Facultad
De
Ciencias

Seguridad utilizando dispositivos NFC

Security using NFC devices

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor : Fidel Abascal López
Director : Domingo Gómez Pérez
Junio - 2016

Índice general

Índice de figuras

Resumen

Hace décadas comunicarse mediante un dispositivo que estuviera conectado a una red cableada y tras mucha espera resultaba algo fantástico. Hoy en día tenemos la posibilidad de realizar un gesto en cualquier lugar y ponernos en contacto con alguien a cientos o miles de kilómetros. En este sentido, las comunicaciones han evolucionado de una forma increíble. La seguridad en estas comunicaciones basada en la criptografía es un elemento primordial para salvaguardar la privacidad de los usuarios y la del contenido.

La criptografía no se ha quedado atrás y durante el último siglo su devenir ha seguido el mismo camino. Desde los métodos más primitivos basados en cambiar una letra por la anexa; hasta complejos sistemas criptológicos (criptosistemas) que aprovechan ciertas propiedades matemáticas para preservar niveles de seguridad elevados con la necesidad de menos recursos (computacionales y de almacenamiento). Optimizar los recursos es esencial para ser competitivo y para ello la metodología de criptografía basada en curvas elípticas reduce exponencialmente la cantidad de almacenamiento necesaria respecto a otros algoritmos. De la mano va la tecnología NFC (*Near Field Comunication*), la cual ha simplificado los dispositivos de comunicación a algo tan pequeño y barato que ha conquistado el planeta en forma de multitud de aplicaciones.

En este Trabajo Fin de Grado (TFG a partir de ahora), comprenderemos la posibilidad de elaborar sistemas seguros con dispositivos de comunicación de bajo coste y criptología avanzada. A su vez, se implementará una aplicación para *smartphones* que, gracias a algorítmos avanzados de criptografía, permitirán a un usuario crear y utilizar un *tag* NFC como dispositivo de autenticación de alta seguridad en un sistema ficticio.

Palabras clave: *Criptografía, Curvas elípticas, NFC, autenticación.*

Abstract

Hace décadas comunicarse mediante un dispositivo que estuviera conectado a una red cableada y tras mucha espera resultaba algo fantástico. Hoy en día tenemos la posibilidad de realizar un gesto en cualquier lugar y ponernos en contacto con alguien a cientos o miles de kilómetros. En este sentido, las comunicaciones han evolucionado de una forma increíble. La seguridad en estas comunicaciones basada en la criptografía es un elemento primordial para salvaguardar la privacidad de los usuarios y la del contenido.

La criptografía no se ha quedado atrás y durante el último siglo su devenir ha seguido el mismo camino. Desde los métodos más primitivos basados en cambiar una letra por la anexa; hasta complejos sistemas criptológicos (criptosistemas) que aprovechan ciertas propiedades matemáticas para preservar niveles de seguridad elevados con la necesidad de menos recursos (computacionales y de almacenamiento). Optimizar los recursos es esencial para ser competitivo y para ello la metodología de criptografía basada en curvas elípticas reduce exponencialmente la cantidad de almacenamiento necesaria respecto a otros algoritmos. De la mano va la tecnología NFC (*Near Field Comunication*), la cual ha simplificado los dispositivos de comunicación a algo tan pequeño y barato que ha conquistado el planeta en forma de multitud de aplicaciones.

En este Trabajo Fin de Grado (TFG a partir de ahora), comprenderemos la posibilidad de elaborar sistemas seguros con dispositivos de comunicación de bajo coste y criptología avanzada. A su vez, se implementará una aplicación para *smartphones* que, gracias a algorítmos avanzados de criptografía, permitirán a un usuario crear y utilizar un *tag* NFC como dispositivo de autenticación de alta seguridad en un sistema ficticio.

Keywords: *Cryptography, elliptic curves, NFC, authentication.*

1 Introducción

Cuando una persona envía un mensaje a un destinatario con información que considera comprometida o personal, pretende realizarlo con el mínimo riesgo de que dicho mensaje llegue de forma alterada y que sea al destinatario indicado. Además, éste quiere que el canal sea seguro y que nadie más intercepte la información; y si ocurriese tal caso, que personas ajenas no sean capaces de interpretar el mensaje y usarlo en su contra de forma perjudicial -o simplemente no desea difundir la información a alguien que no sea el destinatario-. También es evidente la necesidad de sistemas que eviten de la mejor forma posible la suplantación de usuarios; por ello, la autentificación apoyada en la criptografía resulta indespensable.

Existe la consideración generalizada que aquellos elementos más seguros a la hora de identificar y autenticar a un usuario de un sistema son aquellos que implican el uso de parámetros biométricos. Por ejemplo, el algoritmo para el reconocimiento del iris patentado por el investigador de la universidad de Cambridge, John Daugman, se basa en el iris como elemento único e intransferible para cada persona [21]. De forma análoga se utilizan algoritmos basados en la estructura facial o huellas dactilares. Por otra parte, Manuel Lucena López, doctor en informática de la universidad de Jaén, asegura en su publicación [17] que esta clase de requerimientos biométricos se pueden reducir a problemas de autenticación basada en dispositivos. Es decir, una tarjeta puede actuar con el mismo compromiso de seguridad que dichos elementos biológicos.

Al igual, en los sistemas criptográficos (criptosistemas), la implementación más segura suele ser la menos eficiente. En la actualidad, la competencia hace de la optimización un objetivo en el que se invierten ingentes cantidades de recursos. En el campo de la automoción competitiva, un incremento de la velocidad punta de 3km/h puede implicar reducir el tiempo de un competidor en unas décimas vitales que podrían suponer la diferencia entre ser primero o ser segundo. La criptología y la seguridad de comunicaciones también están afectadas por este hecho y no están exentas de la voracidad por optimizar el trinomio de recursos, tiempo y resultados. Dentro de este contexto una tecnología tan

asequible como NFC (*Near Field Comunication*), - dispositivos para la comunicación de información empleando radiofrecuencia de corto alcance- que ha conquistado numerosos ámbitos, será el soporte de estudio de este Trabajo de Fin de Grado (TFG a partir de ahora) como dispositivo de autenticación.

Como suele ser habitual, un dispositivo barato tiene ciertas desventajas. Respecto a la seguridad criptográfica con NFC el mayor inconveniente suele ser el tamaño de almacenamiento - inferior a 500 bits generalmente-. Este problema implica utilizar criptosistemas que puedan trabajar con tamaños reducidos de información sin comprometer la seguridad. La evolución en la criptografía es considerable desde que en la Segunda Guerra Mundial el proyecto ULTRA tratara de descifrar los mensajes del ejército alemán; quienes se encontraban a la vanguardia de la criptografía.

Con el paso del tiempo, se avanzó hacia criptosistemas más complejos. En 1985 Victor Miller y Neal Koblitz propusieron la utilización de curvas elípticas para criptología. Su estructura algebraica es notoriamente más compleja que la mayoría de los criptosistemas anteriores a ésta. Sin embargo su implementación la posiciona entre las más eficientes y capaces de conseguir claves más cortas con el mismo nivel de seguridad [17]. Por ello, se ha implementado esta tecnología en multitud de escenarios: desde ciertas tarjetas de crédito hasta el cifrado de *Whatsapp* [24] trabajan con las propiedades de la criptografía con curvas elípticas.

Finalmente, se verá la potencia de estos sistemas para que, utilizando chips NFC de capacidad reducida, se puede generar un sistema de autenticación seguro aplicable a en escenario real.

2

APP: Autenticación en un sistema ficticio

2.1. Introducción

Anteriormente se han explicado los conocimientos y términos generales relacionados con la seguridad en la autenticidad NFC basada en criptología con curvas elípticas. A continuación se muestra la aplicación de dichos conocimientos en un proyecto software experimental. El objetivo es demostrar la capacidad de los dispositivos NFC para, de la mano de la criptografía con curvas elípticas, llegar a desarrollar un sistema de autenticación óptimo y fiable.

Tanto los nombres, librerías, herramientas, así como el resto del material utilizado se describirán a continuación. A su vez, siguiendo un estándar de desarrollo software basado en metodologías ágiles, se mostrará la utilizada para éste proyecto. Para ello, el autor y director de este TFG (Fidel Abascal y Domingo Gómez) hemos actuado y ejercido tanto de cliente como de contratado para el desarrollo de la aplicación.

Dentro de un ámbito ficticio, se plantea una empresa llamada **Alpha - Consultora S.A.**[*], nueva potencia local dentro del campo de la seguridad bancaria, que ha cosechado unos excelentes resultados a lo largo de sus 2 años de existencia. Cuenta con más de 40 trabajadores y su crecimiento y expansión es notoria. Tanto es el éxito de esta compañía que, para dar cabida a su plantilla, ha decidido trasladarse a una nueva sede más moderna, amplia y mejor ubicada. La empresa, antes de instalarse en la nueva sede, decide contratar a unos expertos en seguridad para gestionar el control de accesos mediante un sistema de tarjetas y lectores en las entradas; teniendo en cuenta una inversión mínima pero garantizando un alto grado de seguridad.

Tras buscar incesantemente recurren a la empresa **F-NFC**. Una vez realizado el estudio por parte de F-NFC, se pone en consonancia un acuerdo para elaborar una aplicación

[*] Cualquier similitud con la realidad es mera coincidencia

que genere información que autentifique a un usuario de la empresa *Alpha* y sea reconocido de forma unívoca para permitir su acceso a la sede. La infraestructura de la empresa propietaria de la nueva sede corresponde a la figura 2.1.

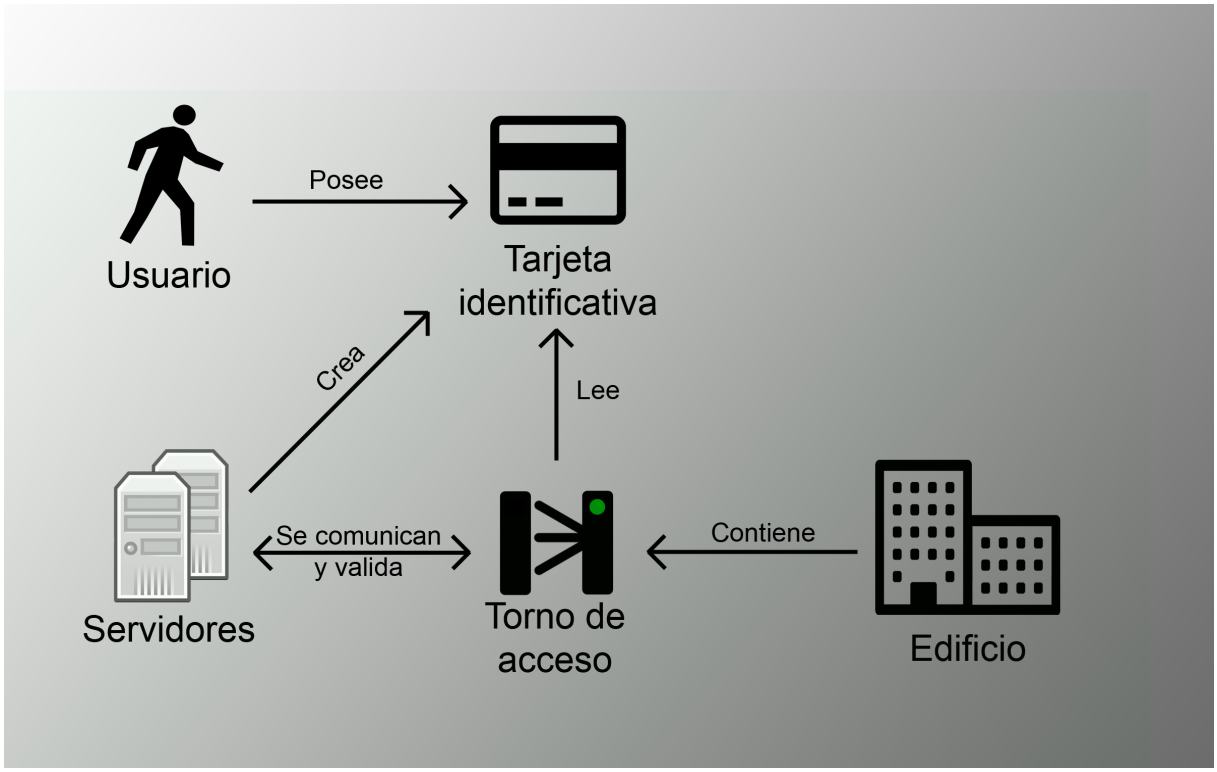

Figura 2.1: Infraestructura del sistema ficticio.

Los detalles de implementación de la aplicación seguirán un objetivo didáctico y experimental que cumplirán los requerimientos de la situación propuesta; adecuándose a la carencia real de la infraestructura anteriormente mencionada. Se explicará en las secciones siguientes en detalle todos los puntos implicados en la consecución de este objetivo. Más concretamente en el apartado de escenario de la aplicación 2.4.

2.2. Materiales y tecnologías utilizadas

Para el desarrollo del proyecto se ha utilizado, ante todo, software de carácter libre junto a imágenes, textos, estructuras y contenido sin restricciones de uso. Ya sea debido al tipo de licencia de cada elemento o por ser de autoría propia.

El desarrollo se ha realizado en el lenguaje de programación *Java* con el kit de desarrollo *Java* (JDK) 1.8.65 (*Java 8 update 65*) de *Oracle Corporation* [3].

El entorno de desarrollo integrado o *IDE* de la aplicación es el oficial de Google para el desarrollo para dispositivos *Android*: *Android Studio*, versión 1.5.1 [12]. Este programa provee las herramientas básicas de desarrollo; entre ellas, un administrador de los kit de desarrollo o *SDKs* para las diferentes versiones y varios elementos descritos a continuación:

- Plataformas SDK
  - API 22: Para la versión Android 5.1.1.
- Herramientas SDK
  - Android Build Tools : Para la construcción de la aplicación.
  - Android SDK Platform Tools v23.1: Soporte para el desarrollo.
  - Repositorio de ayuda Android, rev 30.
  - Libreria de ayuda Android, rev 23.2.1 : Ayudas en la retrocompatibilidad de elementos de interfaz de usuario.
  - Google USB Driver, rev 11 : Conexión entre el servicio de ejecución de la aplicación y los dispositivos USB.
  - Intel x86 HAXM, rev 6.0.1 : Aceleración *hardware* para la emulación.

Estos componentes de *Android Studio* hacen posible el desarrollo de la aplicación objetivo. Este programa, con carencias notorias en ciertos apartados, ha hecho complicado, de cierta forma, la selección de componentes a instalar, versiones , etcétera debido a diversos motivos de compatibilidades de librerías y actualizaciones desfasadas entre los componentes y el propio IDE.

La esencia de la aplicación reside en la criptografía mediante curvas elípticas. Para la implementación de ello ha sido necesario utilizar una librería externa llamada *Bouncy Castle* [19] versión 1.54. Dicha librería suple las deficiencias de la implementación base de la propia API (*Application Programming Interface*) de *Java* en su paquete de *java.security* [2]. La cual tiene limitaciones claras a la hora de la generar curvas elípticas. Gracias a ésta librería se ha podido realizar la parte crítica de la aplicación con una mejora de rendimiento notoria si se tuviera de haber utilizado la API de *Java*.

Respecto al almacenaje de datos se ha optado por utilizar una pequeña base de datos en *SQLite* [22] descrita más adelante. Para el almacenamiento de unos pequeños registros que utiliza la aplicación. El uso de una base de datos de mayor capacidad y funcionalidades no se ha contemplado factible.

En cuanto a los elementos gráficos de la aplicación, un alto porcentaje son de elaboración propia mediante programas de edición de imágenes. El resto son de libre uso comercial. La iconografía de la aplicación es autoría de *Google Inc.* [14]. Dichos iconos han de ser vectoriales debido a la optimización del tratamiento de imágenes y su renderizado, por lo que en este aspecto, *Google* provee estos iconos en formato *SVG* (*Scalable Vector Graphics*) y *PNG*; también *Android Studio* contiene iconografía de forma nativa pero no actualizada. A la hora de incluir estos elementos externos se ha transformado las descripciones vectoriales *SVG* en formato *XML* (*eXtensible Markup Language*) interpretable de forma sencilla por *Android Studio* y fácilmente modificables en los casos que se ha requerido. En el apartado 2.6 de diseño y aspecto de la aplicación se comenta en detalle el resto de la disposición, motivación y elaboración gráfica de la aplicación.

La API de objetivo del proyecto *Android* ha sido la número 22. Desarrollada para la versión 5.1. (*LOLLIPOP MR1*). Se ha decidido utilizar esta esta API debido a las mejoras sustanciales en cuanto al trabajo del adaptador NFC implementadas desde la versión 5.0 [10] y mejoradas en ésta [11].

Para el testado de la aplicación *Android Studio* se dispone de la tecnología *AVD* para la virtualización de dispositivos *Android*. Sin embargo, el rendimiento es pobre en comparación con el testeo y *debug* en un dispositivo físico. Por lo que se ha utilizado un dispositivo *Android smartphone One Plus One* de la compañía americana *Never Settle*, el cuál dispone de la versión *Android* 5.1.1 y *Cyanogen OS* 12.1.1 en el momento de la elaboración de la aplicación.

Figura 2.2: Smartphone One Plus One - Never Settle

Por último, el proyecto ha necesitado de dos elementos físicos principales: el dispositivo *Android* mencionado anteriormente y de tarjetas NFC. Debido a la carencia de un presupuesto, se ha optado por utilizar etiquetas adesibles (desde ahora NFC-T) de bajo coste. Se trata del chip *NTAG213* que siguen el estándar ISO 14443-3 [15]. Las características de estas etiquetas son las siguientes:

- Tipo de etiqueta : ISO 14443-3A.
- Descripción : NXP MIFARE Ultralight (Ultralight C) - NTAG213.
- Tecnología : NfcA, Ndef, MifareUltralight.
- Formato de datos: NFC Forum Type 2.
- Diámetro: 25 mm.
- Identificadores del chip.
    - Valor ATQA: 0x0044.

- Valor SAK: 0x00.
- Firma: NXP Public Key.

▪ Tamaños y capacidad.

- Memoria: 45 páginas de 4bytes por página (180 bytes).
- Tamaño: 137 bytes.
- *UID* (Identificador del contenido): 7 bytes [6].
  - *Byte 7*: Valor *UTF-8*, codificación.
  - *Byte 6*: Valor *0*, reservado para uso futuro.
  - *Byte 5-0*: Tamaño del código del lenguaje *IANA*.
- Tamaño utilizable: 130 bytes (Tamaño menos *UID*).

Estas etiquetas se pueden adherir en una superficie que le haga de soporte. Por ejemplo, se podría plastificar junto a dos tapas que cubran el chip y tener la apariencia de una tarjeta común. En la figura 2.3 se muestra una de estas etiquetas.

Figura 2.3: NFC Tag - NTAG213

Hubiera sido preferible utilizar chips *MIFARE* [18] de alta capacidad y mayores funcionalidades como los *MIFARE classic* 1K y 4K del productor *NXP*. Estos chips están implementados en una enorme cantidad de aplicaciones en todo el mundo, un ejemplo claro de su uso son en las tarjetas de transporte o hasta en las tarjetas de crédito y la mayoría de chips NFC actuales que impliquen la necesidad de encriptación (RSA comúnmente) y seguridad. En estas tarjetas es realmente complicado acceder a su contenido ya que se precisan ciertas claves para la lectura y decodificación. Sin embargo, en las utilizadas en este proyecto se escribe en texto plano (siguiendo el objetivo didáctico); en el apartado de seguridad y criptología 2.7 se explica cómo afectaría esta carencia al sistema final.

Todos los elementos descritos en este apartado conforman lo necesario para simular la estructura descrita en la figura 2.1 y la implementada definida en el apartado 2.4.

2.3. Metodología

Se ha decidido implementar una metodología de desarrollo software basada en iteracciones de funcionalidades de forma incremental. Para una planificación estructurada se han programado los hitos y la consecución de tareas a completar en cada hito de validación.

Gracias a la herramienta Gantt [23] para la elaboración de diagramas de planificación de proyectos, se han propuesto gráficamente la duración de cada tarea valorando su dificultad y las posibles modificaciones que hubieran podido surgir de las mismas en cada hito de validación. La duración de cada tarea implica los días necesarios para su finalización; sin tener relación con las horas laborales ya que, aunque no venga reflejado, se ha realizado trabajo en días no laborables. En las figuras 2.4, 2.5 y 2.6 se observa la organización inicial planteada. Cada apartado principal de la aplicación precedida de un hito se considera una iteracción incremental del proyecto para adecuarse a la metodología planteada.

Nombre	Fecha de inicio	Fecha de fin
Base	23/02/16	29/02/16
Entorno	23/02/16	26/02/16
Estructura	29/02/16	29/02/16
Componentes iniciales	29/02/16	29/02/16
Diseño	1/03/16	2/03/16
Logo	1/03/16	1/03/16
Paleta de colores	2/03/16	2/03/16
Estilo	2/03/16	2/03/16
Validación	3/03/16	3/03/16
BBDD	3/03/16	4/03/16
Definición	3/03/16	3/03/16
Valores predeterminados	3/03/16	4/03/16
Validación	7/03/16	7/03/16
Portada	7/03/16	7/03/16
Menú lateral	8/03/16	9/03/16
Acerca de	10/03/16	10/03/16
Validación	11/03/16	11/03/16
Administración	11/03/16	7/04/16
Base de Datos	11/03/16	14/03/16
Sistema	15/03/16	28/03/16
Validación	29/03/16	29/03/16
Nuevo Usuario	29/03/16	4/04/16
Listado de usuarios	5/04/16	7/04/16
Validación	8/04/16	8/04/16
Verificación NFC	8/04/16	21/04/16
Validación	22/04/16	22/04/16
Pruebas	22/04/16	27/04/16
Despliegue	28/04/16	29/04/16

Figura 2.4: Diagrama Gantt - Subdivisión de elementos de la aplicación para su desarrollo

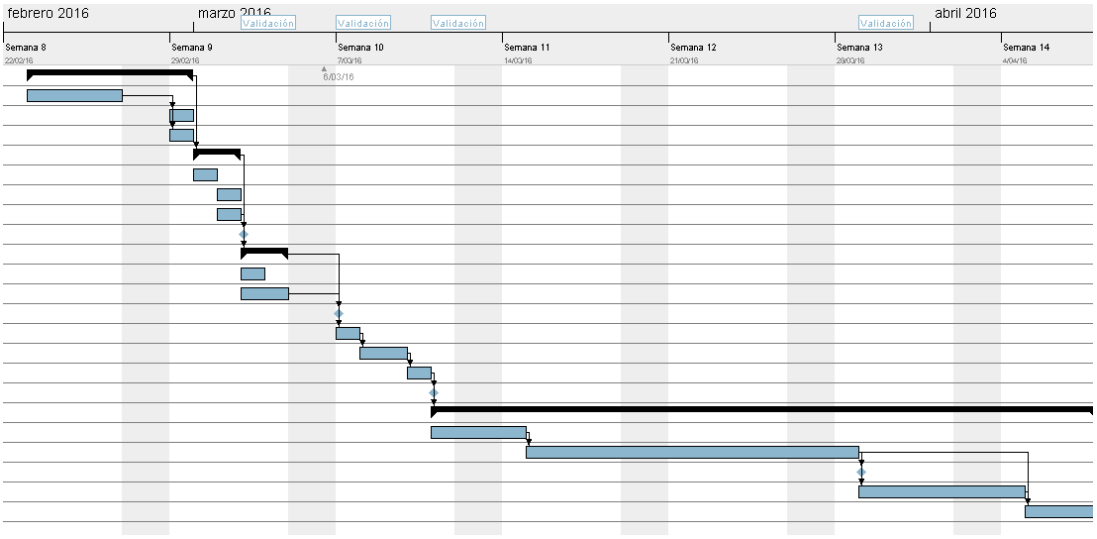

Figura 2.5: Diagrama Gantt : Visualización Parte I

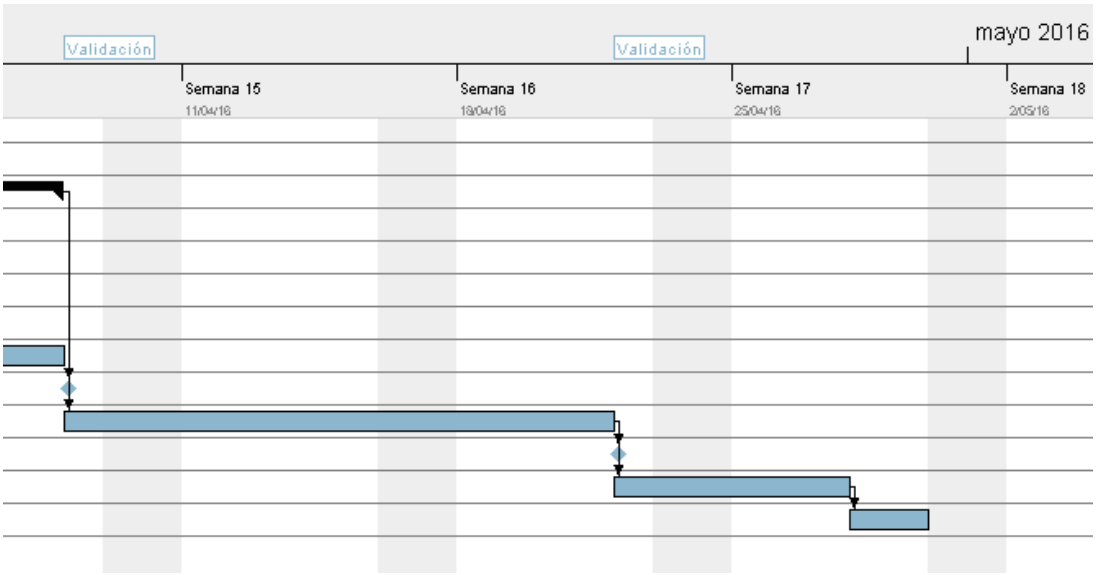

Figura 2.6: Diagrama Gantt : Visualización Parte II

Como se puede observar, la duración final del proyecto ha sido de 17 semanas aproximadamente. Con pequeñas variaciones, los hitos se han ido finalizando satisfactoriamente hasta comprobar que todas las tareas y partes del proyecto se han completado correctamente. A esta planificación habría que añadir el tiempo previo necesario para el estudio y preparación para el uso de las herramientas y la tecnología utilizada.

2.4. Escenario

La idea principal del escenario de la aplicación se ha comentado en el apartado 2.1, en donde se comenta que el objetivo de la aplicación es meramente didáctico y experimental. Alejándose de los modelos pragmáticos del desarrollo profesional.

La complejidad añadida de implementar un sistema cercano a la realidad aplicaría un coste elevado de recursos sin tener relevancia notoria en la finalidad comentada. Elementos tales como: servidores, *WebServices*, *frameworks* de desarrollo, certificación, optimización, etcétera. Dado que la esencia de este TFG se centra en el elemento teórico y didáctico de la criptología en curvas elípticas, la creación de una aplicación de carácter altamente profesional le añadiría beneficios nimios, los cuales no compensan la complejidad agregada. Por tanto, se han realizado labores de simulación y descarte de las partes que se han considerado prescindibles.

La arquitectura de la aplicación representada en la figura 2.1 muestra un escenario típico que da pie a la implementación de este tipo de seguridad. Dentro de este proyecto virtual se dispone de un edificio el cuál posee tornos o puertas de acceso. Estas puertas son capaces de leer el contenido de las tarjetas NFC de cada usuario del sistema. El contenido de las tarjetas se transmite a los servidores de validación de la empresa (sin la necesidad de que se encuentren físicamente dentro del mismo edificio). Los servidores validarán la información recogida en el torno de acceso y validará o no el contenido. Si resulta validado el torno recogerá de los servidores la nueva información a escribir en la tarjeta para hacer válida el paso la próxima vez. Finalmente, el usuario por medio de ésta tarjeta unipersonal podrá acceder al edificio autenticándose en la entrada.

Descrito el escenario virtual, es necesario comentar que éste proyecto se ha realizado. Contando con los elementos del apartado 2.2 se agrupan las funcionalidades del escenario virtual en el dispositivo *Android*. Por lo tanto, no hay conexiones a elementos externos y toda la estructura se compone únicamente del dispositivo *Android* y las tarjetas NFC. A su vez, cuenta con las siguientes funciones que simulan las labores del escenario virtual:

- Información sobre los usuarios del sistema: En lugar de contar con un acceso a una base de datos que recoja la información, se utiliza una pequeña base de datos dentro del dispositivo que contiene la información básica de los usuarios (identificador y nombre del usuario).
- Información del sistema de encriptación: El dispositivo también contiene la información básica del criptosistema implementado (labor de información de servidores).
- Validación: Tras leer el contenido de la tarjeta NFC, denegará o validará la información obtenida (labor de validación y acceso).
- Funciones de administración del sistema: El dispositivo es capaz de restaurar los valores por defecto del sistema, junto a la información inicial. También permite crear nuevas definiciones del sistema de seguridad. Por último, también asignará usuarios al sistema de seguridad que no se encuentren dentro (labor de creación de tarjetas).

Todos los elementos de información se describen en el apartado 2.8 sobre la base de datos de la aplicación; y las funcionalidades del sistema dentro del apartado 2.5 de requisitos y funcionalidades.

2.5. Requisitos y funcionalidades

La toma de requisitos, especificaciones y funcionalidades de la aplicación real, que cumple con los simulados en el contexto virtual descrito en la introducción 2.1, se ha llevado a cabo mediante constantes reuniones entre las partes implicadas (dentro del escenario real, véase el apartado 2.4).

Siguiendo la metodología ágil basada en iteracciones comentada en el apartado 2.3, desde el primer momento se ha ido mostrando el avance del proyecto al supuesto cliente. Éste ha aceptado según sus requerimientos iniciales o ha concretado cambios que se adecuen en el marco de las funcionalidades principales. Iteracción a iteracción se han ido realizando los pasos siguiendo hitos a validar. El diagrama *Gantt* mostrado en la figura 2.4 muestra en detalle la división de las tareas y apartados del proyecto realizado en este TFG.

Cada una de las tareas a realizar han seguido el objetivo de los casos de uso referidos a las funcionalidades de la aplicación. Los casos de uso de la aplicación se muestran en la figura 2.7, realizada mediante la herramienta *online* gratuita para la creación de diagramas UML de este tipo: yUML [20].

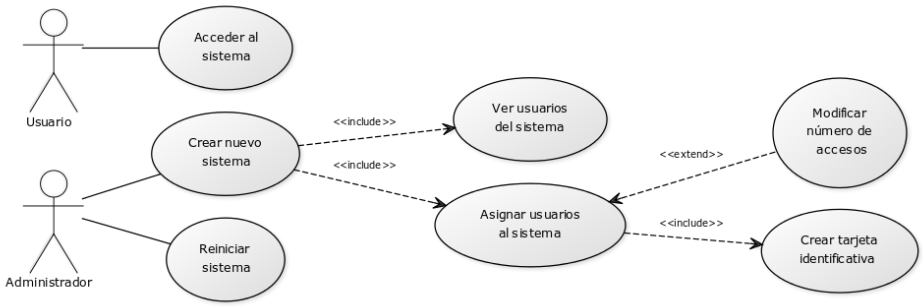

Figura 2.7: Casos de uso

Los casos de uso, cumpliendo las funcionalidades acordadas con el cliente, se corresponden con los requisitos funcionales de la aplicación y la infraestructura de la aplicación referida en la figura 2.1:

- Usuario
  - RF01 - Un usuario, supuesto trabajador de la empresa, poseedor de una tarjeta de identificación proveída por la administración del sistema, es capaz de autenticarse.
- Administrador
  - RF02 - La administración del sistema define las características del criptosistema que utiliza la aplicación.
  - RF03 - Puede restaurar la información a los valores por defecto.
  - RF04 - Puede ver los usuarios pertenecientes al sistema.

- RF05 - Es capaz de asignar usuarios al sistema creando las tarjetas identificadores.
- RF06 - Valida la información recogida de cada tarjeta y sobrescribe con la nueva información acorde al criptosistema empleado.

Como requisitos no funcionales se definen principalmente los siguientes:

- Plataforma: Se utiliza un dispositivo *Android* que cumpla las funciones de administración y validación.
- Criptosistema: Compuesto por criptografía con curvas elípticas de alta seguridad y el sistema *SKEY* para validaciones de un solo uso (*véase apartado sobre 2.7 la seguridad de la aplicación*). La plataforma utilizada puede modificar la curva elíptica implementada por el criptosistema.
- Elementos identificativos: Las tarjetas NFC contendrán la información asignada por el criptosistema. La información se escribirá a la hora e la asignación del usuario al sitema o en el momento de una validación correcta.

Las definiciones en profundidad de estos casos de uso se muestran a continuación.

Caso de uso	**Acceder al sistema**
Actor principal:	Usuario/Empleado
Stakeholders e interesados:	▪ Usuario/Empleado: Validar su identidad para acceder a las instalaciones de la empresa. ▪ Empresa: Autenticar inequívocamente a quienes acceden a emplazamiento de la empresa.
Precondiciones:	▪ El usuario dispone de una tarjeta NFC creada por el sistema. ▪ La aplicación se encuentra instalada en el dispositivo *Android*. ▪ La aplicación posee un sistema de seguridad desplegado. ▪ El dispositivo cuenta con la tecnología NFC y en estado activo.

Postcondiciones:	▪ La aplicación valida el contenido de la tarjeta NFC e informa al usuario del resultado. ▪ La aplicación reescribe la tarjeta NFC preparándola para su siguiente uso.

Escenario principal de éxito:

1 Se inicia el dispositivo.

2 Se aproxima la tarjeta NFC al terminal para la lectura del contenido.

3 La aplicación muestra un mensaje de validación confirmada y escribe el contenido oportuno para la próxima ocasión.

Extensiones:

1 Se inicia la aplicación:

1. Se despliega el menú
2. Se selecciona la opción Verificar NFC

2 Se aproxima una tarjeta NFC que no es del tipo correcto o el contenido es vacío:

1. La aplicación muestra un mensaje de aviso
2. Vuelta al paso 1 del escenario principal.

3 Datos de la tarjeta NFC incorrectos:

1. El sistema muestra el de validación incorrecta.
2. Vuelta al paso 1 del escenario principal.

Frecuencia de ocurrencia:	Cada vez que un usuario del sistema desee autenticarse mediante su tarjeta NFC.

Caso de uso	**Crear nuevo sistema**
Actor principal:	Administrador
Stakeholders e interesados:	▪ Administrador: Definir un nuevo sistema de seguridad suplantando al actual.

Precondiciones:

- La aplicación se encuentra instalada en el dispositivo *Android*.
- La aplicación posee un sistema de seguridad desplegado.

Postcondiciones:

- La aplicación implementa una nueva definición del sistema utilizado. Cambiando las características de la curva elíptica a emplear en el futuro.
- No hay nuevos usuarios asignados en el nuevo sistema.

Escenario principal de éxito:

1 Se inicia el dispositivo.

2 Se inicia la aplicación.

3 Se abre el menú lateral.

4 Se selecciona la opción Despliegue.

5 Se selecciona la definición de una curva elíptica utilizando el desplegable.

6 Se pulsa en crear.

7 Se muestra un cuadro de confirmación.

8 Tras aceptar, se prepara el sistema para asignar nuevos usuarios con la nueva definición.

Extensiones:

4 Se pasa directamente al punto 6 del escenario principal:

1. La definición por elegida es la listada por defecto.

7 Pulsar en Cancelar:

1. Se cierra el cuadro de confirmación.
2. Se vuelve al punto 4 del escenario principal.

8 Error en la creación del sistema:

1. Se muestra un mensaje de error
2. Vuelta al paso 4 del escenario principal.

Frecuencia de ocurrencia:	Cada vez que se desee reiniciar los datos de la aplicación con una nueva definición de seguridad.

Caso de uso	**Reiniciar sistema**
Actor principal:	Administrador
Stakeholders e interesados:	▪ Desarrollador: Revertir todos los cambios respecto a la definición de ejemplo.
Precondiciones:	▪ La aplicación se encuentra instalada en el dispositivo *Android*. ▪ La aplicación posee un sistema desplegado.
Postcondiciones:	▪ La aplicación recupera la implementación del sistema de ejemplo.

Escenario principal de éxito:

1 Se inicia el dispositivo.

2 Se inicia la aplicación.

3 Se abre el menú lateral.

4 Se selecciona la opción Despliegue.

5 Se selecciona la pestaña de Base de datos.

6 Se pulsa en el botón Restaurar Base de Datos.

7 Se muestra un cuadro de confirmación.

8 Tras aceptar, se recupera el sistema de ejemplo inicial.

Extensiones:

7 Pulsar en Cancelar:

1. Se cierra el cuadro de confirmación.
2. Se vuelve al punto 5 del escenario principal.

8 Error en la restauración del sistema:

1. Se muestra un mensaje de error
2. Vuelta al paso 5 del escenario principal.

Frecuencia de ocurrencia:	En cualquier momento que se desee.

Caso de uso	**Ver usuarios del sistema**
Actor principal:	Administrador
Stakeholders e interesados:	▪ Administrador: Listar la información de los usuarios del sistema.
Precondiciones:	▪ La aplicación se encuentra instalada en el dispositivo *Android*. ▪ La aplicación posee un sistema desplegado.
Postcondiciones:	▪ La aplicación muestra los usuarios activos del sistema .

Escenario principal de éxito:

1 Se inicia el dispositivo.

2 Se inicia la aplicación.

3 Abrir el menú lateral.

4 Seleccionar la opción Ver Usuarios.

5 Se lista la tabla con la información de seguridad de los usuarios activos en el sistema.

Extensiones:

4 No hay usuarios activos en el sistema:

1. Se muestra un mensaje indicándolo

Frecuencia de ocurrencia:	En cualquier momento que se desee.

Caso de uso	**Asignar usuarios al sistema**
Actor principal:	Administrador
Stakeholders e interesados:	▪ Administrador: Crear (escribir) tarjetas NFC para autenticar dentro del sistema generado a un empleado de la empresa. ▪ Usuario/Empleado: Obtener una tarjeta NFC con el contenido adecuado.

Precondiciones:

- La aplicación se encuentra instalada en el dispositivo *Android*.
- La aplicación posee un sistema desplegado.
- El dispositivo cuenta con la tecnología NFC y en estado activo.
- La tarjeta NFC objetivo contiene información por defecto (no está vacía) para evitar fallos de escritura.
- La aplicación dispone de un listado de usuarios generales de la empresa.

Postcondiciones:

- Se vuelca el contenido de autenticación de un usuario recién añadido al sistema a una tarjeta NFC.

Escenario principal de éxito:

1 Se inicia el dispositivo.

2 Se inicia la aplicación.

3 Pulsar en abrir el menú lateral.

4 Seleccionar la opción Nuevo Usuario.

5 Seleccionar un usuario que no esté asignado al sistema actual.

6 Pulsar el botón Continuar.

7 Se muestra un mensaje de confirmación.

8 Un mensaje de de información indica que la aplicación está preparada para escribir en la tarjeta. Se acerca la tarjeta para escribir.

9 La aplicación ha escrito correctamente la información en la tarjeta NFC.

Extensiones:

7 Pulsar en Cancelar:

1. Se cierra el cuadro de confirmación.
2. Se vuelve al punto 5 del escenario principal.

8 Fallo al escribir:

1. Se muestra un mensaje indicando el fallo de escritura.
2. No se añade el usuario al nuevo sistema.
3. Se prepara la escritura nuevamente, paso al punto 8 del escenario principal.

Frecuencia de ocurrencia:	Cuando se requiera asignar un usuario al sistema.

2.6. Diseño y aspecto

La aplicación cuenta con un contenedor principal que ocupa la visión principal del dispositivo. En la parte superior dispone de barra principal de la aplicación que contiene el título de cada apartado y un acceso al menú. El menú se muestra desde la parte izquierda ocupando gran parte de la pantalla. Muestra una cabecera de menú y el listado de apartados de la aplicación.

Principalmente estos elementos son los que definen el diseño principal de la aplicación. La disposición sigue las normas básicas para el diseño de una aplicación *Android* utilizando *material desing* de *Google* [13]. Tanto la sombra de la barra principal, la sombra de fondo al mostrar el menú, el menú que se superpone a la barra principal y por debajo de los indicadores del dispositivo, la organización de los elementos del menú, el espaciado entre los elementos, la iconografía y demás elementos siguen en su mayoría las indicaciones de *material desing*. En la figura 2.8 se puede apreciar gran parte de estos elementos de diseño.

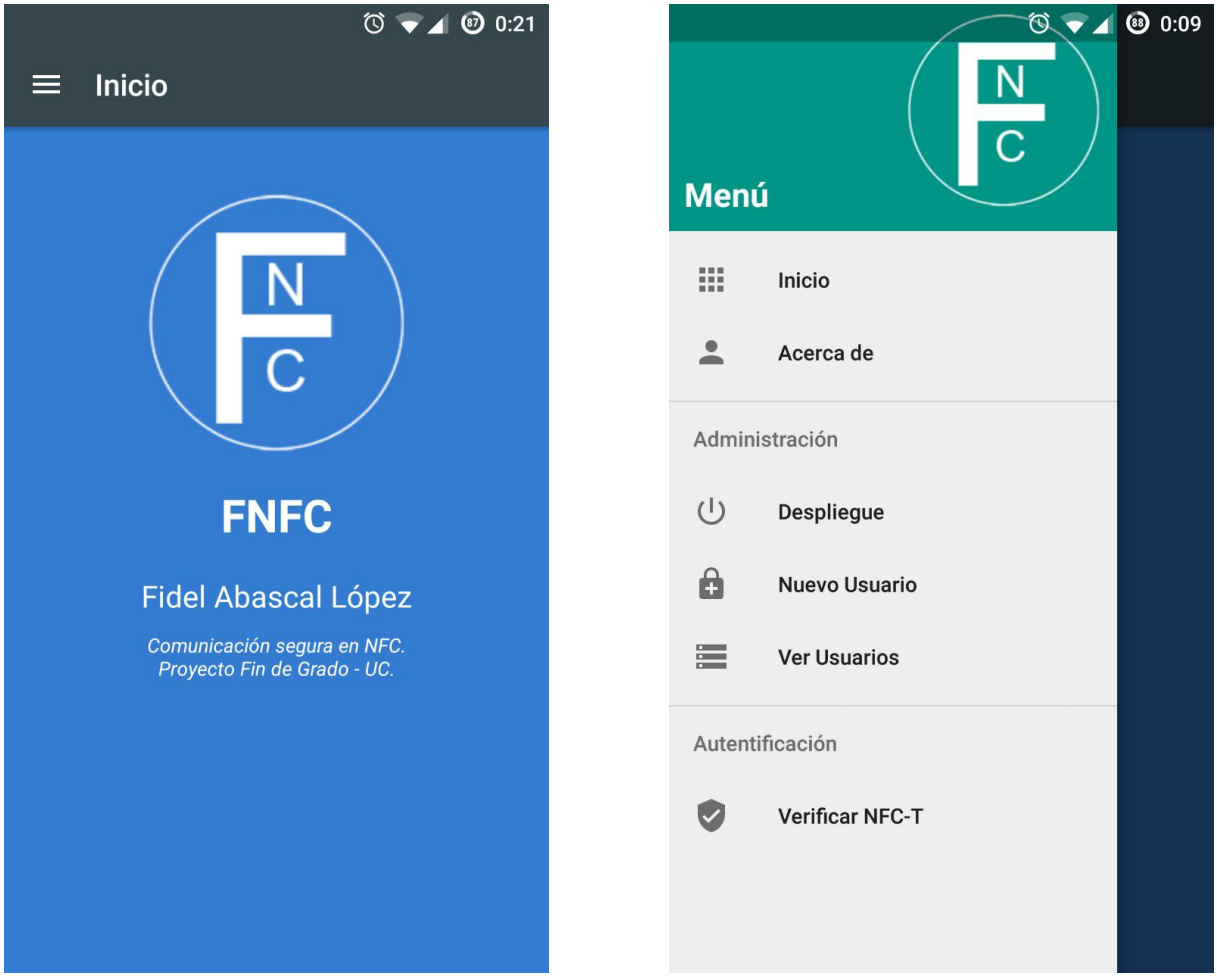

Figura 2.8: Elementos de diseño: Muestra de la disposición espacial del menú y estructura básica de la aplicación

Es oportuno mencionar la paleta de colores utilizada, tanto para el fondo como para las acentuaciones o el color primario y su homónimo oscuro :

- Color primario: #009688 .
- Color primario oscuro: #37474f .
- Color acentuado: #ff4081 .
- Color base principal: #367cd3 .

Se han elegido manteniendo la relación de colores de *material desing* y siguiendo normas conocidas de *marketing* y presentación de elementos visuales a consumidores. En estas normas y patrones es común observar que, por ejemplo, el color rojo implica pasión, efervescencia, impulso; ideal para representar ciertos elementos y estimular sensaciones acordes. Nótese que en la mayoría de marcas de comida, el color rojo predomina, ya que lo hace más apetecible al subconsciente humano y predispone al potencial consumidor a querer hacerse con ello.

Siguiendo el mismo ejemplo, el verde en marcas de alimentos se utiliza para representa lo natural, la limpieza y frescura; muy empleado en marcas ecológicas. El color base de la aplicación es el azul (hex:#367cd3); representa la serenidad, responsabilidad, sinceridad y verdad; apto para ser acogedor y mostrar fiabilidad. Es deliberado el uso de este color al igual que también lo es en la mayoría de las redes sociales. Buscan ganarse la confianza del usuario y dar un ambiente en el que se tenga una experiencia relajante y confortable. Por todo esto, el azul acerca la seguridad del criptosistema utilizado.

2.7. Seguridad y criptología

La seguridad utilizada se basa en la criptología con curvas elípticas. Para la implementación del criptosistema se han utilizado las clases de la librería *Bouncy Castle* y una interfaz *AppEllipticCurveI* a implementar con los métodos necesarios para la aplicación.

El flujo se simplifica básicamente en asignar a los usuarios del sistema un punto P aleatorio en una curva elíptica dentro de un campo finito $G(F(2^m))$. Las curvas están delimitadas por un conjunto de nombres de curvas ya definidas que cumplen ciertas características de seguridad (orden de la curva de más de 130 *bits*) y capacidad objetivo (codificaciones no superiores a la capacidad de las tarjetas NFC). Las curvas siguen diferentes estándares y se cargan en función del nombre que se les atribuye.

Los puntos P obtenidos se multiplican por una variable aleatoria privada k de 160 *bits* de tamaño obteniendo kP. Siguiendo el método *SKEY*, se multiplica el resultado kP tantas veces como usos se hayan indicado a la hora de la creación del punto P (*véase el apartado de la aplicación 2.9.5 sobre la creación de usuarios*). Con ello se computa el valor QkP que es un punto en la curva. Computar kP es sencillo, sin embargo, obtener la variable k es computacionalmente muy complejo debido al problema del logaritmo discreto. Para utilizar códigos de un solo uso se computa Q veces la suma de kP para almacenar en la base de datos el punto $(Q+1)kP$ asociado a un usuario de la aplicación; a su vez, se escribe en la tarjeta NFC el valor QkP y el identificador del usuario.

A la hora de validar la información de la tarjeta se obtiene la codificación del punto QkP de la tarjeta NFC. A este punto se le añade el punto original kP y se comprueba igual a $(Q+1)kP$ de la base de datos. Si coincide se valida la información con lo que se prepara el código del siguiente uso, que se trata de $(X-1)kP$ y se escribe en la tarjeta (guardando el valor comprobante de $(X+1)kP$ hasta alcanzar kP, siendo X el uso actual. Una vez agotados los usos se realiza automáticamente la asignación de un nuevo punto al usuario con los Q usos elegidos originalmente.

Se muestra en la figura 2.9 y 2.10 la estructura de la interfaz mencionada junto a la documentación de cada uno de los métodos que la componen.

```java
public interface AppEllipticCurveI {

    /**
     * Load instance parameters of a elliptic curve from given values
     * @param {@link String} curveName
     * @param {@link java.math.BigInteger} k
     * @return {@link org.bouncycastle.asn1.x9.X9ECParameters} X9ECParameters of curve
     * @throws {@link IllegalArgumentException} IllegalArgumentException
     * @throws {@link fidel.pfg.fnfc.exceptions.CurveNotLoaded} CurveNotLoaded
     */
    public X9ECParameters loadEC(String curveName, BigInteger k) throws IllegalArgumentException , CurveNotLoaded;

    /**
     * Gets a random point from the loaded elliptic curve
     * @return {@link org.bouncycastle.math.ec.ECPoint.F2m} random point of the loaded curve
     * @throws {@link fidel.pfg.fnfc.exceptions.CurveNotLoaded} CurveNotLoaded
     */
    public ECPoint.F2m getRandomPoint() throws CurveNotLoaded;

    /**
     * Generate and set a new secure random key with a default key size
     * @return {@link org.bouncycastle.math.ec.ECPoint.F2m} random point
     */
    public BigInteger newKey();

    /**
     * Multiply the given point {@link org.bouncycastle.math.ec.ECPoint.F2m} p1 n+1 times
     * @param {@link org.bouncycastle.math.ec.ECPoint.F2m} p1 the point to multiply
     * @param n times to multiply +1
     * @return p1*(n+1) as {@link org.bouncycastle.math.ec.ECPoint.F2m}
     */
    public ECPoint.F2m addPoint(ECPoint.F2m p1, int n);
```

Figura 2.9: Interfaz AppEllipticCurveI que se implementa en la aplicación (I).

```java
    public ECPoint.F2m addPoint(ECPoint.F2m p1, int n);

    /**
     * Add the {@link org.bouncycastle.math.ec.ECPoint.F2m} p2 to p1
     * @param {@link org.bouncycastle.math.ec.ECPoint.F2m} p1
     * @param {@link org.bouncycastle.math.ec.ECPoint.F2m} p2
     * @return result of p1+p2 as {@link org.bouncycastle.math.ec.ECPoint.F2m}
     */
    public ECPoint.F2m addPointsCustom(ECPoint.F2m p1, ECPoint.F2m p2);

    /**
     * Encode the {@link org.bouncycastle.math.ec.ECPoint.F2m} given
     * @param {@link org.bouncycastle.math.ec.ECPoint.F2m} point
     * @return {@link String} String encoded point
     * @throws {@link fidel.pfg.fnfc.exceptions.CurveNotLoaded} CurveNotLoaded
     */
    public String encode(ECPoint.F2m point) throws CurveNotLoaded;

    /**
     * Gets the ECPoint.F2m from encoded value
     * @param {@link String} pointValue
     * @return {@link org.bouncycastle.math.ec.ECPoint.F2m} decoded point
     * @throws {@link fidel.pfg.fnfc.exceptions.CurveNotLoaded} CurveNotLoaded
     */
    public ECPoint.F2m decode(String pointValue) throws CurveNotLoaded;

}
```

Figura 2.10: Interfaz AppEllipticCurveI que se implementa en la aplicación (II).

Dentro de la implementación es oportuno comentar el método *AddCustomPoints* el cuál es la realización por cuenta propia del método de suma de puntos en una curva elíptica en campos finitos del tipo $GF(2^m)$. La figura 2.11 muestra en detalle la suma implementada.

```java
@Override
public ECPoint.F2m addPointsCustom(ECPoint.F2m p1, ECPoint.F2m p2){
    ECCurve.F2m ec = (ECCurve.F2m) p1.getCurve();
    ECPoint.F2m other = p2;

    // If the point is the infinity
    if (p1.isInfinity()) { return other; }
    if (other.isInfinity()) { return p1; }

    ECFieldElement.F2m x2 = (ECFieldElement.F2m)other.getX();
    ECFieldElement.F2m y2 = (ECFieldElement.F2m)other.getY();
    ECFieldElement.F2m x1 = (ECFieldElement.F2m)p1.getX();
    ECFieldElement.F2m y1 = (ECFieldElement.F2m)p1.getY();

    // Check if other = this or other = -this
    if (x1.equals(x2))
    {
        if (y1.equals(y2))
        {
            // this = other, i.e. this must be doubled
            return (ECPoint.F2m)p1.twice();
        }

        // this = -other, i.e. the result is the point at infinity
        return (ECPoint.F2m)ecurve.getInfinity();
    }

    ECFieldElement.F2m lambda
            = (ECFieldElement.F2m)(y1.add(y2)).divide(x1.add(x2));
    ECFieldElement.F2m x3
            = (ECFieldElement.F2m)lambda.square().add(lambda).add(x1).add(x2).add(ec.getA());
    ECFieldElement.F2m y3
            = (ECFieldElement.F2m)lambda.multiply(x1.add(x3)).add(x3).add(y1);

    // Return with Normalize
    return (org.bouncycastle.math.ec.ECPoint.F2m) ec.createPoint(x3.toBigInteger(), y3.toBigInteger(), true).normalize();
}
```

Figura 2.11: Suma de puntos implementada en java.

Básicamente se divide en dos partes: si los puntos pasados por parámetro son iguales (duplicar el punto) y o si son distintos. Para el caso de puntos iguales, se realiza la llamada al método *twice()* de la clase *ECPoint.F2m* de la librería de *Bouncy Castle*. Por el contrario, en el caso de puntos diferentes, se realiza la suma de puntos en curva elíptica tras unas comprobaciones previas. La fórmula utilizada y comprobada respecto a la realizada internamente por la librería es la siguiente:

$$y^2 + xy = x^3 + ax^2 + b \ (mod\ n) \equiv E(G(2^m))$$
$$P = (x_1, y_1), Q = (x_2, y_2), R = P + Q = (x_3, y_3)$$
$$P, Q, R \in E(GF(2^m))$$
$$\lambda = (y_1 + y_2)/(x_1 + x_2)$$
$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$$
$$y_3 = \lambda * (x_1 + x_3) + x_3 + y_1$$

La seguridad reside en la privacidad del elemento k difícilmente obtenible pese a que los demás elementos sean públicos gracias a las curvas elípticas en campos finitos y al uso de *SKEY* para utilizar códigos únicos de un solo uso. En el escenario real definido en el apartado 2.4 se utilizaría una comunicación segura de la clave k y otra clave q también empleando curvas elípticas. La idea simplificada es la misma, obtener el valor k de kP es difícilmente computable. En el anexo I 3 se muestra el código de ejemplo de una comunicación de claves de este tipo. Con ésta implementación se podrían comunicar los valores de forma segura entre los tornos de seguridad y los servidores de comprobación y validación.

2.8. Base de datos

La base de datos, mencionada en el apartado 2.2, es una pequeña *SQLite*. Se utiliza este tipo de base de datos debido a la comodidad y simplicidad con la que trabaja en un proyecto *Android*.

En el escenario virtual de la aplicación, en vez de esta base de datos se utilizaría una centralizada en los servidores de la empresa. Se realizarían conexiones seguras para no comprometer la información a enviar. Sin embargo, ésto no es necesario dentro del ámbito real ya que toda la información que se precisa de los servidores ficticios de la empresa se encuentran dentro de esta pequeña base de datos.

Dentro de esta base de datos se encuentra la información de la curva elíptica implementada y la clave privada asociada (*véase el apartado sobre seguridad y critología 2.7*). También está la información de parte de los usuarios ficticios de la empresa. Por último, se encuentra una tabla referida a la información de cada usuario que compone los datos para poder validar el contenido de cada tarjeta NFC de cada usuario. En la figura 2.12 se observa la distribución de tablas y la relación entre ellas (creado mediante la herramienta online *GenMyModel* [8]). Posteriormente en la figura 2.13 se observa el *script* de creación en lenguaje *SQL*.

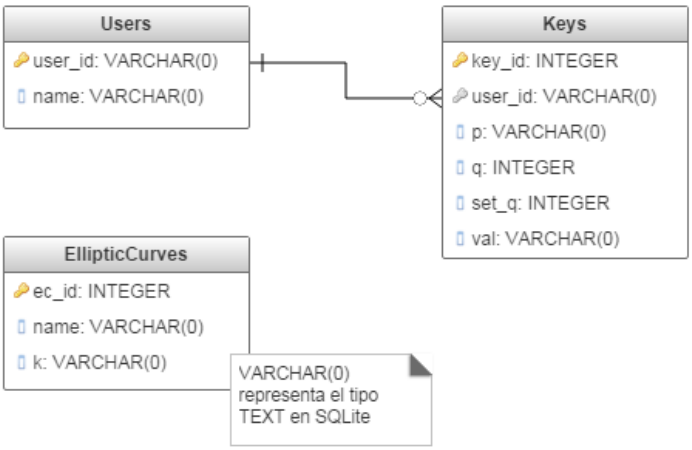

Figura 2.12: Representación de la base de datos implementada.

```
CREATE TABLE Users(
    user_id TEXT PRIMARY KEY,
    name TEXT
    );
CREATE TABLE Keys(
    key_id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id TEXT,
    p TEXT,
    q INTEGER,
    set_q INTEGER,
    val TEXT,
    FOREIGN KEY(user_id) REFERENCES Users(user_id) ON DELETE CASCADE
    );
CREATE TABLE EllipticCurves(
    ec_id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT,
    k TEXT
    );

PRAGMA foreign_keys=ON;
CREATE INDEX user_id_index ON Keys(user_id);
```

Figura 2.13: Script para la creación de la base de datos.

El contenido por defecto del sistema se adecua a la implementación de la curva elíptica $c2pnb163v1$ (cuyos detalles se pueden encontrar en [4]). La clave por defecto es un número entero de 160 bits que es el siguiente: 838828326113658401440043399564525405856963575389. Los distintos valores de los campos *P*, *kP*, *Q* y *Set Q*, *QkP* corresponden con datos válidos del sistema utilizado y de puntos aleatorios de la curva generados bajo la clave mencionada.

2.9. Aplicación desarrollada

La aplicación final se compone principalmente de una actividad principal que dispone del menú lateral, la barra superior de acciones y un contenedor que va modificándose en función de la situación. En las siguientes secciones se muestra el contenido, utilización, significado y ejemplos de cada uno de estos apartados que compondrán los datos que se muestran en el contenedor.

2.9.1. Inicio

Pagina XX

2.9.2. Menú

Pagina XX

2.9.3. Acerca de

Pagina XX

2.9.4. Despliegue

Pagina XX

Sistema

Pagina XX

Base de datos

Pagina XX

2.9.5. Nuevo usuario

Pagina XX

2.9.6. Ver usuarios

Pagina XX

2.9.7. Verificación

Pagina XX

2.10. Pruebas

Para la comprobación y consecución de pruebas unitarias se ha elaborado un proyecto independiente para el testeo de la parte crítica de la aplicación. Ésta parte es la referida a la implementación de la seguridad y criptografía (*véase apartado 2.7*).

En cada uno de los métodos de la clase principal se ha testeado que su ejecución en distintos ámbitos y condiciones devuelve los resultados esperados cubriendo un alto porcentaje de código. Las pruebas se han centrado en el *framework* de pruebas unitarias *JUnit* [16] en su versión 4 elaborada para *Eclipse* [7].

La descripción básica de la clase y su documentación es la recogida por la interfaz que implementa (la descripción en detalle de la clase de puede encontrar en el apartado 2.7). El resultado de los test JUnit se pueden observar en la figura 2.14 con una consecución de cobertura del código de más del 96 % realizado con el complemento *EclEmma* [5] para este propósito. En la misma figura se muestra la prueba de consecución de cobertura código la clase objetivo.

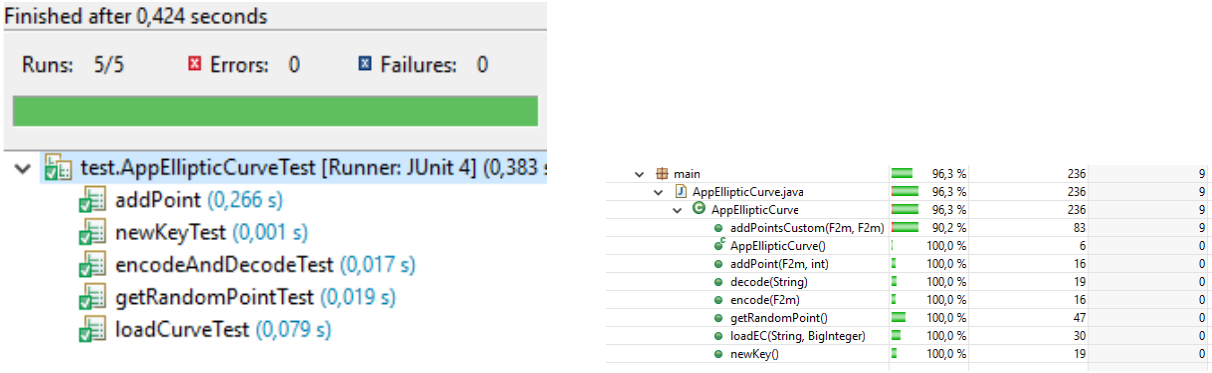

Figura 2.14: Resultado del test JUnit y cobertura de código alcanzada durante el test JUnit.

En la realización de estas pruebas unitarias de la clase que implementa la interfaz *AppEllipticCurveI*, clase principal que implementa la seguridad de la aplicación, se ha utilizado diferentes pruebas de ejecución del código utilizando *Java Reflection* [1] cuando se ha considerado oportuno. A modo de ejemplo se muestra en la figura el código del método principal *loadEC* para la carga de una curva elíptica en el sistema.

```
@Test
public void loadCurveTest(){
    String curveName = "c2pnb163v1", curveNameFalse="false curve name"; // Curve names to use
    try {
        this.aec.loadEC(curveName, null);

        // Reflection params access
        Field f = aec.getClass().getDeclaredField("ecurve");
        f.setAccessible(true);
        ECCurve.F2m createdParams = (ECCurve.F2m) f.get(aec);
        // c2pnb163v1 true order
        BigInteger order = new BigInteger("5846006549323611672814741626226392056573832638401");
        Assert.assertTrue(createdParams.getOrder().equals(order)); //check equals

        // Test loadEC with key
        k = new BigInteger("13706797175872909360991329612936365248199107099 62");
        this.aec.loadEC(curveName, k);

        // Reflection k access
        f = aec.getClass().getDeclaredField("k");
        f.setAccessible(true);
        // Gets loaded k
        BigInteger createdK = (BigInteger) f.get(aec);
        Assert.assertTrue(createdK.equals(k)); // check equals

    } catch (Exception e) {
        e.printStackTrace();
        fail("Exception occured");
    }

    // Exception assertion
    try {
        this.aec.loadEC(curveNameFalse, null);
        fail("Expected exception was not occured.");
    } catch (IllegalArgumentException e) {
        assert true;
    }

}
```

Figura 2.15: Script para el test de la creación de una curva elíptica.

3 Conclusiones

Conclusiones

3.1. Futuras mejoras

Futuras mejoras

Anexo I: Comunicación de claves

A continuación se muestra la comunicación de claves con codificación asimétrica [9].

```
/*
 * @author: Feng Hao, haofeng66@gmail.com
 *
 * This is a simple demo program written in Java, just to show how J-PAKE can
    be implemented using
 * the Elliptic Curve group setting (the same setting as that of ECDSA or
    ECDH).
 *
 * The implementation of J-PAKE in the DSA-like group setting has been
    included into Bouncycastle (v1.48 and above).
 * Details can be found at:
 *
    http://www.bouncycastle.org/viewcvs/viewcvs.cgi/java/crypto/src/org/bouncycastle/crypto/
 *
 * License of the code: none. The code is free to use and modify without any
    restrictions.
 *
 * Dependence: BouncyCastle library (https://www.bouncycastle.org/java.html)
 *
 * Publications:
 *  - The initial workshop paper (SPW'08):
    http://grouper.ieee.org/groups/1363/Research/contributions/hao-ryan-2008.pdf
 *  - The extended journal version (Springer Transactions'10):
    http://eprint.iacr.org/2010/190.pdf
 *
 * Acknowledgment: the author would like to thank Dylan Clarke for useful
    comments on the demo code.
 *
 * Date: 29 December 2013.
 *
 */

import java.math.BigInteger;
import java.nio.ByteBuffer;
import java.security.MessageDigest;
import java.security.SecureRandom;

import org.bouncycastle.jce.ECNamedCurveTable;
```

```
import org.bouncycastle.jce.spec.ECParameterSpec;
import org.bouncycastle.math.ec.ECCurve;
import org.bouncycastle.math.ec.ECPoint;

public class EllipticCurveJPAKEDemo {

   /*
    * See [1] for public domain parameters for NIST standard curves
    * P-224, P-256, P-384, P-521. This demo code only uses P-256 as an
       example. One can also
    * use other curves that are suitable for Elliptic Curve Cryptography
       (ECDSA/ECDH), e.g., Curve25519.
    *
    * [1] D. Johnson, A. Menezes, S. Vanstone, "The Elliptic Curve Digital
       Signature Algorithm (ECDSA)",
    *     International Journal of Information Security, 2001. Available at
    *     http://cs.ucsb.edu/~koc/ccs130h/notes/ecdsa-cert.pdf
    *
    */

   private ECParameterSpec ecSpec =
      ECNamedCurveTable.getParameterSpec("prime256v1");

   // Domain parameters
   private ECCurve.Fp ecCurve = (ECCurve.Fp)ecSpec.getCurve();
   private BigInteger a = ecCurve.getA().toBigInteger();
   private BigInteger b = ecCurve.getB().toBigInteger();
   private BigInteger q = ecCurve.getQ();
   private BigInteger coFactor = ecSpec.getH(); // Not using the symbol "h"
      here to avoid confusion as h will be used later in SchnorrZKP.
   private BigInteger n = ecSpec.getN();
   private ECPoint G = ecSpec.getG();

   /*
    * Shared passwords for Alice and Bob
    * Try changing them to different values?
    */

   private String s1Str = "deadbeef";
   private String s2Str = "deadbeef";

   /*
    * UserIDs for Alice and Bob.
    */

   private String AliceID = "Alice";
   private String BobID = "Bob";

    public static void main(String args[]) {
```

```
    EllipticCurveJPAKEDemo test = new EllipticCurveJPAKEDemo();
    test.run();
  }

  private void run () {

    System.out.println("************ Public elliptic curve domain parameters
        ************\n");
    System.out.println("Curve param a (" + a.bitLength() + " bits): "+
        a.toString(16));
    System.out.println("Curve param b (" + b.bitLength() + " bits): "+
        b.toString(16));
    System.out.println("Co-factor h (" + coFactor.bitLength() + " bits): " +
        coFactor.toString(16));
    System.out.println("Base point G (" + G.getEncoded().length + " bytes):
        " + new BigInteger(G.getEncoded()).toString(16));
    System.out.println("X coord of G (" +
        G.getX().toBigInteger().bitLength() + " bits): " +
        G.getX().toBigInteger().toString(16));
    System.out.println("y coord of G (" +
        G.getY().toBigInteger().bitLength() + " bits): " +
        G.getY().toBigInteger().toString(16));
    System.out.println("Order of the base point n (" + n.bitLength() + "
        bits): "+ n.toString(16));
    System.out.println("Prime field q (" + q.bitLength() + " bits): "+
        q.toString(16));

    System.out.println("");

    System.out.println("(Secret passwords used by Alice and Bob: "+
         "\""+s1Str+"\" and \""+s2Str+"\")\n");

    BigInteger s1 = new BigInteger(s1Str.getBytes());
    BigInteger s2 = new BigInteger(s2Str.getBytes());

    /* Step 1:
     *
     * Alice chooses x1 randomly from [1, n-1], x2 from [1, n-1]
     * Similarly, Bob chooses x3 randomly from [1, n-1] and x4 from [1, n-1]
     *
     * Alice -> Bob: G*x1, G*x2 and ZKP{x1}, ZKP{X2}
     * Bob -> Alice: G*x3, G*x4 and ZKP{x3}, ZKP{X4}
     *
     * Note: in the DSA setting, x1, x3 are chosen from [0, q-1] and x2, x4
        from [1, q-1]
     * However, in the ECDSA setting, the zero element is naturally excluded.
     */
```

```
BigInteger x1 =
    org.bouncycastle.util.BigIntegers.createRandomInRange(BigInteger.ONE,
      n.subtract(BigInteger.ONE), new SecureRandom());
BigInteger x2 =
    org.bouncycastle.util.BigIntegers.createRandomInRange(BigInteger.ONE,
      n.subtract(BigInteger.ONE), new SecureRandom());
BigInteger x3 =
    org.bouncycastle.util.BigIntegers.createRandomInRange(BigInteger.ONE,
      n.subtract(BigInteger.ONE), new SecureRandom());
BigInteger x4 =
    org.bouncycastle.util.BigIntegers.createRandomInRange(BigInteger.ONE,
      n.subtract(BigInteger.ONE), new SecureRandom());

ECPoint X1 = G.multiply(x1);
SchnorrZKP zkpX1 = new SchnorrZKP();
zkpX1.generateZKP(G, n, x1, X1, AliceID);

ECPoint X2 = G.multiply(x2);
SchnorrZKP zkpX2 = new SchnorrZKP();
zkpX2.generateZKP(G, n, x2, X2, AliceID);

ECPoint X3 = G.multiply(x3);
SchnorrZKP zkpX3 = new SchnorrZKP();
zkpX3.generateZKP(G, n, x3, X3, BobID);

ECPoint X4 = G.multiply(x4);
SchnorrZKP zkpX4 = new SchnorrZKP();
zkpX4.generateZKP(G, n, x4, X4, BobID);

System.out.println("************Step 1**************\n");
System.out.println("Alice sends to Bob: ");
System.out.println("G*x1="+new BigInteger(X1.getEncoded()).toString(16));
System.out.println("G*x2="+new BigInteger(X2.getEncoded()).toString(16));
System.out.println("KP{x1}: {V="+new
    BigInteger(zkpX1.getV().getEncoded()).toString(16)+";
    r="+zkpX1.getr().toString(16)+"}");
System.out.println("KP{x2}: {V="+new
    BigInteger(zkpX2.getV().getEncoded()).toString(16)+";
    r="+zkpX2.getr().toString(16)+"}");
System.out.println("");

System.out.println("Bob sends to Alice: ");
System.out.println("G*x3="+new BigInteger(X3.getEncoded()).toString(16));
System.out.println("G*x4="+new BigInteger(X4.getEncoded()).toString(16));
System.out.println("KP{x3}: {V="+new
    BigInteger(zkpX3.getV().getEncoded()).toString(16)+";
    r="+zkpX3.getr().toString(16)+"}");
System.out.println("KP{x4}: {V="+new
    BigInteger(zkpX4.getV().getEncoded()).toString(16)+";
```

```
    r="+zkpX4.getr().toString(16)+"}");
System.out.println("");

/*
 * Alice checks 1) BobID is a valid identity (omitted in this demo code)
    and 2) is different from her own
 */
if (AliceID.equals(BobID)) {
   System.out.println("ERROR: AliceID and BobID must be different.");
   System.exit(0);
}

/*
 * Alice verifies Bob's ZKPs.
 *
 * Note: in the DSA setting, Alice needs to check g^{x4} != 1 (i.e., not
    an identity element).
 * In the ECDSA setting, checking the infinity point (i.e., identity
    element) has been covered in the public key validation step,
 * as part the Schnorr ZKP verification routine.
 */

if (verifyZKP(G, X3, zkpX3.getV(), zkpX3.getr(), BobID) && verifyZKP(G,
    X4, zkpX4.getV(), zkpX4.getr(), BobID)) {
   System.out.println("Alice checks KP{x3}: OK");
   System.out.println("Alice checks KP{x4}: OK");
   System.out.println("");
}else {
   System.out.println("ERROR: invalid KP{x3, x4}.");
   System.exit(0);
}

/*
 * Symmetrically, Bob checks Alice's UserID and her KPs on {x1} and {x2}
 */

if (BobID.equals(AliceID)) {
   System.out.println("ERROR: AliceID and BobID must be different.");
   System.exit(0);
}

if (verifyZKP(G, X1, zkpX1.getV(), zkpX1.getr(), AliceID) &&
    verifyZKP(G, X2, zkpX2.getV(), zkpX2.getr(), AliceID)) {
   System.out.println("Bob checks KP{x1}: OK");
   System.out.println("Bob checks KP{x2}: OK");
   System.out.println("");
}else {
   System.out.println("ERROR: invalid KP{x1, x2}.");
   System.exit(0);
```

```
}

/*
 * Step 2:
 *
 * Alice -> Bob: A and KP{x2s}
 * Bob -> Alice: B and KP{x4s}
 */

ECPoint GA = X1.add(X3).add(X4);
ECPoint A = GA.multiply(x2.multiply(s1).mod(n));

SchnorrZKP zkpX2s = new SchnorrZKP();
zkpX2s.generateZKP(GA, n, x2.multiply(s1).mod(n), A, AliceID);

ECPoint GB = X1.add(X2).add(X3);
ECPoint B = GB.multiply(x4.multiply(s2).mod(n));

SchnorrZKP zkpX4s = new SchnorrZKP();
zkpX4s.generateZKP(GB, n, x4.multiply(s2).mod(n), B, BobID);

System.out.println("************Step 2**************\n");
System.out.println("Alice sends to Bob:");
System.out.println("A="+new BigInteger(A.getEncoded()).toString(16));
System.out.println("KP{x2*s}: {V="+new
    BigInteger(zkpX2s.getV().getEncoded()).toString(16)+",
    r="+zkpX2s.getr().toString(16)+"}");
System.out.println("");

System.out.println("Bob sends to Alice:");
System.out.println("B="+new BigInteger(B.getEncoded()).toString(16));
System.out.println("KP{x4*s}: {V="+new
    BigInteger(zkpX4s.getV().getEncoded()).toString(16)+",
    r="+zkpX4s.getr().toString(16)+"}");
System.out.println("");

/* Alice verifies Bob's ZKP */
if (verifyZKP(GB, B, zkpX4s.getV(), zkpX4s.getr(), BobID)) {
   System.out.println("Alice checks KP{x4*s}: OK");
} else {
   System.out.println("ERROR: invalid KP{x4*s}.");
   System.exit(0);
}

/*
 * Symmetrically, Bob checks Alice's KP on {x1*s}
 */
if (verifyZKP(GA, A, zkpX2s.getV(), zkpX2s.getr(), AliceID)) {
   System.out.println("Bob checks KP{x2*s}: OK");
```

```
        System.out.println("");
      }else {
        System.out.println("ERROR: invalid KP{x2*s}.");
        System.exit(0);
      }

      /* After step 2, compute the common key based on hashing the x
          coordinate of the derived EC point */
      BigInteger Ka = getSHA256(
          B.subtract(X4.multiply(x2.multiply(s1).mod(n))).multiply(x2).getX().toBigInteger())
      BigInteger Kb = getSHA256(
          A.subtract(X2.multiply(x4.multiply(s2).mod(n))).multiply(x4).getX().toBigInteger())

      System.out.println("************After step 2**************\n");
      System.out.println("Alice computes a session key \t K="+Ka.toString(16));
      System.out.println("Bob computes a session key \t K="+Kb.toString(16));

      /*
       * It is recommended that both parties perform an explicit key
          confirmation
       * before using the session key. This provides explicit assurance that
          the
       * two parties have actually obtained the same session key. The key
          confirmation
       * method is the same regardless of the group setting (DSA or ECDSA).
          See the
       * existing J-PAKE key confirmation implementation in Bouncycastle for
          details.
       *
       *
          http://www.bouncycastle.org/viewcvs/viewcvs.cgi/java/crypto/src/org/bouncycastle/c
       */
    }

    public boolean verifyZKP(ECPoint generator, ECPoint X, ECPoint V,
        BigInteger r, String userID) {

      /* ZKP: {V=G*v, r} */
      BigInteger h = getSHA256(generator, V, X, userID);

      // Public key validation based on p. 25
      // http://cs.ucsb.edu/~koc/ccs130h/notes/ecdsa-cert.pdf

      // 1. X != infinity
      if (X.isInfinity()){
        return false;
      }

      // 2. Check x and y coordinates are in Fq, i.e., x, y in [0, q-1]
```

```
  if (X.getX().toBigInteger().compareTo(BigInteger.ZERO) == -1 ||
        X.getX().toBigInteger().compareTo(q.subtract(BigInteger.ONE)) == 1
            ||
        X.getY().toBigInteger().compareTo(BigInteger.ZERO) == -1 ||
        X.getY().toBigInteger().compareTo(q.subtract(BigInteger.ONE)) == 1)
            {
     return false;
  }

  // 3. Check X lies on the curve
  try {
     ecCurve.decodePoint(X.getEncoded());
  }
  catch(Exception e){
     e.printStackTrace();
     return false;
  }

  // 4. Check that nX = infinity.
  // It is equivalent - but more more efficient - to check the coFactor*X
      is not infinity
  if (X.multiply(coFactor).isInfinity()) {
     return false;
  }

  // Now check if V = G*r + X*h.
  // Given that {G, X} are valid points on curve, the equality implies
      that V is also a point on curve.
  if (V.equals(generator.multiply(r).add(X.multiply(h.mod(n))))) {
     return true;
  }
  else {
     return false;
  }
}

public BigInteger getSHA256(ECPoint generator, ECPoint V, ECPoint X,
    String userID) {

  MessageDigest sha256 = null;

  try {
     sha256 = MessageDigest.getInstance("SHA-256");

     byte [] GBytes = generator.getEncoded();
     byte [] VBytes = V.getEncoded();
     byte [] XBytes = X.getEncoded();
     byte [] userIDBytes = userID.getBytes();
```

```
        // It's good practice to prepend each item with a 4-byte length
        sha256.update(ByteBuffer.allocate(4).putInt(GBytes.length).array());
        sha256.update(GBytes);

        sha256.update(ByteBuffer.allocate(4).putInt(VBytes.length).array());
        sha256.update(VBytes);

        sha256.update(ByteBuffer.allocate(4).putInt(XBytes.length).array());
        sha256.update(XBytes);

        sha256.update(ByteBuffer.allocate(4).putInt(userIDBytes.length).array());
        sha256.update(userIDBytes);

      } catch (Exception e) {
        e.printStackTrace();
      }

      return new BigInteger(sha256.digest());
    }

    public BigInteger getSHA256(BigInteger K) {

      MessageDigest sha256 = null;

      try {
        sha256 = MessageDigest.getInstance("SHA-256");
        sha256.update(K.toByteArray());
      } catch (Exception e) {
        e.printStackTrace();
      }

      return new BigInteger(1, sha256.digest()); // 1 for positive int
    }

    private class SchnorrZKP {

      private ECPoint V = null;
      private BigInteger r = null;

      private SchnorrZKP () {
        // constructor
      }

      private void generateZKP (ECPoint generator, BigInteger n, BigInteger x,
          ECPoint X, String userID) {

        /* Generate a random v from [1, n-1], and compute V = G*v */
        BigInteger v =
            org.bouncycastle.util.BigIntegers.createRandomInRange(BigInteger.ONE,
```

```
            n.subtract(BigInteger.ONE), new SecureRandom());
        V = generator.multiply(v);

        BigInteger h = getSHA256(generator, V, X, userID); // h

        r = v.subtract(x.multiply(h)).mod(n); // r = v-x*h mod n
       }

     private ECPoint getV() {
        return V;
     }

     private BigInteger getr() {
        return r;
     }

   }
}
```

Bibliografía

[1] Oracle Corporation. Java reflection.

[2] Oracle Corporation. Java security package.

[3] Oracle Corporation. Software java.

[4] Technische Universität Darmstadt. Curves over gf(2m) defined by ansi x9.62.

[5] EclEmma. Java code coverage for eclipse.

[6] NFC Forum. Nfc forum specifications.

[7] Eclipse Fundation. Eclipse.

[8] GenMyModel. Genmymodel modeling platform.

[9] Feng Hao. J-pake as ecdsa implementation.

[10] Google Inc. Android apis - about version 5.0.

[11] Google Inc. Android apis - about version 5.1.

[12] Google Inc. Android studio - the official ide for android.

[13] Google Inc. Material design guidelines.

[14] Google Inc. Material design icons.

[15] ISO/IEC. Iso/iec standard 14443-3.

[16] JUnit. Junit.

[17] Manuel José Lucena. Criptografía y seguridad en computadores. *versión 0.7*, 5, 1999.

[18] MIFARE. Mifare ic - contactless smart cards.

[19] Legion of the Bouncy Castle Inc. Bouncy castle - api criptográfica.

[20] Pocketworks. yuml diagrams.

[21] Mohamad Ramli, Nurul Akmar, Muhammad Saufi Kamarudin, and Ariffuddin Joret. Iris recognition for personal identification. 2008.

[22] SQLite. Sqlite - public domain embedded sql database engine.

[23] GanttProject Team. Ganttproject.

[24] Whatsapp. Whatsapp encryption overview.