalizar la operación solicitada. Solo algunas de ellas devuelven un resultado síncrono, como la obtención de los servicios primarios o las características de la capa GATT del dispositivo (normalmente y tras el emparejamiento, el smartphone guarda esta información automáticamente).

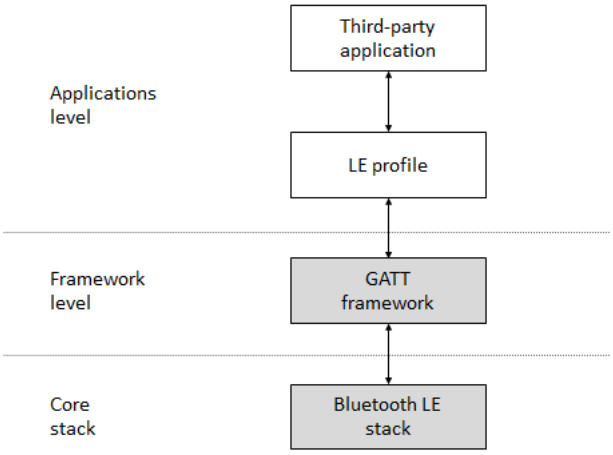


Diagrama de la arquitectura para la API de Motorola. En gris, las funciones proporcionadas por Motorola. En blanco, las funciones proporcionadas por las aplicaciones que usen la API.

En nuestra opinión, quizás lo más engorroso de su uso es la gestión de las notificaciones/indicaciones. Al estar todo dentro de una única interfaz, la gestión y el mantenimiento de la aplicación en caso de que tuviéramos numerosos tipos de notificaciones sería bastante complejo. Motorola podría prever este manejo proporcionando a los desarrolladores algún método para categorizarlas, ya que no todas las aplicaciones envían notificaciones/indicaciones del tipo ACK.

Respecto a la documentación está bastante bien si la intención es usar perfiles típicos de Bluetooth LE, pero se guardan bastantes detalles que a la hora de realizar implementaciones reales suponen auténticos dolores de cabeza.

En nuestra primera aproximación con la tarjeta eSignus, que utiliza el perfil Heart Rate Monitor, decidimos usar la implememementación de este perfil directamente de la API. Sin embargo, tardamos en darnos cuenta que la firma estaba llegando con algunos caracteres menos. La razón era que Motorola en su implementación había presupuesto que el uso que se le dará es con sensores cardíacos, y no con aplicaciones universales que usen este perfil.

Por otro lado, la API sufre de numerosos bugs que los desarrolladores reportan en el foro oficial[6]. Algunos de ellos resultan en reinicios espontáneos del teléfono en condiciones concretas, o la incapacidad de usar más extensivamente las características de los perfiles.

De todas formas, creemos que la tecnología Bluetooth LE está despegando y cuando Android lo soporte en su totalidad y decidan realizar la API oficial, las posibilidades se dispararán y la integración será mejor y más fiable.

7.3. Parámetros de conexión entre el smartphone y la tarjeta eSignus

Uno de los inconvenientes de la API de Motorola es la imposibilidad de negociar los parámetros de conexión entre el dispositivo Bluetooth de la tarjeta y el del smartphone. Estos parámetros de conexión se suelen utilizar cuando buscamos eficiencia en las comunicaciones permitiéndonos ajustar de manera muy fina la conectividad entre ambos dispositivos (latencias, timeouts) obteniendo unas velocidades más elevadas que las soportadas por defecto.

La conexión y envío de los datos desde el smartphone a la tarjeta se realizan de manera muy rápida, no llevando más de 2 segundos en transmitir toda la información. El problema viene cuando comenzamos a recibir la firma desde la tarjeta. Pese a que el tamaño de la firma casi siempre suele estar entorno a los 2 Kb de datos, puede llegar a tardar hasta

14 o 15 segundos en recibirse debido a que los parámetros de conexión que tenemos que usar son los que vienen por defecto. Incluso tratamos de forzar los tiempos modificando el firmware de la tarjeta (que si es configurable), pero recibíamos la firma parcialmente.

Realizando pruebas con un dispositivo Bluetooth LE de Texas [14] (que si proporcionaba la negociación) y usando un applet desarrollado en Java obtuvimos la firma en el programa después de 5 segundos aproximadamente.

Preguntando en los foros oficiales de Motorola, las respuestas obtenidas de los ingenieros fueron la posibilidad de aumentar el tamaño de los paquetes, aspecto inviable si estamos hablando de Bluetooth LE, o usar otro protocolo para el envío y recepción de datos, como RS232, respuesta descartada debido a la arquitectura de nuestro sistema.

Por tanto, la aplicación funciona tan rápida como la API nos lo permite. Como ya hemos comentado, si Google lanza de manera oficial su API para Bluetooth LE y soporta la negociación de parámetros, estimamos que la aplicación será capaz de recibir datos entre dos o tres veces más rápido.

7.4. Envío y conectividad con el banco

En la parte final de la aplicación de transferencia, la firma recibida desde la tarjeta debe ser enviada al banco para que este realice la operación real (comprobando también si la firma es válida). Para el trabajo fin de máster no disponemos de un entorno de desarrollo con un banco ficticio, así que lo que hemos hecho ha sido simular la existencia de un servicio web en un banco y mediante la clase *Runnable* de Android establecimos tiempos aproximados en los que tardaría el banco en realizar los tres pasos básicos: validación de las credenciales del usuario, comprobación de la firma, y envío de la confirmación de la operación.

El código que realiza este proceso es el siguiente:

```
// Creamos la conexión con el banco y enviamos nuestras credenciales
Handler connectingToBank_handler = new Handler();
connectingToBank_handler.postDelayed(new Runnable() {
  public void run() {
     // Enviamos la firma al banco y esperamos su confirmación
     Handler sendingToBank_handler = new Handler();
     sendingToBank_handler.postDelayed(new Runnable() {
       public void run() {
          // Confirmación recibida
       }
     }, 5000);
  }
}, 1000);
```

Básicamente creamos un código y establecemos que se ejecute después de X segundos. El código que hemos incluido ha sido unas animaciones para indicarle al usuario que la operación se está realizando (no lo hemos citado por no considerarlo relevante). Por tanto, la conexión tardará 1 segundo y el envío de la firma al banco más la confirmación unos 5 segundos. Si hiciéramos una implementación real de este proceso, deberíamos estudiar e implementar el código necesario para el protocolo que el servicio web del banco nos proporcione.

7.5. Algoritmo de verificación de número de cuenta bancaria

En España, las cuentas bancarias se identifican con un número de 20 cifras. Este número se separa en cuatro grupos (ejemplo: $1234 - 1234 - 12 - 1234567890$), donde 4 números identifican al banco, 4 números identifican la sucursal, 2 números son dígitos de control y 10 números son el número de cuenta dentro de la sucursal.

Para validar la cuenta se utilizan los dos dígitos de control, siendo el primero para validar los 8 primeros (banco y sucursal), y el segundo para validar los últimos 10 (el número de cuenta). El algoritmo que nos sirve para verificar si un número de cuenta bancaria es válido lo explicaremos cogiendo como cuenta de ejemplo $1234 - 1234 - 12 - 1234567890$, donde 1234 (banco), 1234 (sucursal) y 1234567890 (cuenta). Para los posteriores cálculos, se definen una serie de pesos:$1, 2, 4, 8, 5, 10, 9, 7, 3, 6$.

Empezamos con el primer dígito de control. Cogemos los 4 dígitos del banco y sucursal, 12341234 y como tenemos 10 pesos (listados antes), añadimos dos ceros delante 0012341234, y multiplicamos cada dígito por su peso correspondiente, el orden vienen dado por la posición que ocupan, es decir, el primer número por el primero peso, el segundo número por el segundo peso, etc.. y cada resultado lo sumamos para obtener un solo número:

$$(0*1)+(0*2)+(1*4)+(2*8)+(3*5)+(4*10)+(1*9)+(2*7)+(3*3)+(4*6) = 4+16+15+40+9+14+9+24 = 131$$

Ahora hacemos la siguiente operación para obtener el primer número del dígito de control:

$$11 - (131 \, mod \, 11) = 11 - 10 = 1$$

Y para obtener el segundo dígito de control operamos de la misma forma con los 10 dígitos de la cuenta 1234567890:

$$(1*1)+(2*2)+(3*4)+(4*8)+(5*5)+(6*10)+(7*9)+(8*7)+(9*3)+(0*6) = 1+4+12+32+25+60+63+56+27+0 = 280$$

$$11 - (280 \, mod \, 11) = 11 - 5 = 6$$

Por tanto los dígitos de control son el valor 16, y no 12 como habíamos puesto al principio.

7.6. Codificación en Base64

Los datos que se envían del smartphone a la tarjeta eSignus deben estar codificados en Base64. Android implementa la codificación y decodificación en este formato, pero los resultados obtenidos eran erróneos en nuestro caso. Nos dimos cuenta de este error al comprobar que la trama que le enviábamos a la tarjeta estaba correctamente construida, pero que al codificarla usando el programa Cryptools daba un resultado distinto al devuelto por la codificación usando las funciones internas de Android.

Buscando información encontramos una librería de ApacheCommons[1] que si funciona como se espera. De hecho, realizando el mismo proceso pero codificando los datos con Android y con ApacheCommons, en el primer caso la tarjeta nos devolvía el mensaje de "Datos erróneos".

[1] http://commons.apache.org/codec/download_codec.cgi

Capítulo 8

Pruebas

Las pruebas son procesos que nos permiten verificar la calidad de un producto. Se utilizan para identificar posibles fallas del producto, requisitos que no se cumplen o problemas de usabilidad. Nuestras pruebas irán dirigidas a una aplicación móvil que utiliza varias tecnologías diferentes. Debido a que el emulador incluido con el SDK de Android no incorpora la funcionalidad para Bluetooth Low Energy, probaremos la aplicación directamente en el Motorola RAZR Droid cedido por INELCAN S.L.

En esta fase del trabajo se realizaron pruebas tanto de caja negra como de caja blanca. Las pruebas de caja negra se centran en lo que se espera de un módulo, pero sin preocuparse de lo que esté haciendo el módulo por dentro (suministrándole unas entradas y observando sus salidas). Por otra parte, en las pruebas de caja blanca se revisa el código internamente, asegurando que las variables tengan los valores adecuados y que las funciones se están ejecutando correctamente.

En las sucesivas secciones veremos los tipos de pruebas que hemos realizado sobre la aplicación: de unidad y de integración. Explicaremos teóricamente en qué consisten, pero no citaremos aquí cada una de las pruebas para no volver este capítulo muy extenso.

8.1. Pruebas de unidad

Las pruebas de unidad se centran en el módulo y sirven para asegurar que cada uno de ellos funciona correctamente por separado, independientemente del resto de la aplicación.

Estas pruebas nos aseguran que cada módulo funciona bien por sí mismo, minimizando enormemente los riesgos que la integración añade al proceso de desarrollo. Además, independiza en parte el acoplamiento con las vistas ya que no buscamos que los datos se muestren en su sitio o con un cierto formato en pantalla, sino que estos datos son creados, modificados y enviados correctamente a las vistas.

En nuestro caso, las pruebas de unidad se realizaron en todos los módulos que componen la aplicación, categorizándolos por paquetes para un mejor seguimiento. En el

apartado de Diseño se encuentran explicadas las clases que componen la aplicación y su funcionamiento.

8.2. Pruebas de integración

Una vez realizadas las pruebas de unidad podemos pasar a realizar las pruebas de integración, en las que probaremos cómo se comportan unos módulos con otros. También se verificará que los componentes se integran correctamente y que el sistema siempre queda en un estado estable.

Las pruebas de integración pueden ser realizadas de tres formas:

1. Top-Down: integrar y probar desde niveles de abstracción superiores a niveles de abstracción inferiores.

2. Bottom-Up: inverso al Top-Down, consiste en integrar y probar desde niveles de abstracción inferiores a niveles de abstracción superiores.

3. Big-Bang: integrar todo al mismo tiempo.

En nuestra opinión la mejor opción es integrar en modo Bottom-Up ya que controlamos el proceso desde sus orígenes. Si detectamos fallos a nivel de estructuración de datos o de formatos podemos evitar que se propaguen hacia niveles más altos donde toda esta información estaría enmascarada por procesos más complejos.

Capítulo 9

Resultados y conclusiones

Hemos visto como el protocolo Bluetooth Low Energy y la tecnología Android pueden ser combinados a la perfección para lograr resultados más que aceptables en el campo de las aplicaciones móviles. Además y gracias a la tarjeta eSignus hemos podido aumentar la seguridad en el comercio online mediante la infraestructura de transferencias y cobros desarrollada en la aplicación.

Hablando de los aspectos técnicos de comunicación entre la tarjeta y el móvil, hay que tener en cuenta que la API de desarrollo para Bluetooth LE usada no tiene soporte oficial por parte de Google. Por tanto, no hemos conseguido el rendimiento óptimo que creíamos poder obtener en un principio. Como ya comentamos, la principal limitación es la imposibilidad de negociar parámetros de conexión para, de esta forma, conseguir el mejor rendimiento posible en las comunicaciones entre ambos dispositivos. En este aspecto encontramos serios problemas para obtener velocidades mayores al recibir la firma desde la tarjeta ya que, en nuestra opinión, la recepción de 2k de datos de media no es demasiada información.

Sin embargo, esto no ocurre con la aplicación que INELCAN S.L. ha desarrollado para iPhone. Pese a seguir teniendo problemas con la velocidad, la API de Bluetooth Low Energy de iOS para iPhone (que si soporta de manera nativa el protocolo) permite la negociación de ciertos parámetros con bastantes restricciones. Se han conseguido ligeras mejoras en los tiempos de comunicación, pero no todo lo que se esperaría ya que hay parámetros que no se pueden tocar, y otros que tienen muy pocos valores disponibles para variar. En este aspecto, quizás Apple sea más reacia a soportar la negociación de parámetros en su totalidad para proteger sus propios dispositivos.

De todas formas, estamos tratando con una tecnología que aún está un poco verde y cuando Android soporte de manera nativa en su SDK el protocolo, las velocidades que conseguiremos serán sin duda superiores.

Respecto al desarrollo en Android, hemos echado en falta una herramienta potente para diseñar interfaces gráficas en el propio SDK de Android. En nuestra experiencia en desarrollo móvil, las interfaces gráficas con XCode para iOS resultan mucho más simples

de crear. Además gracias al concepto de Storyboard es muy intuitivo y visual seguir el flujo del programa desde las propias vistas de la aplicación. Sin embargo, las herramientas del SDK para Android no están muy pulidas en el aspecto de maquetación.

Por último, revisando aspectos sobre el comercio electrónico en móviles vemos como la tecnología Bluetooth Low Energy es perfectamente viable para comunicaciones de este tipo. La integración de los móviles con el comercio electrónico es un proceso complejo que tanto las entidades financieras como los fabricantes de dispositivos deben acordar. No se trata de aportar simplemente un nuevo mecanismo de pago, sino que sea seguro y que el usuario se sienta cómodo con él. Hemos visto algunos problemas de seguridad que pueden darse en la tecnología actual, como los ataques del hombre en medio, las escuchas pasivas o la suplantación de identidad por una mala implementación de los mecanismos de autenticación. Actualmente hay diferentes alternativas (uso de NFC en móviles, cobros usando el móvil como TPV) e incluso nosotros hemos aportado la incorporación de la tarjeta eSignus, pero de momento no hay ninguna forma efectiva de saber cuál de ellas predominará en los próximos años. Solo queda esperar cómo las entidades financieras y los fabricantes se van adaptando a estos cambios tecnológicos en el terreno del comercio electrónico desde el móvil.

9.1. Trabajo futuro

Actualmente este proyecto deja abiertas varias puertas. De cara a futuros mantenimientos se nos ocurren varias ideas que, pese a no ser implementadas principalmente por falta de tiempo, quedarán reflejadas en este documento por si alguien quisiera continuar con nuestro trabajo.

En primer lugar encontramos fundamental una mejora de la API Bluetooth LE debido principalmente al problema de la negociación de parámetros entre los dispositivos. Esta mejora podría realizarse de dos formas distintas: implementando una API propia para la tarjeta eSignus o mejorando la de Motorola.

La posibilidad de implementar una API propia trae una ventaja importante y es la posibilidad de personalizarla para las comunicaciones con la tarjeta eSignus. En este caso, todo estaría optimizado al detalle para que su rendimiento fuera el mejor ya que se podrían hacer pruebas incluso a nivel de comunicaciones (usando un sniffer Bluetooth, por ejemplo). Sin embargo, sería necesario un esfuerzo adicional previo a su desarrollo ya que requeriría un conocimiento avanzado tanto del protocolo Bluetooth como de la implementación Java para Android de un protocolo de estas características. Por tanto, necesitaríamos conocer bastantes detalles de bajo nivel del dispositivo Bluetooth del móvil; incluso deberíamos velar por la compatibilidad con futuros móviles que incorporen Bluetooth LE.

Por otro lado, la mejora de la API de Motorola podría no ser posible. Motorola propor-

ciona al desarrollador los ficheros compilados y listos para usar como librerías en nuestros proyectos. Si intentáramos alguna tarea de decompilación podríamos obtener los ficheros fuente originales, pero estos podrían mostrarse parcialmente, o con caracteres fuera del alfabeto. Es por ello que estas mejoras sería conveniente dejarlas en manos de los autores originales, en este caso Motorola.

Otro de los aspectos que deberían tratarse en posteriores iteraciones del desarrollo es la integración real con un banco. En este trabajo el entorno de comunicación con el banco (autenticación, validación, envío y recepción de la firma) ha sido simulado mediante temporizadores que daban al usuario la sensación de estar trabajando con un banco real. El protocolo de comunicación entre la aplicación móvil y el banco debería ser al menos tan rápido como el clásico que usan los datáfonos, ya que en dispositivos móviles buscamos rapidez (ya lo hablábamos con la recepción de datos desde la tarjeta). Por supuesto, no podemos obviar los aspectos de seguridad que hemos venido relatando a lo largo de este trabajo, así que debería ponerse especial énfasis en este aspecto para evitar ataques.

Apéndice A

Manual de la aplicación

A.1. Menú principal

El menú principal de la aplicación nos da acceso a las dos subaplicaciones Transferencia y Datáfono, así como al menú de Preferencias para añadir y borrar tanto cuentas de origen como de destino.

A.2. Cómo hacer una transferencia

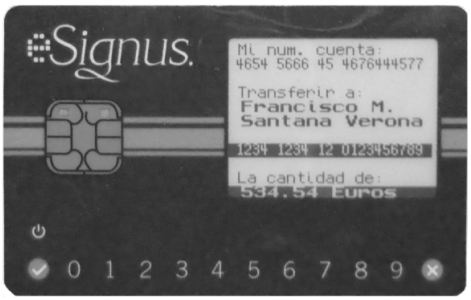
Paso 1. Elegir la opción de Transferencia en el menú principal.

Paso 2. Elegir la cuenta de origen

Paso 3. Elegir la cuenta de destino e introducir el importe

Paso 4. Confirmar la transferencia

Paso 5. Elegir la tarjeta con la que se desea firmar. NOTA: el dispositivo debe haber sido emparejado previamente con el teléfono.



Paso 6. Aceptar la firma en la propia tarjeta.

Paso 7. Esperar la recepción de la firma en el móvil y la confirmación de la misma por parte del banco.

A.3. Cómo usar el datáfono

Paso 1. Elegir la opción de Datáfono en el menú principal.

Paso 2. Introducir la cantidad.

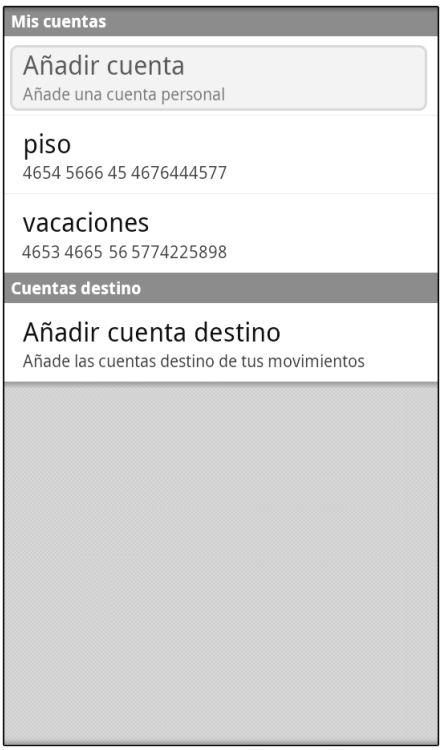
Paso 3. Elegir la tarjeta con la que se desea firmar. NOTA: el dispositivo debe haber sido emparejado previamente con el teléfono.

Paso 4. Aceptar la firma en la propia tarjeta.

Paso 5. Esperar a que se reciba la firma en el móvil.

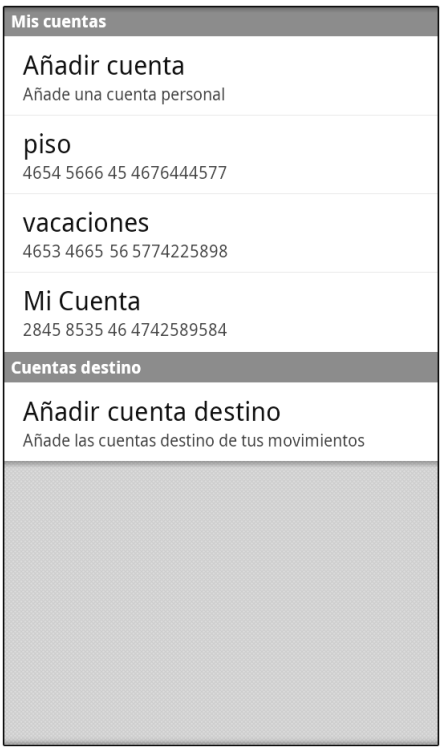
A.4. Cómo añadir una cuenta de origen

Paso 1. Seleccionamos la opción de Configuración del menú principal.



Paso 2. Pulsamos sobre "Añadir cuenta" dentro de la sección "Mis cuentas".

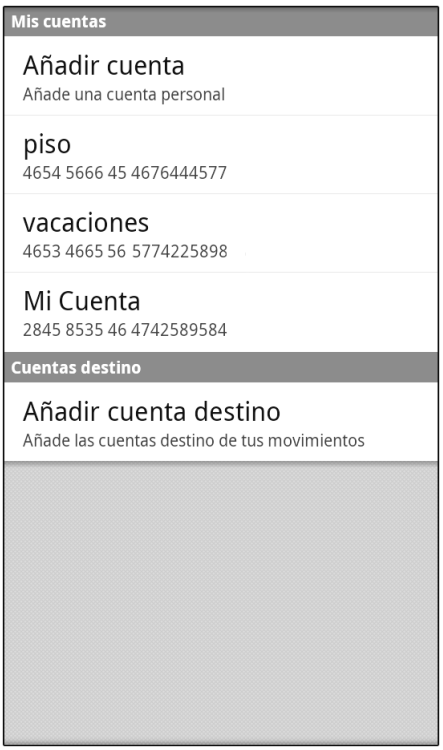
Paso 3. Rellenamos el formulario con los datos de nuestra cuenta bancaria.



Paso 4. Al pulsar Aceptar, la cuenta queda añadida directamente.

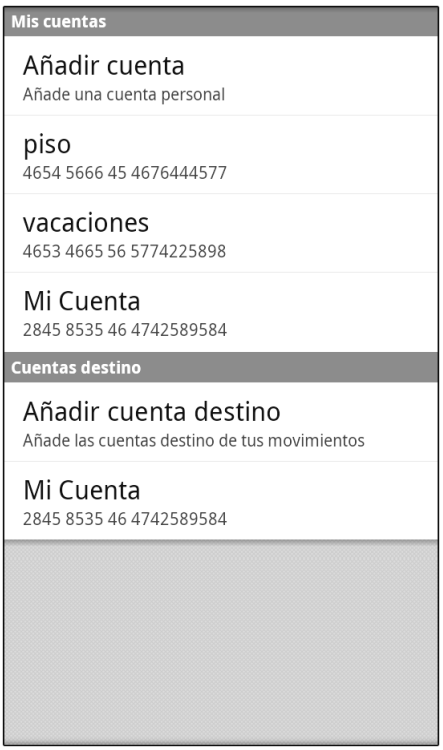
A.5. Cómo añadir una cuenta de destino

Paso 1. Seleccionamos la opción de Configuración del menú principal.



Paso 2. Pulsamos sobre "Añadir cuenta" dentro de la sección Cuentas destino".

Paso 3. Rellenamos el formulario con los datos de nuestra cuenta bancaria.



Paso 4. Al pulsar Aceptar, la cuenta queda añadida directamente.

Bibliografía

[1] Android Dashboard Design Tutorial. http://www.androidhive.info/2011/12/android-dashboard-design-tutorial/

[2] Android Development Tutorial by Lars Vogel. http://www.vogella.com/articles/Android/article.html

[3] Black Hat: Square Credit-Card Reader Hacked!. http://www.pcmag.com/article2/0,2817,2390491,00.asp

[4] Bluetooth Smart Devices. http://www.bluetooth.com/Pages/Bluetooth-Smart-Devices.aspx

[5] eSignus. http://www.e-signus.com/

[6] Foro oficial de desarrollo Android en Motorola. http://community.developer.motorola.com/t5/Android-App-Development-for/bd-p/Android_Development

[7] Google. *Android UI Design Patterns*. Fulcher R. et all. 2010.

[8] INELCAN S.L., Las Palmas de Gran Canaria. 2010. *External signature device with wireless communication capacity*. Fernando de la Puente Arrate et al. España. US 2010/0287376 A1.

[9] Microsoft. ¿Que es un dispositivo de confianza?. http://windows.microsoft.com/es-ES/windows-vista/What-is-a-trusted-device

[10] Motorola Bluetooth Low Energy API. http://developer.motorola.com/docs/bluetooth-low-energy-api/

[11] Motorola Bluetooth Low Energy GATT Framework API. http://developer.motorola.com/docs/bluetooth-low-energy-gatt-framework-api/

[12] Official Android Developers. http://developer.android.com/index.html

[13] Open Bluetooth Low Energy SDK for Android. http://code.google.com/p/broadcomble/

[14] TEXAS INSTRUMENT. *Texas Instruments CC2540 Bluetooth® Low Energy Software Developer's Guide v1.1.*