UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

PROYECTO FIN DE CARRERA

Diseño y desarrollo de una aplicación Android para el uso de identidades digitales, autenticación y firmas digitales en sistemas interactivos

INGENIERÍA DE TELECOMUNICACIÓN

Eva Milagros Blanco Delgado

MAYO 2014

Diseño y desarrollo de una aplicación Android para el uso de identidades digitales, autenticación y firmas digitales en sistemas interactivos

AUTOR: Eva Milagros Blanco Delgado

TUTOR: Francisco de Borja Rodríguez Ortiz

Grupo de Neurocomputación Biológica (GNB)

Dpto. de Ingeniería Informática

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Mayo de 2014

RESUMEN

En este proyecto se ha desarrollado una aplicación Android que permite al usuario obtener una identidad digital de la que puede hacer uso, como por ejemplo, para autenticarse en un sistema. Para la consecución de tal identidad, se ha implementado un protocolo, desarrollado previamente por el GNB para la plataforma Moodle [1][2][3], que permite obtenerla sin necesidad de desplazarse para identificarse a ninguna entidad de registro y con cierto grado de seguridad.

Para alcanzar estos objetivos, se ha realizado, en primer lugar, un estudio sobre el contexto en el que se desarrolla la aplicación en referencia a los conocimientos básicos de seguridad en la implementación de un sistema y la seguridad existente en una plataforma de amplia expansión como es Android.

En segundo lugar, se ha realizado una descripción de las diferentes herramientas que facilitan la implementación de tal aplicación y cómo han de usarse o cuál es la metodología para facilitar y automatizar el trabajo.

A continuación, se expone el diseño que se ha llevado a cabo sobre el protocolo de seguridad implementado y su uso en relación a la aplicación desarrollada, explicando que pasos se han seguido para la realización tanto de la aplicación, como del protocolo.

Después del desarrollo de la aplicación y de la implementación del protocolo, se muestra cuál es el resultado final de la aplicación, sus funciones, sus características y cualidades y se comprueba cuál es el rendimiento final del sistema.

Por último, se lleva a cabo una discusión sobre las conclusiones a las que se ha llegado a lo largo de la realización del proyecto además de las propuestas para un trabajo futuro.

PALABRAS CLAVE

Android, identidad digital, protocolo de seguridad, certificados digitales, Autoridad certificadora, PKI, firma digital, autenticación, cifrado simétrico, cifrado asimétrico.

ABSTRACT

In this Master Thesis, an Android application, which lets a user obtain a digital identity from which he can make use e. g. to authenticate himself in a system, has been developed. A protocol previously developed by the GNB group for Moodle platform [1][2][3], has been implemented to obtain that identity without the needing from a user to move to identificate himself in any registration entity and having certain level of security.

In order to achieve these objetives, firstly, there has been made a research of the context in which the application is developed regarding basic knowledge of a system security and existing security in a wide spread platform like Android.

In the second place, there has been carried out a description of the different tools that facilitate the implementation of such an application and how they have to be used or what is the metodology for facilitating and automating the work.

Following, there has been exposed the design carried out on the security protocol here implemented and its use regarding the developed application, explaining the steps that there have been followed to the realization of both the application and the protocol.

After the development of the application and the protocol implementation, it is shown which is the final result of the application, its functions, its features and attributes, and it is checked which is the final efficiency of the system.

Finally, there has been carried out a discussion on the conclusions that have been reached during the Master Thesis realization, in addition to the suggestions for future work.

KEY WORDS

Android, digital identity, security protocol, digital certificates, Certificate Authority, PKI, digital signature, authentication, symmetric encryption, asymmetric encryption.

"La vida no es fácil, para ninguno de nosotros. Pero... ¡qué importa! Hay que perseverar y, sobre todo, tener confianza en uno mismo. Hay que sentirse dotado para realizar alguna cosa y que esa cosa hay que alcanzarla, cueste lo que cueste"

Marie Curie (1867-1934)

"Aunque me quede solo, no cambiaría mis libres pensamientos por un trono"

Lord Byron (1788-1824)

AGRADECIMIENTOS

Cuando leo los proyectos de mis compañeros, la parte en la que primero me fijo es en los agradecimientos, porque lo que se pone en ellos refleja más que la meta alcanzada y los objetivos conseguidos: los agradecimientos muestran el camino, las bases y el apoyo necesarios para alcanzar dichos objetivos y como sin estos apoyos, no estaríamos donde ahora estamos.

Las personas más importantes que me han llevado a alcanzar la meta de finalizar mis estudios, son mi familia. Por ello, quiero dar las gracias a mis padres. A mi madre Toñi, que no sólo es mi madre, sino que también es mi mejor amiga. Una persona que se ha esforzado, luchado y sacrificado para que todos sus hijos tengan una carrera, algo que ella hubiese querido para si, pero que nunca pudo tener por cuidar de los demás. Por todas las risas que nos pasamos juntas, cómo no se enfada cuando nos metemos y nos reimos con ella porque se inventa palabras y por sus chistes malos. A mi padre Jesús, que al igual que mi madre siempre ha trabajado mucho, que siempre me trae algún capricho cuando sale con mi madre a hacer la compra, que me calienta mis pies de hielo glacial y me hace unos maravillosos masajes en los pies y en la espalda.

Quiero dar las gracias también a mis hermanos por toda la diversión y consejos que han aportado a mi vida. A mi hermano Raúl (o como yo le decía cuando era pequeña: mi hermano preferido), por todas las veces que me ha acompañado a estudiar a la biblioteca o me ha venido a buscar, por reírse de mis chistes malos aunque me diga que no me dedique al humor y que mucho menos vaya al Club de la Comedia y por recordarme que siempre pienso que no voy a conseguir aprobar nada o sacar el proyecto adelante y que al final siempre lo consigo todo. A mi hermano Javi, que me ha enseñado una cosa muy importante en la vida: reírse de uno mismo. Por su sentido del humor y porque me enseña, no sólo, a no tomarme demasiado en serio a mi misma, sino también a las personas absurdas que provocan situaciones absurdas y con las que nos vemos obligados a cruzarnos todos los días. A mi hermano Ismael (el pequeñín que me saca una cabeza y media) porque aunque es un chico serio, aguanta todas las bromas de sus hermanos mayores y porque, aunque al final tuve que repetir esas primeras pruebas llevadas a cabo en este proyecto, me las vigiló cuando por obligaciones ajenas tuve que ausentarme mientras se realizaban. También quiero dar las gracias a mi cuñada Eva por cuidar de mi hermano Javi y por hacer que las celebraciones familiares sean aún más divertidas.

Por supuesto, no puedo olvidarme de dar las gracias a mi tutor en este proyecto, Francisco de Borja Rodriguez, por toda su ayuda, por resolverme todas mis dudas amablemente, por todo el tiempo invertido en ello y porque desde el principio tuvo confianza en mi trabajo a pesar de mi falta de experiencia y conocimientos en algunos

de los terrenos que aborda el proyecto. A Jesús Díaz Vico, por toda la amabilidad y dedicación con la que ha resuelto mis dudas de una forma tan detallada y por siempre sacar tiempo en resolverlas aún teniendo otras obligaciones y responsabilidades lejos de la UAM. Cuando gente más preparada que uno, como lo son Paco y Jesús, ponen tanta confianza en ti, quieres dar lo mejor de tu trabajo y esfuerzo para no decepcionarles. También quiero acordarme del resto de los profesores de la carrera.

Quiero dar las gracias a Olga, mi amiga/hermana de toda la vida, por estos 17 o 18 años de amistad, por haber sido siempre tan leal a mi, por no haberte avergonzado nunca de ser mi amiga, por no haberme dado nunca de lado, por nunca dejarte convencer por nadie para alejarte de mi y por explicarme algunas cosas que me cuestan entender de las personas. A Milena, una amiga a la que no conozco desde hace demasiado tiempo y que sin embargo, se ha convertido en uno de mis principales apoyos durante el año pasado y lo que llevamos de este. Por ser una de las personas más positivas, divertidas y directas que conozco y por ese viaje que hicimos las tres. A Najary que aunque hace mucho que no nos vemos tengo muy buenos recuerdos del tiempo que pasábamos en el centro cultural, los gublins y el chocolate.

A todos los compañeros de clases, de prácticas y amigos con los que he compartido estos años de universidad. Me ha llevado mucho tiempo decidir poner sus nombres o no, pero realmente me he sentido tan querida por tanta gente, que si tuviese que poner los nombres y por qué me siento agradecida hacia ellos, es posible que me ocupase demasiado espacio y aún así me olvidase de alguien. Quiero acordarme de algunas de las compañeras con las que me he sentado en clase y compartido prácticas en estos dos últimos años, por aprender juntas y por respetar y tener en cuenta mis ideas cuando eran buenas y no menospreciarme cuando eran malas. Agradecer a todas las personas que en su momento se acordaron de mi en los agradecimientos de sus proyectos por las cosas bonitas que dijeron sobre mi y la certeza de que mis sentimientos hacia ellos son muy similares. A la gente que con su ejemplo hacen que queramos ser mejores ingenieros y personas. Quiero dar las gracias a las personas que han colaborado en el test. También las personas con las que compartí la hora que para mi era la más divertida: la de la comida. Siento una gran gratitud hacia esos amigos que nunca me exigen más de lo que me dan, pues en realidad siento que me dan más de lo que yo doy a ellos, más allá de lo material, como persona. Por las personas que van más allá de las apariencias, que se forman sus propias ideas, que no le siguen la corriente a todo el mundo, que toman partido. Y en general a todas las personas que me han ayudado a superar la casi enfermiza timidez con la que empecé la uni, por haberme apoyado tanto, respetado, ayudado a integrarme, a conocer a las personas y dejar que ellas me conozcan a mi para que vean que soy tan normal como el resto.

Por último, me gustaría acordarme de las personas que he conocido durante este pasado 2013 que me han ayudado a mejorar como profesional y como persona.

Índice de contenido

Índice de Figuras

Índice de tablas.

Glosario

AA: *Autoridad de atributo.*

Entidad que expide un tipo de certificados llamados certificados de atributos.

AES: *Advanced Encryption Standard.*

Estándar de cifrado simétrico por bloques publicado por el NIST en el año 2001.

API: *Application Programming Interface.*

Conjunto de funciones o métodos que ofrece una biblioteca como capa de abstracción para ser usados por otros software.

ARM: *Advanced RISC Machine.*

Marca de procesadores de la empresa ARM Holding, que es una empresa de semiconductores, cuyo principal negocio consiste en los procesadores y cuyos procesadores permiten una implementación muy pequeña usando bajas cantidades de potencia.

ART: *Android RunTime.*

Nueva máquina virtual lanzada de forma experimental por Android en su última versión (KitKat).

ASN.1: *Abstract Syntax Notation One.*

Es una norma para representar datos independientemente de la máquina que se esté usando y sus formas de representación internas.

C: *Country (Pais).*

Campo de un certificado o de un CSR que indica el país del que procede la persona o entidad que solicita el certificado o la localización de la entidad que emite los certificados.

CA: *Certificate Authority (Autoridad Certificadora).*

Entidad que se encarga de expedir los certificados solicitados por una persona, servidor o cualquier entidad que lo solicite.

CN: *Common Name (Nombre común).*

Campo de un certificado o CSR que especifica el nombre de la persona, servidor o entidad que solicita el certificado, o bien, idem de la identidad certificadora.

CRL: *Certificate Revocation List (Listas de revocación de certificados).*

Lista con los números de serie de los certificados con la fecha y los motivos por los que se han revocado. Está firmada digitalmente por una CA e incluye el nombre del emisor, la fecha en el que la lista fue creada, la fecha en el que se prevé que se va a emitir el próximo CRL y la entrada para cada certificado revocado.

CSR: *Certificate Signing Request (Certificado de petición de firma).*

Certificado que permite a una persona, servidor o entidad realizar la solicitud de un certificado a una CA.

DEA: *Data Encryption Algorithm.*

Algoritmo de cifrado simétrico por bloques, escogido en 1977 como un Estándar de Procesamiento de Información Feceral (Federal Information Processing Standard o FIPS) por el NIST.

DER: *Distinguish Encoding Rule*.

Formato binario de un certificado usado principalmente en Java.

DES: *Data Encryption Standard.*

Estándar de cifrado simétrico por bloques. Es otra forma de llamar a DEA.

DN: *Distinguised Name*.

Tipo de notación de un certificado que se compone de diversos campos para identificar a la persona o entidad dueña de un certificado. Estos campos son CN (Common Name), OU (Organizational Unit), O (Organization) y C (Country).

DSA: *Digital Signature Algorithm.*

Algoritmo de Firma digital del FIPS propuesto por el NIST y con un alto tiempo de cómputo.

DTLS: *Datagram Transport Layer Security.*

Protocolo que permite a las aplicaciones cliente/servidor comunicarse de manera que se eviten las escuchas no deseadas, accesos no permitidos, o modificación de mensajes.

DVM: *Dalvik Virtual Machine.*

Máquina virtual que utiliza la plataforma Android. Permite ejecutar aplicaciones programadas en Java.

FIPS: *Federal Information Processing Standard.*

Son estándares anunciados públicamente desarrollados por el gobierno de los Estados Unidos para la utilización por parte de todas las agencias del gobierno no militares y por los contratistas de su gobierno.

GID: *Group Identifier.*

Identificador de grupo usado en plataformas de tipo UNIX que permite organizar a los usuarios por grupos.

GSM: *Global System for Mobile Communication.*

Es un estándar de comunicaciones móviles (2G) y el más extendido hasta la actualidad.

JCA: *Java Cryptography Architecture (Arquitectura de Criptografía de Java).*

Interfaz de programación de Java que proporciona una serie de servicios como un juego de APIs para firmas digitales, funciones resumen, certificados, validación de certificados, generación de claves y muchos otros servicios de seguridad.

JCE: *Java Cryptography Extensions (Extensión de Criptografía de Java).*

Es una extensión criptográfica a JCA que añade servicios que años anteriores no fueron posible añadir debido a las restrictivas políticas de seguridad de E.E.U.U.

JSSE: *Java Secure Socket Extension (Extensión Segura de Sockets de Java).*

Interfaz de Java que permite establecer comunicaciones seguras a través de internet.

LTE: *Long Term Evolution.*

Estándar de comunicaciones móviles (4G), evolución de 3G.

MAC: *Message Authentication Code.*

Una función hash que utiliza dos parámetros de entrada: un mensaje y una clave secreta y se usa para comprobar la autenticidad del origen de un mensaje.

MDC: *Modification Detection Code.*

Función hash que utiliza un parámetro de entrada (el mensaje) y se usa para verificar la integridad del mensaje.

NIST: *National Institute of Standard and Technology.*

Es una agencia de la Administración de Tecnología del Departamento de Comercio de los Estados Unidos La misión de este instituto es promover la innovación y la competencia industrial en Estados Unidos.

O: *Organization (Organización).*

Campo del DN de un certificado o de un CSR que identifica a la organización a la que se adscribe la persona, servidor o entidad que solicita un certificado.

OU: *Organizatonal Unit (Unidad Organizacional).*

Campo del DN de un certificado o un CSR que sirve para identificar a una única organización dentro de una unidad de organizaciones.

PEM: *Privacy Enhanced Mail.*

Formato para codificar certificados en Base64.

PGP: *Pretty Good Privacy.*

Es un tipo de certificado que se usa principalmente para cifrar y firmar digitalmente mensajes de email y archivos.

PKI: *Public Key Infraestructure (Infraestructura de Clave Pública).*

Una PKI es un conjunto de procedimientos, aplicaciones y servicios que permiten proveer un nivel adecuado de seguridad.

RA: *Registration Authority (Autoridad de Registro).*

Es un componente opcional que se encarga de todas las tareas administrativas que en general podrían ser llevadas a cabo por una CA.

RC5: *Rivest Ciphering 5.*

RC5 es un cifrado simétrico por bloques que fue creado por Ronald Rivest en 1994. RC5 se compone de tres elementos: expansión de clave y algoritmo de cifrado y descifrado.

RISC: *Reduced Instruction Set Computer.*

Es un tipo de arquitectura computacional usado en el diseño de CPU generalmente usado en microprocesadores o microcontroladores.

ROM: *Read Only Memory.*

Es un medio de almacenamiento utilizado en ordenadores y dispositivos electrónicos, que permite sólo la lectura de la información y no su escritura, independientemente de la presencia o no de una fuente de energía.

RSA: *Rivest, Shamir, Adleman.*

Algoritmo de cifrado asimétrico de bloques desarrollado en 1977 por Ron Rivest, Adi Shamir y Len Adleman en el MIT.

SPKI: *Simple Public Key Infraestructure.*

Es un tipo de certificado que establece un vínculo entre la clave pública de una entidad y un permiso o autorización. Su propósito es proveer permisos o dar el poder de conceder permisos a terceros.

SSL: *Secure Socket Layer (Capa de conexión segura).*

Es un protocolo que permite establecer una conexión segura a través de internet.

TDEA: *Triple Data Encryption Algorithm.*

Algoritmo de cifrado simétrico que se creó para subsanar los problemas de seguridad del cifrado DES.

TDES: *Triple Data Encryption Standard.*

Es el estándar del Triple DES u otra forma de llamar al TDEA.

TLS: *Transport Layer Security (Seguridad de la capa de transporte).*

Es una versión mejorada del protocolo SSL.

UID: *User Identifier.*

Representación de los usuarios en un sistema UNIX.

UMTS: *Universal Mobile Telecomunication System.*

Sistema de comunicaciones móviles (3G) que es la evolución de 2G.

Capítulo 1:

Introducción

1. INTRODUCCIÓN

En este proyecto final de carrera se desarrolla una aplicación para la obtención de una identidad digital por parte del usuario que haga uso de dicha aplicación.

Para la obtención de una identidad digital se ha implementado además, un protocolo que permite solicitar con cierto nivel de seguridad un certificado digital que identifique al usuario. El protocolo implementado en este proyecto, se basa en una mejora realizada sobre un protocolo de registro llamado "*EBIA*" *(Email Based Identification Authentication)*[1][2][3]. En ese protocolo se envía a un usuario un ticket o un enlace a una dirección de correo electrónico y este enlace tiene que ser verificado por el usuario. El principal problema es que ese email que se envía al usuario puede ser interceptado por un atacante, por lo que para incrementar la seguridad del protocolo se envía el código también por un canal seguro y en la aplicación se compara el código enviado por el servidor de correo con el enviado por el canal seguro.

En el caso de este proyecto, no se usa un servidor de correo electrónico, si no que se simula el comportamiento que tendría el protocolo usando un servidor de SMS. Se simula, porque usar un servidor de SMS real requiere la contratación de unos servicios telefónicos. Por tanto, se ha preferido implementar el protocolo sin hacer uso de los mismos. No obstante, en el *capítulo 4*, apartado *4.2* y en el *anexo D*, se explica como se podría implementar usando realmente un servidor de SMS.

La obtención de un certificado digital como método para identificar al usuario, permite que este pueda autenticarse en un sistema haciendo uso del mismo. Pero un certificado digital puede ser muy útil en muy diversos ámbitos, ya que también puede ser usado para firmar digitalmente un documento, firmas grupales, cifrar información en correo electrónico y abrir una infinidad de posibilidades en el entorno de la seguridad. Esto se debe a que los certificados digitales cumplen un estándar, en el caso de este proyecto se usan certificados X.509 que se definen en el apartado *2.3.4.1.*

La aplicación se ha desarrollado para dispositivos que dispongan del sistema operativo Android, por ello los lenguajes utilizados son Java y XML.

1.1. MOTIVACIÓN

El aumento del uso de tecnologías móviles que existe actualmente, hace que se haya creado un gran mercado en torno al uso y creación de aplicaciones. Existe un gran número de aplicaciones de todo tipo y para todo tipo de usos.

La cantidad de dispositivos que usan el sistema operativo Android es significativamente mayor a la cantidad de dispositivos que usan otros sistemas operativos, bien por el precio, bien por la calidad, etc. por ello, y por la facilidad y accesibilidad a documentación relacionada con Android, al tratarse este de un sistema operativo de libre acceso, hace que sea muy útil para un usuario cierto tipos de aplicaciones y a su vez, resulta fácil para un desarrollador crear una aplicación para este sistema operativo.

La principal motivación para el desarrollo de la aplicación que se ha llevado a cabo en este proyecto, es la falta de aplicaciones existentes actualmente para la obtención de identidades digitales.

Aunque existen aplicaciones que permiten obtener identidades digitales, todas requieren acercarse a un centro de registro para obtenerlas, siendo la aplicación útil únicamente para obtener un código de activación. El hecho de tener que desplazarse se debe a la necesidad que tiene el usuario de identificarse ante una autoridad que determine la identidad de la persona. Se analizan algunos casos en el apartado *2.4.*

Por ello, otro de los motivos para el desarrollo de la aplicación para la obtención de identidades digitales, es la implementación de un protocolo que ofrezca cierto grado de seguridad y que permita la obtención de una identidad digital sin que el usuario tenga que desplazarse para identificarse.

Como se ha comentado, una identidad digital permite autenticarte en determinados servicios o plataformas. El hecho de que se haya elegido el certificado digital como método para autenticarse en lugar de otros métodos como las contraseñas, se debe a que estas pueden implicar problemas de seguridad si son débiles, si se produce suplantación de un usuario o se puede producir una revelación de contraseñas, pero los certificados evitan muchos de estos problemas.

Otra motivación para llevar a cabo este proyecto es comprobar que se puede llevar a la práctica la implementación de un protocolo que no se ha realizado con anterioridad en una plataforma como Android y comprobar que sus funciones realmente pueden ser realizadas por un sistema real.

Además, se quiere mostrar la utilidad que puede tener una aplicación de este tipo si una entidad utilizase este tipo de tecnología para la identificación de sus usuarios. Por ejemplo, una entidad podría expedir sus propios certificados para que los usuarios de

esa entidad hiciesen uso de ellos para identificarse y acceder a sus servicios por internet.

1.2. OBJETIVOS

El principal objetivo de este proyecto es el desarrollo de una aplicación en Android que haga uso de una estructura *PKI* orientada a la consecución de una identidad digital por parte de un usuario sin que este tenga que desplazarse, pero que además, ofrece un grado de seguridad, dando cierta garantía de que el usuario que solicita el certificado es realmente la persona que lo va a recibir. Para ello, se requiere la creación de una Autoridad Certificadora y conocer los mecanismos de seguridad que permitan la implementación de la estructura.

Se quiere además, dadas las motivaciones para la creación de este tipo de aplicación, llevar a la práctica un protocolo de seguridad que permite la obtención de un certificado y que este certificado se pueda utilizar para identificar a una persona. Por ello, se pretende que este certificado obtenido sea usado por el usuario para autenticarse en una plataforma, para lo que se quiere construir un sistema que le permita hacerlo.

En resumen, el objetivo primordial es la implementación real de este protocolo ideado únicamente de manera teórica y comprobar que es implementable en una plataforma de amplia extensión como lo es Android.

Para la consecución de estos objetivos principales, se tiene además, que haber cumplido una serie de pre-requisitos o haber alcanzado una serie de objetivos secundarios pero previos a los objetivos principales. Se podría considerar entre estos sub-objetivos los siguientes:

- Adquirir cierto nivel de conocimientos del desarrollo de aplicaciones en Android, así como en seguridad. Con más precisión, se deben de conocer qué librerías de seguridad posee Android para la implementación de ciertos servicios de seguridad y en general qué librerías existen en Java para una implementación de seguridad.
- Adquisición de conocimientos de protocolos de seguridad básicos.
- Conocer cuales mecanismos de seguridad existen para poder implementar una aplicación que cumpla ciertos requisitos de seguridad.
- Conocer que herramientas se pueden utilizar para implementar una PKI.

Además, se pretende que la memoria de este proyecto aquí escrita, sirva como guía para que cualquier persona interesada en la realización de una aplicación similar a la desarrollada pueda llegar a reproducir o al menos adquirir los conocimientos y los

medios necesarios para reproducir el trabajo expuesto o uno que requiera características similares de seguridad.

Una vez que el usuario pueda obtener un certificado digital con cierto grado de seguridad, haciendo uso del protocolo que se desarrolla en este proyecto y que este certificado pueda ser utilizado para acceder a una plataforma, se habrá cumplido los objetivos para los cuales ha sido desarrollado este proyecto.

1.3. METODOLOGÍA

Para la realización de este proyecto no se ha realizado una metodología en particular, sino que a partir de un objetivo principal, se ha seguido una línea secuencial de tareas hasta la finalización del proyecto.

En primer lugar, se ha realizado un aprendizaje del contexto en el que se debería de desarrollar la aplicación [41][43][51], conociendo los sistemas de seguridad que rigen las tecnologías actualmente[5], para posteriormente aplicar estos conocimientos a la realización de la aplicación.

Para el desarrollo de la aplicación se ha fijado como objetivo principal, la obtención de un certificado digital que identifique al usuario. Para ello, se han seguido una serie de pasos:

1.- Desde la propia aplicación tiene que ser posible la creación de un certificado de solicitud de firma (Certificate Signing Request) que se envía a un servidor para que este devuelva al usuario el certificado solicitado y que será el que le identifique (esto será explicado con más detalla en el *capítulo 2*). La creación del certificado de solicitud de firma se realiza de forma transparente al usuario por lo que este no será consciente de en qué momento se crea. Pero para crear un certificado de solicitud de firma, un usuario tiene que rellenar una serie de campos que serán incluidos en este certificado. Se emplean CSRs cuando la entidad que solicita un certificado es diferente a la entidad que los emite. Cuando no se emplea un CSR para que sea firmado por una CA, se generan certificados autofirmados, los cuales no acreditan su identidad ante otras entidades.

2.- Implementación de un protocolo que permite al usuario la obtención de un certificado sin necesidad de que este tenga que desplazarse a ningún lugar para su identificación. El uso que hace la aplicación y el protocolo de los SMS se ha modelizado, pues como se ha comentado, la implementación real requiere la contratación de unos servicios telefónicos. Sin embargo, se explica cómo podría realizarse en el apartado *4.2* y en el *anexo D*.

Una vez conseguido esto, se puede considerar que se cumple con el objetivo principal de la aplicación, pero se ha considerado necesario que se demuestre el uso práctico que puede tener para un usuario la obtención de este certificado.

3.- Como demostración y validación del uso que se le puede dar a la aplicación y a los certificados obtenidos por el usuario, se ha desarrollado en la aplicación otra función que permite al usuario autenticarse en un sistema interactivo simulado. La autenticación como se explicará a lo largo de este proyecto se realizará en un servidor mediante la verificación de una firma digital hecha desde la aplicación con el

certificado del usuario y de forma transparente a este y mediante la verificación de que el certificado del usuario ha sido expedido por una Autoridad Certificadora que, como se explicará posteriormente, es la encargada de firmar las solicitudes de certificado de los usuarios. Además, se ha añadido a la aplicación la capacidad de que el usuario pueda eliminar un certificado del cual ya no vaya a hacer uso. Es decir, la aplicación permite al usuario el manejo de certificados.

4.- Por último, para comprobar el rendimiento del sistema se ha realizado una serie de pruebas:

- Realización de un grupo de 100, 500 y 1000 iteraciones para medir el tiempo medio y la desviación típica que tarda la aplicación desde que solicita un certificado hasta que lo recibe, es decir, el tiempo que tarda en realizar todo el protocolo (explicado en el apartado *4.3.1*), para un tamaño de clave de 1024, 2048, 3072 y 4096 bits.

- Realización de un grupo de 100, 500 y 1000 iteraciones para medir el tiempo medio y la desviación típica que tarda la aplicación desde que se genera el CSR hasta que se recibe el certificado para un tamaño de clave de 1024, 2048, 3072 y 4096 bits.

- Realización de un grupo de 100, 500 y 1000 iteraciones para medir el tiempo medio y la desviación típica que tarda la aplicación en realizar el protocolo, introduciendo un tiempo aleatorio entre cada muestra para comprobar el comportamiento del sistema. Otra vez el tamaño de clave es de 1024, 2048, 3072 y 4096 bits.

- Realización de un grupo de 100, 500 y 1000 iteraciones para medir el tiempo medio y la desviación típica desde que se genera un CSR hasta que se obtiene un certificado, introduciendo un tiempo aleatorio entre cada muestra, con tamaños de clave de 1024, 2048, 3072 y 4096 bits.

- Realización de un grupo de 100, 500 y 1000 iteraciones para medir el tiempo medio y la desviación típica de lo que tarda la aplicación en realizar la autenticación de un usuario.

- Realización de un grupo de 100, 500 y 1000 iteraciones para medir el tiempo medio y la desviación típica que tarda la aplicación en realizar la autenticación, introduciendo un tiempo aleatorio entre cada iteración.

1.4. CONTRIBUCIÓN

La principal contribución de este proyecto y que marca la diferencia en relación a otras aplicaciones similares, es la obtención por parte de un usuario de un certificado digital sin necesidad de desplazarse pero aportando un grado de seguridad y aportando la garantía de que la persona que recibe el certificado es la que lo solicita.

Esta aplicación puede usarse para autenticarse en un sistema de una forma diferente a la manera habitual, usando certificados en lugar de contraseña como la mayoría de las aplicaciones.

Una característica del protocolo desarrollado en el proyecto, es que en un futuro el uso de certificados podría sustituirse por el uso de contraseña, pero la seguridad del protocolo seguiría siendo el mismo.

1.5. ORGANIZACIÓN DE LA MEMORIA

Este documento se distribuye en 6 capítulos según como se expone a continuación:

Capítulo 1: Introducción. En este capítulo se realiza una introducción que explica a muy grandes rasgos las características de este proyecto: cuál es la motivación para el desarrollo de una aplicación para el uso de identidades digitales, los objetivos que se persiguen, la metodología llevada a cabo y que contribución aporta este proyecto.

Capítulo 2: Estado del arte. En este capítulo se estudia la seguridad de Android y qué vulnerabilidades tiene. También se analiza los sistemas de seguridad que existen dentro del mundo de las comunicaciones en general, que métodos existen para garantizar la seguridad en una comunicación, en qué consiste una identidad digital y otros métodos para identificar a una persona a través de internet y para la obtención de una identidad digital de una forma segura, etc., es decir, se estudia el contexto en el que se va a desarrollar la aplicación. Por último, se realiza un análisis comparativo con otras aplicaciones existentes actualmente para la obtención de identidades digitales y gestión de certificados.

Capitulo 3: Metodología y uso de las herramientas. En este capítulo se realiza un análisis de cómo se desarrolla una aplicación para Android y cuáles son las herramientas que se usan para el desarrollo de una aplicación. Además se explica que librerías existen en Android y en Java que permite realizar funciones y aplicaciones relacionadas con la seguridad. También se explica que herramientas permiten el uso y gestión de certificados digitales y para la obtención de una identidad digital.

Capitulo 4: Diseño y desarrollo de la aplicación. En este capítulo se explica cómo se ha llevado a cabo el diseño y desarrollo de la aplicación para el uso de identidades digitales. También se explica la estructura y el sistema de los que hace uso la aplicación y que posibilita la obtención y uso por parte del usuario de una identidad digital.

Capítulo 5: Resultados. En este capítulo se muestra la aplicación resultante y se analiza la aplicación y el sistema creado. También se realiza un análisis de los tiempos que tarda la aplicación en la obtención de cada uno de los datos principales (certificado, acceso a un sistema después de autenticarse, etc.)

Capitulo 6: Conclusiones y trabajo futuro. En este capítulo se expresan las conclusiones a las que se ha llegado tras la realización del proyecto y las posibles mejoras que se pueden realizar en el futuro.

Esta memoria, está organizada de tal manera que si un lector no tuviese ningún conocimiento sobre cómo desarrollar un sistema de seguridad en el entorno tecnológico actual podría adquirir cierto nivel de conocimiento si su lectura se comenzase en el *capítulo 2*.

Si dicho lector, tuviese cierto grado de conocimiento teórico sobre seguridad, pero no tuviese alguna idea sobre que herramientas podría utilizar para desarrollarlo, podría empezar su lectura a partir del *capítulo 3*.

Si tal lector, ya tuviese conocimiento teórico y además supiese que herramientas puede utilizar para implementar un sistema, pero tuviese curiosidad en cómo se ha implementado este sistema en particular, podría empezar por la lectura del *capítulo 4*.

Para cualquier persona que tenga curiosidad sobre los resultados obtenidos y que esté familiarizado con Android y seguridad, puede consultar directamente el *capítulo 5*.

La persona que quiera saber que posibles mejoras o quiera adquirir nuevas ideas sobre la implementación de un sistema de seguridad en el entorno de Android, puede consultar el *capítulo 6*.

Además, para ciertos detalles técnicos o teóricos que no se muestran en estos capítulos, se encuentra esta información, de manera adicional, en los anexos de este proyecto.

Capítulo 2:

Estado del arte y contexto del proyecto

2. ESTADO DEL ARTE Y CONTEXTO DEL PROYECTO

2.1. INTRODUCCIÓN

A lo largo de este capítulo se realizará un análisis del contexto en el que se va a desarrollar la aplicación de la que trata este proyecto. Se estudiarán cuales son las estructuras y los mecanismos de seguridad de los que se podría valer una aplicación para la obtención de una identidad digital basada en certificados digitales.

Para empezar se va a realizar un estudio sobre Android, sus versiones, como se estructura su seguridad y que posibles vulnerabilidades presenta.

Android es el sistema operativo más difundido actualmente y esto se debe a la utilización de software libre y de código abierto. El aumento de smartphones y en particular los que usan el sistema operativo Android ha hecho que también hayan aumentado los problemas de seguridad, de ahí la necesidad de realizar un estudio de qué mecanismos de seguridad pueden necesitarse para la obtención de un certificado.

A continuación, en este capítulo, se estudiarán algunos conceptos teóricos en los que se basa la seguridad de las tecnologías actualmente.

Entre otros conceptos, se verá que en una comunicación segura **cada entidad posee una clave privada** que sólo es conocida por la propia entidad **y una clave pública** que permite la identificación de dicha entidad y es conocida por todos los miembros de una comunicación. La relación matemática entre las dos claves es tal que una acción realizada con una clave puede estar relacionada a la otra clave pero sin revelar los datos de esta.

Los certificados digitales son documentos digitales que permiten establecer una unión entre el nombre o identidad de una entidad y su clave pública, o dicho de otra forma, **son la parte pública de una identidad digital.**

La aplicación móvil desarrollada en este proyecto, forma parte de una *infraestructura de clave pública o PKI*.

Una PKI es un conjunto de procedimientos, aplicaciones y servicios que permiten proveer un nivel adecuado de seguridad. Una infraestructura PKI consiste en:

- Una política de seguridad. Son normas que permiten mantener cierto nivel de seguridad.

- Una Autoridad Certificadora (CA). Es el emisor de certificados digitales.

- Un sistema de administración de certificados. Esto se lleva a cabo mediante la Autoridad Certificadora o mediante la Autoridad de Registro que es la que se encarga de registrar los datos de los usuarios.

- Un conjunto de aplicaciones que hacen uso de la tecnología PKI.

La infraestructura PKI es una tecnología de servicios de seguridad y proporciona autenticación basada en una combinación de criptografía de clave pública (o asimétrica) y clave privada (o simétrica). También proporciona otros servicios de seguridad que incluyen la confidencialidad de datos, la integridad de los datos y el manejo de claves, etc. PKI representa la integración de la criptografía de clave pública para el uso de firmas digitales y el manejo de claves y la criptografía de clave privada[1] usada para cifrado. Las bases de una PKI está definida en la recomendación ITU-T X.509. [4]

X.509 es una infraestructura que define un marco para proveer servicios de autenticación y define una estructura de certificado y protocolos de autenticación que se usan en una gran variedad de contextos. [5][6]

Como parte de esta estructura PKI, a parte de la aplicación móvil, en este proyecto se va a desarrollar un protocolo de registro y la autenticación de los usuarios. Esto será explicado en capítulos posteriores.

Como ejemplo de una estructura PKI se muestra la siguiente imagen:

[1]La criptografía de clave pública y la criptografía de clave privada son sinónimos de criptografía asimétrica (se estudia en el apartado 2.3.1.2) y criptografía simétrica (se estudia en el apartado 2.3.1.1). La firma digital es un mecanismo criptográfico que se utiliza para identificar a una persona (apartado 2.3.3)

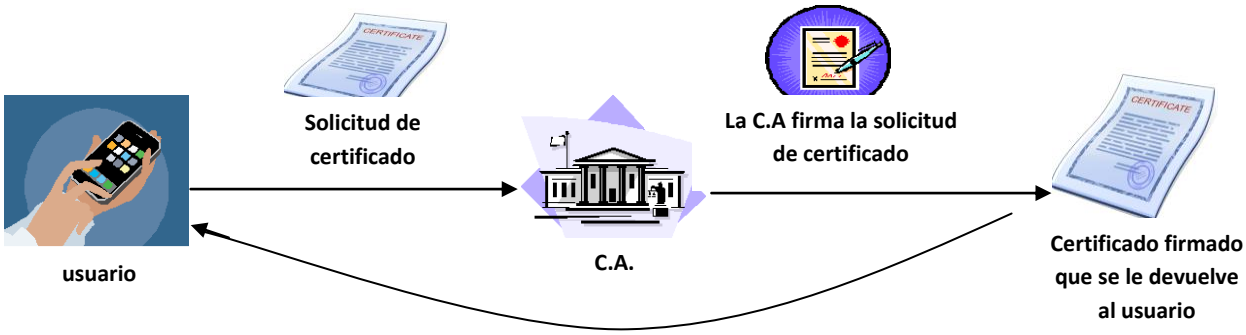


Figura 2.1. Uso de una PKI para la obtención de una identidad digital. Un usuario envía un certificado de solicitud de firma a una CA, la CA firma la solicitud dando como resultado el certificado de usuario y este se envía al usuario, con lo cual el usuario ya tiene una identidad digital basada en certificado digital.

Por último, al final de este capítulo, se hará un estudio sobre las aplicaciones que existen actualmente para la obtención de identidades digitales y se comprobará, que no existen demasiadas aplicaciones que permitan la obtención de identidades digitales, aunque sí que hay bastantes aplicaciones que permiten la gestión de certificados e incluso permiten cifrar y firmar correos electrónicos haciendo uso de certificados digitales.

2.2. SISTEMAS ANDROID

Android es un sistema operativo diseñado por *Android Inc.* (una compañía ubicada en Palo Alto, California) para dispositivos móviles, comprada por Google en el año 2005, que usa código abierto.

En un principio se pensó que Google diseñaría dispositivos móviles que hicieran uso exclusivo del sistema operativo de Android, pero en 2007 se anunció la unión a la *Open Handset Alliance (OHA)*[2] lo que significaba que oficialmente Android se establecería como una plataforma de código abierto y con estándares públicos. Esto permite que cualquier desarrollador haga uso del código y del sistema, extendiéndolo o mejorándolo. De hecho, una persona podría descargarse el código fuente, modificarlo y venderlo instalado en terminales.

Android fue diseñado para ejecutarlo en cualquier clase de dispositivo sin hacer ninguna clase de presunción sobre el tamaño de la pantalla, la resolución, etc. ya que, por primera vez se hace distinción entre el hardware del dispositivo y el software que se ejecuta en él. Su núcleo está diseñado para que sea portable.

El hecho de que Google haya decidido que Android fuera un sistema operativo libre y de código abierto se debe a la filosofía de la compañía con respecto a que la información debe ser universalmente accesible y útil. Aunque Google tiene algunas licencias propietarias (p. e. *Gmail* y *Maps*) y obtiene ciertos beneficios en el Android market *(Google Play)*, sus principales ingresos siguen viniendo de la publicidad. Por tanto, el hecho de que se use código abierto no influye en sus beneficios, a diferencia de lo que ocurre con otras compañías como *Apple.*

El sistema operativo de Android está formado por varias capas y cada una de las capas realiza una función. Las principales cuatro capas en la que se divide Android son:

1. El kernel.
2. Las librerías y la máquina Virtual *Dalvik* (máquina virtual *ART* a partir de la versión KitKat de Android, ver en apartado *2.2.1*).
3. Marco de aplicación.
4. Las aplicaciones.

La arquitectura del sistema operativo de Android se muestra en la siguiente imagen:

[2] Open Handset Alliance es un consorcio formado por un gran número de empresas punteras divididas en cinco categorías: Operadoras, fabricantes de teléfonos, empresas de software, empresas semiconductoras y empresas de comercialización.

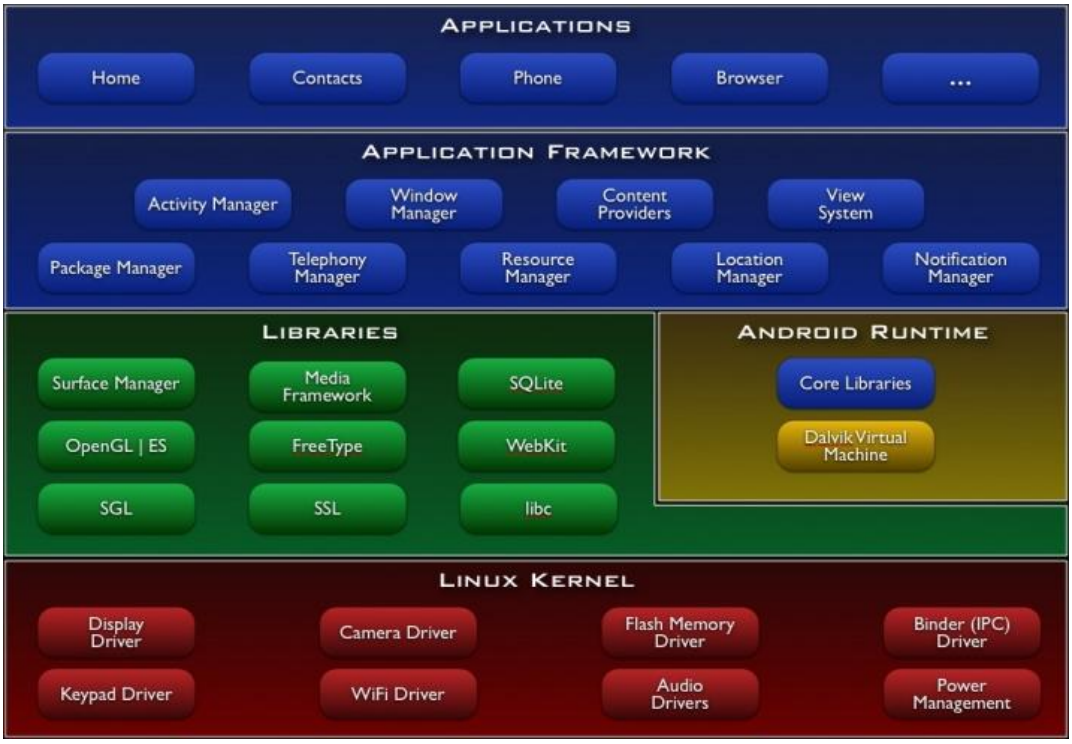


Figura 2.2. Arquitectura del sistema operativo de Android.[3] En color rojo se muestra el kernel, en verde las librerías, en amarillo la máquina virtual, la primera capa de azul pertenece al marco de aplicaciones y la segunda capa de azul pertenece a las aplicaciones.

Las aplicaciones que utilizan Android están implementadas en el lenguaje de programación de Java. Las herramientas de *Android SDK*[4] agrupan el código en un paquete con formato *.apk.*

Android está basado en Linux, de hecho se ejecuta sobre el kernel de *Linux 2.6*. Algunas de las razones por las que se basa en Linux son la portabilidad, la seguridad y las características de Linux.

Linux es una plataforma portable que puede ser compilada en varias arquitecturas hardware. Gran parte de la arquitectura software a bajo nivel está construido sobre código C, lo que permite que sea utilizable en una gran cantidad de dispositivos.

Linux es un sistema fuertemente seguro, y ha sido probado en diversas situaciones inseguras. Todas las aplicaciones de Andorid se ejecutan como procesos separados de Linux con permisos establecidos por el sistema, pero además cada aplicación tiene su propia máquina virtual[5], por lo que el código de una aplicación se ejecuta de forma

[3] Imagen obtenida de la página http://picandocodigo.net/2007/android-la-nueva-plataforma-de-desarrollos-moviles/
[4] Android SDK es el conjunto de herramientas de desarrollo que utiliza Android.
[5] La máquina virtual de Android simula un dispositivo físico que utiliza Android como sistema operativo.

aislada con respecto al código de las demás aplicaciones, en procesos separados, además se permiten varias instancias de la maquina virtual. Esto implica que si una aplicación falla puede ejecutarse aún el resto de aplicaciones que se estén ejecutando.

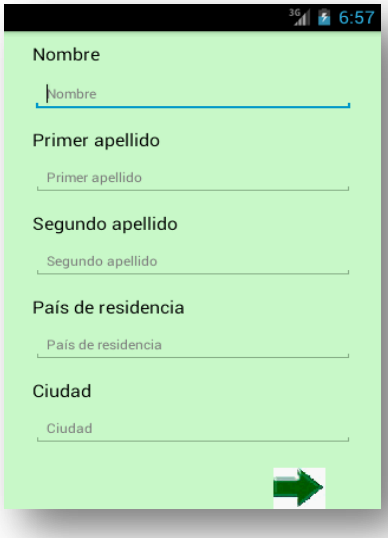
Android aprovecha muchas de las características útiles de Linux, como por ejemplo, soporte para la gestión de la memoria, gestión de potencia y networking en sus aplicaciones. [7]

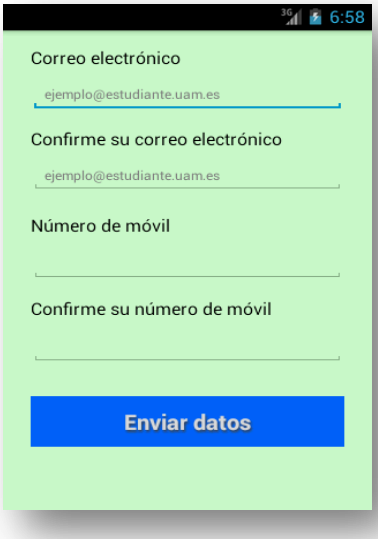
Una aplicación *(.apk)* está formada por diferentes componentes. Cada uno de estas componentes tiene una función específica y está definida por su ciclo de vida que indica cuando cada componente se crea y cuando es destruido.

Hay cuatro tipos posibles de componentes en una aplicación:

- **Actividades**

  Es cada una de las pantallas que forman la interfaz de usuario. Cada aplicación es una agrupación de actividades que hace que la experiencia de usuario sea coherente, si bien, cada actividad es independiente a otra. Como ejemplo, se muestran dos actividades correspondientes a dos pantallas de la aplicación desarrollada en este proyecto.





Figuras 2.3 y 2.4. Dos actividades de la aplicación Android desarrollada en este proyecto. Cada una de las actividades representan una pantalla de la aplicación.

- **Servicios**

  Son componentes que se ejecutan en segundo plano para la realización de acciones que requieran un mayor tiempo de ejecución o que se realizan de

forma periódica. Algunos ejemplos de servicios son los antivirus, las actualizaciones de aplicaciones, servidores web, etc.

- **Proveedores de contenido**

  Los proveedores de contenido almacenan, recuperan y hacen accesibles a las aplicaciones un conjunto de datos, es decir, actúan como repositorios de datos a los cuales las aplicaciones pueden acceder o incluso pueden modificar esos datos si el proveedor del contenido lo permite. Esta es la única forma de compartir contenido entre aplicaciones. Un ejemplo de esto sería los contactos de la agenda telefónica.

- **Receptor de anuncios**

  Es un componente que recibe y reacciona ante anuncios globales de tipo Broadcast. No tienen una interfaz de usuario pero pueden crear una actividad o crear una notificación para informar al usuario. Algunos ejemplos de esto, son el aviso de batería baja o las llamadas entrantes.

En la aplicación desarrollada en este proyecto, sólo se han utilizado Actividades, ya que, dadas las características del mismo y su sencillez en relación a que las acciones más importantes se realizan de forma transparente al usuario, no se requiere el uso del resto de componentes.

La forma de enlazar una componente con otra de forma que la aplicación componga una unidad es usando objetos *"intent"*. Estos objetos inician la siguiente actividad, servicio o anuncio y también pueden recibir resultados desde estos componentes. Por ejemplo, cada vez que una actividad termina el programador, instancia en el código un *"intent"* para que la aplicación pase a la siguiente pantalla.

Una característica particular de Android es que una aplicación puede ejecutar un componente de otra aplicación. De esta forma una aplicación puede iniciar una acción realizada por otra sin necesidad de que sea desarrollada en la primera.

Otro elemento importante para la cohesión de los diferentes componentes y que da unicidad a una aplicación es un archivo *.xml* conocido como el *"manifest"*. En el *manifest* se definen cada uno de los componentes que componen una aplicación, los diferentes permisos que pueden ser necesarios y se especifica el nivel de API que va a utilizar la aplicación a desarrollar. (Como ejemplo de manifest en el *anexo D* se encuentra el manifest de este proyecto)[8]

La interfaz gráfica de las aplicaciones en Android se programa en lenguaje XML y las operaciones y acciones que puede realizar las aplicaciones se programan en JAVA.

Las *API (Application Programming Interface)*, se utilizan para obtener cierto nivel de abstracción haciendo de intermediario entre un usuario y las diferentes operaciones que realiza una aplicación, es decir, representa la capacidad de comunicación entre componentes software. Por ejemplo, se pueden considerar APIs ciertas librerías que un programador puede usar para desarrollar una aplicación que a la vez hacen uso de librerías ya existentes.

Desde su creación el sistema operativo de Android ha tenido varias versiones. Las versiones no dan información sobre el software, sino sobre los niveles de API. Algunas veces, la versión cambia debido a un cambio en las APIs y otras por mejoras hechas gracias a errores detectados en el código.

A continuación se muestra una tabla con las diferentes versiones de Android y sus niveles de API (tabla extendida con las características de cada versión en el Anexo A):

Lanzamiento	Versión	Nivel de API
23 de Sept. 2008	Android 1.0	Nivel de API 1
9 de Feb. 2009	Android 1.1	Nivel de API 2
Cupcake		
Abril 2009	Android 1.5	Nivel de API 3
Donut		
Sept. 2009	Android 1.6	Nivel de API 4
Eclair		
26 de Nov. 2009	Android 2.0	Nivel de API 5
3 de Dic. 2009	Android 2.0.1	Nivel de API 6
12 de Enero 2010	Android 2.1	Nivel de API 7
Froyo		
20 de Mayo 2010	Android 2.2	Nivel de API 8
18 de Enero 2011	Android 2.2.1	
22 de Enero 2011	Android 2.2.2	
21 de Nov. 2011	Android 2.2.3	
Gingerbread		

6 de Dic. 2010	Android 2.3	Nivel de API 9
9 de Feb. 2011	Android 2.3.3	Nivel de API 10
28 de Abril 2011	Android 2.3.4	
25 de Julio 2011	Android 2.3.5	
2 de Sept. 2011	Android 2.3.6	
21 de Sept. 2011	Android 2.3.7	
HoneyComb		
Feb. 2011	Android 3.0	Nivel de API 11
10 de Mayo 2011	Android 3.1	Nivel de API 12
15 de Julio 2011	Android 3.2	Nivel de API 13
20 de Sept. 2011	Android 3.2.1	
Ice-Cream Sandwich		
Oct. 2011	Android 4.0/4.0.3	Nivel de API 14 y 15
Jelly Bean		
27 de Jun. 2012	Android 4.1	Nivel de API 16
23 de Julio y 9 de Octubre 2012	Android 4.1.1/4.1.2	
29 de Oct. 2012	Android 4.2	Nivel de API 17
24 de Julio de 2013	Android 4.3	Nivel de API 18
	KitKat	
31 de Octubre de 2013	Android 4.4	Nivel de API 19

Tabla 2.1. Versiones de Android [9], [10], [48]. Características de las diferentes versiones en el *Anexo A*.

Uno de los problemas que presenta Android es que la programación de las aplicaciones puede depender de la versión del sistema operativo y actualmente la mayoría de estas versiones están conviviendo a la vez. Aunque existen versiones de Android cuyas APIs no varían mucho, por lo que se pueden desarrollar aplicaciones similares para varias versiones, si es verdad que no es lo mismo desarrollar una aplicación para una API de nivel 17 a desarrollar una aplicación para una API de nivel 1.

Las aplicaciones se guardan en memoria *ROM* (Memoria integrada). En esta memoria está la partición del sistema operativo y la partición dedicada a las aplicaciones.

El espacio particular donde los usuarios guardan sus archivos de música, fotos, videos, etc. es la tarjeta *microSD*. Algunos dispositivos no tienen una tarjeta *microSD* sino otro módulo flash de gran tamaño (16 o 32 GB) que a nivel de sistema de archivos actúa como una tarjeta SD. Según los permisos de cada aplicación, estas pueden tener acceso a la tarjeta o no. En cualquier caso, a partir de la versión 2.2 de Android se pueden instalar aplicaciones en las tarjetas SD.

Otros sistemas de archivos que utiliza Android es *yaffs2*[6] *(Yet Another Flash File System2)*. Funciona bien cuando hay espacio libre, pero es lenta si la memoria flash en la que se está escribiendo está llena.

Además, Android puede utilizar otros sistemas de archivos basados en Unix, como son el *ext1* y el *ext2*, etc.

El tipo de sistema de archivos que se utiliza en cada dispositivo es elegido por el fabricante.

Los dispositivos que utilizan Android como sistema operativo, suelen seguir la tendencia actual consistente en que la mayoría de los componentes (procesador, modem, radio, adaptador de pantalla, cámaras, GPS, micrófonos, etc) estén integrados en un solo chip (System on a Chip o *SoC*). Estos tipos de chips se suelen utilizar en sistemas embebidos, que son sistemas de computación usados para realizar una o varias tareas en un sistema de computación a tiempo real. El hecho de que todos los componentes se encuentren en un solo chip ahorra espacio, lo cual es muy útil para dispositivos móviles.

Además, se suelen utilizar procesadores súper escalares. La capacidad alcanzada en los últimos dispositivos creados, tanto tablets como móviles, han igualado y en algunos casos superado la capacidad de los famosos Netbooks. [9]

Los procesadores súper escalares, permiten la ejecución de varias instrucciones a la vez en el mismo ciclo de reloj, por tanto, se mejora la velocidad del dispositivo móvil.

A diferencia de lo que ocurre con los ordenadores personales, en los que la arquitectura y el set de instrucciones lo definen principalmente dos marcas: *AMD* e *Intel*; en la arquitectura y el set de instrucciones de los dispositivos que utilizan Android, ha destacado principalmente la marca *ARM* de la empresa *ARM Holding*, que es una empresa de semiconductores, cuyo principal negocio consiste en los

[6] Yaffs2 es la evolución de Yaffs y Yaffs1. Yaffs es un sistema de ficheros diseñado por Charles Manning para memorias flash NAND. Prolonga la vida de la memoria flash y es robusta a cambio de energía. El tamaño máximo del sistema de ficheros que proporciona es de 8 GiB

procesadores. Desarrollan una arquitectura *RISC (Reduced Instruction Set Computer)* de 32 bits.

ARM ha sido diseñado para permitir implementaciones muy pequeñas, lo que permite que los dispositivos consuman baja potencia. Dado que utilizan una arquitectura *RISC*, presentan características típicas de *RISC*:

1. Un gran archivo de registro uniforme.
2. Una arquitectura de load/store donde las operaciones de procesamiento de datos solo operan en el contenido de los registros, no sólo en el contenido de la memoria.
3. Modos de direccionamiento simple, con todas las direcciones de load/store solo determinadas por el contenido del registro y por los campos de las instrucciones.
4. Los campos de las instrucciones son uniformes y de longitud fija, para simplificar la decodificación de instrucciones. [11]

2.2.1. Seguridad en Android

Se puede afirmar que no existe ningún sistema que sea totalmente seguro. Por tanto, Android no es una excepción, ya que también presenta ciertas vulnerabilidades.

Android fue diseñado con seguridad multicapa que proporciona la flexibilidad requerida para una plataforma abierta. Además, no sólo fue diseñada pensando en la seguridad del usuario, sino también en la seguridad del desarrollador.

A nivel global, Google tiene la capacidad de eliminar remotamente una aplicación no sólo de *Android Market* (actualmente *Google Play Store*), sino también directamente de un dispositivo. La característica de eliminación de una aplicación de forma remota es un control de seguridad que Android posee donde una aplicación peligrosa podría ser eliminada de la circulación activa de una forma rápida y escalable para prevenir de mayores exposiciones a los usuarios. El control de seguridad fue ejercido por primera vez en Junio de 2010 cuando un investigador de seguridad distribuyó una aplicación de prueba que la podía permitir descargar e instalar otras aplicaciones en el dispositivo. [12].

La aplicación diseñada por este investigador, permitía tener control de forma remota sobre el dispositivo en el cual se instalaba. De este modo, el diseñador de la aplicación accedía a los datos de los móviles de las personas que habían descargado esta aplicación. Se distribuyó como dos aplicaciones distintas, aunque hacían lo mismo.

Eran aplicaciones aparentemente sencillas: en una aparecía la típica frase *"Hello world"*, la otra distribución simulaba un avance de la película *"Eclipse"* de la Saga *"Crepúsculo"*. Estas aplicaciones no fueron diseñadas para hacer un uso malicioso, sólo como una prueba y no hacían uso de permisos para acceder a datos privados pero podían acceder al kernel.

Cuando Google fue informado de la vulnerabilidad provocada por estas aplicaciones de prueba, las eliminaron de forma remota, tanto de cada uno de los dispositivos en los que se habían instalado, como de Google Play Store. [13]

A nivel de sistema operativo, tal y como se ha comentado anteriormente, Android se basa en el modelo de seguridad de Linux.

En el sistema operativo de Linux, cada usuario es identificado con un número *(UID)* que tiene una serie de permisos. Por otro lado, un grupo también se identifica con un identificador *(GID)* y también pueden tener una serie de permisos como grupo.

Cada recurso en Linux tiene asignados tres clases de permisos: como dueño, grupo o público. Los permisos que se pueden asignar a un usuario, a un grupo o al público pueden ser permisos de lectura (read, R), de escritura (write, W) y de ejecución (execute, X).

Cuando se instala un paquete en Android, se crea un nuevo identificador de usuario (User ID, UID) distinto a cualquier otro identificador que se haya creado anteriormente y la nueva aplicación se ejecuta bajo esa UID. Todos los datos almacenados por esa aplicación se identifican con esta misma UID, esto es una característica nueva introducida por el sistema operativo Android, ya que en Linux múltiples aplicaciones se ejecutan con permisos de usuario. Además, no hay garantía de que este UID sea utilizado por la misma aplicación en otros dispositivos.

Por tanto, cada aplicación se ejecuta en un proceso diferente y cada ejecución dentro de una *Máquina Virtual Dalvik (Dalvik Virtual Machine, DVM)*[7] separada. Sin embargo, todas las aplicaciones pueden incluir código nativo, que es código que se ejecuta fuera del *DVM* (por ejemplo, acceso del dispositivo a wi-fi) y es compilado para ejecutarse directamente en el procesador de un dispositivo Android, lo cual sin embargo, no altera el modelo de seguridad, debido al sistema de permisos de android (al instalarse una aplicación se le muestra al usuario el acceso que necesita la aplicación a los diferentes datos, de forma que la responsabilidad de que una aplicación pueda acceder a los datos recae en el usuario al elegir instalarse la aplicación o no), así como

[7]Dalvik Virtual Machine, DVM: es una máquina virtual optimizada para dispositivos móviles, específicamente diseñada para ejecutarse rápido en todos los dispositivos que utilizan Android como sistema operativo. Una máquina virtual es un sistema operativo que se ejecuta dentro de otro sistema operativo.

a la separación mediante UIDs, impide que afecte al comportamiento general del dispositivo.

El hecho de que se utilice UID diferentes para cada proceso previene:

- Que un usuario pueda leer los archivos de otro usuario.
- Se asegura que un usuario no agote la memoria de otro usuario.
- Que un usuario no agote los recursos de otro usuario.
- Que un usuario no agote los dispositivos de otro usuario.

Esto es debido, a que como se ha explicado anteriormente un usuario con UID específico, tiene unos permisos determinados para ese usuario que impiden que tengan acceso a los recursos de otro usuario, ni que otro usuario acceda a sus recursos a menos que se indique expresamente.

Una aplicación puede compartir datos con otra aplicación haciendo que compartan también la misma UID, esto provoca que el núcleo de Linux los identifique como la misma aplicación. Esto se consigue declarándolo en el manifiesto mediante un string que identifica al UID. [14]

Cabe destacar que Android ha lanzado de forma experimental una nueva máquina virtual llamada *ART (Android RunTime)* en la última versión de Android *(KitKat)*. La diferencia que existe con *DVM* es que en *DVM*, la aplicación no se compila hasta que no se haya lanzado por primera vez y entonces se almacena en ese momento. Sin embargo *ART*, cuando se instala una aplicación directamente se realiza una pre-compilación, ocupa más espacio e implica que la aplicación tarda más en instalarse, pero las aplicaciones se ejecutan más rápido y por tanto, se consume menos batería.

Con respecto a la seguridad en el sistema de archivos, Android 3.0 y posteriores proporcionan la posibilidad de realizar un cifrado del sistema de archivos completo por parte del kernel. La clave de cifrado se obtiene a partir de la contraseña del usuario (no se admite el bloqueo de pantalla basado en patrones), previniendo accesos desautorizados.

A nivel de aplicación, los permisos (declarados en el manifest y también llamados permisos del Manifiesto) son componentes centrales en la seguridad de Android y no tienen relación con los permisos del núcleo de Linux mencionados anteriormente. (Ejemplo de manifest de la aplicación desarrollada en este proyecto en el *Anexo D*)

Los permisos permiten acceso a diferentes servicios como internet, componentes de memoria, galerías de fotos, a los datos privados del usuario, etc. Cuando un usuario

quiere descargarse una aplicación observa en la pantalla los permisos que requiere esa aplicación, lo que hace que conozca mejor su comportamiento y pueda decidir si instalar la aplicación resultará perjudicial para su dispositivo.

El problema con los permisos es que no se puede asegurar que una persona lea todos los permisos que requiere una aplicación. Por ese motivo, un desarrollador debe ser conciso a la hora de declarar los permisos.

Existe la posibilidad de que un desarrollador cree su propio permiso, el cual debe ser declarado en el manifiesto. Además existe la posibilidad de categorizar los permisos. Existen cuatro tipos de categorías:

Constante	Valor	Descripción
Normal	0	Permiso que informa de un riesgo de bajo nivel que da a una aplicación acceso a características aisladas del nivel de aplicaciones, con riesgos mínimos para otras aplicaciones, el sistema o el usuario. No requieren una aprobación explícita, aunque el usuario siempre tiene la posibilidad de ver cuáles son los permisos.
Dangerous	1	Un permiso con un riesgo mayor que da acceso a las aplicaciones a datos privados del usuario o proporciona un control sobre el dispositivo que puede impactar negativamente sobre el usuario. Requieren permiso explicito por parte del usuario.
Signature	2	Este permiso sólo se concede si la aplicación solicitada está firmada con el mismo certificado que la aplicación que declara el permiso. Si los certificados coinciden, la aplicación se instala sin preguntárselo al usuario.
signatureOrSystem	3	Siguen la misma norma que los permisos signature, excepto que se le da automáticamente a la imagen del sistema de Android además de solicitar el certificado de la aplicación.

Tabla 2.2. Permisos de android [19]

Como se ve en la tabla, las aplicaciones están firmadas digitalmente, es decir, se les aplica una operación criptográfica que sirve para la identificación del desarrollador de la aplicación (en el apartado *2.3.3* se estudia con más detenimiento en qué consisten las firmas digitales). Esto sirve para probar que una aplicación ha sido realmente creada por un desarrollador, es decir, para identificar a un desarrollador pero no

implica un aumento de la seguridad porque la firma sólo identifica a la persona que desarrolla la aplicación, pero no implica que la aplicación sea segura o que no sea maliciosa. Cada desarrollador genera un certificado digital para firmar las aplicaciones.

Como se verá más adelante, hay dos tipos de certificados: uno expedido por una Autoridad Certificadora y el otro es un certificado autofirmado creado por el propio desarrollador. Aunque las aplicaciones tienen que estar firmadas antes de ser instaladas, Android no requiere que sean firmadas por un certificado expedido por una Autoridad Certificadora, de hecho se suelen utilizar certificados autofirmados. Durante la ejecución de la aplicación, durante el proceso de desarrollo, esto se hace de una forma transparente al desarrollador. Cada vez que éste compile la aplicación, el juego de herramientas de desarrollo genera una clave automáticamente.

Los certificados de las aplicaciones al igual que ocurre con los permisos, sólo se comprueban en el momento en el que se instala la aplicación. [14]

Existen dos formas de firmar una aplicación. Durante el proceso de desarrollo de una aplicación, cada vez que el desarrollador compila y depura su aplicación, se crea lo que se conoce como *"debug key"*. Sin embargo, esta *"debug key"* no debe ser utilizada cuando la aplicación se ha terminado completamente, en ese momento la aplicación debe ser firmada por certificado real. Este certificado puede ser creado por la herramienta *"Keytool"* (se hablará de esta herramienta en el siguiente capítulo) si se utiliza un certificado autofirmado. En la aplicación desarrollada en este proyecto se ha utilizado la clave que se crea durante la compilación y depuración, pero no se ha realizado la firma formalmente, pues esta debe hacerse al finalizar por completo la aplicación y de cara a lanzar la aplicación en el mercado. Sin embargo, el proceso para firmar una aplicación se explicará en el capitulo siguiente.

Otra forma de firmar una aplicación (y en realidad la forma recomendable de hacerlo) es utilizando un certificado expedido por una autoridad certificadora de confianza. Los pasos que se seguirían son los siguientes:

1. Una CA valida la identidad y la autenticidad de un desarrollador como autor de la aplicación o de su contenido.

2. La CA envía al desarrollador un ID de desarrollador específico que se usa para autenticarle cuando desea firmar el código.

3. El desarrollador usa su ID específico para firmar los archivos de su aplicación que luego envían a la CA.

4. El contenido que se vuelve a firmar o autenticar, ya está listo para una distribución de confianza.

Es recomendable que los desarrolladores firmen siempre sus aplicaciones con el mismo certificado y que el periodo de validez sea superior al tiempo de vida que se considera que va a tener la aplicación. Si se necesita usar otro certificado la aplicación deberá recibir otro nombre [15][16]

2.2.2. Vulnerabilidades y limitaciones de Android

A parte del problema de seguridad detectado en 2010, mencionado anteriormente, durante el año 2013 se descubrió una vulnerabilidad en los sistemas operativos de Android que afectó al 99% de los dispositivos. Esta vulnerabilidad permitía reemplazar ciertas *.apk* debido a un fallo en el sistema criptográfico y en las actualizaciones.

Todas las aplicaciones de Android están firmadas digitalmente de la misma forma en la que se explica en el siguiente capítulo, lo que permite identificar al desarrollador de la aplicación. Cada vez que una aplicación necesita ser actualizada, esta tiene que estar firmada digitalmente por el mismo desarrollador, ya que si la aplicación no está firmada por este desarrollador no podría ser actualizada.

La vulnerabilidad radica en un fallo en el sistema de verificación de firma digital de tal forma que la firma seguiría siendo válida, por lo que una aplicación podría ser actualizada a pesar de estar firmada por otro desarrollador.

Este problema podría permitir a un atacante crear una aplicación maliciosa "disfrazada" de una aplicación conocida y confiable y crear problemas en los dispositivos en los que sea actualizado. Al ser una actualización, la aplicación no pide permisos, por lo que en realidad el usuario no podría de alguna forma ser consciente de los problemas que esta aplicación puede ocasionar, ni a que parte del dispositivo puede tener acceso las aplicaciones.

Si esta actualización se hace sobre alguna aplicación simple, como ciertos juegos, no ocasiona gran daño, el problema radica en que se haga uso de algunas aplicaciones como *Gmail* que permite acceso a cuentas del usuario, por lo que un atacante podría conocer los datos privados de la persona que se instala la aplicación.

Para que los usuarios no tuviesen estos problemas, estos no deberían descargarse aplicaciones fuera de *Google Play*, ya que para publicar aplicaciones en *Google Play* se sigue un proceso más seguro y se realizan más controles que con otros métodos de descarga de aplicaciones. Todas las versiones de Android a partir de la versión 1.6 son vulnerables a sufrir este problema. *Samsung S4* es el único dispositivo que tiene un parche para este fallo. [17]

En marzo del año 2014 se detectó un importante fallo en Android que afecta a todas las versiones de Android. Este fallo implica problemas de malware al realizar actualizaciones en el sistema operativo de Android.

Este fallo consiste en que un usuario puede instalarse una aplicación aparentemente inofensiva y que requiere pocos permisos o que no requieren el acceso a datos importantes, pero a la hora de realizar actualizaciones, la aplicación eleva sus privilegios accediendo así a cierta información importante o a ciertas funciones del dispositivo como los mensajes de voz, los credenciales de acceso, mensajes de texto o registros de llamadas y más acciones fraudulentas y peligrosas para el usuario. Actualmente, Google está trabajando para solucionar este fallo.

La forma de evitar este fallo es comprobar quién o quiénes son los desarrolladores de la aplicación (por ejemplo, si es una empresa conocida), comprobar la puntuación que los usuarios le dan a la aplicación en Google Play e informarse sobre la aplicación en otras fuentes de información. [68][69]

Comparando Android con otros sistemas operativos como *iOS*, el número de vulnerabilidades de Android (27) es ocho veces menor que *iOS* (228), es decir, que presenta más debilidades que podrían comprometer la seguridad. En realidad, la mala fama que puede tener Android en relación a *iOS* es debido a que Android suele sufrir más ataques de seguridad (vienen del exterior), ya que a mayor número de usuarios, mayor es la probabilidad de sufrir ataques. [18]

Se muestra a continuación un par de tablas, en las que se muestra el número de vulnerabilidades según el sistema operativo (en la primera tabla) y el número de malware según el sistema operativo (segunda tabla) que se produjeron durante el año 2012 recogido en un informe de seguridad en internet de *Symantec* del año 2013:

Mobile Vulnerabilities by OS

Source: Symantec

Platform	Documented Vulnerabilities
Apple iOS	387
Android	13
BlackBerry	13
Nokia	0
LG Electronics	0
Windows Mobile	2

Tabla 2.3. Vulnerabilidades de los Sistemas Operativos para móviles [49]

Mobile Threats by Device Type in 2012

Source: Symantec

Device Type	Number of Threats
Android malware	103
Symbian malware	3
Windows Mobile malware	1
iOS malware	1

Tabla 2.4. Cantidad de software malicioso según el sistema operativo [49]

2.3. SISTEMAS DE SEGURIDAD EN LAS COMUNICACIONES.

Actualmente existe una gran tendencia a la interconectividad e interoperabilidad entre redes, entre un usuario y una red, entre aplicaciones etc. por lo que cada vez se hace más necesario un sistema de seguridad que en años anteriores sólo se consideraba para la seguridad en sistemas militares o diplomáticos.

Cualquier sistema de información es sensible a ataques de seguridad. Un ataque de seguridad es cualquier acción que comprometa la seguridad de la información perteneciente a una organización.

Los ataques de seguridad se dividen en:

- **Ataques pasivos:** El objetivo del atacante es obtener información que ha sido transmitida. Estos ataques son muy difíciles de detectar porque no supone ninguna alteración de los datos. No es peligrosa en el sentido de que no se modifica la información transmitida, sin embargo puede acarrear problemas de seguridad si la información obtenida se utiliza para el beneficio del atacante.

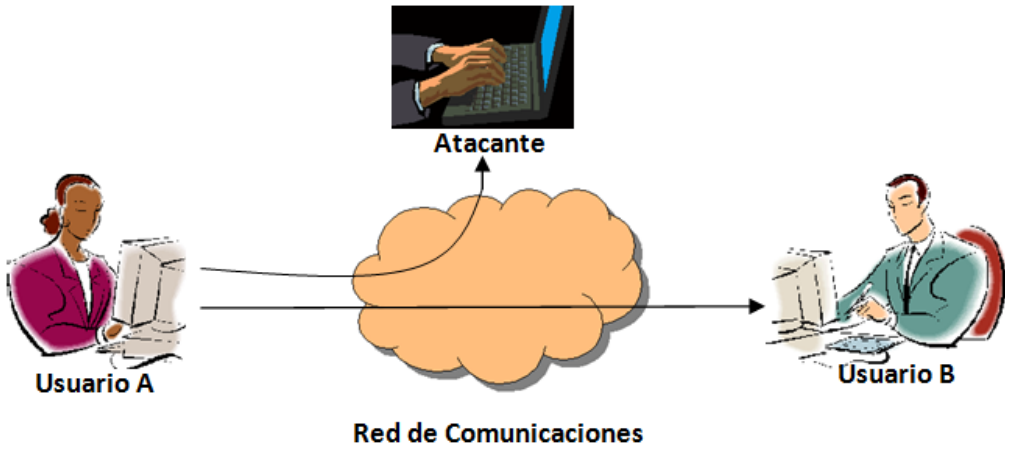


Figura 2.5. Esquema de un ataque pasivo. El atacante obtiene la información que el usuario A envía al usuario B pero no la modifica.

- **Ataques activos:** Supone alguna modificación de los datos o la creación de datos falsos. Los ataques activos se pueden clasificar en:

- Suplantación o mascarada: consiste en que una entidad pretenda o finja ser una entidad diferente.

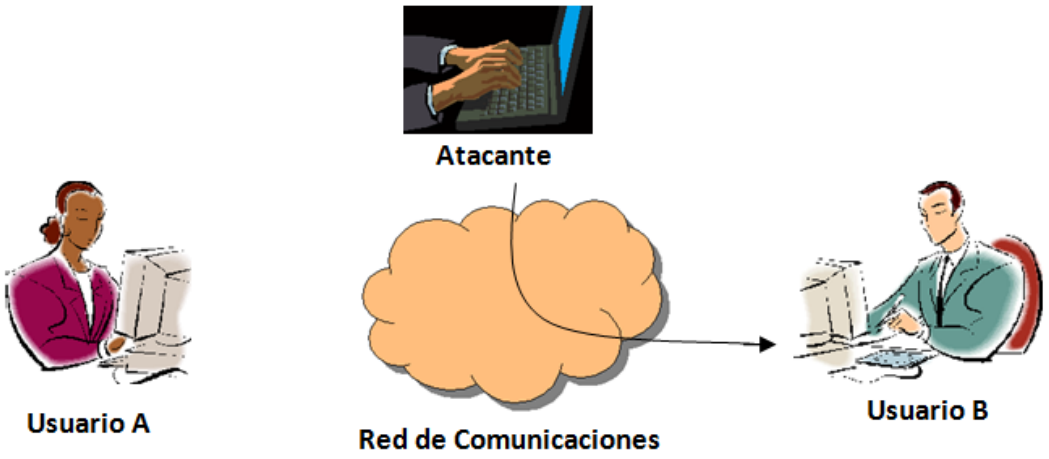


Figura 2.6. Esquema de un ataque activo. Mascarada: El atacante se hace pasar por el usuario A.

- Repetición: Supone la captura pasiva de una unidad de datos y su subsecuente retransmisión, para producir un efecto no autorizado.

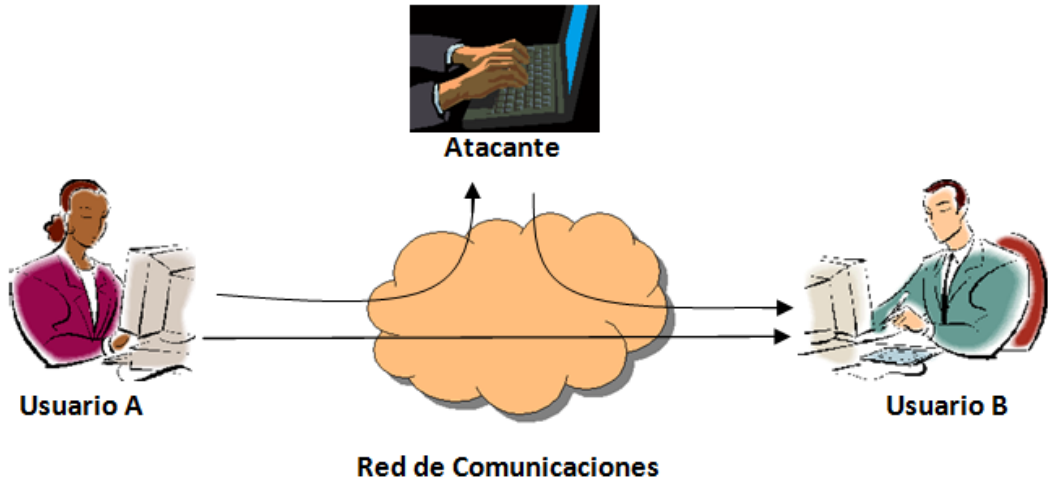


Figura 2.7. Esquema de un ataque activo. Repetición: Retransmisión de datos.

- Modificación de mensajes: Una porción de un mensaje legítimo es alterado o que las diferentes porciones de un mensaje son reordenadas para producir un efecto no autorizado.

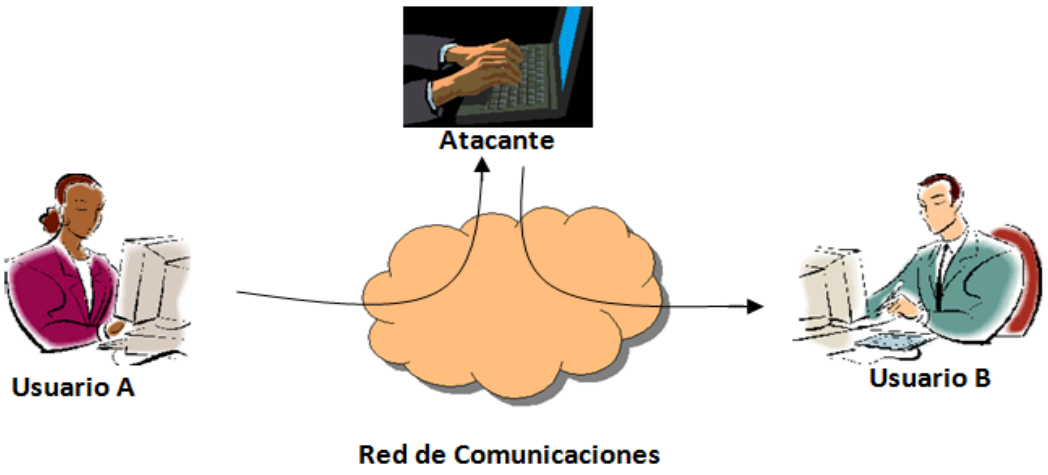


Figura 2.8. Esquema de un ataque activo. Modificación de mensaje.

- Denegación de servicios: impide la utilización normal o la gestión de las actividades de comunicación.

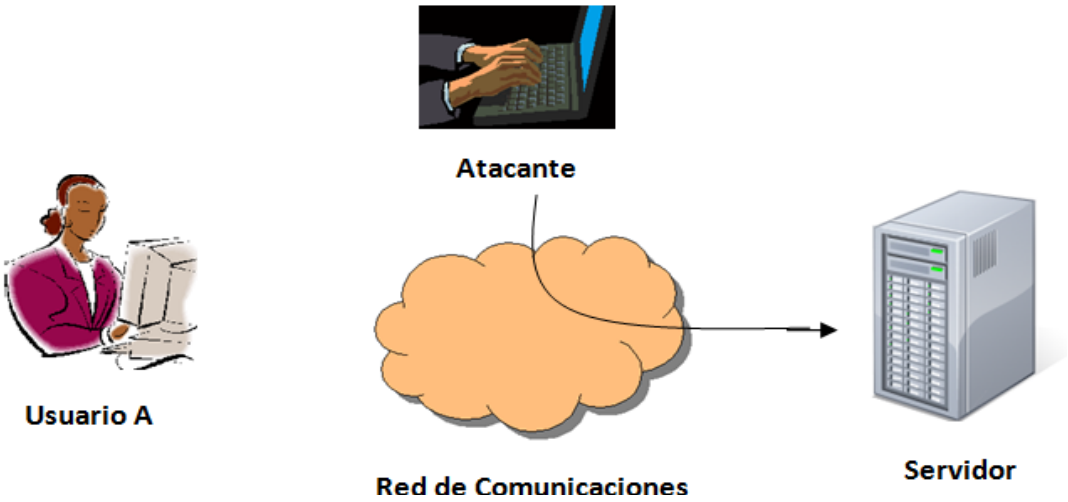


Figura 2.9. Esquema de un ataque activo. Denegación de servicio.

Existen diferentes mecanismos para detectar, prevenir o recuperarse de alguno de estos ataques de seguridad. Algunos de estos mecanismos son:

- **Cifrado:** Consiste en el uso de algoritmos matemáticos para transformar información a una forma que no sea fácilmente inteligible.

- **Firma digital:** Transformación criptográfica de una unidad de datos que permite al receptor de la misma verificar su fuente e integridad y protegerla frente a falsificaciones.

- **Control de acceso:** Es la capacidad de limitar y controlar el acceso de los dispositivos y aplicaciones a través de enlaces de comunicaciones. Para ello cada entidad que desee acceder debe primero ser identificada o autorizada.

- **Integridad de los datos:** Una variedad de de mecanismos usados para asegurar la integridad de una unidad de datos o una cadena de datos.

- **Autenticación:** Mecanismo dirigido a asegurar la identidad de una entidad mediante el intercambio de información, es decir, asegurarse que la comunicación sea autentica y que la información sea de la fuente de la que dice ser.

- **Padding de tráfico:** Inserción de bits para frustrar los intentos de análisis.

- **Control de rutado:** Habilita la selección de una ruta segura.

- **Notarización:** el uso de una tercera parte de confianza para asegurar la transmisión segura de la información.

Otras características adicionales que un servicio de seguridad puede ofrecer son las siguientes:

- **Confidencialidad de los datos:** Es la protección ante ataques pasivos de los datos que se quieren transmitir.

- **Integridad de los datos:** La integridad de un servicio orientado a conexión asegura que los mensajes son recibidos igual que fueron enviados, sin duplicados, inserciones, modificaciones, reordenaciones, etc. La destrucción de datos también es cubierta por este servicio.

- **Integridad de un sistema:** es la cualidad que tiene un sistema que no ha sido manipulado de forma no autorizada y que permite que se realice la función deseada.

- **No repudio:** Previene la denegación de un mensaje transmitido por parte del transmisor o del receptor del mensaje.

- **Disponibilidad de servicio:** Propiedad de un sistema o de la fuente de un sistema para ser accesible o utilizable bajo demanda por la entidad de un sistema autorizado de acuerdo a las especificaciones del sistema. [5]

La aplicación desarrollada en este proyecto utiliza algunos de estos mecanismos. Por ejemplo, el usuario puede *autenticarse* ante una plataforma, además para realizar la autenticación se necesita enviar una *firma digital* realizada con el certificado del usuario. El servidor de la plataforma, al autenticar al usuario realiza un *control de acceso*, pues un usuario que no demuestre que su certificado ha sido firmado por una Autoridad Certificadora de confianza, no podrá acceder al sistema.

Además, podemos decir que la seguridad es un objetivo que requiere la cooperación y coordinación de varias entidades. Las diferentes entidades que participan de forma activa o pasiva en la seguridad de un sistema son las siguientes:

- Desarrolladores de software.
- Fabricantes de productos.
- Integradores de datos en el sistema.
- Usuarios finales.
- Organizaciones de evaluación de la seguridad.
- Administraciones de sistema y de seguridad.
- Seguridad de las comunicaciones, centrada en la seguridad en la transmisión del flujo de datos.
- Seguridad contra emanaciones electromagnéticas que evite que se produzcan interferencias que modifiquen la información transmitida.
- Seguridad de los equipos. Se ocupa de los puestos de trabajo o de cualquier tipo de dispositivos de computación de cualquier tipo.

Los sistemas de información y comunicación requieren una constante evaluación y seguimiento de la seguridad, pues se encuentra en constante cambio y evolución. [20]

Por tanto, que una aplicación en Android sea segura, depende tanto de cómo se ha implementado; si hace uso de intercambio de información, que este intercambio sea seguro; que el usuario haga un uso adecuado de las aplicaciones y que informe si ha comprobado algún problema, etc.

2.3.1. Criptografía

La criptografía es un mecanismo que permite establecer canales de comunicaciones seguras entre dos puntos, cumpliendo además las características mencionadas en el punto anterior.

Cuando establecemos comunicación con otro equipo asumimos que nuestros mensajes son depositados generalmente en un medio hostil y necesitamos proveer sobre ellos cierta seguridad que permita su protección en los casos más desfavorables.

Los mensajes que se envían a través de un medio hostil se enfrentan principalmente a dos peligros:

- Acceso por agentes no autorizados: Un atacante puede acceder al mensaje. Nuestros sistemas de protección deben garantizar que el mensaje resulte ininteligible para este atacante.
- Alteraciones en el mensaje: Las alteraciones pueden aplicarse tanto sobre el mensaje, como sobre la información acerca de su verdadera procedencia. [23]

La criptografía depende de tres características principalmente:

1.- El tipo de operación usada para transformar un mensaje plano a un mensaje cifrado. Todos los algoritmos de cifrado se basan en dos principios: sustitución en el que cada elemento del mensaje plano se mapea o transforma en otro elemento y transposición que consiste en la reordenación de los elementos del mensaje plano. Además, estos cambios se deben producir sin que se produzca una pérdida de información.

2.- La forma en la que el texto se procesa. Puede ser un cifrado en bloque, en el que un bloque de datos se cifra y a la salida se obtiene el bloque de datos cifrados. O también puede ser un cifrado de una cadena en el que los elementos de entrada continuamente están produciendo una salida de elementos cifrados, lo cual se conoce como *"Stream cipher"* o cifrado de flujo.

3.- El número de claves usadas. Si el emisor y el receptor del mensaje utilizan la misma clave se dice que el sistema es simétrico. Si por el contrario, emisor y receptor utilizan dos claves (cada uno de ellos un par de claves diferentes, una pública y una privada), se dice que es un sistema asimétrico o cifrado de clave pública. [5]

2.3.1.1. Criptografía Simétrica

La criptografía simétrica consiste en la utilización de una clave para cifrar un texto sin la cual no puede ser descifrado, por tanto, las dos partes de la comunicación necesitan

saber la clave que cifra el mensaje. Cuanto más larga sea la clave más difícil será descifrar el mensaje.

Un esquema de cifrado simétrico tiene 6 componentes:

- Mensaje plano: Es el mensaje legible que se mete al algoritmo como entrada.
- Algoritmo de cifrado: realiza varias transformaciones del mensaje plano.
- Clave de cifrado: Clave utilizada para cifrado.
- Texto cifrado: Este es el mensaje mezclado que se obtiene a la salida. Depende del mensaje plano y la clave.
- Algoritmo de descifrado: Acepta el texto cifrado y la correspondiente clave y produce el mensaje plano original.

Por tanto, la comunicación segura usando criptografía simétrica se establecerá de la siguiente forma:

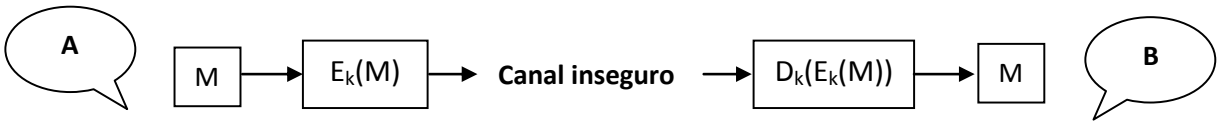


Figura 2.10. Esquema de criptografía simétrica. El emisor A, utiliza una clave para cifrar el mensaje M y lo transmite por un canal y el receptor B, descifra el mensaje usando la misma clave que usó A.

Donde:

- A es el emisor del mensaje.
- M es el mensaje original.
- $E_k(M)$ es la transformación del mensaje mediante cifrado.
- K es la clave utilizada tanto por el emisor como por el receptor.
- Dk(Ek(M)) es la transformación inversa mediante cifrado (descifrado).
- B es el receptor del mensaje.

Existen dos requisitos principales para el uso seguro del cifrado simétrico:

1.- Un fuerte algoritmo de cifrado tal que un oponente que sepa el algoritmo y tenga acceso a uno o más textos cifrados no podría descifrar el texto o averiguar la clave.

2.- Emisor y receptor deben haber obtenido copias de la clave secreta de un modo seguro y mantener dicha clave segura.

Estos dos requisitos se basan en los conocidos como *"Principios de Kerckhoffs"* que establecen las propiedades deseables de un sistema criptográfico. Estos principios se enumeran a continuación:

- Si el sistema no es teóricamente irrompible, al menos debe serlo en la práctica.
- La efectividad del sistema no debe depender de que su diseño permanezca en secreto.
- La clave debe ser fácilmente memorizable de manera que no haya que recurrir a notas escritas.
- Los criptogramas deberán dar resultados alfanuméricos.
- El sistema debe ser operable por una única persona.
- El sistema debe ser fácil de utilizar. [24]

El primer principio se refiere a que aunque en la teoría se planteen claves largas o difícilmente calculables, al final será posible romperlas en la práctica, pues a medida que mejoran la capacidad de cálculo de las máquinas, será más fácil descifrarlas, por lo que entonces se debe aumentar el tamaño de la clave.

El segundo principio se refiere a que se debe poder conocer como está implementado un sistema y aún así no ser capaz de romper su seguridad.

El tercer principio hace referencia al uso de claves que, por ejemplo, no sean excesivamente grandes para evitar que una persona no se olvide y necesite apuntarla en algún sitio, de forma que la seguridad se puede ver comprometida.

Los últimos principios se refieren a que los sistemas no deben ser imposibles de operar por una persona, es decir, hace referencia a las limitaciones de las personas.

Algoritmos de criptografía simétrica en *Anexo B*.

2.3.1.2. Criptografía Asimétrica

La criptografía asimétrica usa dos claves diferentes: una para el cifrado y otra para el descifrado. Las claves tienen una relación matemática especial que permite a los mensajes encriptados con una clave ser descifrados por la otra.

Normalmente en una aplicación criptográfica asimétrica, cada una de las partes de la comunicación tiene un par de claves. Una es conocida por todas las partes que

participan en la comunicación y se llama clave pública. Esta clave puede ser guardada, administrada y difundida por determinados servidores. La otra clave que se utiliza, se llama clave privada y no puede ser conocida salvo por el dueño de dicha clave. Una de las partes de la comunicación puede cifrar el mensaje utilizando la clave pública, mientras que la otra parte de la comunicación descifra este mensaje utilizando su clave privada.

Un esquema de cifrado de clave pública tiene 6 componentes:

- Mensaje plano: Es el mensaje legible que se mete al algoritmo como entrada.
- Algoritmo de cifrado: realiza varias transformaciones del texto llano.
- Clave pública y privada: Par de claves que han sido seleccionadas para que una se use para cifrado y otra para descifrado.
- Texto cifrado: Este es el mensaje mezclado que se obtiene a la salida. Depende del texto llano y la clave.
- Algoritmo de descifrado: Acepta el texto cifrado y la correspondiente clave y produce el texto llano original.

Se establece comunicación segura usando criptografía asimétrica o de clave pública, de la siguiente forma:

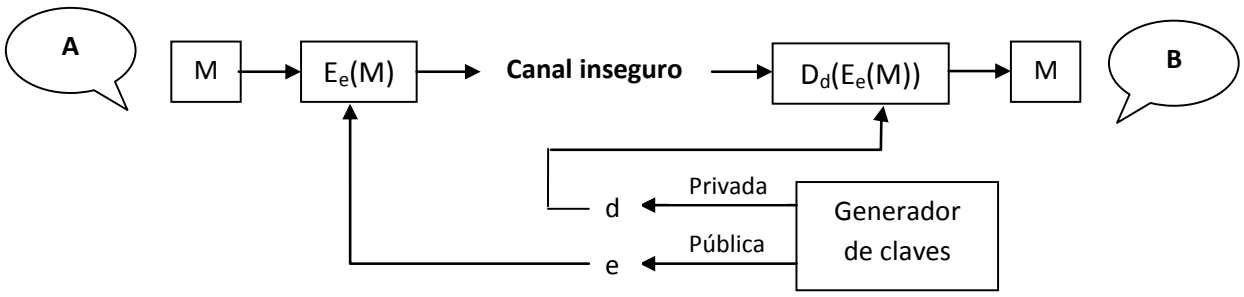


Figura 2.11. Esquema de criptografía asimétrica. El emisor A cifra el mensaje con la clave pública del receptor B y el receptor B, descifra el mensaje usando su clave privada.

Donde:

- A es el emisor.
- M es el mensaje original.
- $E_e(M)$ es el mensaje cifrado con la clave pública.

- e es la clave pública del emisor.
- d es la clave privada del receptor.
- $D_d(E_e(M))$ es el mensaje descifrado usando la clave privada del receptor.
- B es el receptor.

Algoritmos de criptografía asimétrica en anexo C.

2.3.2. Funciones resumen

Una función resumen o función hash criptográfica se usa para garantizar la integridad de los datos y la autenticación de mensaje. Proporciona una secuencia de bits de pequeña longitud, que va asociada al mensaje (aunque contiene menos información que este) y que debe resultar muy difícil de falsificar. Existen funciones resumen que emplean en sus cálculos una clave adicional (MAC, message authentication code) y otras que no la usan (MDC, modification detection codes). A continuación se definen estos dos tipos:

- *Modification Detection Codes (MDC):* función hash que utiliza un parámetro de entrada (el mensaje) y se usan para verificar la integridad del mensaje.

Existen dos tipos de funciones MDCs:

  - One-way hash functions (OWHF) o funciones hash de un solo sentido: Es difícil encontrar una entrada cuya hash sea un valor hash preespecificado.
  - Collision Resistant Hash Function (CRHF) o funciones hash de resistencia a colisiones: Encontrar cualesquiera dos funciones que tengan el mismo valor de hash es difícil. [25]

- *Message Authentication Codes (MAC):* Una función hash que utilizan dos parámetros de entrada: un mensaje y una clave secreta y se usa para comprobar la autenticidad del origen de un mensaje, asegurando además la integridad de dicho mensaje.

Una autenticación debe hacerse empleando una función resumen o hash y no codificando el mensaje completo. El propósito de una función Hash criptográfica es producir una *"huella"* de un archivo, mensaje u otros bloques de datos [60]. Para ser útil para una autenticación de mensaje, una función hash debe tener las siguientes propiedades:

1) Puede ser aplicado a un bloque de datos de cualquier tamaño.
2) Produce una salida de una longitud fija (Compresión).

3) Una función hash debe ser relativamente fácil de calcular para cualquier mensaje.

4) Dado el resultado de una función hash, debe ser irrealizable encontrar el mensaje original. A esto se le llama la propiedad unidireccional.

5) Resistencia de colisión débil. Se da si dado un mensaje *"x"*, es fácil encontrar otro mensaje *"y"* distinto al primero en el que se cumpla que las dos funciones hash resultado sean iguales. [5]

Como ya se ha comentado anteriormente, las funciones hash tienen diferentes utilidades:

- **Autenticación de mensaje**

  Es un mecanismo o servicio usado para verificar la integridad de un mensaje. Asegura que los datos recibidos son exactamente iguales a los enviados. En algunos casos, existe el requisito de asegurar que la identidad del emisor es válida.

- **Firma digital**

  Otra aplicación que tienen las funciones hash es la de firma digital (estudiado también en el apartado *2.3.3*). La operación de la firma digital es similar a la que se hace con la MAC, ya que se obtiene una hash del mensaje con la clave privada del usuario y la integridad del mensaje se comprueba con la clave publica.

- **Otras aplicaciones**

  Otros usos que se suelen hacer de las funciones hash son: la detección de intrusos, la detección de virus, construir una función pseudo-aleatoria (Pseudorandom Function o PRF) o un generador de números pseudoaleatorios (Pseudorandom Number Generator o PRNG). [25]

2.3.3. Identidades digitales y firma digital

La identidad digital, al igual que ocurre con la identidad física, consiste en el conjunto de datos e información que permiten identificar a una persona, a un colectivo, instituciones, etc., en general a una entidad, a través de la red.

Cuando un usuario quiere registrarse en un sistema interactivo, introduce determinados datos personales que permiten identificarlo, como por ejemplo, un nombre de usuario, un número de teléfono, una IP, un dominio de internet o el correo electrónico. Estos datos haciendo uso de diferentes métodos criptográficos, permiten identificar a una persona.

Existen diversos métodos para la identificación de una persona a través de internet. Algunos de estos métodos son los ya mencionados certificados digitales, que identifican a una persona mediante la relación entre los datos de esa persona con su clave pública. Otro método es el uso de una firma digital que identifique a esta persona.

La firma digital sirve para determinar la autenticidad y la integridad de un archivo mediante el uso de criptografía y el uso de funciones resumen o Hash, además de identificar a la persona que firma dicho documento. Los requisitos que se deben cumplir para que resulte útil el uso de una firma digital son los siguientes:

- El receptor tiene que poder verificar la identidad del transmisor.

- El transmisor no puede repudiar o negar el contenido del mensaje enviado.

- Se debe poder verificar que el receptor no ha creado el archivo él mismo.

Para la creación de una firma digital corrientemente se genera una hash del mensaje que se quiere firmar. Posteriormente el mensaje se cifra con la clave privada del autor del mensaje. El resultado de este proceso se añade a los datos, lo que permite por un lado identificar al emisor mediante su clave privada y verificar que el texto enviado es el que el autor deseaba enviar mediante la comprobación de la función resumen del texto enviado.

El receptor del mensaje accede a la función resumen mediante la clave pública del emisor. A la vez, se genera un hash del texto enviado, que se compara con la función hash obtenida al descifrar la firma digital adjunta al mensaje y al comparar ambos resultados se comprueba que el texto no ha sido modificado por un atacante. [26][27]

El proceso de firma digital se muestra en la siguiente imagen:

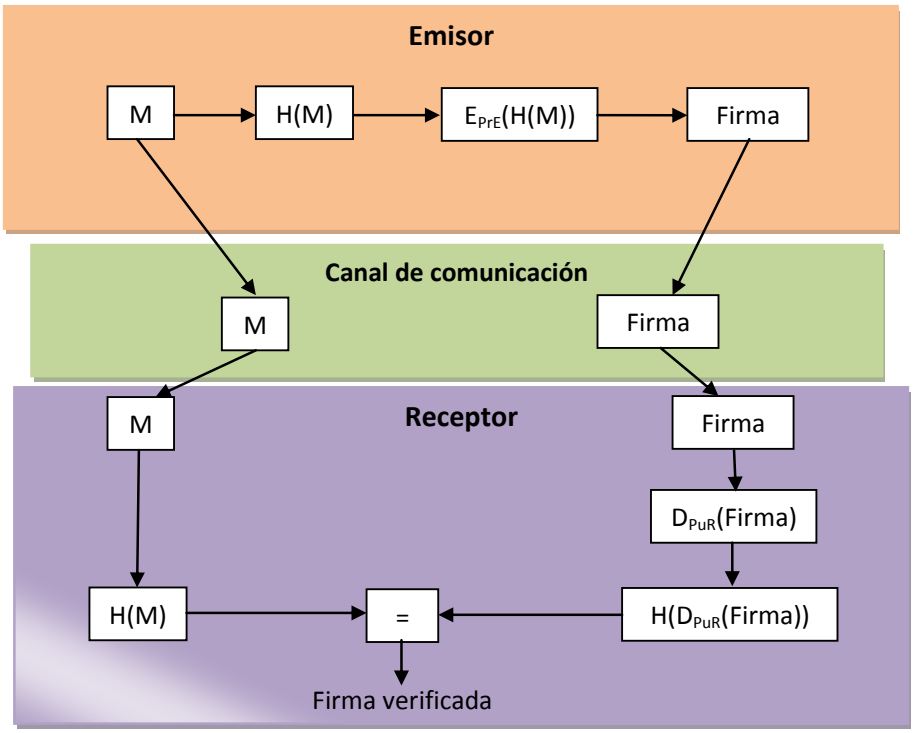


Figura 2.12. Proceso de firma digital. El emisor realiza una Hash del mensaje y después lo cifra y con ello se obtiene la firma. El emisor envía la firma y el mensaje. El receptor descifra la firma y el Hash y lo compara con el mensaje y la firma enviada, si los resultados son iguales se verifica la firma.

Donde:

- M representa el mensaje original.
- H(M) Función Hash del mensaje.
- $E_{PrE}(H(M))$ representa el cifrado con clave privada de la Hash del mensaje original.
- $D_{PuR}(Firma)$ representa el descifrado de la Firma recibida utilizando la clave pública del receptor.
- $H(D_{PuR}(Firma))$ es la función Hash que se realiza después de descifrar la firma.
- PrE es la clave privada del emisor.
- PuR es la clae pública del receptor.

Se pueden emplear varios algoritmos para la generación de una firma digital: RSA y DSA. Otros algoritmos que se usan son El-Gamal (*Anexo C*), Schnorr y Curvas Elipticas. Con RSA se requiere que la longitud de las claves no sea inferior a 1024 bits para que sean lo suficientemente segura. Con respecto a DSA se sabe que tiene una complejidad computacional mayor a RSA. (Algoritmos descritos en el *Anexo C* junto con más algoritmos de firma). [61]

2.3.4. Certificados digitales

Los certificados digitales se usan para establecer una unión entre el nombre de una entidad o algún dato que lo identifique y su clave pública.

Existen diferentes tipos de certificados digitales entre los que cabe destacar:

- Certificados de clave pública X.509
- Certificados de infraestructura simple de clave pública (Simple Public Key Infraestructure o SPKI)
- Certificados Pretty Good Privacy (PGP)
- Certificados de Atributo.

Todos estos certificados tienen distintos formatos y en algunos casos distintas versiones.

En muchos casos cuando se habla de certificados digitales, se suele hacer referencia al certificado de clave pública X.509 (se hablará de este tipo de certificado en el apartado *2.3.4.1*), que es el tipo de certificado más extendido. Sin embargo, es necesario mencionar las principales características de los otros tipos de certificado:

SPKI

Se pensó como un tipo de certificado en el que la clave del certificado y los distintos protocolos asociados fuesen fácilmente entendibles, implementables y usables. Está más pensado para autorización que para identificación. Tiene características en común con el certificado X.509 pero la sintaxis es diferente. [22]

En un certificado SPKI se establece un vínculo entre la clave pública de una entidad y un permiso o autorización. Su propósito es proveer permisos o dar el poder de conceder permisos a terceros.

SPKI es un estándar que pretende proveer servicios de seguridad en un amplio rango de aplicaciones de internet, incluyendo a protocolos IPSEC, correos cifrados electrónicamente, protocolos de pagos (por ejemplo, para pedir permiso para escribir cheques electrónicos), obtener permisos para utilizar determinado software, tarjetas de crédito, tarjetas de teléfono, tarjetas de descuento de los supermercados, etc.

La autorización o concesión de permisos, es algo que no requiere la interacción con terceros, sino que está totalmente en manos de la persona que usa el certificado. La información que contiene estos certificados tiene que ser la menor posible, es decir, la persona que quiere confirmar que tiene permiso debe poder dar pocos datos.

Los certificados SPKI deberían ser utilizables en entornos limitados, como smart card, pequeños sistemas embebidos, etc. El código que los procese, así como la memoria que los almacene deben ser de pequeño tamaño también. [28][29]

PGP

Es un método que se usa principalmente para cifrar y firmar digitalmente mensajes de email y archivos. Existen diferencias importantes entre este tipo de certificados y los certificados X.509 que han hecho que la comunidad se haya dividido entre el uso de certificados PGP y certificados X.509. La principal diferencia entre PGP y x.509 radica en que cualquier usuario puede firmar claves públicas, es decir, se sigue una estructura horizontal, mientras que x.509 sigue una estructura jerárquica. En PGP la confianza no se basa en Autoridades Certificadoras, si no, en la propia confianza que los usuarios van depositando gradualmente entre ellos mismos. Una posible solución a este problema es que este tipo de certificado se use conjuntamente a la versión 3 de X.509 pero existe una gran división entre los proveedores de estos dos servicios. PGP no se considera una buena opción para su uso en intranets corporativas, ya que su confianza no dependería de la empresa, sino en los individuos pertenecientes a dicha empresa. [22]

En PGP cada clave pública se almacena en lo que se denomina certificado de clave. Estos certificados contienen:

- Una clave pública.
- Uno o más IDs de usuario para el creador de la clave (normalmente el nombre de la persona y la dirección de correo electrónico).
- La fecha en la que la clave fue creada.
- Opcionalmente incluye una lista de firmas digitales sobre la clave cuya veracidad es confirmada por otros usuarios.

Como se ha comentado, PGP, no es sólo un tipo de certificado, sino que define una serie de protocolos de seguridad cuyo uso compite directamente con los protocolos X.509.

Entre otras cosas, PGP puede usarse para las siguientes acciones:

- Cifrar archivos: Para ello se suele utilizar un algoritmo de clave privada IDEA.
- Crear las propias claves privadas y públicas: Estas se necesitan para cifrar y firmar los mensajes que se envían a través de la red y descifrar los mensajes recibidos.

- Gestión de claves: Se puede crear y mantener una base de datos con las claves de los usuarios con los que él dueño de la base de datos se relaciona.
- Cifrar y descifrar los correos electrónicos que se envían y reciben.
- Firmas digitales: Se puede usar PGP para firmar documentos así como para verificar las firmas de los documentos recibidos.
- Claves de certificados: Se puede usar PGP para firmar digitalmente claves públicas de otros usuarios.
- Revocar y deshabilitar claves: Una persona puede deshabilitar o revocar los certificados que sean comprometidos o peligrosos.
- Usar los servidores de claves PGP de internet: Un usuario puede añadir su clave pública a una base de datos de claves, a la vez que puede obtener la clave pública de otros usuarios. [30]

Certificados de atributo

Fue estandarizado en la versión de 1997 del certificado X.509. Está diseñado para facilitar la transmisión de atributos sobre un sujeto para facilitar los privilegios de gestión. [22]

Un certificado de atributo es una estructura separada de los certificados de clave pública. Un sujeto podría tener varios certificados de atributos asociados con cada uno de sus certificados de clave pública. No existe el requisito de que la misma autoridad que expide los certificados de clave pública expida también los certificados de atributo. Por lo general, los certificados de clave pública son expedidos por una CA, mientras que los certificados de atributos son expedidos por una Autoridad de Atributo (AA) y por ello normalmente son firmados por distintas claves privadas. En los casos en los que la CA y la AA sean la misma entidad, está recomendado que los certificados sean firmados por la misma clave privada. [31]

2.3.4.1. Protocolo y Certificados x.509

X.509 es un estándar UIT-T para PKI. Especifica formatos estándar para certificados de claves públicas y un algoritmo de validación de la ruta de certificación. Los formatos de codificación más comunes para certificados X.509 son DER (Distinguish Encoding Rules) o PEM (Privacy Enhanced Mail).

La recomendación X.509 es parte de la serie de recomendaciones X.500 que define un servicio de directorio. Un directorio es un servidor que mantiene una base de datos de información sobre los usuarios. El protocolo X.509 define un marco para proveer

servicios de autenticación y define una estructura de certificado y protocolos de autenticación que se usan en una gran variedad de contextos. Incluye también estándares para implementación de listas de certificados de revocación (CRL) y aspectos de sistemas PKI.

Este protocolo gira en torno al uso de certificados de clave pública, los cuales son creados por alguna Autoridad de Certificación de confianza (CA), que son almacenados por el usuario para su posterior uso. Se habla de una estructura jerárquica, al depender la confianza de una entidad que reparte los certificados (la CA), en contraste con modelos de redes de confianza como PGP, donde la estructura es horizontal al depositar la confianza en los propios usuarios [5].

El estándar X.509 sólo define la sintaxis según el lenguaje ASN.1 (Abstract Syntax Notation One) de los certificados por lo que no está atado a ningún algoritmo en particular, y contempla los siguientes campos:

- Versión
- Número de serie
- Identificador del algoritmo empleado para la firma digital
- Nombre del certificador
- Periodo de validez
- Nombre del sujeto (DN o Distinguised Name que además se divide en más campos: CN o common name, OU o Organozatonal Unit, O o Organization, C o Country).
- Clave pública del sujeto (Consta de dos campos: el algoritmo empleado para crear la clave y la propia clave)
- Identificador único del certificador (introducido en la versión 2)
- Identificador único del sujeto (introducido en la versión 2)
- Extensiones (introducido en la versión 3)
- Firma digital de todo lo anterior generado por el certificador. Como se ha comentado anteriormente, la firma digital es la función hash (en este caso del certificado), cifrado por determinado algoritmo de firma.

Se muestra en la siguiente página un ejemplo de certificado obtenido del acceso al correo de la Universidad Autónoma de Madrid.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            7a:6f:3c:d0:27:19:03:17:4f:f2:01:43:44:eb:69:ee
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=NL, O=TERENA, CN=TERENA SSL CA
        Validity
            Not Before: Dec 16 00:00:00 2013 GMT
            Not After : Jan 28 23:59:59 2017 GMT
        Subject: OU=Domain Control Validated, CN=webmail.uam.es
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
00:d2:1e:7d:c3:13:9a:cc:58:b7:2a:92:bf:90:06:
a2:2a:56:d3:b5:52:33:6b:53:a7:d5:77:7e:2c:25:
9e:ee:6b:4b:a1:1b:36:0a:dd:71:17:d7:eb:61:33:
93:56:6a:d5:a5:ca:10:75:d8:5b:30:44:d4:55:36:
ac:da:a6:39:9e:32:a0:4b:06:8f:a4:04:5b:83:cb:
4d:ba:ae:f7:c6:d2:a9:8a:b3:fe:84:f0:bc:d3:8d:
66:e5:01:3e:03:3f:b0:60:46:4c:af:d6:00:d3:0f:
ec:a3:29:c9:af:b7:3c:4c:77:5a:a6:c5:27:1a:fb:
1d:6e:c9:fe:33:d4:d6:bf:77:9b:dc:e7:d6:69:a5:
21:63:c9:45:29:e4:ed:bf:b9:e0:61:55:5a:44:e5:
72:26:b8:c9:8e:90:bf:81:a9:35:a0:be:c1:f4:be:
d7:e9:80:cb:70:82:88:49:60:63:f3:ba:fb:fe:45:
1f:1d:4f:d6:a7:a2:ed:72:28:e8:55:c1:6c:8a:1d:
59:ca:5a:90:54:37:13:b0:fe:0e:22:06:cf:e5:92:
6e:9a:81:9a:b6:b1:59:68:e9:1a:89:c1:5d:f5:7d:
55:fe:94:7f:72:2b:cd:9b:ce:1a:54:95:44:3f:9f:
10:86:a2:aa:a3:90:9b:64:52:da:fc:dd:d9:91:fb:
44:0b
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Authority Key Identifier:
keyid:0C:BD:93:68:0C:F3:DE:AB:A3:49:6B:2B:37:57:47:EA:90:E3:B9:ED
            X509v3 Subject Key Identifier:
4C:AC:9D:D8:45:FA:49:9B:BA:4C:34:E6:44:41:E6:BE:F8:2A:8E:C6
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 Certificate Policies:
                Policy: 1.3.6.1.4.1.6449.1.2.2.29
                Policy: 2.23.140.1.2.1

            X509v3 CRL Distribution Points:
                URI:http://crl.tcs.terena.org/TERENASSLCA.crl

            Authority Information Access:
                CA Issuers - URI:http://crt.tcs.terena.org/TERENASSLCA.crt
                OCSP - URI:http://ocsp.tcs.terena.org

            X509v3 Subject Alternative Name:
                DNS:webmail.uam.es
    Signature Algorithm: sha1WithRSAEncryption
        50:a1:a4:97:8d:1d:83:dc:11:5c:47:0d:1d:a8:bd:41:df:ad:
        8a:ce:a7:f7:1a:61:d0:b0:14:a9:7e:91:bc:0b:d7:42:61:2a:
        26:6e:d7:6f:f0:b4:f5:41:fa:3c:72:c7:2f:03:49:c8:e8:2a:
        ba:04:78:54:2c:10:de:7b:63:f7:4a:b9:b4:84:48:82:c0:0b:
        5e:28:20:52:01:bd:ea:6b:d1:23:70:b9:33:c6:52:fe:4c:cf:
        2e:4e:03:53:61:11:74:3c:06:82:f5:9e:08:b2:67:7d:74:df:
        00:ba:ea:2d:b6:9f:62:d4:05:f7:17:48:36:2c:d8:e2:10:29:
        28:99:b5:a2:42:0b:95:ed:7e:74:65:81:a9:14:ae:c0:26:09:
        4b:db:40:18:70:8f:8f:b5:f3:33:4f:c2:cf:f2:39:64:af:f1:
        60:63:28:07:71:2c:e0:9a:e5:d7:a7:2a:be:13:12:c0:bd:de:
        0e:74:f4:39:45:04:00:cb:40:f0:1a:e2:af:47:8d:3f:3d:5f:
        f1:40:c4:d5:95:a0:87:60:c8:08:ae:9c:f4:0a:ea:be:ef:19:
        4d:d3:dd:40:df:fd:52:00:16:ae:a9:06:fc:c1:2f:39:01:3a:
        61:30:33:03:87:a3:c9:9d:23:59:b3:cc:88:e7:43:53:03:82:
        69:70:73:09
```

Figura 2.13. Certificado digital X.509 v3 extraido del correo de la UAM. Contiene la versión del certificado, el número de serie del certificado, la clave pública del servidor, los datos del servidor, los datos de la CA, la firma de la CA, el algoritmo de firma, el tamaño de las claves, etc.

En relación a la verificación de los certificados, una autoridad de certificación suele tener un ámbito relativamente local. Si fuera necesario verificar un certificado digital de un certificador ajeno, del cual desconocemos su fiabilidad, existe la posibilidad de

que la clave pública del propio certificador esté a su vez firmada por otra entidad de la que si nos fiemos. Esta circunstancia puede ser aprovechada de forma jerárquica (como en las PKI o Infraestructura de Clave Pública) o distribuida (como hace PGP). [23]

Desde la creación de este estándar, se han desarrollado tres versiones de certificados. La versión 1 (v1) fue publicada en el año 1988. En al año 1993 se publicó una nueva versión (v2) que añadió dos campos más en relación al control de acceso de directorio (Identificador Unico de emisor o Identificador Unico de Sujeto). Sin embargo, estas dos versiones no eran lo suficientemente adecuadas pues se necesitaba más información para adecuarse al formato PEM[8] que se utiliza en los certificados, por ello se creó la versión 3 (v3). [33]

Cuando un certificado pierde su periodo de validez es necesario revocarla, también si se ha perdido el certificado, para ello se genera un certificado de revocación. Un certificado de revocación es un mensaje que identifica a la clave pública que se desea anular, firmada por la clave privada del dueño del certificado.

Es necesario que las claves de revocación se creen justo después de haber creado el par de claves privadas/públicas y que se guarden en algún lugar seguro, ya que hay que acceder a ella en caso de pérdidas o expiración del periodo de validez.

2.3.4.2. Solicitud de firma de certificado o Certificate Signing Request (CSR)

Un CSR es una petición de certificado que una entidad envía a una CA para obtener un certificado digital. Al generar un CSR, se genera también un par de claves, la clave privada está unida al sujeto del certificado. Una vez que el CSR es enviado a una CA, esta lo firma con su clave privada y de esta forma la entidad que ha enviado un CSR obtiene un certificado.

Se emplean CSRs cuando la entidad que solicita un certificado no es la misma que los emiten. Las entidades que emiten certificados normalmente generan su propio certificado, es decir, generan certificados autofirmados. Usar certificados autofirmados normalmente no acredita la identidad de una persona.

[8] PEM es un tipo de extensión para los certificados. Lo que le caracteriza es que está codificado en Base64 y se encuentra encerrado entre "-----BEGIN CERTIFICATE-----" y "-----END CERTIFICATE-----". Además existen otros tipos de extensión: .cer, .der, .p7b, .p7c, pfx, .p12.

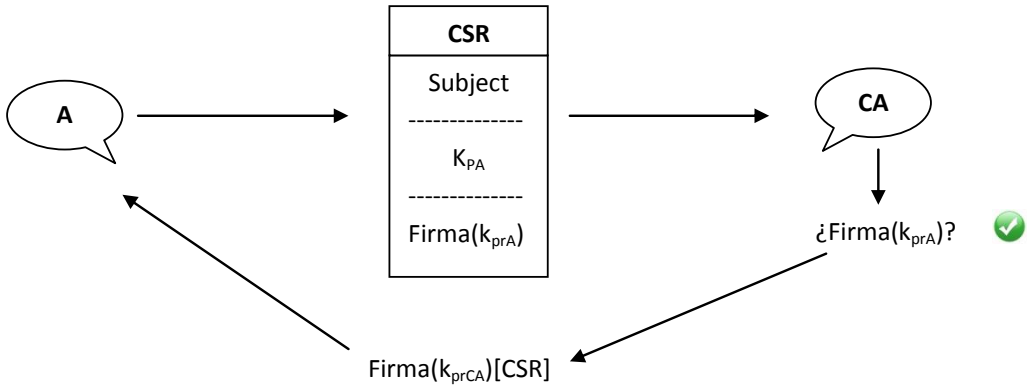


Figura 2.14. Proceso para la obtención de un certificado usando un CSR. El usuario envía un CSR que contiene un subject con datos del usuario, la clave pública (K_{PA}) del usuario y una firma digital hecha con la clave privada (K_{prA}) del usuario. La CA verifica que la firma del CSR es del usuario y realiza una firma sobre el CSR de forma que se obtiene un certificado que se envía al usuario.

Un CSR se compone de información de la solicitud de certificado, de un identificador del algoritmo de firma, la clave pública de la entidad que solicita el certificado y una firma digital sobre la información de solicitud de certificado hecha con la clave privada de la entidad que solicita el certificado. A su vez la información de la solicitud de certificado consiste en un nombre distintivo de la entidad que lo solicita (Distinguished Name o DN), su clave pública y otra serie de atributos proporcionados por la entidad.

Una CA firma el CSR después de autenticar la entidad que solicita el certificado y verificar la firma digital de la entidad y si la solicitud es válida se construye un certificado X.509 a partir del nombre distintivo (DN) y la clave pública, el emisor del nombre y la elección por parte de la CA del número de serie, el periodo de validad y el algoritmo de firma. [34]

Se muestra a continuación un esquema con los campos de un CSR:

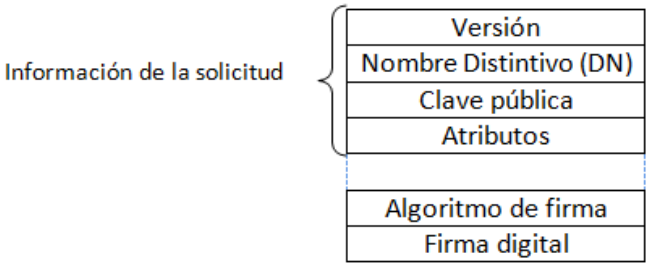


Figura 2.15. Estructura de un CSR. Contiene versión del certificado, nombre distintivo con los datos del usuario o del servidor que solicite el certificado, clave pública, atributos, algoritmo de firma y la firma digital del usuario hecha con la clave privada del usuario.

En principio, la estructura de un CSR es muy similar a la de un certificado X.509, pero son certificados diferentes. Lo que diferencia a ambos certificados, es que el CSR contiene la clave pública del usuario y la firma digital del propio usuario hecha con la clave privada del usuario, mientras que un certificado usual contiene la clave pública del usuario y la firma digital de la CA, hecha con la clave privada de la CA.

En el siguiente capítulo se mostrará el CSR que el usuario desde la aplicación desarrollada envía a la CA para obtener el certificado del usuario.

2.3.4.3. Lista de revocación de certificados (CRL)

Fueron introducidos en el ITU-T en la recomendación X.509 en el año 1988. Es una lista con los números de serie de los certificados con la fecha y los motivos por los que se han revocado [32]. Esta lista está firmada digitalmente por una CA y se actualiza periódicamente. Además, incluye el nombre del emisor (el nombre de la CA que firma), la fecha en el que la lista fue creada, la fecha en el que se prevé que se va a emitir el próximo CRL y la entrada para cada certificado revocado.

Un certificado se encuentra revocado si tras recuperar y verificar la firma de la lista, encontramos su número de serie en la lista de certificados revocados. [35]

Los certificados se pueden revocar, además de por estar expirado, por alguna de las siguientes razones:

- La clave privada está comprometida o poco segura.
- El usuario ya no será certificado por dicha CA.
- El certificado de la CA se encuentra comprometido o es poco seguro.

Cada CA debe mantener una lista de todos los certificados revocados pero no de certificados que han expirado, incluyendo los certificados emitidos a usuarios u otras CAs.

Cuando un usuario recibe un certificado en un mensaje, el usuario debe comprobar si el certificado ha sido revocado. El usuario puede revisar el CRL cada vez que un certificado es recibido. [5]

La siguiente figura muestra el formato que suele presentar una lista de revocación de certificados:

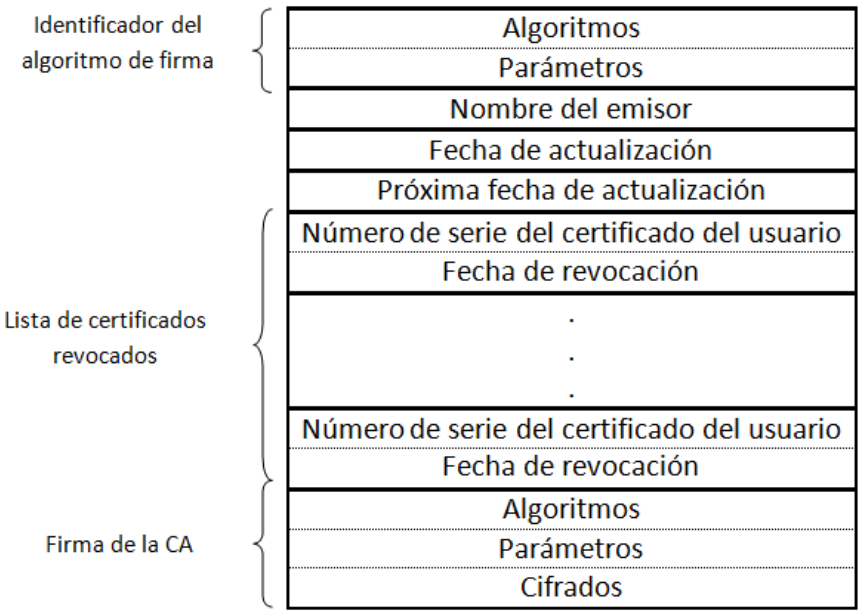


Figura 2.16. Estructura de una lista de revocación de certificados. Contiene el algoritmo de firma, el nombre del emisor, la fecha en la que se ha expedido el certificado, la fecha en la que expira el certificado, el número de serie del certificado, y la firma de la CA que a su vez contiene el algoritmo que se ha usado, y la firma en sí.

2.3.5. Infraestructura de clave pública o PKI

Una buena forma de intentar mantener la seguridad es mediante el uso de una Infraestructura de clave pública (Public Key Infraestructure o PKI de aquí en adelante).

Una PKI es un conjunto de herramientas (software, hardware, políticas de seguridad o protocolos de seguridad) que permiten a los usuarios de una red a priori insegura como es el caso de internet, transmitir información de una forma segura. Esta transmisión de datos se hace mediante el uso de una clave privada y una clave pública que es obtenida a través de una autoridad de confianza. Una estructura PKI debe ser capaz de proporcionar un certificado digital a un usuario u organismo que haga uso de él para identificarse, además debe de encargarse de su revocación en el momento en el que se haya expirado la fecha en el que su uso es válido.

Una estructura PKI no sólo engloba el uso de certificados digitales y de un par de claves privada y pública, sino que también incluye el uso de redes y protocolos de seguridad. Además, existen protocolos como son el protocolo Secure Sockets Layer (SSL) y el protocolo Hypertext Transfer Protocol Secure Sockets (HTTPS), etc. pueden hacer uso de una PKI para aumentar su seguridad.

Una PKI hace uso de criptografía de clave pública. La criptografía de clave pública es el método más usado para la autenticación del usuario que transmite un mensaje y para

el cifrado de mensajes. La criptografía tradicional comparte entre el transmisor y el receptor de un mensaje una clave que utiliza para cifrar y descifrar un mensaje. Sin embargo este método puede ser peligroso, pues la clave puede ser interceptada por un atacante. Por eso, una infraestructura de clave pública y la criptografía de clave pública son métodos mucho más seguros.

Una infraestructura de clave pública consiste en lo siguiente:

- Una Autoridad Certificadora (CA) que emita y verifique los certificados digitales.
- Una Autoridad de Registro (RA) que verifica los certificados antes de que la CA emita los certificados al usuario que lo solicita.
- Uno o más directorios donde los certificados y sus claves privadas son almacenadas.
- Un sistema de gestión de certificados. [21]

De todos los servicios de seguridad mencionados en el apartado anterior, una PKI ofrece principalmente tres servicios descritos a continuación:

Autenticación

Consiste en garantizar que una entidad o usuario es quien dice ser. Esta garantía se debe dar en dos contextos: verificar el origen de los datos y verificar la identidad de un usuario.

Con respecto a la identificación de un usuario o entidad, la autenticación nos sirve para identificar quien es el la persona que quiere acceder a un determinado servicio. Por ejemplo, cuando una persona quiere acceder a un servicio introduce una serie de credenciales que le identifican, de esta forma se impide que el usuario acceda a datos confidenciales o privados si este no está autorizado, esto aseguraría además la confidencialidad. La identificación de una entidad en un sistema PKI es posible gracias a que la clave privada que utiliza una entidad para firmar es usada para autentificarse ante otras entidades.

En cuanto a la verificación del origen de los datos, sirve principalmente para impedir que alguien vaya a introducir datos maliciosos en el sistema por ello se hace necesaria su identificación. De esta forma, si alguien introduce un dato que pueda ser dañino es fácilmente identificable.

Integridad

Con la integridad se asegura que los datos no han sido alterados. Algunas veces la no integridad de los datos puede ser debida a alteraciones en la transmisión de la señal

que modifiquen la señal y que hacen que el conjunto de bits recibidos por un equipo no sea el mismo conjunto de bits que se transmitieron, para corregir estos errores accidentales existen una serie de algoritmos. Pero el mayor problema no son los errores accidentales, sino el ataque por parte de una persona que modifique de forma maliciosa los datos.

El servicio de integridad de los datos proporcionado por una PKI hace uso de una serie de algoritmos criptográficos que permiten asegurar que los datos han sido transmitidos correctamente.

Confidencialidad

Es la garantía de que existe privacidad de los datos, evitando que la información transmitida sea leída por personas no autorizadas. Para proporcionar confidencialidad se usan técnicas criptográficas que permiten mantener los datos secretos a cualquier atacante con intenciones maliciosas. [22]

2.3.5.1. Componentes de una PKI

La estructura que puede tener una PKI es la siguiente:

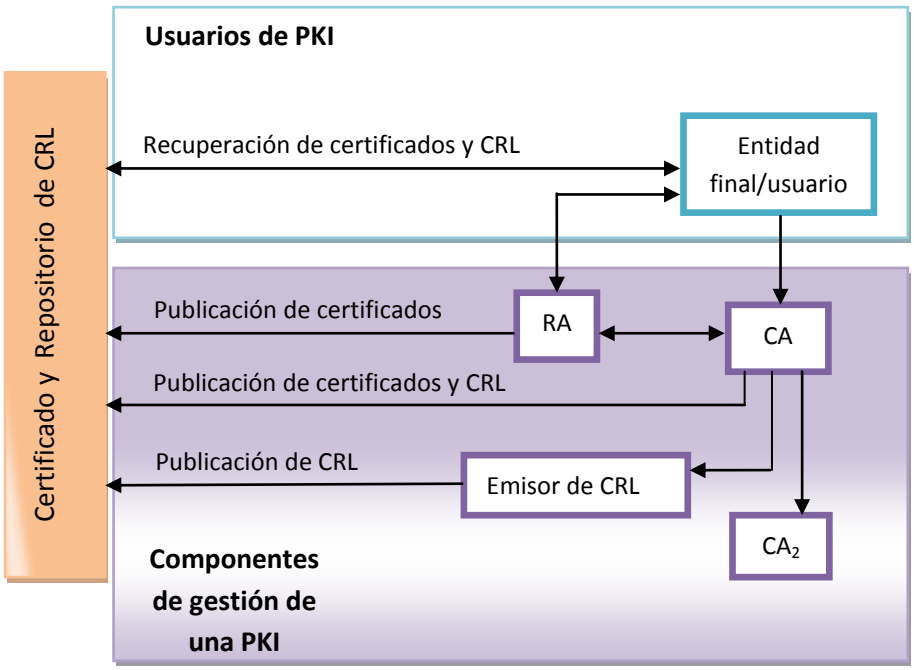


Figura 2.17. Arquitectura de una PKI [32]. Consta de una Autoridad de Registro, una Autoridad Certificadora, un emisor de CRL, etc.

Como se ve en la imagen, una entidad final (usuario) solicita un certificado a una CA, está puede realizar tres acciones: solicitar el certificado a otra CA que tenga más autoridad, emitir una CRL o enviar el certificado a una RA para que compruebe su

contenido y lo verifique. Además, tanto los CRL como los certificados se almacenarán en una jerarquía de directorios usados por la CA.

De forma más específica los componentes de una estructura PKI son los siguientes:

Entidad final

Una entidad final se refiere a un usuario, un dispositivo o cualquier otro tipo de entidad que puede ser identificada con un certificado de clave pública. Una entidad final consume o soporta servicios relacionados con una PKI.

Autoridad certificadora (CA)

Es el emisor de certificados y por lo general también de CRL. Los certificados de clave pública son firmados por una CA y es quien une la identidad del usuario con el certificado.

También puede realiza diferentes tareas administrativas como el registro de un usuario (aunque por lo general esta tarea es llevada a cabo por una RA), la recuperación de seguridad de una clave o en general las facilidades de recuperación, etc.

Autoridad de registro (RA)

Es un componente opcional que se encarga de todas las tareas administrativas que en general podrían ser llevadas a cabo por una CA. Está asociada al proceso de registro de la entidad final y se encarga de verificar su identidad. Algunas otras funciones son:

- Validación de los atributos del solicitante del certificado.
- Comprobación de que el solicitante posee la clave privada registrada.
- Generación del par de claves.
- Intermediario entre el usuario final y la CA.

Aunque la RA se puede ocupar de muchas tareas administrativas opcionales de la CA, nunca puede emitir un certificado digital.

Repositorio

En general hace referencia a cualquier método utilizado para almacenar datos de interés para una PKI, como certificados de clave pública o CRLs.

Emisor de listas de revocación de certificados.

Como su propio nombre indica es la entidad encargada de emitir la lista de revocación de certificados. Como se ha comentado anteriormente, un certificado es necesario revocarlo si ha expirado, si se ha perdido el certificado, si ha sido robado, si ha cambiado la autoridad certificadora, etc.

Algunas de las funciones principales de gestión que realiza una PKI se pueden resumir en las siguientes:

- Registro.
- Inicialización.
- Certificación.
- Recuperación del par de claves.
- Actualización del par de claves.
- Petición de revocación.

2.3.6. Protocolo SSL/TLS

El protocolo SSL es un protocolo que permite establecer una conexión segura a través de internet. El objetivo inicial fue proteger las comunicaciones entre cliente y servidor que se realizaban con el protocolo HTTP. [62]

Proporciona confidencialidad, integridad y verificación de la identidad de ambas partes. Forma parte de la capa de aplicación y va sobre el protocolo TCP, por debajo de protocolos como HTTP, IMAP, LDAP, etc. Se usa principalmente para proporcionar seguridad a los protocolos HTTP (web), SMTP (email) y NNTP (news). Proporciona codificación a los mensajes antes de ser enviados por internet.

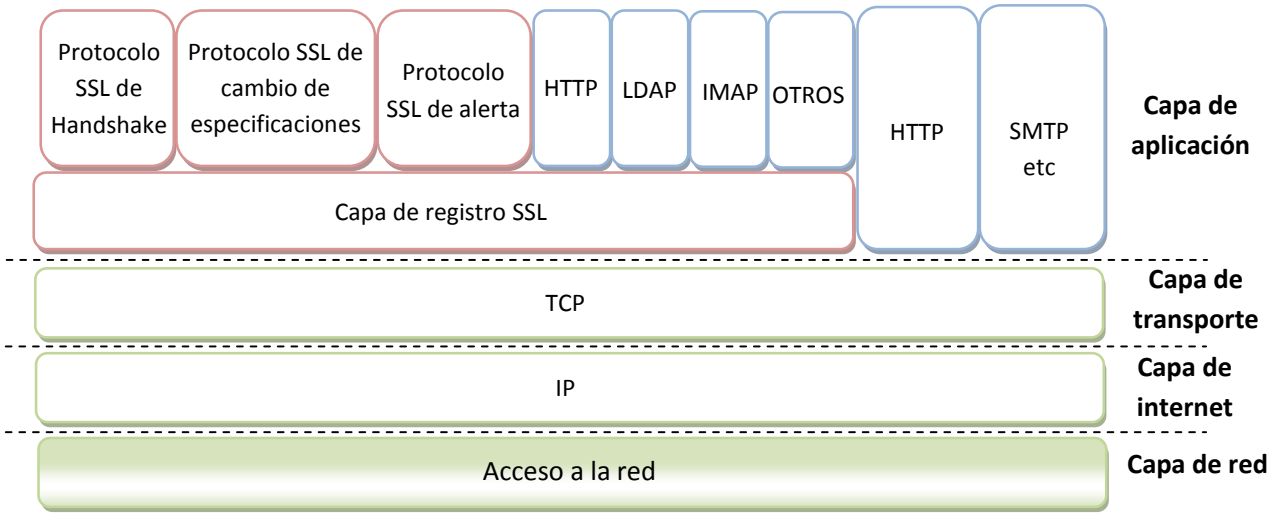


Figura 2.18. Representación de SSL en la jerarquía de protocolos. La imagen representa cada una de las capas que componen el protocolo SSL.

El protocolo SSL handshake se usa para negociar los parámetros de seguridad de una conexión. Cuando se va a establecer una conexión entre un servidor y un cliente, estos se ponen de acuerdo en la versión de protocolo se va a usar (SSL v1, SSL v2, SSL v3, etc), se selecciona un el algoritmo criptográfico que se va a utilizar, se autentican, y se utilizan técnicas de cifrado de clave pública para compartir información. Constituye la primera fase de una comunicación SSL.

El protocolo de cambio de especificaciones de cifrado se encarga de actualizar el repertorio de cifrado, realizando las transiciones entre las distintas políticas de cifrado o los cambios que se pueden producir durante la conexión.

El protocolo de alerta envía mensajes avisando de los problemas que pueden surgir durante la conexión, cerrando la conexión si existe un problema grave.

El protocolo de registro asegura la autenticación, integridad y confidencialidad de los mensajes enviados. La autenticación y la integridad se garantizan mediante el uso de firmas digitales configuradas según los parámetros enviados durante el handshake o negociación. La confidencialidad se consigue mediante cifrado simétrico y la clave establecida durante la negociación. [36]

SSL tiene un mecanismo de autenticación que permite la comunicación segura. Lo habitual es que lo que se autentique sea el servidor, aunque también puede existir el caso en el que se autentique tanto el cliente como el servidor y el caso en el que no se tengan que autenticar ni cliente ni servidor, en ese caso se utiliza para cifrado el algoritmo Diffie-Hellman. En general, todas las aplicaciones tienen almacenado el certificado de la CA que permite verificar la identidad del servidor, de este modo, cuando un servidor envía su certificado al establecer la conexión, se verifica la autenticidad del mismo.

Una comunicación SSL implica cuatro fases:

- FASE 1: Establecimiento de la conexión y negociación de los algoritmos criptográficos que van a usarse en la comunicación. El cliente le envía un mensaje de *"Hola"* al servidor, si el servidor no responde con otro mensaje de *"Hola"* ocurrirá un error fatal y la conexión fallará. Específicamente los atributos que se establecen son: versión de protocolo, ID de sesión, cipher suite[9] y método de compresión. Además se pueden intercambiar dos valores generados aleatoriamente.

- FASE 2 y 3: Intercambio de claves de una longitud suficiente, mediante algún mecanismo de clave pública o criptografía asimétrica y autenticación de los

[9] El cipher suite define la versión del protocolo, el algoritmo de cifrado y la longitud de la clave asociada.

interlocutores a partir de sus certificados digitales. Se usan dos claves: una para la comunicación entre el cliente y el servidor y otra en sentido contrario. Más específicamente el servidor envía su certificado que será autenticado, se envía un mensaje de intercambio de claves del servidor (sólo se requiere el envío de este mensaje si el servidor no tiene certificado o es sólo de firma), si el servidor es autenticado, opcionalmente se puede pedir al cliente su certificado, si sus características coinciden con el cipher suite. Finalmente, el servidor envía un mensaje de finalización de "Hola" que indica que la fase anterior de establecimiento de conexión (saludo de hanshake) ha finalizado. El cliente puede responder con un certificado o que no tiene certificado (la autenticación de cliente es opcional). Se realiza entonces el intercambio de claves del cliente. El certificado enviado por el cliente puede tener la capacidad de ser firmado, entonces se enviará al cliente un mensaje de verificación de firma.

- FASE 4: Comunicación secreta y protección de la integridad de los datos mediante cifrado simétrico del tráfico. Se realiza una última actualización de especificaciones de cifrado. En este punto el handshake ha finalizado y ya cliente y servidor pueden intercambiar datos de la capa de aplicación. [23][37]

El funcionamiento del handshake, es decir, la negociación que se realiza entre cliente y servidor para que se establezca la conexión, se muestra en la siguiente página.

Los algoritmos empleados por SSL son:

- Parte asimétrica: RSA, Diffie-Hellman o DSA.
- Parte simétrica: RC2, RC4, IDEA, DES TripleDES o AES.
- Funciones resumen: SHA-1 o MD5.

Si el algoritmo que se usa para la autenticación del servidor y el intercambio de claves es RSA, entonces se genera un número aleatorio secreto de 48 bytes por parte del cliente, cifrado con la clave pública del servidor. Este número se envía al servidor, el servidor utiliza su clave privada para descifrarlo y entonces se genera la clave de sesión que finalmente establece la conexión.

Las ventajas de SSL son que liberan a las aplicaciones de llevar a cabo las operaciones criptográficas antes de enviar la información, y su transparencia permite usarlo de manera inmediata sin apenas modificar los programas ya existentes.

En cuanto al protocolo TLS, se trata de una versión mejorada de SSL (está basada en la versión 3.0) que lo hace incompatible con este.

Una de las mejoras con respecto a SSL es que permite una comunicación sobre una conexión TCP ya existente, esto permite utilizar los mismos puertos que protocolos no cifrados.

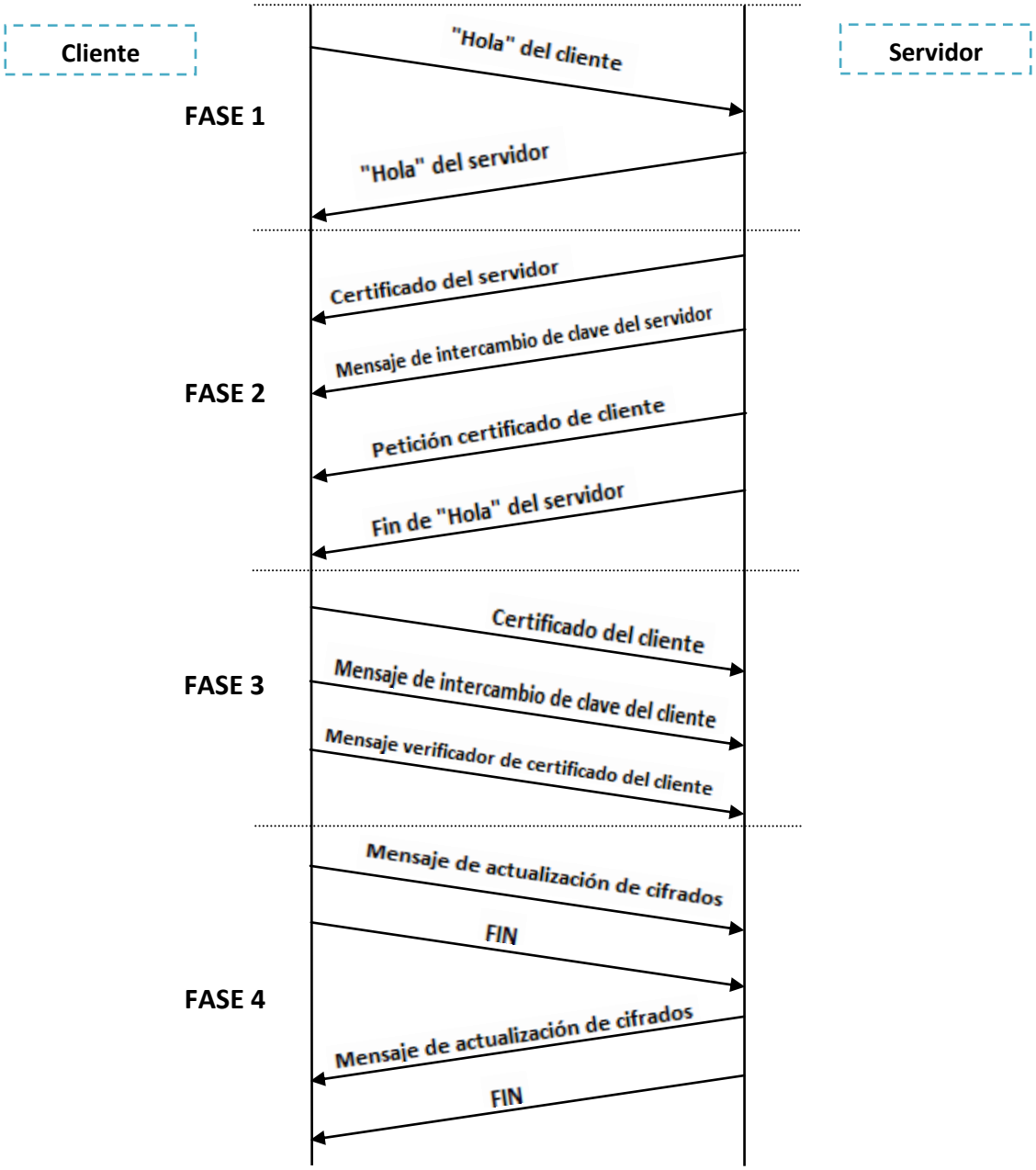


Figura 2.19. Representación del establecimiento de una conexión SSL o Handshake. La imagen representa cada una de las fases definidas anteriormente que componen el handshake.

En este protocolo se emplea una serie de medidas de seguridad adicionales, encaminadas a protegerlo de distintos tipos de ataque:

- Uso de funciones MAC en lugar de funciones MDC únicamente.
- Numeración secuencial de todos los campos que componen la comunicación, e incorporación de esta información al cálculo de los MAC.

- Protección frente a ataques que intentan forzar el empleo de versiones antiguas (menos seguras) del protocolo o cifrado más débiles.
- El mensaje que finaliza la fase de establecimiento de la conexión incorpora una signatura (hash) de todos los datos intercambiados por ambos interlocutores. [23]

Los mecanismos de seguridad que ofrecen los protocolos SSL/TLS son los siguientes:

- **Confidencialidad.** Mediante el intercambio de paquetes con datos cifrados mediante claves simétricas. Se negocian las claves que se utilizarán para cifrar los datos. Se utilizan dos claves diferentes: una para la comunicación entre cliente-servidor y otra para el sentido contrario. Para evitar que un atacante pueda saber que claves se han acordado, se sigue un mecanismo seguro de intercambio de claves basándose en criptografía asimétrica o de clave pública y el algoritmo empleado en este intercambio se negocia también durante el establecimiento de la conexión.
- **Autenticación de entidad.** Mediante el uso de firmas digitales, el cliente puede confirmar la identidad del servidor al cual se conecta. Estas firmas, se verifican con el certificado o clave pública del servidor.
- **Autenticación de cliente.** Mediante el uso de contraseñas introducidas por el usuario.
- **Autenticación de mensaje.** Los paquetes enviados pueden ir cifrados y además incorporar un código MAC para que el destinatario compruebe que nadie ha modificado el paquete.
- **Eficiencia.** Mediante la definición de sesiones, se pueden iniciar conexiones seguras o simultaneas sin necesidad de repetir la autenticación y las claves y mediante la compresión para datos intercambiados.
- **Extensibilidad.** Como se ha comentado, al inicio de cada sesión se negocian entre el cliente y el servidor los algoritmos que se utilizarán para el intercambio de claves, la autenticación y el cifrado. Las especificaciones de los protocolos incluyen unas combinaciones predefinidas de algoritmos criptográficos, pero existe la posibilidad de añadir nuevos algoritmos que puedan ser más eficientes o más seguros.

Los protocolos SSL/TLS están diseñados para prevenir los siguientes ataques:

- Lectura de los paquetes enviados por el cliente y el servidor.
- Suplantación de servidor o cliente.

- Alteración de los paquetes.
- Repetición, eliminación o reordenación de paquetes.

Aunque SSL/TLS son sistemas bastante seguros para el establecimiento de una conexión, se puede producir ataques a la seguridad si se engaña a un usuario con la dirección con la que se conecta o con el certificado digital que autentifica al servidor.

En el año 2011 dos profesionales de la seguridad demostraron una vulnerabilidad en el protocolo TLS. Idearon una aplicación que permitía descifrar una cookie de autenticación usada para acceder a una cuenta de Paypal. [38]

Los protocolos SSL/TLS son una de las herramientas de las que se sirven las PKI para proporcionar una conexión segura, sin embargo, toda la seguridad que proporciona una PKI, no depende exclusivamente de este protocolo. El protocolo SSL sólo puede autenticar al servidor y al cliente de una comunicación, pero no puede identificar ni autenticar a la persona que establece la conexión, por tanto, una persona que no fuera dueño de un ordenador podría acceder a una página protegida por SSL, lo que en ocasiones podría suponer un problema de seguridad.

En un sistema que use SSL con autenticación de servidor únicamente, existe la posibilidad de que alguien tenga acceso mediante credenciales falsas. Por otra parte, no es factible utilizar autenticación de clientes en sistemas en los que haya muchos clientes y un solo servidor.

Si alguien intercepta una comunicación haciéndose pasar por el servidor real, podría tener acceso a datos personales de la persona que se conecta.

Por tanto, una PKI utiliza SSL para establecer una conexión segura, pero SSL no ofrece otros servicios, como por ejemplo, servicios de registro o login o firma digital de documentos que si ofrece una PKI.

2.4. ESTUDIO Y COMPARATIVA DE LA OBTENCIÓN Y USO DE IDENTIDADES DIGITALES EN LAS TECNOLOGÍAS ACTUALES.

En la actualidad existe una gran cantidad de servicios que requieren el uso de identidades digitales. Estas identidades se usan principalmente para el acceso a varios servicios y para la firma de datos que requieren la identificación de la persona que los ha enviado (por ejemplo, la firma electrónica en un correo electrónico). Esta firma legalmente tiene la misma validez que la firma manuscrita.

Para conseguir un certificado digital, una persona puede solicitarlo a una entidad encargada de emitir certificados. Por ejemplo, en España la Fabrica Nacional de Moneda y Timbre (FNMT) se encarga de distribuir certificados de un tipo u otro a las personas que lo soliciten dependiendo de si se refiere a una persona física, a una persona jurídica, una entidad sin personalidad jurídica, a la administración pública o certificados de componentes (para aplicaciones informáticas). Existen más entidades emisoras de certificados como Camerfirma; la Dirección General de Policía que emite los DNIs electrónicos; y las respectivas entidades correspondientes a Comunidades Autónomas, como la de la de Cataluña (Agència Catalana de Certificació), la de la Comunidad Valenciana (Autoridad de Certificación de la Comunidad Valenciana o ACCV) o la del País Vasco (Izenpe).

En la FNMT [63], un certificado digital se puede obtener de varias formas:

1. Como un archivo descargable en el ordenador.
2. En una tarjeta criptográfica.
3. En un dispositivo Android (la aplicación no existe en dispositivos con iOS).
4. Utilizando el DNIe.

En el caso 1 y 2 el certificado se solicita via internet (http://www.fnmt.es/home), se obtiene un código y después se acude a una Oficina de Registro para que la persona presente una serie de credenciales de que le identifiquen (como por ejemplo, el DNI) y el código obtenido, de esa forma el certificado se activará y podrá ser descargado por el usuario desde la página web de la FNMT.

El certificado se instala en el navegador que usa el usuario, de forma que ahora el usuario tiene acceso a muchos más servicios de los que tenía antes, unas veces de forma transparente a él y otras tendrá que buscar su certificado en el navegador.

En el caso 4 el usuario no necesita presentarse en una oficina

de registro pues ya se identifica mediante el DNIe, a diferencia de lo que ocurre en el caso 3, en el que el usuario si necesita presentarse a una entidad de registro.

La FNMT tiene una aplicación Android para la obtención de certificados digitales.

El usuario tiene que descargarse la aplicación en el móvil. Como en los casos 1 y 2, el usuario recibe un código en el móvil. Este código tiene que ser presentado en la oficina de registro y después se le permitirá al usuario descargar el certificado a su dispositivo. La clave privada se genera en el propio dispositivo en el momento en el que se solicita el código, pero en la información sobre la aplicación, no se especifica si el CSR se genera en la aplicación o no.

Figura 2.20. Aplicación de la FNMT para la obtención de certificados. Estas imágenes muestran dos actividades de la aplicación para la distribución de certificados digitales de la aplicación de la FNMT.

Este sistema es similar al que se desarrolla en este proyecto, en el sentido de que el usuario recibe un código para poder obtener el certificado, con la diferencia que en este proyecto, la identidad y el registro se realiza en la propia aplicación. Además, el formato del almacén de claves en la aplicación de la FNMT es diferente a la usada en la aplicación desarrollada en este proyecto.

Una de las empresas más importantes que se dedican al desarrollo de software de clave pública es Safelayer. Esta empresa colabora en varios proyectos importantes en relación a la seguridad, certificación y uso de identidades digitales, tales como la FNMT, la implantación y uso de DNIe, etc.[64]. También trabajan en el desarrollo de aplicaciones para móviles. Según esta empresa, el uso de identidades en dispositivos móviles presenta las siguientes ventajas:

- Los certificados digitales son de fácil uso en los dispositivos móviles.
- La revocación de un certificado bloquea el acceso del dispositivo en la red y otros servicios corporativos como el correo electrónico.

- El uso de certificados digitales permite realizar operaciones tales como firma digital o cifrado de datos en las aplicaciones. [39]

El ayuntamiento de Barcelona dispone de una aplicación similar a la de la FNMT, que permite activar una identidad digital. En este caso el usuario tiene primeramente que acercarse a una oficina donde se registra y recibe un documento con un código y un código QR que le permite descargar la aplicación en su móvil y activar su identidad. Esta aplicación le permite el acceso a diferentes servicios como los servicios de empadronamiento, los servicios de grúa municipal y la gestión de multas. En caso de perder el móvil el usuario tiene que volver a la oficina de registro para dar de baja la identidad anterior y solicitar una nueva. [65][66]

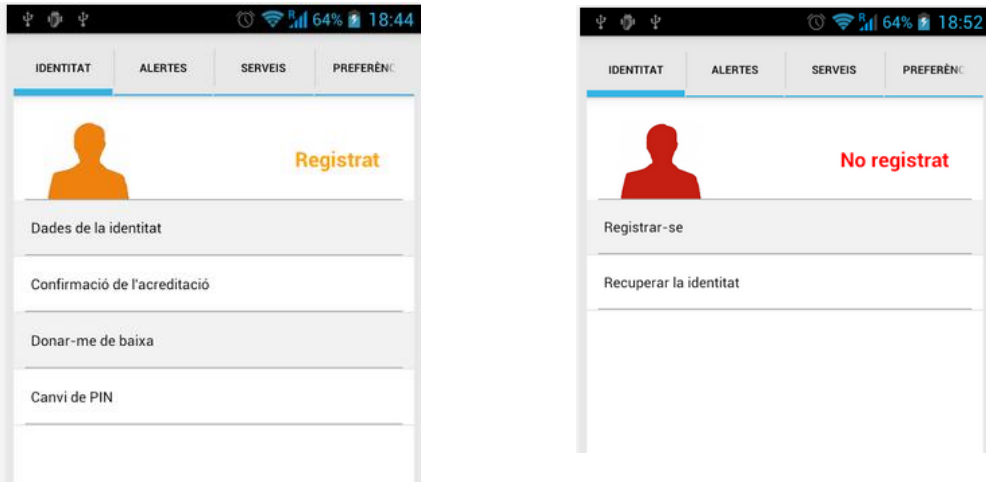


Figura 2.21. Aplicación idBCN del Ayuntamiento de Barcelona. Esta imagen representa dos de las actividades de la aplicación idBCN.

Existen además, aplicaciones que permiten crear identidades digitales no a partir de un certificado digital, si no a partir de un usuario y una contraseña y el uso de señales biométricas como la voz. Este es el caso de una aplicación llamada idfor.me, que activa un usuario y contraseña para cualquier plataforma, después de haber identificado a la persona mediante su voz.

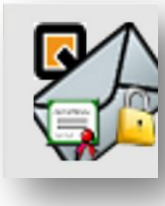
Según el conocimiento que tiene la autora de este proyecto, hasta la fecha no existen más aplicaciones que proporcionen identidades digitales a los usuarios, pero sí que existen bastantes aplicaciones que hacen uso de identidades digitales o certificados digitales o los gestionan para que los usuarios hagan un uso adecuado.

A parte de las aplicaciones anteriormente nombradas, existen una serie de aplicaciones que sirven para la gestión de certificados por parte del usuario o incluso permiten ciertas operaciones de firma digital, cifrado de correos electrónicos, etc. Aunque no tiene relación con la obtención de un certificado digital como una identidad digital y su posterior uso (como en el caso de este proyecto), es importarte mostrar como funcionan este tipo de aplicaciones.

Se hace a continuación una breve descripción de algunas de las aplicaciones existentes para la gestión de certificados:

- **Mobile Identity Agent:** Gestiona los certificados que usa el usuario y permite que estos certificados se usen para firma S/MIME y cifrar emails; establecer una conexión SSL con autenticación de cliente y servidor; y VPN. Las siguientes versiones que se hagan de esta aplicación incluirán la capacidad de almacenar claves públicas y privadas para el uso exclusivo dentro de la aplicación.

- **X509Tools y R2Mail2:** Al igual que en el ejemplo anterior, esta aplicación es un cliente de correo electrónico que permite gestionar los certificados con los que los usuarios cifran sus correos (correos enviados sobre el protocolo SMTP) y también firmarlos con ellos.



- **Symantec PKI client:** es una aplicación de la empresa Symantec y permite a los usuarios hacer uso de una estructura PKI que gestiona los certificados que el usuario usa cotidianamente. Por ejemplo, un usuario hace uso de un certificado otorgado por la empresa para la que trabaja y requiere hace uso de esta aplicación para gestionar su almacenamiento, su uso o lo necesita para establecer una conexión mediante VPN con la empresa.

- **CACertMan:** es una aplicación creada por el grupo de The Guardian Project, que es una iniciativa de desarrolladores para la creación de código abierto de aplicaciones seguras. Esta aplicación permite gestionar el uso de certificados raíz que usan los dispositivos Android.

- **Gnu Privacy Guard:** es otra aplicación desarrollada por el grupo de The Guardian Project que haciendo uso de líneas de comando y GPG que es la alternativa libre a PGP, permite el establecimiento de comunicaciones seguras y cifrados de datos entre un extremo y otro de la comunicación. Esta aplicación está desarrollándose actualmente.

Además de la aplicación anterior que hace uso del protocolo PGP en lugar del X.509, existen también algunas aplicaciones similares que permiten la gestión de los certificados PGP y cifrado de datos para comunicaciones seguras. Aunque el desarrollo de la aplicación llevada a cabo en este proyecto usa el protocolo X.509, es necesario mencionar estas otras alternativas. Algunos ejemplos de aplicaciones que usan PGP a parte de la anteriormente nombrada, se muestran a continuación:

- **APG:** es el de desarrollo de OpenPGP para Android. La meta de esta **aplicación** es la de realizar una implementación OpenPGP similar a la de GnuPG. Permite gestionar claves privadas y públicas; cifrar, firmar y verificar emails; creación y edición de claves, etc.

- **OpenPGP Manager:** es cliente PGP para Android, que permite crear, importar, exportar y publicar claves PGP, enviar correo cifrados, cifrado de ficheros con password y PGP y por último firma y verificación de firma.

También existen implementaciones que combinan el uso tanto de hardware, mediante el uso de smartcard y de software, que permiten instalar funciones criptográficas en la tarjeta del móvil y que permiten las mismas opciones de cifrar emails y establecer conexiones seguras.

Por tanto, como conclusión, puede decirse que si bien existen muchas aplicaciones que hacen uso de certificados digitales, principalmente para la firma de documentos y correos electrónicos, así como aplicaciones dedicadas a la gestión de certificados; no podemos decir que haya un gran número de aplicaciones que permitan la obtención de una identidad digital y menos aún sin que el usuario necesite acercarse a una oficina para registrarse, tal como es uno de los objetivos de este proyecto.

Capítulo 3:

Metodología y uso de las herramientas

3. METODOLOGÍA Y USO DE LAS HERRAMIENTAS

3.1. INTRODUCCIÓN

En este capítulo se hará una descripción y análisis sobre el uso que se han hecho de las diferentes herramientas para poder conseguir el resultado obtenido en la aplicación final de la que trata este proyecto.

Se pretende explicar cómo se han utilizado las herramientas de programación, en particular las usadas para crear una aplicación en Android y las diferentes librerías necesarias.

Además, se quiere dar una descripción de ciertas herramientas criptográficas que permiten la implementación de firmas digitales, creación de claves criptográficas, etc.

Las herramientas criptográficas utilizadas son:

- *Keytool:* un programa que viene proporcionado por JRE[10] y proporciona utilidades de gestión de claves y certificados.

- *OpenSSL:* es un proyecto de software libre que consiste en un paquete de herramientas de administración y bibliotecas relacionadas con la criptografía.

- *Portecle:* es una interfaz gráfica de usuario que permite la gestión de claves, certificados, CRLs, CSRs, keystore, truststore, etc.

También, se explican qué librerías se han utilizado para la realización del proyecto, espcialmente se hará un análisis de las librerías criptográficas que utiliza Java y Android: JCA, JCE y JSSE; así como librerías externas que también se hayan usado o se podrían usar: Bouncy Castle o Spongy Castle.

[10]JRE: Java Runtime Environment. Es un conjunto de utilidades que permiten la ejecución de programas Java.

3.2. DESARROLLO DE APLICACIONES EN ANDROID

Para poder desarrollar una aplicación en Android es necesario tener instalado en el ordenador de trabajo una serie de herramientas. [41][51]

Actualmente la instalación de las herramientas se realiza de una forma muy sencilla. La persona que quiera desarrollar una aplicación tendrá que descargarse e instalarse las siguientes herramientas:

Java SE Development Kit (JDK)

Android hace uso de JDK para el desarrollo de aplicaciones. Este paquete se encuentra de forma libre en la página de Oracle que es la propietaria de Sun Microsystems, la compañía que desarrolló java:
http://www.oracle.com/technetwork/java/javase/downloads/index.html

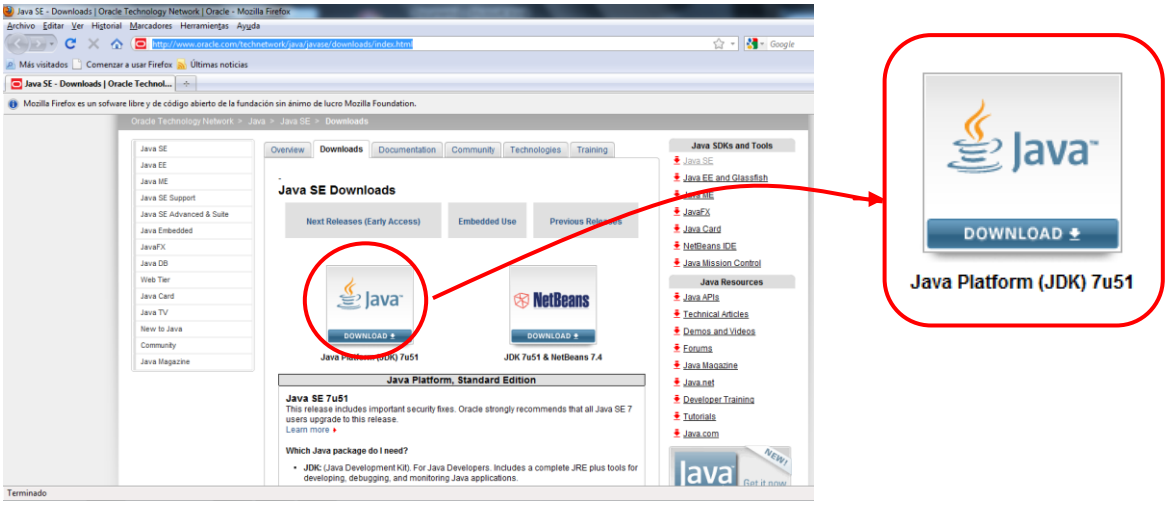
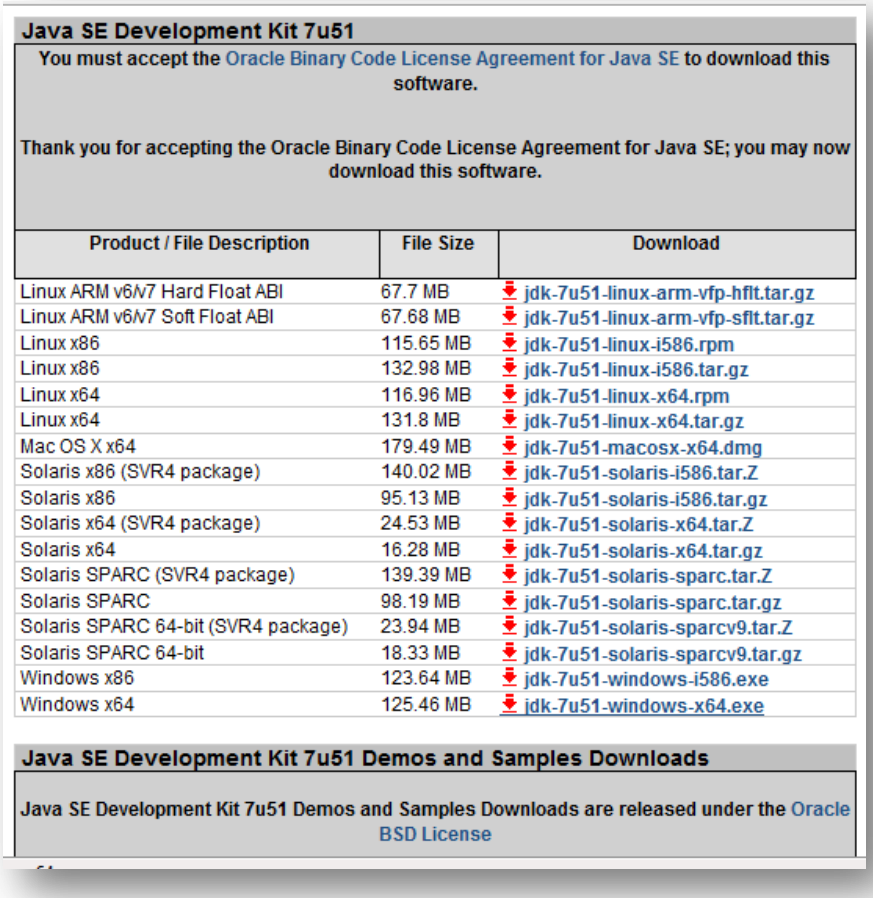


Figura 3.1. Descarga de JDK. Esta imagen muestra donde un desarrollador debe descargarse el JDK de Java en la página de Oracle

Al pulsar sobre el enlace se muestra una lista para descargar JDK según el sistema operativo que usemos:



Java SE Development Kit 7u51

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	67.7 MB	jdk-7u51-linux-arm-vfp-hflt.tar.gz
Linux ARM v6/v7 Soft Float ABI	67.68 MB	jdk-7u51-linux-arm-vfp-sflt.tar.gz
Linux x86	115.65 MB	jdk-7u51-linux-i586.rpm
Linux x86	132.98 MB	jdk-7u51-linux-i586.tar.gz
Linux x64	116.96 MB	jdk-7u51-linux-x64.rpm
Linux x64	131.8 MB	jdk-7u51-linux-x64.tar.gz
Mac OS X x64	179.49 MB	jdk-7u51-macosx-x64.dmg
Solaris x86 (SVR4 package)	140.02 MB	jdk-7u51-solaris-i586.tar.Z
Solaris x86	95.13 MB	jdk-7u51-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	24.53 MB	jdk-7u51-solaris-x64.tar.Z
Solaris x64	16.28 MB	jdk-7u51-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	139.39 MB	jdk-7u51-solaris-sparc.tar.Z
Solaris SPARC	98.19 MB	jdk-7u51-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	23.94 MB	jdk-7u51-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	18.33 MB	jdk-7u51-solaris-sparcv9.tar.gz
Windows x86	123.64 MB	jdk-7u51-windows-i586.exe
Windows x64	125.46 MB	jdk-7u51-windows-x64.exe

Java SE Development Kit 7u51 Demos and Samples Downloads

Java SE Development Kit 7u51 Demos and Samples Downloads are released under the Oracle BSD License

Figura 3.2. Lista de JDK según el sistema operativo. El JDK de Java está disponible en varios sistemas operativos y el desarrollador lo puede descargar dependiendo del sistema operativo que use en su ordenador de trabajo.

Android SDK, Android ADT y Eclipse

Android SDK proporciona las librerías de las APIs, documentación y ejemplos y todas las herramientas para que el desarrollador pueda construir, comprobar y depurar aplicaciones. Además esta herramienta permite emular las aplicaciones sin necesitar de un dispositivo real.

Android SDK se descarga desde la página del desarrollador de Android:

http://developer.android.com/sdk/index.html

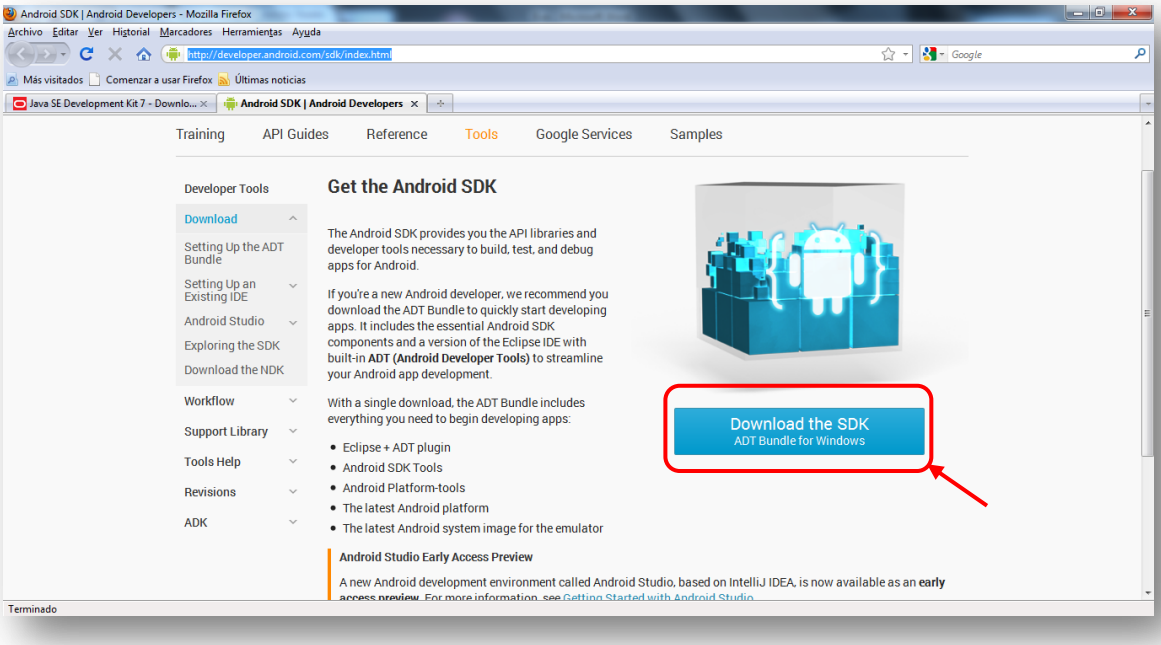


Figura 3.3. Descarga de Android SDK. En la página del desarrollador de Android, se indica a los desarrolladores principiantes los pasos que deben seguir para instalarse las herramientas. En la imagen se muestra donde está el enlace para descargarse el SDK de Android.

Al pulsar sobre el botón, se descargará un archivo .rar que contiene el directorio de instalación de las herramientas y dentro de este, otros dos directorios:

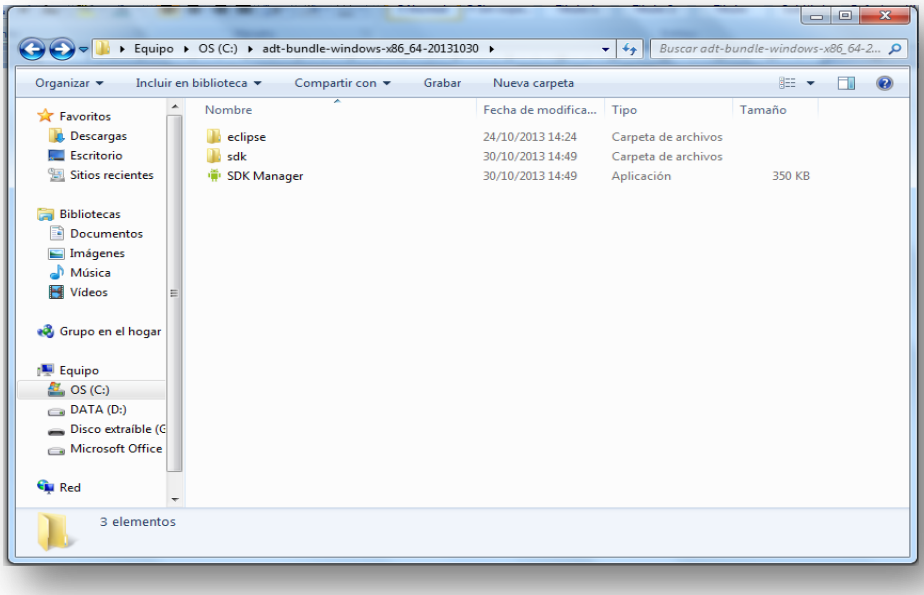


Figura 3.4. Contenido del directorio de instalación de las herramientas de Android. Dentro del archivo *.rar* que se descarga de la página de desarrollador de Android, se encuentra un directorio que contiene los otros tres directorios que se muestran en la imagen.

En el directorio del sdk, hay otra serie de directorios que contienen datos que configuran el sdk:

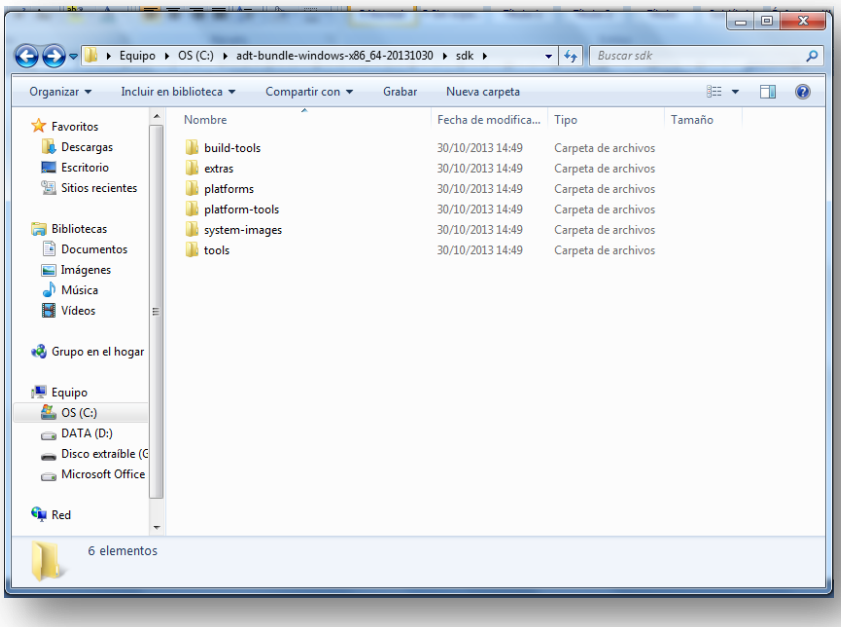


Figura 3.5. Contenido del directorio del sdk. Dentro del directorio del sdk, se encuentran una serie de directorios con todas las caracteristicas del sdk.

Para terminar la instalación hay que acceder al directorio de Eclipse y pulsar sobre su ejecutable:

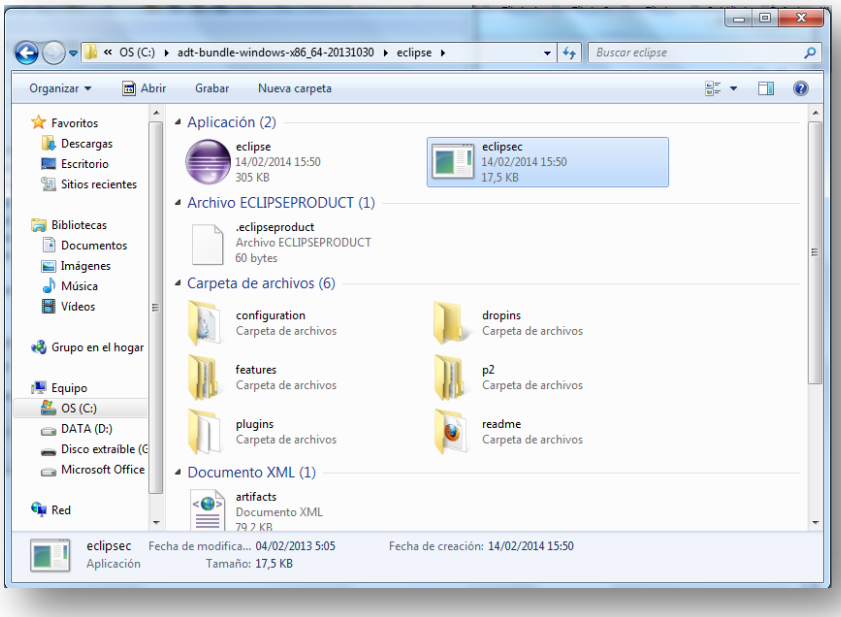


Figura 3.6. Contenido del directorio de Eclipse. Dentro del directorio de Eclipse, se encuentran los ejecutables de Eclipse y algunas carpetas extra con diferente documentación.

Con ello, se instala directamente Android ADT:

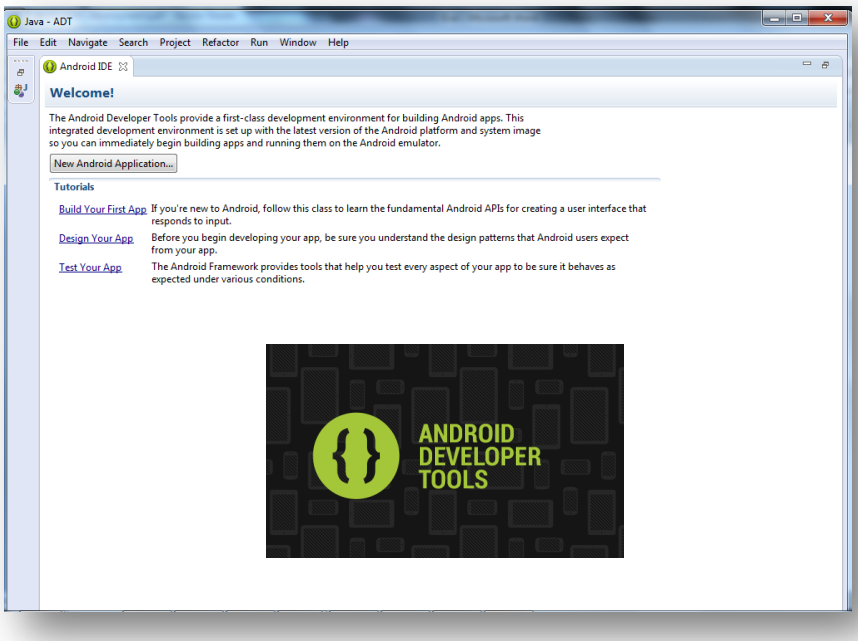


Figura 3.7. Ejecución de Android ADT. Al pulsar sobre el ejecutable de Eclipse se ejecuta directamente el ADT y se instala Eclipse.

El ADT (Android Development Tools o Herramientas de Desarrollo de Android) es el plug-in de Eclipse. El conjunto de herramientas de ADT nos suministra:

- El asistente de proyectos de Android para generar los archivos que requiere un proyecto.
- Editores de recursos específicos que necesita Android.
- Gestor de AVD. Esta herramienta que proporciona una interfaz gráfica para el lanzamiento de emuladores para la simulación de una aplicación sin necesitar el uso de un dispositivo real.
- DDMS (Dalvik Debug Monitor Server): esta herramienta se usa para acciones tales como el depurado y control de aplicaciones, captura de pantalla, logcat, procesos, llamadas entrantes y gestión de archivos.
- Integración con la utilidad de log LogCat de Android y AHV (Android Hierarchy Viewer).
- Compilación y ejecución de aplicaciones Android para su ejecución en emuladores y dispositivos.

- Herramientas de firma de código con certificados digitales y empaquetado (APK, Android Packages Kit) para la distribución de aplicaciones.

A la vez que se instala Android ADT, se instala Android SDK:

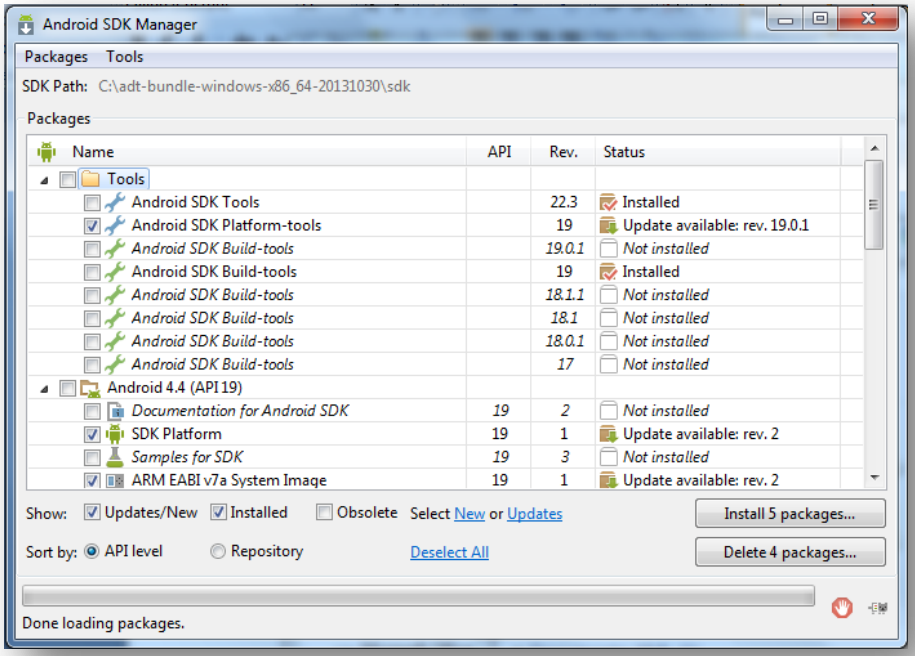


Figura 3.8. Android SDK Manager. Cuando se abre esta ventana el desarrollador debe elegir qué paquetes o características se quieren añadir a Eclipse.

Con esto, ya estarían instaladas las herramientas necesarias. La forma de instalar estas herramientas han cambiado hace poco, pues cuando se comenzó este proyecto, había que descargar cada una de las herramientas por separado.

En el momento en el que se comenzó este proyecto, Eclipse IDE había que descargarlo desde la página http://www.eclipse.org/downloads/. Aunque en teoría se puede trabajar con otros IDEs[11], Eclipse es el único IDE totalmente compatible con el plug-in de Android Development Tools (ADT).

En ese momento, la forma de instalar el ADT era dentro del entorno de Eclipse pulsando *Help->Install New Software*. De esta forma, se abría una nueva ventana que había que rellenar con una serie de datos:

[11] IDE (Integrated Development Environmet o Entorno de Desarrollo Integrado en Español). Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI).

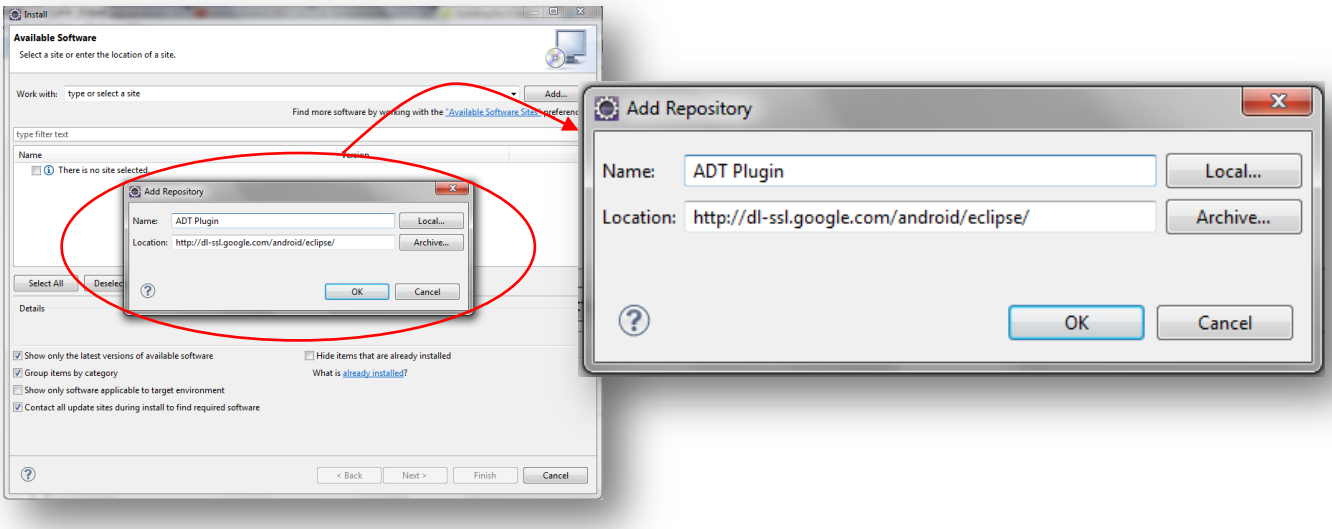


Figura 3.9. Instalación del ADT de Android. Durante el año 2013 la forma de instalarse el ADT era a través de Eclipse. Se mostraba las anteriores ventanas y era necesario rellenar los campos que se indican de igual forma que se muestra en la imagen y con las mismas palabras.

Se pulsa sobre el botón *"Add"* y se añade los valores que se muestran en la figura anterior. [39][40]

Como se ha comentado, esta última forma de instalar ADT se estuvo haciendo de esta forma hasta mediados de 2013. Actualmente, la forma de instalar todas las herramientas es mucho más sencilla y se realiza como se ha explicado a principio de este apartado.

Además, actualmente se está poniendo en marcha el uso de un nuevo Bundle (actualmente Eclipse + ADT) distinto de ADT que se llamará "*Android Studio*". Este es un nuevo entorno de desarrollo de Android basado en intelliJ IDEA. Proporciona herramientas integradas para desarrollar y depurar las aplicaciones.

3.2.1. Creación de un proyecto.

El desarrollo de una aplicación Android se puede llevar a cabo desde diferentes sistemas operativos y mediante la herramienta Eclipse o a través de lineas de comando.

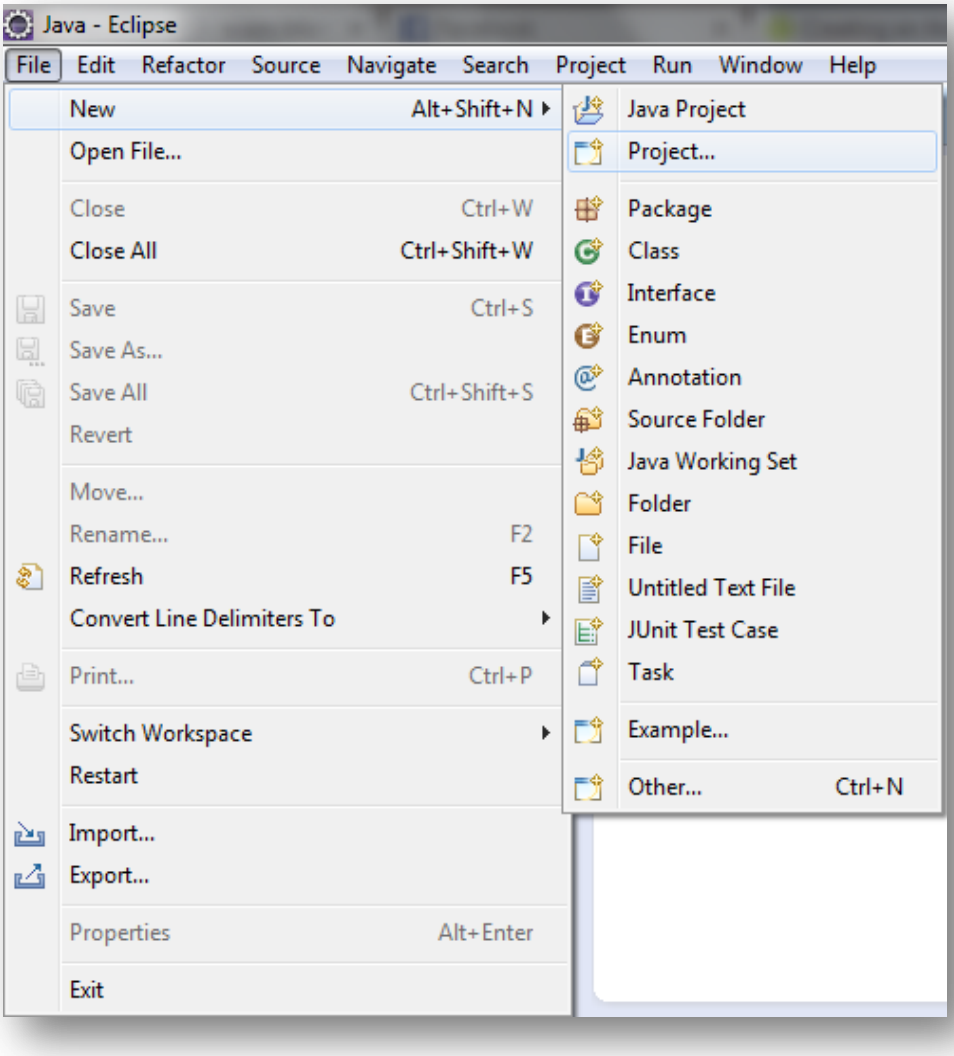
Para la realización de este proyecto se ha usado el sistema operativo Windows mediante el uso del IDE de Eclipse.

Una vez que se instalan todas las herramientas mencionadas en el apartado anterior, se puede empezar al desarrollo de una aplicación.

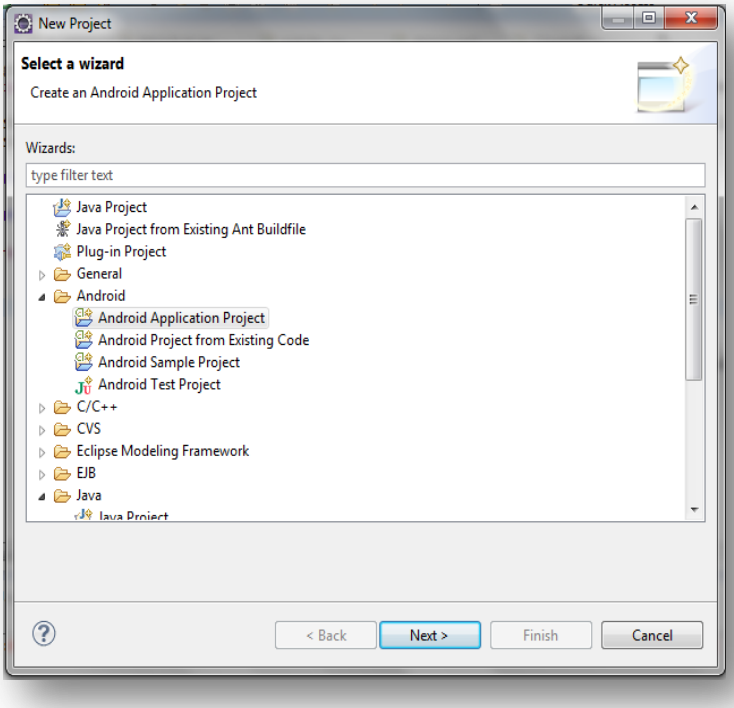
Para mostrar como se crea una aplicación, se va a realizar un pequeño ejemplo de aplicación con dos actividades. En la primera se le mostará al usuario un mensaje y un botón y en la segunda se le mostrará la frase estándar *"Hola mundo"*.

Cuando el usuario pulse sobre el botón de la primera actividad, se pasará a la segunda actividad.

En primer lugar se debe crear un proyecto: *File->Project->Android Application Project.*



Figuras 3.10. Creación de un proyecto en Android. Lo primero que se debe hacer para crear un proyecto es pulsar sobre *File-> New -> Project…*



Figuras 3.11. Creación de un proyecto en Android. Después de seguir las instrucciones de la imagen anterior, se pulsa sobre *"Android Application Project"*

A continuación se abre una nueva ventana que el desarrollador rellena según los criterios que él considere adecuados:

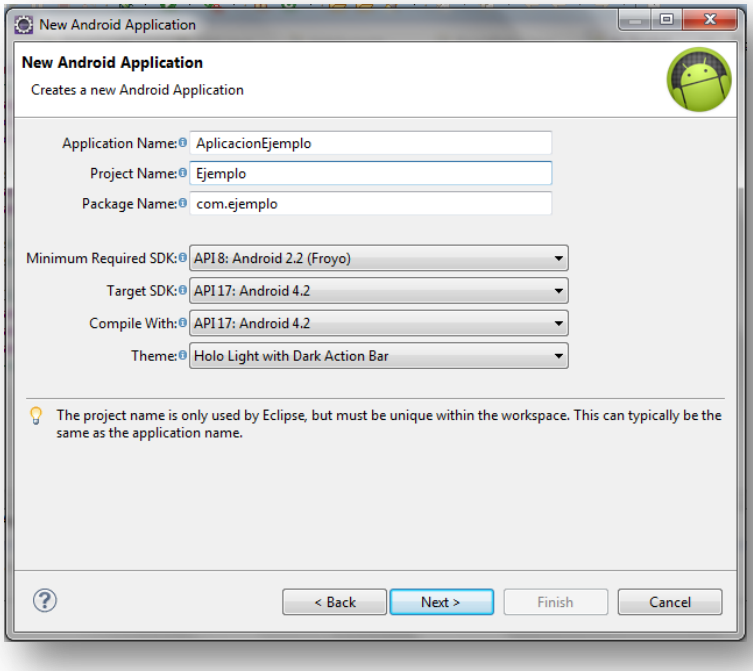


Figura 3.12. Características de una aplicación elegidas por el desarrollador. En esta ventana se especifica el nombre que se le va a dar al proyecto, al paquete de la aplicación, desde qué nivel a qué nivel de API se desarrolla y el estilo de la aplicación.

Los campos que un desarrollador debe rellenar en esta ventana son los siguientes:

- **Application Name:** Es el nombre que le vamos a dar a la aplicación y también el nombre con el que se nombra a la aplicación en el menú inicial del dispositivo móvil.
- **Project Name:** Es el nombre del directorio del proyecto en el que se almacenan todos los archivos generados durante el desarrollo de la aplicación.
- **Package Name:** Es el nombre de los paquetes de la aplicación (sigue el mismo estilo que los paquetes en el lengauje de programación de Java). El nombre sigue el orden inverso del dominio de la organización que desarrolla la aplicación.
- **Minimum Required SDK:** Es la versión más baja que el desarrollador elige para que su aplicación pueda funcionar, es decir, por debajo de ese nivel la aplicación podría no funcionar perfectamente. La versión viene determinado por el nivel de API. Usar un nivel elevado puede permitir dotar a la aplicación de características más novedosas, sin embargo, de cara a poder hacer un uso más general y dependiendo de si se quiere abarcar un mercado mayor, es conveniente poner un nivel más bajo.
- **Target SDK:** Indica la versión más alta de Android con la que el desarrollador quiere comprobar su aplicación. De forma, que a medida que nuevas versiones de Android aparecen en el mercado, será necesario aumentar este valor para poder dotar a la aplicación de nuevas características que no existían en las versiones anteriores. Este valor también se indica según el nivel de API.
- **Compile with:** Identifica el número de versión con el que se va a compilar la aplicación durante el desarrolla la aplicación. Por defecto se suele poner el valor más alto indicado por el Target SDK, pero se puede cambiar. Sin embargo ponerlo en un nivel más alto permite unas características mejores y una gran experiencia usuario pues se adapta a las últimas versiones de dispositivos.
- **Theme:** Especifica la Interfaz de Usuario de Android que el usuario quiere dar a la aplicación. Existen cuatro temas:
    - None: el estilo lo diseña el propio desarrollador.
    - Holo Dark: se aplica un estilo con un fondo oscuro, al igual que el Action Bar[12].

[12] Action Bar, es la barra que aparece en la parte superior de la mayoría de las aplicaciones, con diversos botones de opciones, buscador, etc.

- Holo Ligth: se aplica un estilo con un fondo claro, al igual que el Action Bar.

- Holo Ligth with Dark Action Bar: se aplica un estilo con un fondo claro y un Action Bar oscuro.

A continuación se mostrará una pantalla donde se debe indicar en la carpeta en la que se quiere guardar el proyecto que acabamos de crear.

La siguiente pantalla que aparece permite definir el icono que se va a utilizar como icono de la aplicación:

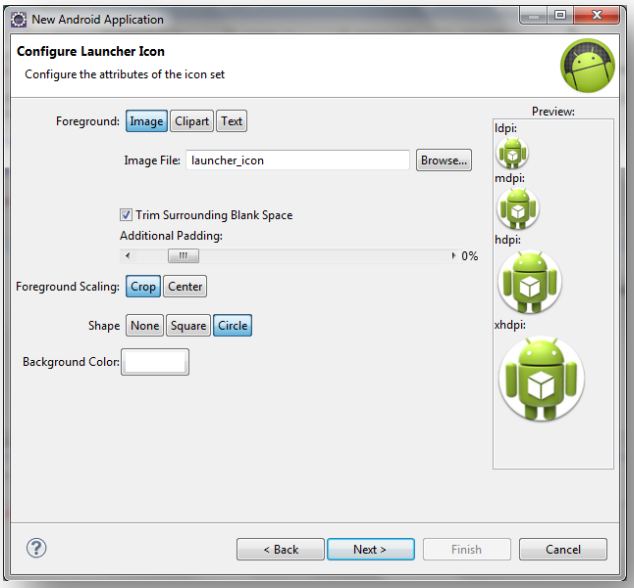


Figura 3.13. Pantalla para la definición del icono de la aplicación. En esta ventana se pueden elegir las características del icono que se va a usar para la aplicación.

Se ven varias opciones:

- Foreground: Permite elegir entre varias opciones para el icono: una imagen elegida por nosotros (opción image), un símbolo existente en el sistema (opción clipart) y por último un texto (opción text).

- Trim surrounding y Aditional Padding: determina el porcentaje del icono que es ocupado por la imagen, símbolo o texto elegidos.

- Foreground Scaling: al igual que las opciones anteriores determina una mayor o menor visibilidad de la imagen, el símbolo o texto elegidos.

- Shape: determina la forma que tendrá el icono. Podrá seleccionarse entre cuadrado, redondo o ninguno de los dos.

- Background Color: sirve para determinar el color del fondo del icono.

3.2.1.1. Estructura del directorio de un proyecto

Una vez que el proyecto se ha creado, se muestra el directorio del proyecto dentro del espacio de trabajo que usa Eclipse para almacenar los proyectos:

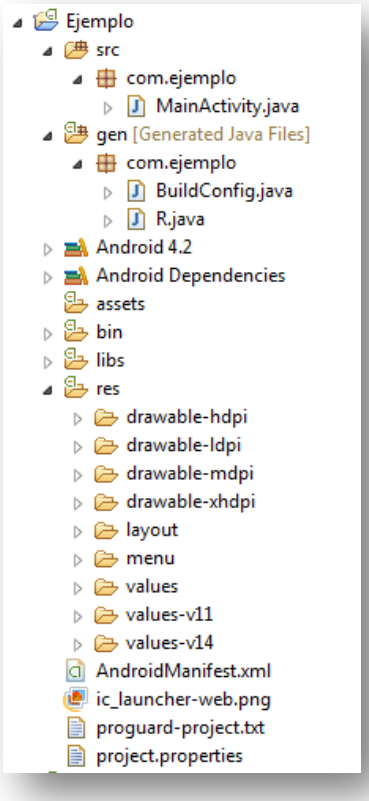


Figura 3.14. Carpetas de un proyecto del directorio de trabajo. Se muestran a la izquierda de Eclipse una lista con los directorios que contienen los fuentes Java, los archivos XML y el AndroidManifest.xml entre otros ficheros.

Las carpetas que se pueden considerar más importantes son:

- Src: en esta carpeta se guarda el paquete del proyecto y dentro del mismo, los archivos *.java* que se generan en cada nueva actividad y los que generan a parte el desarrollador.

- Gen: Contiene el archivo *R.java* que es un archivo que referencia todos los recursos que se encuentran en el proyecto. Este archivo no se debe modificar. Todos los recursos del proyecto se compilan en esta clase y si se necesita se puede hacer referencia a él en el código.

- Assets: contiene archivos adicionales que el usuario necesite utilizar, como bases de datos, archivos de texto, etc. En este proyecto, no ha sido necesario añadir ningún archivo a esta carpeta.

- Bin: contiene todos los archivos creados por el ADT durante el proceso de ejecución. Por ejemplo, contiene el .apk que es el archivo binario que permite ejecutar la aplicación en cualquier dispositivo móvil.

- Libs: en esta carpeta se guardan las librerías que se usan durante el desarrollo de la aplicación. También se almacenan librerías externas. Por ejemplo, en este proyecto, se han añadido en esta carpeta librerías externas como las de BouncyCastle de las que se hablará en este capítulo.

- Res: en esta carpeta se almacenan a su vez una serie de carpetas que contienen los archivos .xml que permiten definir las características gráficas de la aplicación, es decir, los estilos de la aplicación, las imágenes que se han utilizado, los colores, las capas, las cadenas de caracteres, etc.

Un archivo muy importante es el AndroidManifest.xml. Este archivo es el archivo de configuración general para la aplicación. Contiene información sobre el paquete, la versión mínima SDK y los permisos. Además, especifica cada una de las actividades de la aplicación, así como su tema y el titulo de cada actividad. El AndroidManifest.xml de este proyecto se encuentra en el *anexo D*.

3.2.1.2. Ejemplo de configuración de un archivo XML en una aplicación y relación con su archivo Java.

Cuando se han seguido los pasos anteriores, el proyecto ya ha sido creado y ya se puede configurar los archivos para ver una aplicación sencilla, que consiste en dos actividades.

Como se ha comentado con anterioridad, cada actividad representa una pantalla de la aplicación. El concepto de actividad se encuentra explicado en el apartado 2.2 y representado en las figuras *2.3* y *2.4*.

Configuración del archivo XML de la primera actividad.

Se puede hacer de dos formas:

a) Usando la interfaz gráfica que presenta Android, que permite mediante el arrastre de elementos de una lista que aparece a la izquierda, incluir elementos en la actividad:

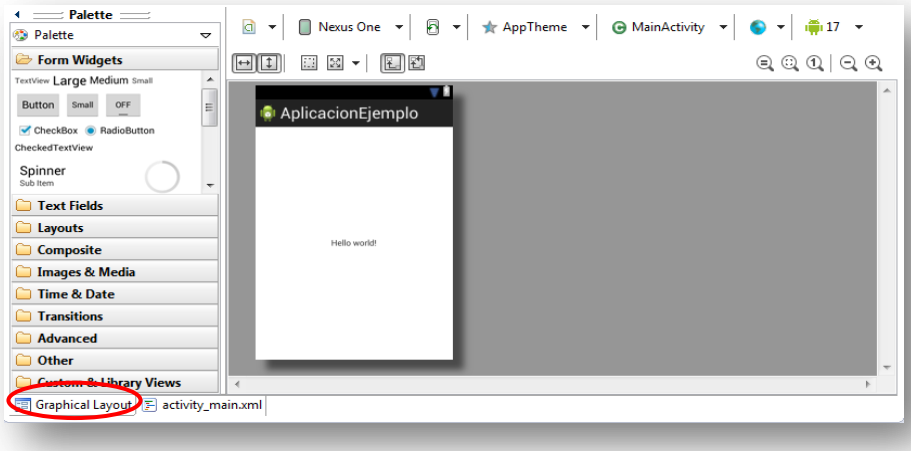


Figura 3.15. Interfaz gráfica para la programación de una actividad. La interfaz ofrece una paleta de características que el desarrollador puede utilizar para programar la apariencia de la aplicación.

b) Usando código en el archivo XML:

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

     <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="141dp"
        android:text="@string/pulse_sig" />

    <Button
        android:id="@+id/boton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView1"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="37dp"
        android:onClick="SiguienteActividad"
        android:text="@string/boton" />

</RelativeLayout>
```

Los *Layouts* son *ViewGroups* que son agrupaciones de *Views* (vistas). Un *view* es un elemento que puede aparecer en una actividad, como un botón, un *TextView* (la vista de un texto), un *EditText* (permite editar un texto), etc.

Existen varios tipos de *Layouts*:

- LinearLayout: permite la vista de sus controles hijos o elementos en horizontal o vertical, según se indique con la línea:

```
android:orientation="vertical"
```

- AbsoluteLayout: permite especificar la posición exacta de sus controles hijos mediante el uso de las siguientes líneas de comando y variando los valores entre comillas:

```
android:layout_x="150px"
android:layout_y="150px"
```

- TableLayout: permite la vista de sus controles hijos en filas y columnas.

- RelativeLayout: permite posicionar sus controles hijos en relación con el resto de elementos.

- FrameLayout: coloca todos sus controles hijos en su esquina superior izquierda, suele usarse únicamente para colocar imágenes. [51]

Cada View y ViewGroup tienen una serie de atributos en común que determinan sus características:

Atributo	Descripción
layout_width	Especifica el ancho del View o del ViewGroup
layout_height	Especifica la altura del View o del ViewGroup
layout_marginTop	Especifica el margen con el elemento superior del View o del ViewGroup
layout_marginBottom	Especifica el margen con el elemento posterior del View o del ViewGroup
layout_marginLeft	Especifica el margen con respecto al elemento de la izquierda
Layout_marginRight	Especifica el margen con respecto al elemento de la derecha
layout_gravity	Especifican como se colocan los controles hijos
layout_weight	Especifica la cantidad de espacio extra se debe asignar a la vista
layout_x	Especifica la coordenada *x* de la vista
layout_y	Especifica la coordenada *y* de la vista

Tabla 3.1. Tabla con la definición de los atributos que se le pueden dar a una vista.[51]

El tamaño de la letra que se mostrará al usuario tanto en el textView como en el botón se determina con un número y un diminutivo correspondiente al tipo de medida, pudiendo ser: *sp (scaled pixels), px (pixels), dp (density-independent pixels), in (inches), mm (millimiters)*.

Las cadenas de caracteres que se muestran al usuario se tienen que declarar en el archivo como un String, esto se puede hacer de varias formas:

a) Introduciendo en el archivo XML la cadena entre comillas en la parte indicada para el texto, señalándolo con el ratón y después pulsando sobre: *Refactor ->Android->Extract Android String…*

b) Mediante líneas de código en el archivo Java:

```
private Button bot;
private TextView texto;

@Override
protected void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	setContentView(R.layout.activity_main);

	//Colocar texto en el TextView
	texto =(TextView)findViewById(R.id.textView1);
	texto.setText("Pulse el botón para pasar a la siguiente
actividad");

	//Colocar texto en el botón
	bot = (Button)findViewById(R.id.boton1);
	bot.setText("Botón");
}
```

c) Abriendo el directorio values, el archivo *String.xml* y declarando la cadena ahí:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">AplicacionEjemplo</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="boton">Botón</string>
    <string name="pulse_sig">Pulse el botón para pasar a la siguiente
actividad</string>

</resources>
```

Los tres primeros String se crean por defecto, de hecho, se podría modificar la primera línea para que saliese otro nombre en la aplicación.

d) Agregando el texto pulsando de manera visual sobre la pestaña *Resources* del archivo XML.

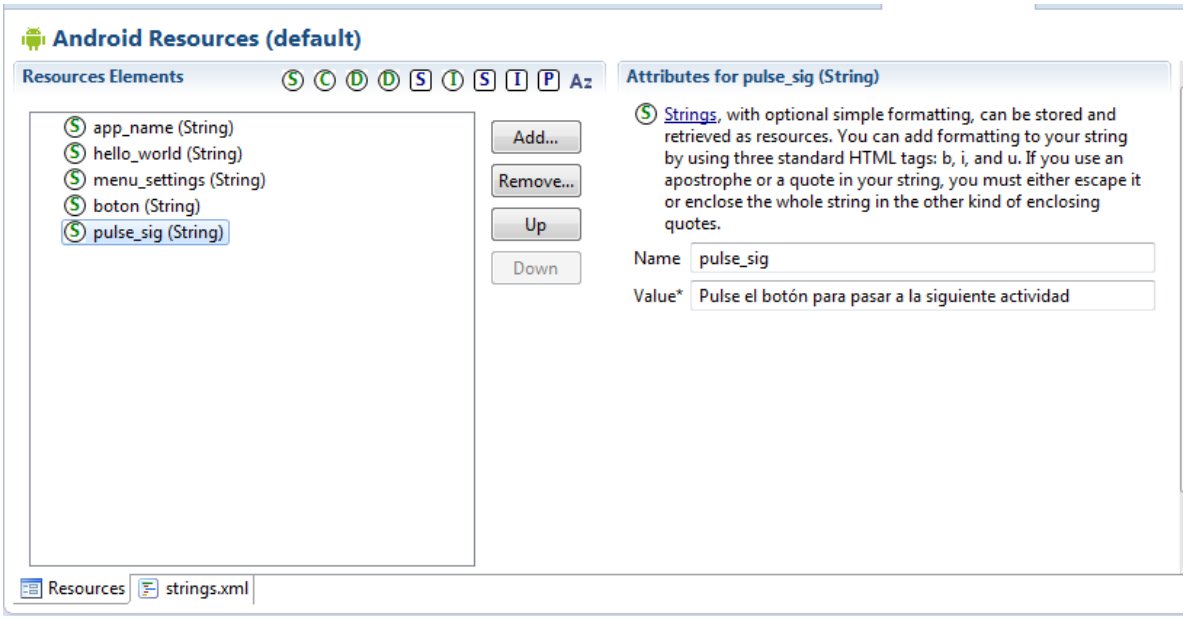


Figura 3.16. Interfaz de Android para añadir Strings. Para añadir un String, se pone el nombre con el que se va a designar a este en el apartado *"Name"* y se introduce la cadena que se va a mostrar en el apartado *"Value"*. Luego se pulsa en *"Add"*.

De igual forma que se añaden Strings, es posible añadir colores en el archivo color.xml, un estilo en el archivo *Styles.xml*, un menú en los archivos para cada actividad dentro del directorio menu en la carpeta de recursos de la aplicación.

Por último, en archivo XML de esta actividad, se ha puesto la siguiente línea en la parte del archivo dedicado al botón:

```
android:onClick="SiguienteActividad"
```

Esto permite crear un método en el archivo Java para esta actividad con el nombre *"SiguienteActividad"*. Este método permitirá recibir la acción del botón que dará la instrucción para pasar a la siguiente actividad.

Configuración del archivo Java de la primera actividad.

En el archivo Java, se creará el método comentado en el párrafo anterior, quedando el siguiente código:

```
package com.ejemplo;


import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
```

```
public class MainActivity extends Activity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);

	}

	public void SiguienteActividad(View view)
	{
		Intent intent = new Intent(this, Segunda.class);
		startActivity(intent);
		this.finish();
	}

	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
	// Inflate the menu; this adds items to the action bar if it is present.
		getMenuInflater().inflate(R.menu.activity_main, menu);
		return true;
	}
}
```

El último método, sólo es necesario si se requiere poner un menú en el Action Bar, lo cual en este momento no es necesario, por lo que se puede eliminar.

El método *"SiguienteActividad"* reacciona cuando el usuario pulsa sobre el botón. En este momento, se crea un *"Intent"* para pasar a la siguiente actividad. Un *Intent* es un elemento de comunicación entre distintos elementos de una aplicación. Los *intent* se usan para iniciar una actividad, otra aplicación, un servicio, etc.

La actividad siguiente se incicia con *"startActivity(intent);"* y la actividad actual finaliza con la línea de código *"this.finish()"*.

Configuración del archivo XML de la segunda actividad.

En la segunda actividad sólo se mostrará un mensaje: *"Hola mundo"*.

En este apartado solo se va a mostrar el contenido del archivo, pues lo fundamental se ha explicado en el subapartado anterior:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Segunda" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hola_mundo" />

</RelativeLayout>
```

Configuración del archivo Java de la segunda actividad.

Esta actividad puesto que sólo muestra el comentario de *"Hola mundo"*, no es algo difícil de configurar y dado que no se va a utilizar ningún menú, el archivo Java de esta segunda actividad será así de sencillo:

```
package com.ejemplo;

import android.os.Bundle;
import android.app.Activity;

public class Segunda extends Activity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_segunda);
	}

}
```

3.2.1.3. Simulación de una aplicación.

Una vez que el proyecto ha sido creado, se puede utilizar dos formas para comprobar si la aplicación funciona de la forma que queremos: mediante el uso de un dispositivo móvil con el sistema operativo de Android o mediante la simulación en un emulador de Android (Dispositivo Virtual Android o DVA).

Para la realización de este proyecto se ha utilizado emuladores la mayor parte del tiempo. Se pueden crear tantos emuladores como consideremos necesarios.

También se ha usado un teléfono LG Optimus L5 para simular la aplicación. Sin embargo, para simular la aplicación en un dispositivo, hay que descargarse una serie de drivers. Para simularlo en el dispositivo a veces da problemas pues los drivers no siempre funcionan bien, por lo que para simularlo se puede usar otra técnica, que consiste en copiar el .apk que se genera durante la ejecución del programa en el emulador AVD en el directorio de .apk del dispositivo y se puede comprobar el funcionamiento de la aplicación. Por supuesto, la aplicación no estará firmada como si se hubiese descargado del Google Play, pero utilizará la debug key, que es la clave que se genera durante la ejecución en el emulador.

El emulador al no ser un dispositivo real, tiene algunas limitaciones y diferencias en relación al uso de un móvil real:

- Simula el funcionamiento de un dispositivo pero sin poder tener una referencia real del hardware del dispositivo.

- Los datos obtenidos a partir de sensores en el dispositivo, como la ubicación geográfica, la energía disponible o la conectividad de red se simulan en relación al estado del ordenador del desarrollador.
- Algunos periféricos, como la cámara digital, no son del todo funcionales o no lo son en absoluto.
- Las llamadas telefónicas y la recepción de llamadas también se tienen que simular.
- El puerto USB y el Bluetooth no están accesibles. [40]

Para crear un emulador, se pulsa sobre window->Android Device Manager o sobre el icono  . De esta forma, se muestra la siguiente pantalla:

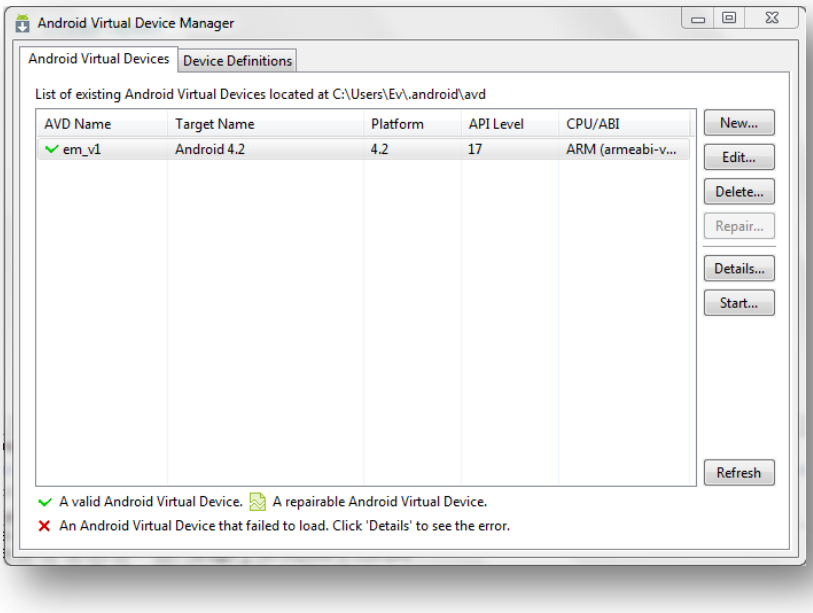


Figura 3.17. Gestor de ADV. Esta ventana muestra todos los emuladores que el desarrollador esté usando. Cuando quiere crear uno nuevo pulsa sobre *"New"*. Cuando quiere modificar una característica del emulador da sobre *"Edit"*. Para borrar un emulador pulsa sobre *"Delete"*. Y para poner un emulador en marcha, se pulsa sobre *"Start"*.

Esta pantalla es el gestor de ADV. Este gestor muestra una lista con todos los ADV (emuladores) que se han creado. Puede ser interesante usar varios emuladores para distintas versiones de la misma aplicación que se está desarrollando y comprobar los diferentes comportamientos.

Se pulsa en New para generar un nuevo emulador. Se muestra la siguiente pantalla donde cada desarrollador puede elegir las características que se le van a dar al emulador:

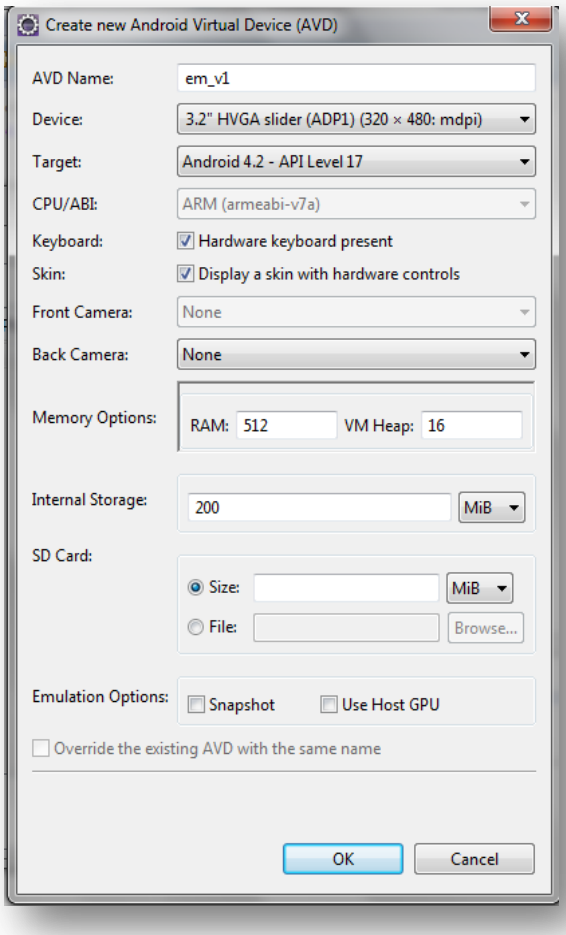


Figura 3.18. Configuración de un emulador. Aquí se debe indicar el nombre que se le da al emulador, el tamaño y tipo de dispositivo, el nivel de API y el tamaño de memoria si se va a hacer uso de ella, etc.

Se le da un nombre al emulador y se elige el tipo de dispositivo según la resolución y las pulgadas de la pantalla. También se indica el nivel de API con el que se quiere simular.

Se pueden elegir una serie de cualidades para la simulación, como por ejemplo, si queremos simular el dispositivos con controles de hardware o no. Se puede elegir el tamaño de memoria. Además, si para la simulación necesitamos introducir un archivo, hay que añadir memoria adicional en la opción SD Card.[13]

Una vez que ya se han elegido todas las características que deseamos que tenga la aplicación que se ha desarrollado, se acepta y se muestra la siguiente pantalla:

[13] La adición de un archivo para usarlo en una simulación se hace a través del DDMS.

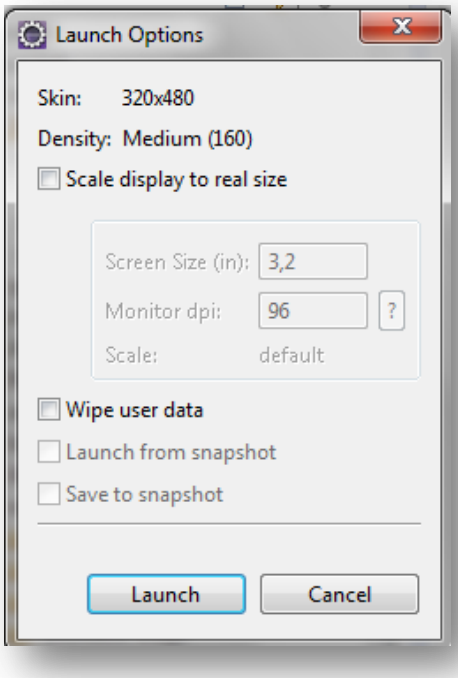


Figura 3.19. Configuración de un emulador. Muestra características que el desarrollador debe elegir

Por fin, se obtiene el emulador antes de que se cargue la aplicación diseñada:

Figura 3.20. Imagen de un emulador. El emulador presenta a la derecha una serie de controles que pueden ser usados por el desarrollador y representan los típicos controles de cualquier móvil.

3.2.1.4. Depurar una aplicación.

Para depurar una aplicación se selecciona *Run/Debug Configurations*. Obtenemos la siguiente pantalla, en la cual se debe pulsar sobre Android Application:

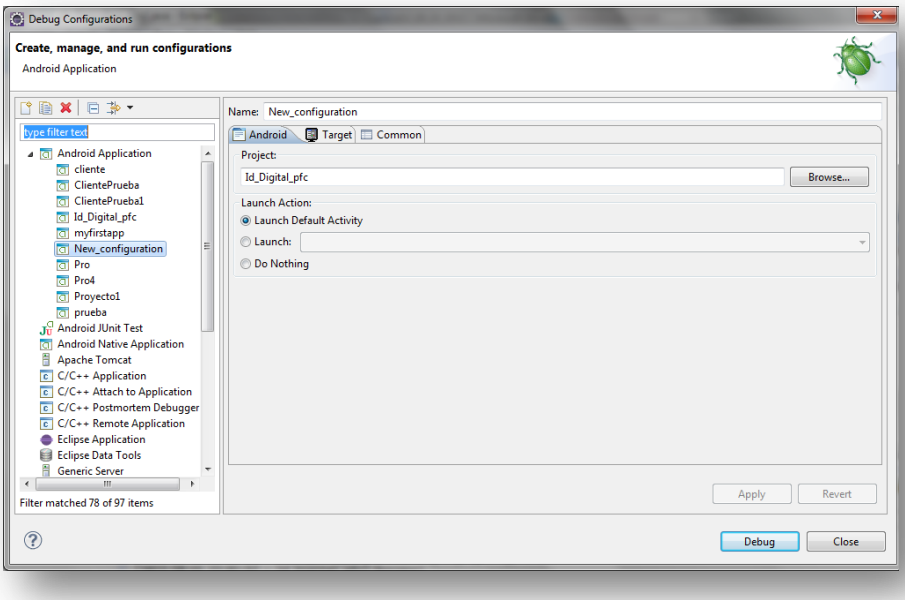


Figura 3.21. Configuración para la depuración de la aplicación. Esta ventana muestra las opciones que el desarrollador puede elegir para depurar o ejecutar una aplicación.

En la parte derecha del panel, se puede configurar las características del depurador. Para elegir el emulador que se va a utilizar para depurar la aplicación se pulsa sobre Target:

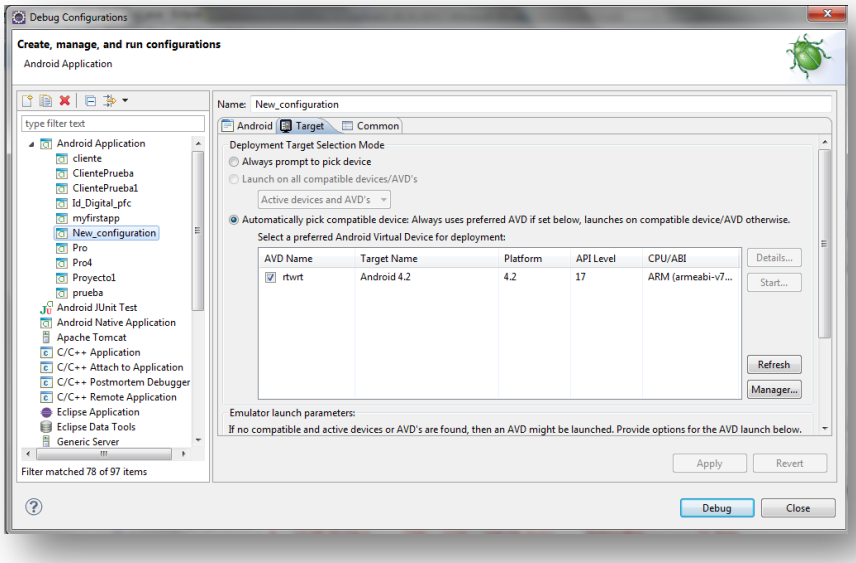


Figura 3.22. Elección del emulador que se va a utilizar para la depuración. Esta ventana da al desarrollador la posibilidad de elegir un emulador cada vez que se ejecuta la aplicación, ejecutar la aplicación en todos los emuladores existentes o que siempre se ejecute en el mismo emulador.

Después de este proceso se usan *breakpoints*, se pone la perspectiva de debug en la parte superior derecha, se pulsa sobre el icono de *Debug*  y se comprueba paso a paso con los iconos que se muestran en la barra de herramientas.

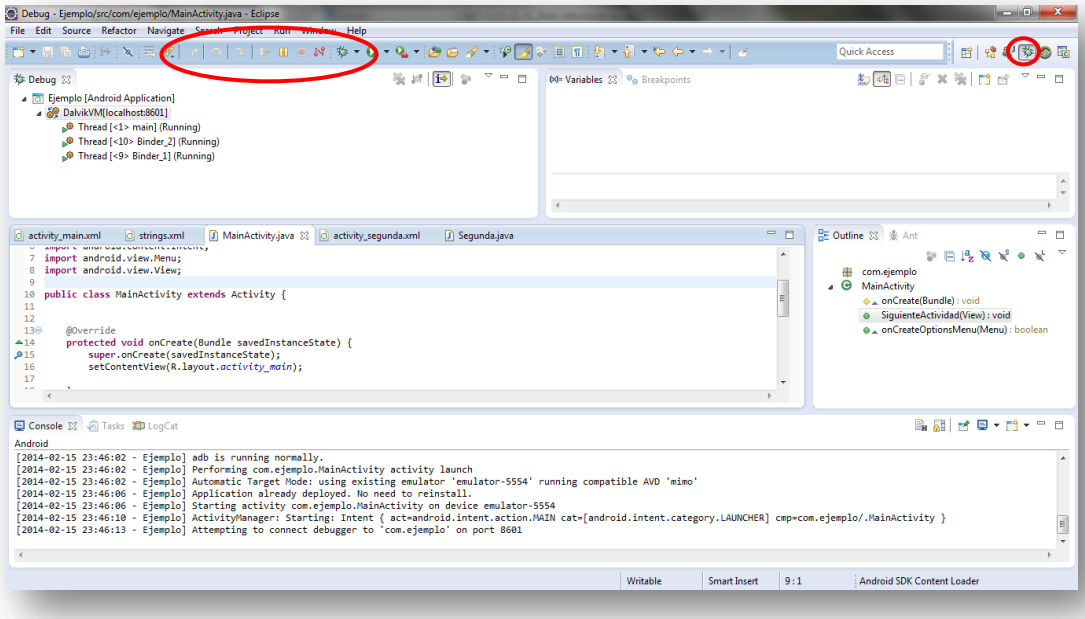


Figura 3.23. Perspectiva del depurador. En la perspectiva se muestran las líneas en las que se añade un breakpoint, el valor de las variables de la aplicación, etc.

3.2.1.5. Ejecución de una aplicación.

Una vez que todos los pasos anteriores se han llevado a cabo correctamente, ejecutar la aplicación es muy sencillo. La forma más fácil es pulsando sobre el icono de run 

Todos los pasos, excepto los que se comentan en el último parrafo del apartado anterior, sirven también para la ejecución de la aplicación. De hecho, tanto la depuración, como la ejecución del programa se pueden hacer desde el mismo emulador.

Por tanto, al ejecutar la aplicación, usando la aplicación de ejemplo creada en el apartado *3.2.1.2* se obtiene como resultado la siguiente aplicación:

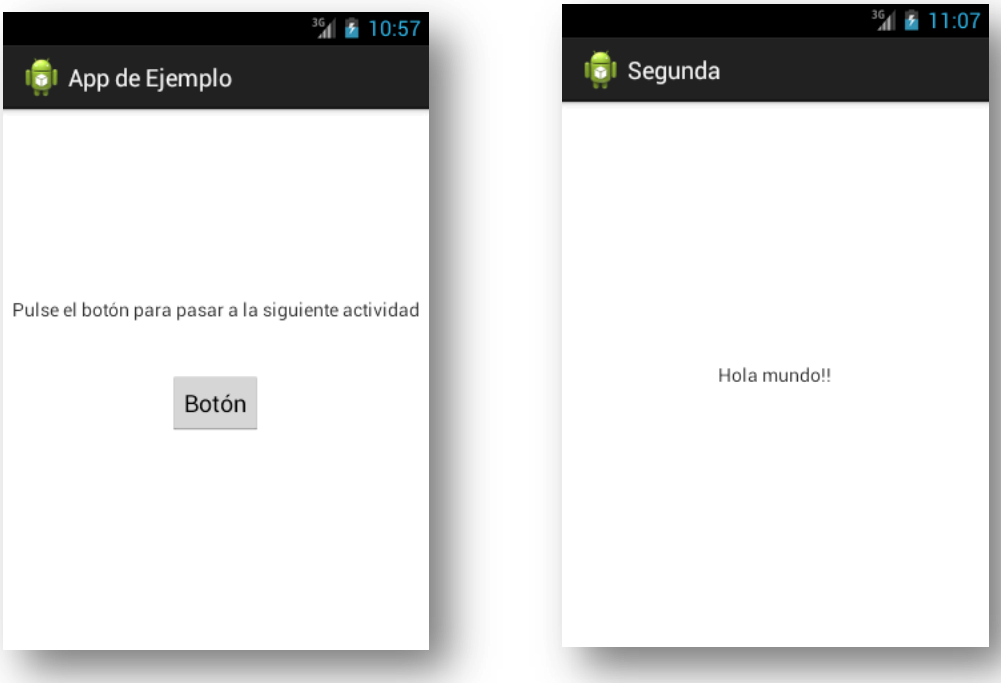


Figura 3.24. Aplicación de ejemplo después de haber sido ejecutada. La aplicación de ejemplo consta de dos actividades: la primera muestra un String y un botón y la segunda, sólo un String con el mensaje *"Hola mundo!"*

3.2.1.6. Estructura de AndroidManifest.xml

AndroidManifest.xml es el archivo de configuración de una aplicación y en él se definen las características de una aplicación. La estructura de AndroidManifest.xml es la siguiente:

Definición del .xml de AndroidManifest.xml
- Paquete de la aplicación
- Versión de la aplicación
- Nombre de la versión

    Definición de permisos, configuración y sdk

    Definición de la aplicación
    - Icono
    - Etiqueta

        Actividad 1
        - Nombre
        - Etiqueta
        - Tema
        - Filtros:

            Actividad principal

            Actividad asignada como lanzador

        Resto de Actividades
        - Nombre

- Etiqueta
- Tema

Los permisos, definición de sdk y otras configuraciones se pueden definir después de la definición de la aplicación.

El AndroidManifest.xml de la aplicación para la obtención de identidades digitales desarrollada en este proyecto, se encuentra en el *anexo D*.

A pesar de todo lo que se ha explicado en el apartado *3.2* y todos sus subapartados, merece la pena detenerse a leer todo el conjunto del anexo D en el que se encuentran todos los detalles técnicos destacables de la aplicación desarrollada para este proyecto.

3.2.1.7. Firmar una aplicación

Firmar una aplicación de Android es bastante sencillo si se usan las propias herramientas de las que dispone Android para ello.

En este apartado se explicará cómo hacerlo por primera vez, ya que hecho por primera vez es muy sencillo hacerlo las siguientes veces.

Se detallan a continuación cada uno de los pasos para la firma de aplicaciones (*.apk*):

- Acceder a Android Manifest y pulsar sobre la pestaña *"manifest"*. Una vez dentro pulsar sobre *"Exporting"* y sobre *"Use export wizard"*:

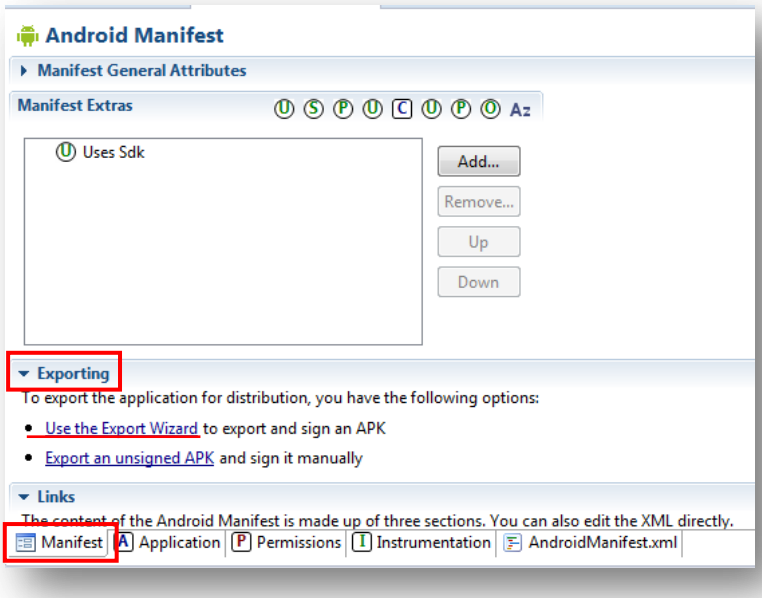


Figura 3.25. Imagen del AndroidManifest.xml. El desarrollador debe pulsar sobre Manifest y después sobre Exporting.

- En la siguiente pantalla se mostrará por defecto el proyecto con el que se está trabajando. Se pulsa sobre *"next"*:

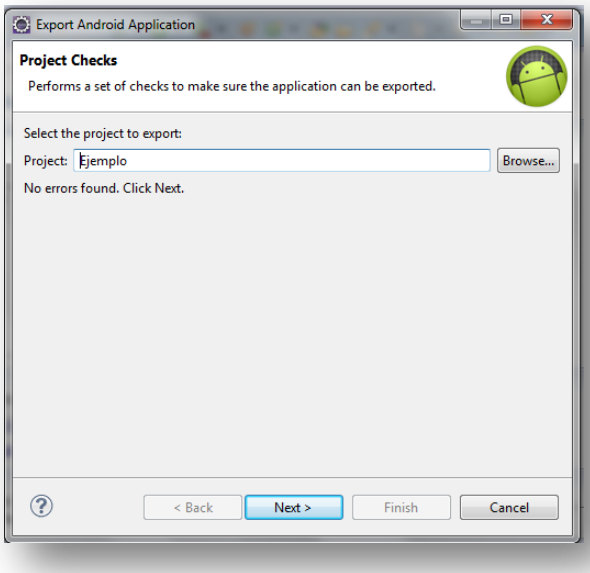


Figura 3.26. Segunda pantalla de la herramienta para la firma del apk. Se debe elegir el proyecto que se quiere firmar.

- En la siguiente pantalla, si no es la primera vez que firmamos una aplicación pulsamos sobre *"Use existing Keystore"*, ahí elegiremos un certificado de los que hayamos guardado en el keystore para firmar la aplicación (podríamos añadir nuestro propio certificado en el keystore usando la herramienta keytool de la que se hablará posteriormente). Si es la primera vez que firmamos un certificado, pulsamos sobre *"Create new keystore"* y señalamos la ubicación donde vamos a almacenar la keystore, le ponemos una contraseña a la misma:

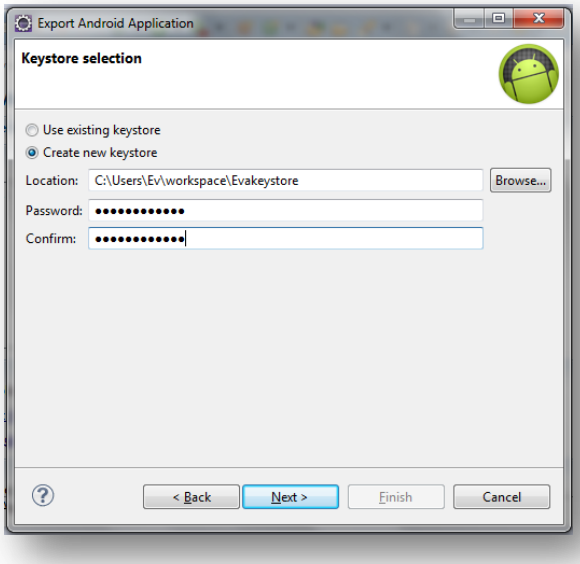


Figura 3.27. Creación de un keystore en Eclipse. Se elige el lugar en el que se quiere almacenar el keystore y se indica la contraseña que se va a usar para ese keystore.

- En la siguiente pantalla, el desarrollador introduce una serie de datos que le identifican y el *alias* con el que va a almacenar el certificado autofirmado con esl que se firmará la aplicación.

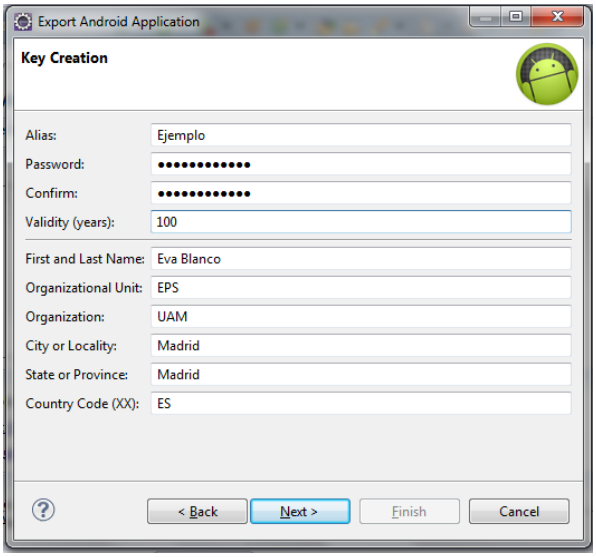


Figura 3.28. Creación de un certificado autofirmado en Eclipse. Se indican los datos que identifican al desarrollador y que se incluirán en el certificado autofirmado.

Para consultar el código que corresponde a cada país se puede usar el estándar ISO 3166-1: http://es.wikipedia.org/wiki/ISO_3166-1

- Por último se selecciona la carpeta en la que se almacenará el apk de la aplicación firmada:

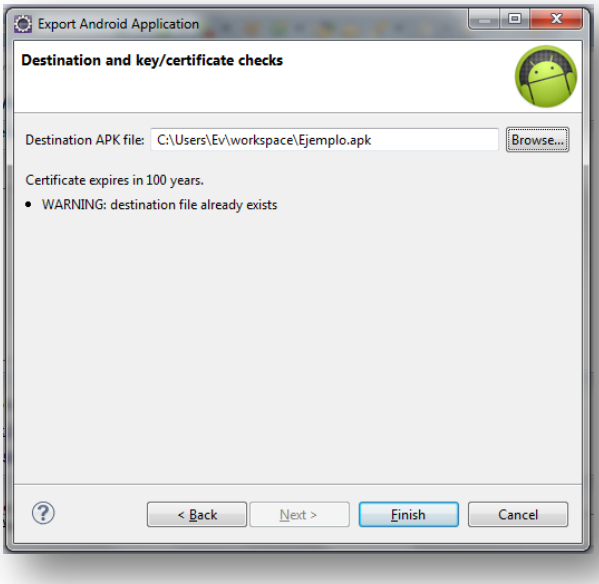


Figura 3.29. Último paso para la firma de una aplicación. En esta ventana se indica la ruta del .apk de la aplicación que se desea firmar.

Para subir la aplicación a Google Play hay que acceder a la pantalla principal de la consola de Google Play para desarrolladores: https://support.google.com/googleplay/android-developer/answer/113469?hl=es

Para subir aplicaciones a Google Play hay que pagar una cuota y rellenar una serie de datos bancarios.

Después se introduce el .apk de la aplicación, que como tamaño máximo puede tener 50 MB. Este se puede guardar como borrador hasta que se rellenen todos los datos que se solicitan. Además, se tienen que subir un par de capturas de pantalla.

También hay que añadir una serie de detalles:

- Idioma: indica el idioma que utilizará la aplicación.
- Nombre: indicará el nombre con el que se verá la aplicación en Google Play y se puede poner un nombre para cada país que se quiera utilizar.
- Descripción: es la descripción que aparecerá sobre la aplicación en Google Play.
- Cambios recientes: que permite añadir una descripción los cambios que se han hecho en las diferentes versiones que se suben.
- Texto promocional: texto algo más llamativo que se verá al lado de las aplicaciones en Google Play.
- Tipo de aplicación: existen de dos tipos: aplicaciones o juegos.
- Categoría: se indica una lista de categorías dependiendo de si se ha elegido aplicaciones o juegos.

Por último, se pueden indicar otras opciones de publicación como la ubicación, información de contacto, etc. [59]

3.3. LIBRERÍAS E INTERFACES DE PROGRAMACIÓN DE SEGURIDAD

Android no sólo tiene un conjunto de librerías propias, si no que además usa librerías de Java y de alguna otra distribución externa como Apache y BouncyCastle.

En este proyecto final de carrera se hace uso de la librería externa BouncyCastle y de ciertas APIs de programación que posee Java.

En los siguientes apartados se hace una descripción de estas APIs.

3.3.1. Librerías e interfaces de seguridad internas de Java/Android.

Arquitectura de Criptografía de Java (JCA) y Extensión de Criptografía de Java (JCE)

JCA y JCE son dos marcos de programación que permiten la implementación de diferentes soluciones criptográficas.

Para versiones anteriores a JDK[14] 1.4, JCE estaba desagregado de JCA, por lo que se mencionaban por separado, ahora se considera que JCE es parte de JCA.

Esta separación era debida a que las leyes de los EE.UU restringen la exportación de ciertos tipos de software criptográfico fuera de los EE.UU y Canadá, por lo que si JCE formase parte del software de Java su uso en otros lugares fuera de EE.UU estaría muy limitado. JCE, además, no tiene una política restrictiva del tamaño de claves.

La JCA es una parte importante de Java, que permite proporcionar una serie de servicios como una arquitectura de proveedor[15] (los servicios de seguridad pueden ser aportados por diferentes librerías e infraestructuras) y un juego de APIs para firmas digitales, funciones resumen, certificados y validación de certificados, cifrado, generación de claves, generación de números aleatorios seguros, etc. Esta arquitectura fue diseñada en torno a una serie de principios:

- *Independencia de implementación:* los desarrolladores no necesitan implementar algoritmos ni funcionalidades criptográficas en sus aplicaciones, si no que mediante el uso de una serie de funciones y a través de una serie de interfaces pertenecientes a la plataforma Java pueden implementar sistemas de seguridad haciendo uso de las mismas.

[14] JDK es el Java Development Kit o Herramientas de desarrollo de Java
[15] Un proveedor es un paquete o un conjunto de paquetes que proporcionan una serie de funciones criptográficas.

- *Interoperabilidad de la implementación:* una aplicación no tiene por qué estar unida a un solo proveedor, así como un proveedor no tiene por qué estar unido a una sola aplicación.

- *Extensibilidad de algoritmos:* Java provee el uso de una serie de algoritmos, pero debido al avance de los métodos criptográficos, es posible que existan algoritmos que se queden fuera y que sean proporcionados por otros proveedores. Java permite la incorporación de proveedores externos. [43]

Además, es necesario decir, que aunque la aplicación podría funcionar correctamente para tamaños de clave de 1024 o 2048 bits, para tamaños superiores esto no se puede asegurar. Para evitar que haya ningún problema al respecto, hay que añadir al directorio de librerías del proyecto dos librerías de JCE que evita estos problemas. Estas librerías se pueden descargar en la siguiente dirección:

http://www.oracle.com/technetwork/java/javase/downloads/jce-6-download-429243.html

Las clases e interfaces que forman parte de la JCA son:

- `java.security`: conjunto de clases e interfaces del nucleo de la estructura de proveedor de servicios y API de funcionamiento criptográfico de JCA.
- `java.security.cert`: conjunto de clases e interfaces de administración de certificados.
- `java.security.interfaces`: conjunto de interfaces que se usan para encapsular y administrar claves públicas y privadas DSA y RSA. (Consultar el anexo C)
- `java.security.spec`: conjunto de clases e interfaces que se usan para describir especificaciones de algoritmos y parámetros con clave privada.

Los paquetes aportados a la JCA por parte de la JCE son los siguientes:

- `javax.crypto`: proporciona 14 clases, una interfaz y cuatro excepciones que soportan algoritmos criptográficos básicos.
- `javax.crypto.interfaces`: proporciona tres interfaces que soportan claves Diffie-Hellman.
- `javax.crypto.spec`: proporciona 12 clases que definen especificaciones de clave y especificaciones de parámetros de algoritmos.

Los motores criptográficos[16] que proporciona JCA son los siguientes:

[16] Los motores criptográficos implementan una interfaz estándar de proveedor de servicios criptográficos.

- MessageDigest: Crea y verifica funciones resumen seguras.
- Signature: crea y verifica firmas digitales.
- KeyPairGenerator: genera pares de clave pública y privada.
- KeyFactory: convierte entre especificaciones de claves seguras.
- KeyStore: modifica la información en un almacén seguro de claves.
- CertificateFactory: genera certificados y listas de revocación de certificados.
- AlgorithmParameters: codifica parámetros de algoritmos criptográficos.
- AlgorithmParameterGenerator: crea parámetros de algoritmos criptográficos.
- SecureRandom: crea números aleatorios.

Los motores criptográficos aportados a JCA por parte de JCE son los siguientes:

- Cipher: proporciona cifrado y descifrado.
- KeyAgreement: proporciona un protocolo de intercambio de claves.
- KeyGenerator: es un generador de claves simétricas.
- Mac: algoritmo de autentificación de mensajes.
- SecretKeyFactory: Representa una factoría de claves secretas.[17]

Estas clases son clases abstractas cuyas funciones también pueden ser realizadas por librerías externas como las de Bouncy Castle que se explican posteriormente. En este proyecto, se han utilizado tanto clases de JCA y JCE como clases de Bouncy Castle.

El proveedor[18] de servicios criptográficos que por defecto usa Java es SUN. Este proveedor proporciona una combinación de motores y algoritmos criptográficos que permiten las siguientes operaciones:

- Algoritmos para las funciones resumen MD5 y SHA-1.
- DSA para firmas digitales.
- Generador de par de claves usando DSA.
- Generador de parámetros de algoritmos DSA.
- Fabrica de claves DSA. Las fabricas son técnicas de diseño que permiten crear objetos de tipo genérico que permite a las subclases elegir cuál es la mejor opción.

[17] Las factorías de claves se emplean para convertir claves en especificaciones de claves (representaciones transparentes de claves) y viceversa.
[18] Un proveedor proporciona la implementación de ciertos algoritmos. En este proyecto, se usa como proveedor BouncyCastle.

- Keystores JKS (como se ha comentado BKS para Android que sigue la implementación de BouncyCastle).
- Generador de números pseudoaleatorios SHA1PRNG. [42]

Extensión Segura de Socket de Java (JSSE o Java Secure Socket Extension)

JSSE permite establecer comunicaciones seguras a través de internet. Proporciona un marco y una implementación para la versión de Java de los protocolos SSL y TLS. Incluye funcionalidades para el cifrado de datos, autenticación de servidor, integridad de mensaje y una autenticación opcional de cliente.

El protocolo de handshake se realiza de forma transparente al programador, de forma que este no es consciente de si se está realizando o no.

JSSE proporciona el marco de una API y la implementación de esa API. La API de JSSE soporta las versiones 2 y 3 de SSL y la versión 1 de TLS.

JSSE está implementado completamente en Java. Además, proporciona soporte para muchos algoritmos criptográficos usados en el cipher suite del protocolo handshake. Alguno de estos algoritmos se lista a continuación:

- RSA: usado para la autenticación e intercambio de claves, permitiendo una longitud de clave de 512 bits o más.
- RC4: usado en cifrado con una longitud de clave de 128 bits.
- DES: usado para cifrado con longitud de clave de 64 bits.
- Triple DES: usado para cifrado con longitud de 192 bits.
- AES: usado para cifrado con longitud de 256 o 128 bits.
- Diffie-Helman: para el acuerdo de claves con longitud de clave de 1024 o 512 bits.
- DSA: usado para autenticación con longitud de claves de 1024 bits. [43]

Los paquetes que conforman la arquitectura JSSE son los siguientes:

- `Javax.net.ssl`: Este paquete contiene el conjunto de clases e interfaces que forman el núcleo de las API de JSSE.
- `Javax.net`: Este paquete no es específico en la JSSE, sin embargo es necesario para poder soportar la funcionalidad básica de factoría de socket cliente y servidor.
- `Javax.security.cert`: No es específico de la JSSE, pero es necesario para soportar una funcionalidad básica de administración de certificados.

Se muestra a continuación una lista con las clases e interfaces de JSSE:

- SSLSocket: establece un socket[19] que soporta los protocolos de socket seguro SSL, TLS y WTLS.
- SocketFactory: es una factoría para los objetos Socket.
- SSLSocketFactory: es una factoría para los objetos SSLSocket.
- SSLServerSocket: Un socket de servidor que soporta los protocolos de socket seguro SSL, TLS y WTLS.
- ServerSocketFactory: Una factoría para los objetos ServerSocket.
- SSLServerSocketFactory: Una factoría para los objetos SSLServerSocket.
- SSLSession: interfaz de un objeto que encapsula una sesión SSL.
- SSLSessionContext: interfaz de un objeto que encapsula un conjunto de sesiones SSL identificadas con un ID de sesión.
- SSLBindingEvent: clase de evento que encapsula eventos de enlace y desenlace de sesión SSL.
- SSLBindingListener: interfaz de auditor que está implementada por objetos que desean conocer los eventos de enlace y desenlace de sesión SSL.
- HandshakeCompletedEvent: una clase de evento que encapsula el hecho de que ha terminado una despedida SSL.
- HandshakeCompletedListener: interfaz de auditor que está implementada por objetos que desean conocer los eventos de terminación de despedidas SSL. [42]

3.3.2. Librerías externas de seguridad.

Bouncy Castle

Son un conjunto de APIs y paquetes criptográficos que implementan una serie de algoritmos criptográficos. Tienen versiones en Java y en C#.

Los primeros creadores de estos paquetes son australianos, por lo que las restricciones de seguridad en la exportación de software criptográfico impuestas por los EE.UU no les afectaban.

Bouncy Castle nació como una necesidad de sus creadores de tener un conjunto de librerías criptográficas que no tuviesen que modificarse cada vez que estas dos personas cambiaban de puesto de trabajo.[47]

[19] Un socket es un método o conexión que permite el intercambio de un flujo de datos entre dos programas que pueden estar situados en dos ordenadores diferentes.

En su creación, los requisitos que plantearon los creadores fueron dos principalmente. El primero consiste en que sean APIs de bajo peso que permitan ser utilizadas en aplicaciones que tengan poca capacidad. El segundo requisito es que se usase JCE como proveedor. Además, estas APIs haciendo uso del proveedor de JCE implementan funcionalidades más allá de las implementadas por JCE, como por ejemplo, soporte para PGP.

El hecho de usar JCE como proveedor y el bajo peso de las APIs, permite que se pueda acceder a funciones criptográficas cuando no es posible acceder fácilmente a las funcionalidades de JCE. [43]

En resumen las APIs de Bouncy Castle poseen las siguientes características:

- Una API ligera de criptografía.
- Un proveedor para JCE y JCA.
- Una librería para la lectura y escritura de objetos ASN.1.
- Unas APIs ligeras para TLS y DTLS.
- Generador de certificados X.509 para la versión 1 y 3, para la versión 2 de CRLs y archivos PKCS12.
- Generadores de certificados de atributos de X.509 para la versión 2.
- Generadores y procesadores para los siguientes protocolos: S/MIME y CMS, OCSP, TSP, CMP y CRMF, OpenPGP, Control de Acceso Extendido (EAC) y para validación de datos y servidor de certificados.
- Versiones .jar compatibles con JDK 1.4-1.7 y el proveedor Sun JCE. [44]

Las APIs y los paquetes de Bouncy Castle se han hecho tan populares en la red, que incluso Android las utiliza en su sistema operativo. Este hecho ha dado como resultado la creación de nuevos paquetes criptográficos para Android conocidos como Spongy Castle.

Spongy Castle

Como se ha comentado, en versiones anteriores a la 4.2 del sistema operativo de Android incluye versiones personalizadas de Bouncy Castle. Debido a conflictos con el nombre de las clases, esto hace que a veces, dependiendo de las clases que se vayan a utilizar para una aplicación, está no funcione correctamente. Para solucionar este problema se creó una nueva distribución renombrada y llamada Spongy Castle. [43]

La diferencia básica radica en el nombre de las librerías y en el nombre del proveedor que pasa de *"BC"* con Bouncy Castle a *"SC"* con Spongy Castle.

En este proyecto se ha usado Bouncy Castle porque la implementación de ciertas funciones de la aplicación fue más sencilla, la información sobre como implementarlas con Bouncy Castle es más extensa y dio menos problemas usando Bouncy Castle, además se comprobó que no había ningún método que interfiriese con Android por lo que se decidió seguir usando Bouncy Castle.

Las librerías de Spongy Castle se encuentran en la dirección https://github.com/rtyley/spongycastle

3.4. HERRAMIENTAS CRIPTOGRÁFICAS

Existen dos conocidas herramientas criptográficas mediante las cuales es posible realizar algunos de las opraciones criptográficas más importantes. Estas dos herramientas son: Keytool y OpenSSL. Además, existe una herramienta llamada Portecle que es una Interfaz gráfica de usuario para crear, gestionar o ver el contenido e keystores, claves, certificados, CSR, CRL, etc.

Estás herramientas son necesarias a la hora de realizar importantes operaciones relacionadas con la creación de certificados, firma digital de certificados, creación de almacenes de claves, etc., y han sido una parte fundamental a la hora de llevar a cabo la realización de este proyecto.

Keytool es una herramienta que se instala en el momento en el que se instalan las disitintas herramientas de Java [46]. OpenSSL es una herramienta criptográfica de distribución libre que se puede descargar en las páginas oficiales de OpenSSL [52]. Portecle se distribuye como un ejecutable de Java y permite realizar ciertas operaciones que también se pueden hacer con las dos herramientas antes mencionadas pero de una forma gráfica y más cómoda para la persona que lo utilice [53].

En los próximos subapartados se explicará algúnas de las características más importantes de estas herramientas.

Los motivos por lo que es necesario usar estas herramientas, es porque, por lo general, un sistema requiere muchos recursos que no pueden ser creados por una sola persona. Uno de los objetivos de este proyecto es la creación de un protocolo de registro con cierto nivel de seguridad. Implementar esto de una forma realista, requiere crear un servidor web real, utilizar una CA real, etc. pero esto implica una gran cantidad de tiempo y una complejidad demasiado elevada, por lo que se usan estas herramientas para simular el comportamiento real que debería tener la aplicación y el sistema en general, pues han facilitado el trabajo que se ha realizado.

3.4.1. Keytool

Keytool es una herramienta basada en el uso de comandos para la gestión de claves y certificados. Permite a los usuarios administrar sus propios par de claves públicas/privadas y certificados asociados para su uso en autenticación, tanto de si mismo y de otros usuarios, mediante firmas digitales.

Keytool también permite a los usuarios administrar claves secretas usadas en cifrado y descifrado simétricos. Además, permite almacenar las claves y los certificados en Keystores.

Keytool ofrece las siguientes opciones:

- Generación de par de claves.
- Generación de certificados
- Generación de peticiones de firma de certificados.
- Administración de entradas de clave.
- Administración de entradas de certificado fiables.
- Administración de contraseñas.
- Importación de bases de datos.
- Ayuda. [43]

A continuación se listan una serie de comandos utilizados por Keytool:

- *Comandos para crear y añadir datos a la Keystore*

  `-genkeypair:` genera un par de claves (una pública y otra privada). La clave pública tiene formato de certificado autofirmado X.509 v3.

  `-genseckey:` genera una clave secreta y la almacena en una nueva entrada del Keystore identificado por su alias.

  `-genseckey:` lee el certificado o cadena de certificados y lo almacena en una entrada de la Keystore idenficado por su alias.

  `-importkeystore:` importa una única entra o todas las entradas de una keystore de origen a una keystore de destino.

- *Comandos para exportar datos*

  `-certreq:` genera una solicitud de firma de certificado (Certificate Signing Request o CSR) usando un formato PKCS#10 (De los formatos de los certificados se hablará más adelante).

  `-exportcert:` lee de una keystore el certificado asociado a un alias y lo almacena en un archivo. Si no se especifica un archivo la salida se muestra por stdout.

- *Comandos para mostrar datos*

  `-list:` imprime a la salida (stdout) el contenido de la entrada de la keystore identificada por el alias. Si no se especifica el alias, se muestran todas las entradas de la Keystore.

  `-printcert:` lee el certificado de un archivo y imprime su contenido en un formato legible por una persona. Si no se especifica ningún archivo, el certificado se lee del flujo de entrada (stdin).

- *Comandos para la gestión de la keystore*

  `-storepasswd:` cambia la contraseña usada para proteger la integridad de los contenidos de la keystore.

  `-keypasswd:` cambia la contraseñabajo la cual la clave privada/secre identificada por un alias es protegida.

  `-delete:` borra la entrada de la keystore identificada por su alias. Si no se introduce un alias en el comando, se pide por pantalla.

  `-changealias:` mueve una entrada existente de la keystore especificada por un alias a un nuevo alias.

- *Comandos para solicitar ayuda*

  `-help:` lista los comandos básicos y sus opciones.

Todos estos comandos se complentan con varias opciones que determinan diferentes características dependiendo del comando. Algunas de las opciones más usuales son:

`-storetype` *`storetype`*

Especifica el tipo de keystore que se va a instanciar. Se sustituye la palabra en cursiva por el nombre del tipo de keystore que se va a usar. Por ejemplo, JKS para Java y BKS para Android.

`-keystore` *`keystore`*

Especifica la localización de la keystore.

`-storepass` *`storepass`*

Especifica la contraseña que se usa para proteger la integridad de la Keystore. Debe tener como mínimo 6 caracteres.

`-providerName` *`provider_name`*

Usado para identificar el nombre de un proveedor de servicios criptográficos cuando se lista en un archivo de propiedades de seguridad.

`-providerClass` *`provider_class_name`*

Usado para especificar el nombre de una clase maestra de un proveedor de servicios criptográficos cuando el proveedor de servicios no se lista en el archivo de propiedades de seguridad.

`-providerArg` *`provider_arg`*

Usado junto con –providerClass. Representa una cadena de enentrada opcional como argumento al constructor de provider_class_name.

`-protected`

Puede ser true o false. Este valor se debe especificar como true si una contraseña se da mediante un campo de autenticación protegido, como por ejemplo, un lector dedicado PIN.

Además, existen otras opciones que aún son más usadas que las anteriores, por lo que incluso vienen con valores por defecto aunque también pueden ser configurables por el programador. Estas opciones junto con sus valores por defecto son:

`-alias “mykey”`

`-keyalg`

“DSA” (cuando se usa -genkeypair)
“DES” (cuando se usa -genseckey)

`-keysize`

`1024` (cuando se usa -genkeypair)
`56` (cuando se usa –genseckey y –keyalg es “DES”)
`168` (cuando se usa –genseckey y –keyalg es “DESede”)

`-validity 90`

`-keystore` el archivo con extensión .keystore que se encuentre en el directorio que esté usando el usuario

`-storetype` el tipo por defecto definido en el archivo de configuración.

`-file` flujo de entrada (stdin) en lectura y flujo de salida en escritura (stdout).

`-protected false.` [45]

3.4.2. OpenSSL

OpenSSL es un proyecto de Software Libre desarrollado por Eric Young y Tim Hudson. Es un conjunto de herramientas de administración y bibliotecas relacionadas con la criptografía. Colabora en la implementación de protocolos de seguridad en red como SSL y TLS y además se puede utilizar para la creación de certificados digitales para su uso en servidores. [67]

OpenSSL es una herramienta basada en el uso de lineas de comando y se puede usar para las siguientes operaciones:

- Creación y gestión de claves privadas, claves públicas y distintos parámetros.
- Operaciones de criptografía de clave pública.
- Creación de certificados X.509, CSRs y CRLs.
- Cálculo de funciones resumen.
- Cifrado y descifrado.
- Comprobación de cliente y servidor SSL/TLS.
- Manejo de emails cifrados.
- Peticiones de sello de tiempo, generación y verificación.[46]

Existe una cantidad muy grande de comandos, por lo que a continuación se van a nombrar los más importantes, al menos en la realización de este proyecto:

- Generar una clave privada y su correspondiente CSR:

```
>> openssl req –out nombreCSR.csr –pubkey –new –keyout nombreClave.key
```

- Descifrar una clave privada:

```
>> openssl rsa –in nombreClave.key >> clave_noCifrada.key
```

- Generar un CSR para una clave privada existente:

```
>> openssl req –out nombreCSR.csr –key nombreClave.key –new
```

- Generar un CSR basado en un certificado X.509 existente:

```
>> openssl x509 –x509toreq –in miCertificado.crt –out nombreCSR.csr –
signkey nombreClave.key
```

- Crear un certificado autofirmado (se puede usar para firmar otros certificados):

`>> openssl req -x509 -new -out miCertificado.crt -keyout nombreClave.key -days 365` (puede ser más días o menos días)

- Firmar un CSR:

`>> openssl x509 -req -in nombreCSR.csr -CA certDeCA.crt -Cakey claveDeCA.key -Cacreateserial -out miCertificado.crt -days` (este valor se elige)

3.4.3. Portecle

Como se ha comentado, Portecle es una interfaz grafica de usuario que permite realizar las siguientes operaciones:

- Crear, cargar, guardar y convertir Keystores (se hablará de los keystores en el siguiente apartado).
- Generar entradas de par de claves DSA y RSA con certificados X.509 v1 autofirmados.
- Importar certificados X.509 como certificados de confianza.
- Importar par de claves en archivos con el estándar PKCS#12 y PEM.
- Clonar y cambiar la contraseña de las entradas del par de claves y el keystore.
- Ver los detalles de los certificados contenidos en las entradas de los keystore, certificados y conexiones SSL/TLS.
- Exportar entradas del Keystore en una gran variedad de formatos.
- Generar y ver CSRs.
- Importar replicas de CAs.
- Eliminar, clonar y renombrar entradas del keystore.
- Ver los detalles de los CRLs.

3.5. ALMACENES DE CLAVES: KEYSTORES Y TRUSTSTORES

Los almacenes de claves son repositorios de certificados en donde se almacenan claves privadas o públicas. Normalmente se usan dos tipos de almacenes de claves: keystore y truststore.

La diferencia principal entre keystore y truststore radica en que generalmente las truststore se usan para almacenar certificados en los que confía un servidor, mientras que las keystore se usan para almacenar claves privadas, aunque también pueden almacenar claves públicas.

Es posible crear almacenas de claves con la herramienta criptográfica keytool. Java, también permite mediante código el uso de diferentes métodos crear y hacer uso de los almacenes de claves.

Los almacenes de claves permiten establecer conexiones seguras entre un cliente y un servidor. Por ejemplo, si una aplicación cliente quiere tener acceso a ciertos datos almacenados en un servidor o acceder a un servicio web, la aplicación cliente almacenará en su truststore el certificado del servidor, de forma que cuando la aplicación cliente quiere acceder a este servicio comprueba que es un servicio de confianza pues tiene almacenado su certificado en el truststore. De igual forma, es posible la autenticación del cliente. Además, tanto el cliente como el servidor podrán confirmar su identidad al demostrar que los certificados almacenados en sus truststore les corresponden. Esta comprobación se realiza durante el proceso de negociación al iniciarse una conexión.

En resumen, un truststore permite almacenar los certificados de ciertos servicios web en los que una aplicación cliente confía, mientras que la keystore permite autenticar la identidad de la aplicación cliente.

En el entorno de Android, las keystores se utilizan para autenticar al desarrollador que firma la aplicación antes de ser lanzada en el Play Store de Android.

Capítulo 4:

Diseño y desarrollo de la aplicación

4. DISEÑO Y DESARROLLO DE LA APLICACIÓN

4.1. INTRODUCCIÓN

A lo largo de este capítulo, se llevará a cabo una descripción del proceso que se ha seguido para el desarrollo de la aplicación objeto de este proyecto. Se pretende hacer una descripción a grandes rasgos del desarrollo de la aplicación y del sistema, para posteriormente profundizar en los detalles que componen cada uno de los mismos.

La mayoría de los sistemas actualmente requieren por parte de un usuario, un código de usuario y una contraseña para autenticarse en dicho sistema. Esto puede implicar problemas de seguridad si las contraseñas son débiles, si se produce suplantación de un usuario, si se produce una revelación de las contraseñas, etc. El uso de certificados para el acceso permite una gestión más sencilla al no tener que tener en cuenta este tipo de problemas. Por lo tanto, si un usuario obtiene una identidad digital, permitirá que pueda acceder también a ciertos servicios, los cuales no serían posibles sin el uso de un certificado digital. Por ejemplo, existen muchos servicios institucionales que requieren el uso de un certificado digital para autenticarse. Otros servicios muy usados en dispositivos móviles y que hacen uso de certificados digitales, son los servicios de correo electrónico, que los usan para la firma digital de los correos que un usuario envía o para cifrar los mismos.

Como se ha comentado en capítulos anteriores, no existen muchas aplicaciones que permitan la obtención de una identidad digital y si existen, requieren el desplazamiento del usuario para poder obtener definitivamente la identidad digital que el usuario había solicitado previamente. Estas aplicaciones obtienen un código que necesitan presentar en la oficina de registro donde se comprueba que realmente la persona es quién dice ser y posteriormente puede obtener su identidad digital a través de la propia aplicación.

Muchos servicios por internet como entidades bancarias, tiendas online, etc. utilizan protocolos seguros como el HTTPS que permite el envío de información cifrada, de forma que los datos privados del usuario no sean descubiertos. En HTTPS el cifrado está basado en los protocolos SSL/TLS, los cuales requieren el uso de certificados digitales, pero si además, estos certificados son los que identifican al usuario, esto implica que la conexión se está realizando directamente con el usuario que posee dicha identidad digital y sólo requiere que el servidor de dicho servicio compruebe los credenciales del usuario.

El establecimiento de la seguridad en el desarrollo de este proyecto final de carrera está basado en un protocolo desarrollado en el Grupo de Neurocomputación Biológica en la Escuela Politécnica Superior, que se establece mediante varias conexiones seguras entre diferentes servidores y con el usuario. El protocolo requiere que exista confianza entre los servidores para poder enviar información entre sí [1][2][3]. Este protocolo se definirá adelante en este capítulo, pues constituye la parte central de la aplicación.

4.2. ESTRUCTURA DEL SISTEMA

En la siguiente imagen se muestra la forma en la que está estructurado este proyecto, en la que no sólo interviene la propia aplicación en sí, sino una serie de elementos que conforman la PKI de este sistema:

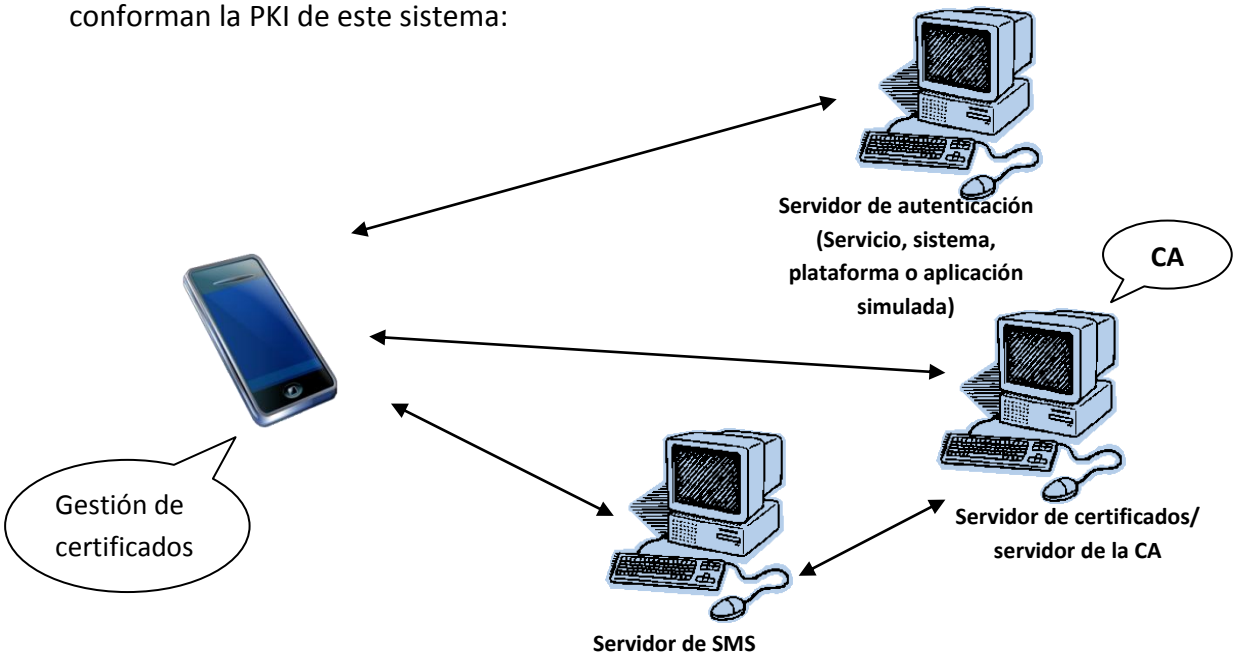


Figura 4.1. Estructura del sistema. Se han desarrollado tres servidores. En la imagen cada servidor se encuentra representado por un ordenador, sin embargo, la implementación real se realiza en un único ordenador. La implementación de la aplicación se hizo en otro ordenador y se instaló en un móvil.

Como se ve en la imagen, se han implementado tres servidores. En la imagen se han representado como si realmente cada servidor estuviese en un equipo diferente, lo cual es posible realizar si se dispusiese de ese número de equipos, pero en este caso los tres servidores se han implementado en un mismo equipo y en el lenguaje de programación Java.

La conectividad de los programas servidores y con el cliente, se establece mediante el uso de sockets tanto seguros como no seguros (en el caso del servidor de SMS), por lo que solo es necesario saber la dirección IP y el puerto, que se indican dentro de los programas. En el caso de usar los servidores en distintos equipos, simplemente se necesitaría modificar unas líneas de código indicando las diferentes IPs y puertos.

Para poder utilizar distintos puertos, hay que abrirlos o habilitarlos tal y como se indica en el *anexo E*.

Físicamente la implementación de la aplicación y el sistema se ha desarrollado de la siguiente forma:

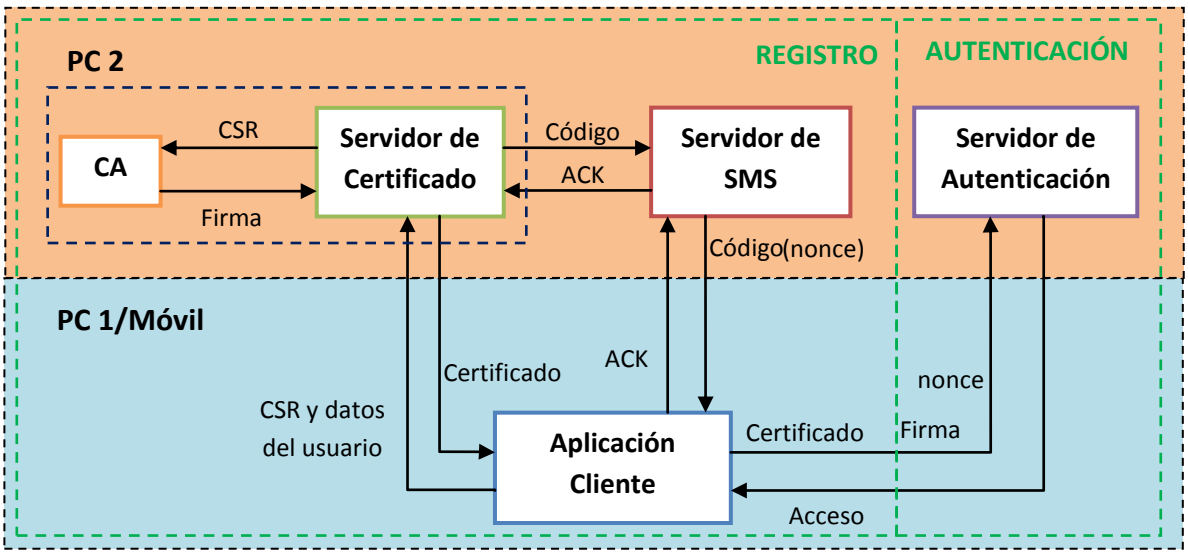


Figura 4.2. Distribución física de la implementación del protocolo. Existen dos partes diferenciadas: el registro para el cual se ha implementado un protocolo [1][2][3] y la autenticación que sirve como comprobación de que realmente ha funcionado correctamente.

Los servidores se han implementado en el lenguaje de programación Java y durante la implementación se han programado y ejecutado en un ordenador con el sistema operativo Linux. La utilización de Linux se debe a que facilita la operación de firma por el servidor, ya que esta se realiza mediante el uso de un script y porque el uso de OpenSSL en Windows requiere que se ejecute como administrador, lo cual complicaría el uso de este script comentado anteriormente.

Para la implementación de la aplicación se han utilizado tanto un móvil como el emulador de Android en un segundo ordenador con el sistema operativo Windows.

La conexión entre el servidor de la CA y el usuario, se realiza con un socket SSL seguro con autenticación de servidor, al igual que ocurre con el servidor de autenticación. La conexión del servidor de la CA con el servidor de SMS se realiza utilizando sockets SSL seguros con autenticación de cliente y servidor.

Hay que comentar que el servidor de la CA tampoco tendría que estar necesariamente en el mismo equipo que la CA, sino que simplemente tiene que ser de confianza del servidor real en el que se encuentre la CA, pero como se ha comentado anteriormente, esta forma de implementación tiene que ver con la practicidad. Podría considerarse, que el servidor de la CA es en realidad una RA, puesto que realiza las gestiones de registro, pero también permite que la CA firme los CSR del usuario.

En cuanto al uso de un servidor de SMS, es necesario comentar que en este caso es una simulación de un sistema real, se trata de una prueba de concepto. Una implementación real supondría unos gastos económicos adicionales que no son

compatibles con la realización de un gran número de pruebas necesarias para comprobar el funcionamiento de la aplicación. Para simular que se trata de un tipo de canal distinto a internet, el servidor de SMS se ha implementado utilizando socket normales, es decir, sin un tipo de conexión segura, pero a continuación se explica una posible implementación

En internet hay múltiples compañías que ofrecen servicios de mensajería que requieren de una contratación para la prestación de estos servicios. Algunos ejemplos son:

- Mensatek (http://www.mensatek.com/).
- SMSCertificado (http://www.smscertificado.es/): envía un sms al destinatario y una notificación de que el mensaje se ha enviado correctamente a la persona que ha enviado el email.
- Ozeki NG SMS Gateway (http://www.ozekisms.com/)
- SMSCover (http://www.smscover.com/)

Estas empresas permiten mediante la contratación de sus servicios, el envío de mensajes con mediación de páginas html, ejecutables, etc. Esto podría ser muy útil para un desarrollador o para una empresa que quisiera hacer usos de ellos, pues simplemente mediante la ejecución de una línea de código (como el caso de Ozeki NG SMS Gateway que posee ejecutables con código), una persona podría recibir un sms.

Otro problema que dificulta, aunque no imposibilita, una implementación usando un servidor de sms real, a parte de los gastos económicos adicionales, es que es difícil su simulación en el emulador de Android, ya que la recepción de sms requiere de una tarjeta SIM, igualmente si la recepción se realiza en un móvil sin la misma. Por tanto, si es posible crear código para la recepción de SMS, siempre que el código se comprueba en un móvil con tarjeta. Un ejemplo de código para la recepción de mensajes SMS se encuentra en el *anexo D*.

Para realizar la autenticación, desde la aplicación se genera un número aleatorio y se firma. El número aleatorio, la firma y el certificado se envían al servidor y este verifica que el certificado está firmado por la CA. El servidor además, a partir del certificado recibido firma el número aleatorio recibido y compara la firma con la recibida desde la aplicación, si ambas son iguales, se ha verificado la identidad del usuario.

Además, la aplicación permite la gestión de certificados, eliminando los certificados que el usuario ha decidido que no necesita utilizar más.

4.3. DISEÑO DE LA APLICACIÓN

El sistema anteriormente expuesto, se estructura de tal forma que si un usuario quiere acceder a una plataforma necesita haber obtenido previamente un certificado digital, pues la forma de autenticarse en el sistema es mediante el uso de certificado en lugar de usuario y contraseña como en la mayoría de sistemas actuales.

El diseño de la aplicación se estructura en tres partes principales, que se corresponden con tres acciones diferenciadas: registro, inicio de sesión y eliminar certificado.

Si el usuario usa la aplicación por primera vez, lógicamente no habrá obtenido ningún certificado por lo que no tendrá la capacidad de acceder a la plataforma. Por tanto, el primer paso que tiene que hacer un usuario es registrarse. Esto se hace igual en la mayoría de plataformas que requieren un proceso de registro y requiere que el usuario introduzca una serie de datos que le identifiquen como usuario del sistema.

Una vez que el usuario ha conseguido un certificado o varios, ya tiene la capacidad de acceder a una plataforma. El usuario elige cual de los certificados puede usar.

Por último, el usuario tiene la capacidad de eliminar los certificados de los que ya no haga uso.

En los siguientes apartados se explican detenidamente cada una de las partes principales que componen la aplicación y la interacción que realizan con los servidores.

4.3.1. Registro

El primer paso para el desarrollo de la aplicación se basó en conseguir generar un CSR por parte del usuario desde la aplicación. Para generar un CSR, se necesita conocer una serie de datos que se van a enviar a la CA para que está compruebe que son correctos y los firme. Normalmente, los datos que constan en un CSR son:

- Common Name o CN (Nombre común): Se refiere al nombre de la persona o entidad que quiere solicitar un certificado.

- Country o C (País): País en la que reside la persona que solicita el certificado.

- Locality o L (Localidad): Se refiere a la ciudad en la que reside la persona que lo solicita.

- EmailAddress (correo electrónico): Es el correo electrónico de la persona que los solicita.

- Organitation (Organización): Organización que solicita el certificado.

- Organitation Unit (Unidad de organización): Se refiere específicamente a la unidad a la que pertenece el solicitante.

Las dos últimas opciones, son opciones que se suelen utilizar cuando es un servidor el que necesita un certificado digital en lugar de una persona. Por supuesto, en ese caso también se deben rellenar el resto de apartados.

Durante el proceso de registro, la aplicación solicita al usuario los datos arriba mencionados para la creación del CSR. Además, se le solicita el número de teléfono, que se transmite al servidor de la CA para la generación del código que se envía al usuario y para que el servidor de SMS sepa donde tiene que enviar el código.

El establecimiento de la seguridad para la obtención de un certificado en el proceso de registro, se basa en una mejora del protocolo EBIA (Email Based Identification and Authentication). La seguridad de este protocolo se basa principalmente en que para que el usuario verifique sus datos, se le envía un ticket (código) o enlace a través de correo electrónico, pero esto puede ser algo inseguro, ya que el email puede ser interceptado por un atacante y la conexión entre un usuario y el servidor de correo no es segura. Por ello, si este ticket se envía también por el canal seguro (usando sockets SSL/TLS) entre el servidor de la CA y el usuario en la misma sesión que ha iniciado el propio usuario y se comprueba que ambos son iguales, esto indicaría que el usuario que ha iniciado la sesión es el dueño de esa cuenta de correo electrónico. En el caso de este proyecto, el servidor de correo se sustituye por un servidor de SMS.

A continuación se muestra gráficamente el funcionamiento de este protocolo utilizado en el sistema durante el proceso de registro:

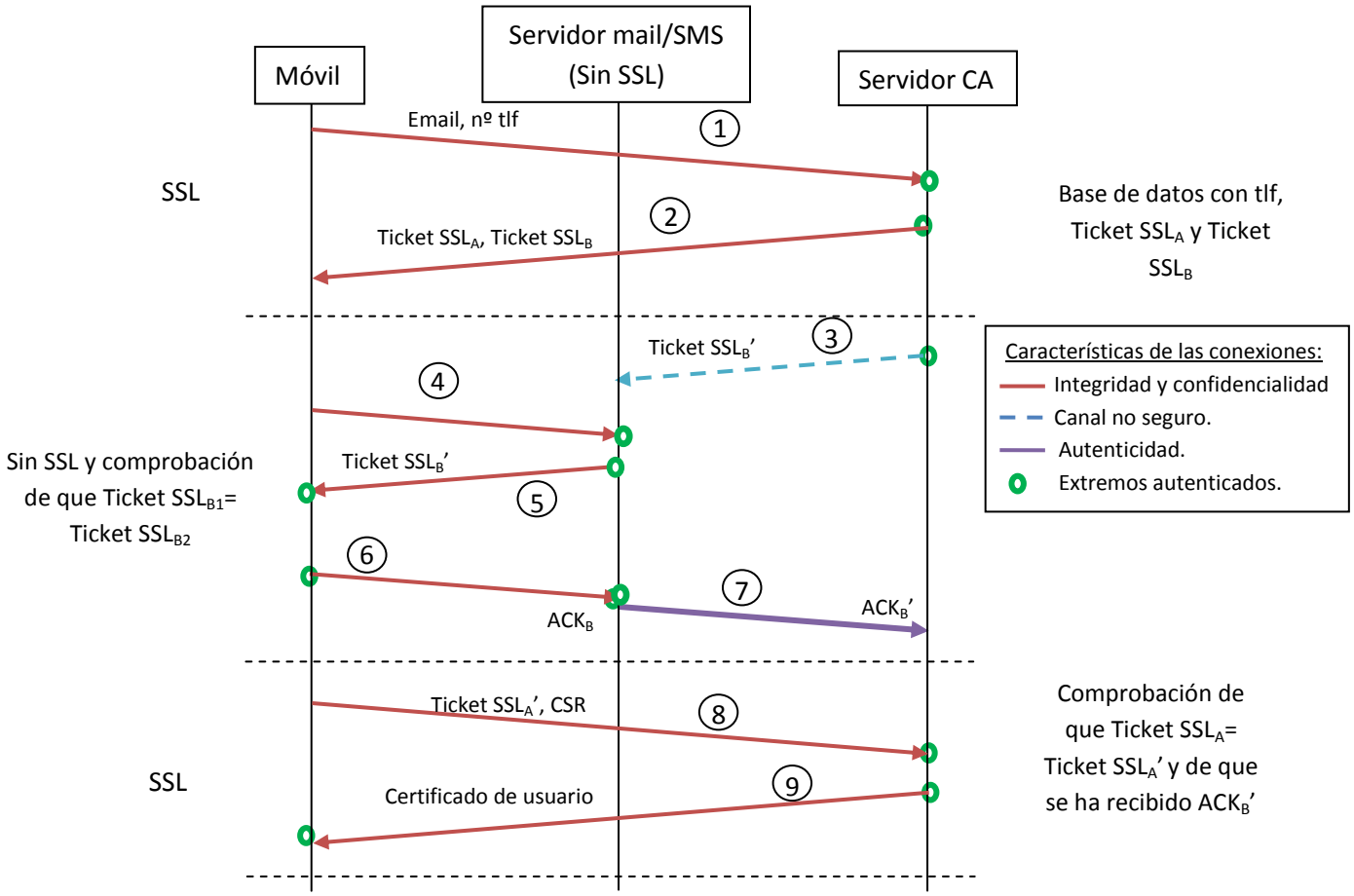


Figura 4.3. Diagrama de secuencia del protocolo usado en este proyecto. Esta imagen representa el protocolo que se ha implementado en este proyecto. El usuario se conecta al servidor de la CA y esta devuelve un código (nonce) al usuario y al servidor de SMS, si este código es correcto se envía un ack al servidor y se envía el csr para ser firmado por la CA.

Se expone a continuación detenidamente y paso a paso el funcionamiento de este protocolo correspondiendo cada paso con el número señalado en la *figura 4.2*:

1. El usuario se conecta al servidor de la CA con un socket SSL y le envía su email y su número de teléfono. El servidor de la CA genera un par de nonces (números aleatorios que solo se van a utilizar una vez) a partir del número de teléfono enviado por el usuario.
2. El servidor de la CA envía al usuario los nonces: un código A (ticket SSLA) y un código B (ticket SSLB).
3. El servidor de la CA envía el mismo código B al servidor de SMS.
4. El usuario se conecta al servidor de SMS, usando los sistemas de seguridad y autenticación de una red móvil.
5. El servidor de SMS le envía al usuario el código B que previamente el servidor de SMS había recibido del servidor de la CA.

6. El usuario comprueba que el código B del servidor de la CA y el código B del servidor de SMS es el mismo y envía un ACK al servidor de SMS.
7. El servidor de SMS envía el ACK recibido desde el usuario al servidor de la CA.
8. El usuario envía el código A recibido en el paso 2 junto con el CSR al servidor de la CA. El servidor de la CA comprueba que ha recibido el ACK del código B y que el Ticket SSLA existe en la base de datos y es el mismo que se ha enviado al mismo usuario.
9. El servidor de la CA después de comprobar que todos los datos son correctos, envía el certificado firmado al usuario.

Como se ha comentado en el apartado *4.2*, la conexión entre el usuario y el servidor de la CA se establece con un socket SSL/TLS con autenticación de servidor. La conexión entre el servidor de la CA y el servidor de SMS también se realiza con un socket SSL/TLS, pero esta vez existe tanto autenticación de cliente como autenticación de servidor, ya que los servidores deben confiar mutuamente. Por último y como se ha comentado anteriormente, la conexión entre el servidor de SMS y el usuario se realiza con un socket que no usa SSL para esta implementación.

Para establecer una conexión segura usando servidores en Java, es preciso tener almacenes de claves para que en el momento de la conexión se compruebe de comprueben los certificados de los extremos de la comunicación. En el caso de la conexión entre el usuario y el servidor de la CA en la que sólo hay autenticación de servidor, el truststore contiene el certificado del servidor de la CA y cuando se conecte, se comprobará de forma transparente al usuario y al desarrollador, que el certificado sea del servidor de la CA y que se tiene confianza en la conexión. En el caso de la conexión entre los servidores en los que existe autenticación de cliente y servidor, el servidor de SMS tiene almacenado en su truststore el certificado del servidor de la CA y este el del servidor de SMS.

Es necesario explicar, que los sistemas móviles tienen sus propios mecanismos de seguridad bastante similares a los sistemas informáticos. Los sistemas GSM (2G) basan su seguridad en la autenticación de la identidad del usuario, la confidencialidad de la identidad del abonado, confidencialidad de los datos de señalización y confidencialidad de los datos del usuario.

La forma de identificar de forma única en GSM a cada usuario es mediante el IMSI (International Mobile Subscriber Identity) que consiste en un código compuesto de 15 cifras decimales y es única en el mundo para cada usuario y con la clave individual de autenticación del abonado (Ki). Estos códigos se almacenan en la SIM junto con el algoritmo de generación de claves de cifrado, el algoritmo de autenticación y el número de identificación personal (PIN).

El sistema de comunicación GSM se compone de una serie de subsistemas, entre ellos el Centro de Autenticación (AuC) que es una base de datos que almacena para cada usuario el IMSI y su clave de autenticación de abonado (Ki).

Se le envía al móvil un número aleatorio de 128 bits, se calcula su firma que tiene una longitud de 32 bits, basándose en el cifrado del número aleatorio con el algoritmo de autenticación utilizando la clave individual de autenticación del abonado (Ki). Esto se envía a la estación GSM, que calcula la firma con los datos para cada usuario almacenados en el AuC y se comparan ambos resultados, si coinciden se permite el acceso.

La seguridad en sistemas UMTS (3G) es similar a la seguridad de GSM, excepto en que no sólo se produce autenticación del usuario, si no que en el dispositivo del usuario también se autentica la identidad de la red servidora (Server Network o SN) a la que se conecta. Es un esquema de autenticación mutua. El mecanismo de seguridad de UMTS se denomina AKA (Authentication and Key Agreement) y en él, el móvil y la red manejan una clave de cifrado y una clave de integridad.

En los sistemas LTE (4G), la seguridad es igual a las redes UMTS. Esto tiene sentido al derivar 4G de 3G y 3G de 2G.

Por tanto, en general la forma de autenticar a un usuario en una red móvil, se realiza de forma muy similar a la autenticación que se realizaría en un sistema informático.

El servidor de la CA recibe un CSR creado por la aplicación. Este certificado consta de las siguientes partes:

- Versión del tipo de CSR:

```
Version: 0 (0x0)
```

- Subject: contiene una serie de datos que identifica al usuario.

```
Subject: CN=Nombre Apellido1 Apellido2, C=Spain,
 L=Madrid/emailAddress=nombre@estudiante.uam.es
```

- Información sobre la clave pública del usuario:
    - Algoritmo usado para generarla:

```
Public Key Algorithm: rsaEncryption
```

    - Tamaño de las claves:

```
RSA Public Key: (1024 bit)
```

    - Clave pública:

```
Modulus (1024 bit):
    00:c5:40:b3:76:fb:40:cd:51:0c:ec:87:86:5a:4f:
    d7:a5:99:a8:4f:ab:0c:44:5e:4d:e3:26:cb:7c:a6:
```

```
                    95:c4:ed:9f:db:4b:11:f5:5e:bd:6b:3d:a5:4c:85:
                    5c:16:c9:7c:ad:42:50:b7:0f:61:c4:4d:76:a3:ca:
                    63:61:00:ae:f6:34:41:c5:b0:82:f4:12:3e:65:96:
                    6a:83:90:06:de:49:e1:ee:b7:c3:a1:79:a8:c1:0b:
                    f0:8e:a6:00:0e:5e:2e:17:90:08:f8:1e:b4:14:4a:
                    5a:36:3d:db:c2:78:b7:77:ca:b1:41:1e:f5:ce:38:
                    71:2d:c6:4d:a2:6f:2a:80:6d
                Exponent: 65537 (0x10001)
```

- Atributos:

```
Attributes:
  challengePassword        :123456789
```

- Firma digital del CSR hecha con la clave privada del usuario:

```
Signature Algorithm: sha1WithRSAEncryption
        9e:5c:90:2d:ad:88:62:1e:fb:d1:04:66:31:40:5c:db:05:e8:
        ac:94:88:fb:36:68:bd:88:8e:a5:3c:ba:64:dc:4a:87:54:c2:
        0b:4c:b1:a4:94:61:7b:f4:a9:11:cf:68:4d:a8:7b:d2:55:88:
        e7:aa:ef:f0:70:38:b9:4a:6b:a8:17:67:8d:48:2b:e6:d9:9f:
        71:19:d2:77:55:48:a1:70:9c:b7:3d:c7:9e:0d:79:fa:b0:97:
        3e:e7:6b:75:d7:9d:00:94:80:ac:27:18:e1:3a:62:db:ce:1d:
        76:a2:b1:fc:3c:cb:c3:27:a6:ca:76:1c:d5:59:86:2f:97:66:
        e1:fd
```

El CSR final es como el que sigue:

```
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: CN=Nombre Apellido1 Apellido2, C=Spain,
L=Madrid/emailAddress=nombre@estudiante.uam.es
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:c5:40:b3:76:fb:40:cd:51:0c:ec:87:86:5a:4f:
                    d7:a5:99:a8:4f:ab:0c:44:5e:4d:e3:26:cb:7c:a6:
                    95:c4:ed:9f:db:4b:11:f5:5e:bd:6b:3d:a5:4c:85:
                    5c:16:c9:7c:ad:42:50:b7:0f:61:c4:4d:76:a3:ca:
                    63:61:00:ae:f6:34:41:c5:b0:82:f4:12:3e:65:96:
                    6a:83:90:06:de:49:e1:ee:b7:c3:a1:79:a8:c1:0b:
                    f0:8e:a6:00:0e:5e:2e:17:90:08:f8:1e:b4:14:4a:
                    5a:36:3d:db:c2:78:b7:77:ca:b1:41:1e:f5:ce:38:
                    71:2d:c6:4d:a2:6f:2a:80:6d
                Exponent: 65537 (0x10001)
        Attributes:
            challengePassword        :123456789
    Signature Algorithm: sha1WithRSAEncryption
        9e:5c:90:2d:ad:88:62:1e:fb:d1:04:66:31:40:5c:db:05:e8:
        ac:94:88:fb:36:68:bd:88:8e:a5:3c:ba:64:dc:4a:87:54:c2:
        0b:4c:b1:a4:94:61:7b:f4:a9:11:cf:68:4d:a8:7b:d2:55:88:
        e7:aa:ef:f0:70:38:b9:4a:6b:a8:17:67:8d:48:2b:e6:d9:9f:
        71:19:d2:77:55:48:a1:70:9c:b7:3d:c7:9e:0d:79:fa:b0:97:
        3e:e7:6b:75:d7:9d:00:94:80:ac:27:18:e1:3a:62:db:ce:1d:
        76:a2:b1:fc:3c:cb:c3:27:a6:ca:76:1c:d5:59:86:2f:97:66:
        e1:fd
```

Figura 4.4. CSR enviado por el usuario. Un CSR contiene un conjunto de datos que identifican al usuario, la clave publica generada desde la aplicación y una firma digital hecha con la clave privada del usuario.

El servidor no tiene forma de conocer la identidad del usuario que solicita el certificado a parte de los datos personales que se incluyen en el subject, pero el certificado generado sólo se puede usar si está asociado a la clave privada de ese certificado. El CSR contiene la clave pública del usuario y la firma del usuario, por lo que internamente y de forma transparente, la CA puede verificar la identidad del usuario.

Como resultado de todo este proceso de registro, el usuario obtiene un certificado. Este certificado consta de varias partes que definen sus características y se explican a continuación:

- Version: versión de estándar de X.509 que utiliza el certificado.

```
Version: 1 (0x0)
```

- Serial Number: número de serie del certificado que es un número que identifica de manera única al certificado y ha sido generado por la CA.

```
Serial Number:
            88:96:1e:fe:74:f9:75:a6
```

- Signature Algorithm: nombre del algoritmo de firma que ha usado la CA para firmar el certificado

```
Signature Algorithm: sha1WithRSAEncryption
```

- Issuer: identidad de la CA que emite el certificado

```
Issuer: C=ES, ST=Madrid, L=Madrid, O=UAM, OU=EPS, CN=Servidor
        CA/emailAddress=evam.blanco@gmail.com
```

- Validity: periodo de validez del certificado recibido por el usuario

```
Validity
            Not Before: Feb  4 21:04:18 2014 GMT
            Not After : Feb  2 21:04:18 2024 GMT
```

- Subject: datos personales del usuario que ha solicitado el certificado

```
Subject: CN=Nombre Apellido1 Apellido2, C=Spain,
L=Madrid/emailAddress=nombre.apellido@estudiante.uam.es
```

- Subject Public Key Info: clave pública del usuario y algoritmo de clave pública asociado a esa clave pública.

```
Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (1024 bit)
          Modulus (1024 bit):
              00:9d:4b:2e:7b:b3:12:b7:ab:fd:1c:48:39:11:f0:
              c2:cb:07:3e:f7:81:31:11:94:66:a0:06:c8:db:8a:
              ba:53:c5:75:7b:62:82:70:2b:94:7e:f4:5a:8c:ea:
              48:e6:3c:8f:4d:61:1d:6e:cd:2f:36:93:02:74:27:
              f5:25:4c:2c:ac:07:ad:91:07:c8:74:1d:55:bf:73:
              51:fe:69:2e:96:24:8d:42:ea:40:36:50:84:64:55:
              8c:f9:c4:88:5e:40:7e:84:51:db:ac:29:c0:f1:e7:
              a1:d5:0a:6d:6e:79:15:14:91:07:10:9d:62:e2:92:
              a7:a8:32:e8:99:b2:32:45:5b
          Exponent: 65537 (0x10001)
```

- Firma digital de la Autoridad Certificadora que consiste en un hash de la información anterior y su cifrado con la clave privada de la CA.

```
Signature Algorithm: sha1WithRSAEncryption
        bc:30:ad:fa:2f:22:d8:92:73:36:02:ff:dd:77:65:99:d5:f4:
        2b:1c:1e:97:db:b9:1e:95:65:74:62:75:9a:5d:6e:36:5e:61:
        e3:63:cf:8e:a5:fd:6b:15:0d:08:e0:ad:f3:3c:d2:b5:2f:71:
        cb:49:90:5a:87:d8:00:18:f4:ab:68:d6:8f:3b:d8:07:7c:c1:
        e3:d2:9b:f1:c4:6b:51:79:fb:68:49:07:ba:73:70:79:99:b2:
        a9:1e:dd:92:e9:8d:75:cd:c6:18:72:4b:f1:3c:34:f1:bc:b3:
        27:62:65:68:00:69:4c:e3:07:82:92:0c:16:37:9d:fe:99:f8:
        9f:97:f0:54:9e:fc:81:44:e0:f7:f8:75:c0:e2:85:ac:c3:bf:
        4a:de:ea:7c:c6:24:cc:f6:fc:e0:bd:07:10:54:8a:95:f5:64:
        9c:6e:41:96:5d:da:d5:71:3a:3a:f1:3c:80:46:3e:14:35:27:
        1d:fc:05:3d:cf:9e:7e:85:96:2e:1a:65:79:07:92:80:56:87:
        65:d7:23:a4:22:18:14:68:11:44:ab:42:52:14:3e:e7:46:14:
        1e:bd:d1:0b:8b:9a:9f:2f:09:2c:ca:c1:ee:27:05:22:f3:83:
        eb:71:8b:1a:0e:8d:b2:cf:d9:e0:53:46:22:95:a6:dd:a4:ea:
        cd:b5:05:fe:63:db:17:4a:35:ac:7c:34:dd:c5:77:18:af:30:
        27:c7:ec:09:e6:8b:e9:48:49:f8:d2:d5:17:7e:12:85:77:69:
        b4:87:cc:11:c7:0c:bb:ef:67:b0:3f:c5:27:58:46:b0:20:bc:
        62:80:26:38:ae:be:3a:a3:25:89:e1:40:89:b6:bd:a2:8d:23:
        a6:71:0f:52:4f:da:ec:7e:4a:13:a9:c7:96:58:de:d9:5c:f4:
        d7:06:96:e3:31:40:88:a5:be:12:46:e4:cd:f2:0f:06:2f:b6:
        e0:c8:b4:e1:83:cd:fb:6d:cd:70:c9:e5:fe:2f:db:85:be:79:
        16:c6:04:db:94:7b:20:5c:1a:a1:cf:ff:70:5c:55:fa:99:7b:
        34:9b:af:57:1b:81:21:cd:36:75:8b:a4:6a:64:96:8d:f9:4d:
        de:e8:25:9a:5e:d1:86:dc:c5:4f:f9:4c:6f:cd:1a:14:c2:ab:
        8b:af:0a:e1:dd:82:8e:e5:d9:db:c9:a0:b7:e2:f9:4e:c3:b6:
        94:cd:5d:c1:03:8a:c0:9d:ea:d9:cc:6e:dd:f0:58:89:5d:5c:
        e1:06:50:0b:ce:09:88:0b:15:cd:65:21:b5:a1:e7:8e:d4:79:
        02:9e:b2:c4:51:2c:8d:fe:73:02:b2:b7:a7:d4:c3:c7:ac:64:
        93:59:ba:b4:39:67:0e:74
```

Por último, se muestra la estructura real que muestra el certificado que se envía a un usuario:

```
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            88:96:1e:fe:74:f9:75:a6
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=ES, ST=Madrid, L=Madrid, O=UAM, OU=EPS, CN=Servidor
CA/emailAddress=evam.blanco@gmail.com
        Validity
            Not Before: Feb  4 21:04:18 2014 GMT
            Not After : Feb  2 21:04:18 2024 GMT
        Subject: CN=Nombre Apellido1 Apellido2, C=Spain,
L=Madrid/emailAddress=nombre.apellido@estudiante.uam.es
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:9d:4b:2e:7b:b3:12:b7:ab:fd:1c:48:39:11:f0:
                    c2:cb:07:3e:f7:81:31:11:94:66:a0:06:c8:db:8a:
                    ba:53:c5:75:7b:62:82:70:2b:94:7e:f4:5a:8c:ea:
                    48:e6:3c:8f:4d:61:1d:6e:cd:2f:36:93:02:74:27:
                    f5:25:4c:2c:ac:07:ad:91:07:c8:74:1d:55:bf:73:
                    51:fe:69:2e:96:24:8d:42:ea:40:36:50:84:64:55:
                    8c:f9:c4:88:5e:40:7e:84:51:db:ac:29:c0:f1:e7:
                    a1:d5:0a:6d:6e:79:15:14:91:07:10:9d:62:e2:92:
                    a7:a8:32:e8:99:b2:32:45:5b
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha1WithRSAEncryption
        bc:30:ad:fa:2f:22:d8:92:73:36:02:ff:dd:77:65:99:d5:f4:
        2b:1c:1e:97:db:b9:1e:95:65:74:62:75:9a:5d:6e:36:5e:61:
        e3:63:cf:8e:a5:fd:6b:15:0d:08:e0:ad:f3:3c:d2:b5:2f:71:
        cb:49:90:5a:87:d8:00:18:f4:ab:68:d6:8f:3b:d8:07:7c:c1:
        e3:d2:9b:f1:c4:6b:51:79:fb:68:49:07:ba:73:70:79:99:b2:
        a9:1e:dd:92:e9:8d:75:cd:c6:18:72:4b:f1:3c:34:f1:bc:b3:
        27:62:65:68:00:69:4c:e3:07:82:92:0c:16:37:9d:fe:99:f8:
        9f:97:f0:54:9e:fc:81:44:e0:f7:f8:75:c0:e2:85:ac:c3:bf:
        4a:de:ea:7c:c6:24:cc:f6:fc:e0:bd:07:10:54:8a:95:f5:64:
        9c:6e:41:96:5d:da:d5:71:3a:3a:f1:3c:80:46:3e:14:35:27:
        1d:fc:05:3d:cf:9e:7e:85:96:2e:1a:65:79:07:92:80:56:87:
        65:d7:23:a4:22:18:14:68:11:44:ab:42:52:14:3e:e7:46:14:
        1e:bd:d1:0b:8b:9a:9f:2f:09:2c:ca:c1:ee:27:05:22:f3:83:
        eb:71:8b:1a:0e:8d:b2:cf:d9:e0:53:46:22:95:a6:dd:a4:ea:
        cd:b5:05:fe:63:db:17:4a:35:ac:7c:34:dd:c5:77:18:af:30:
        27:c7:ec:09:e6:8b:e9:48:49:f8:d2:d5:17:7e:12:85:77:69:
        b4:87:cc:11:c7:0c:bb:ef:67:b0:3f:c5:27:58:46:b0:20:bc:
        62:80:26:38:ae:be:3a:a3:25:89:e1:40:89:b6:bd:a2:8d:23:
        a6:71:0f:52:4f:da:ec:7e:4a:13:a9:c7:96:58:de:d9:5c:f4:
        d7:06:96:e3:31:40:88:a5:be:12:46:e4:cd:f2:0f:06:2f:b6:
        e0:c8:b4:e1:83:cd:fb:6d:cd:70:c9:e5:fe:2f:db:85:be:79:
        16:c6:04:db:94:7b:20:5c:1a:a1:cf:ff:70:5c:55:fa:99:7b:
        34:9b:af:57:1b:81:21:cd:36:75:8b:a4:6a:64:96:8d:f9:4d:
        de:e8:25:9a:5e:d1:86:dc:c5:4f:f9:4c:6f:cd:1a:14:c2:ab:
        8b:af:0a:e1:dd:82:8e:e5:d9:db:c9:a0:b7:e2:f9:4e:c3:b6:
        94:cd:5d:c1:03:8a:c0:9d:ea:d9:cc:6e:dd:f0:58:89:5d:5c:
        e1:06:50:0b:ce:09:88:0b:15:cd:65:21:b5:a1:e7:8e:d4:79:
        02:9e:b2:c4:51:2c:8d:fe:73:02:b2:b7:a7:d4:c3:c7:ac:64:
        93:59:ba:b4:39:67:0e:74
```

Figura 4.5. Certificado recibido por el usuario. El usuario recibido por el usuario contiene todos los apartados definidos anteriormente: algoritmo de fima, clave pública del usuario, firma digital hecha con la clave pública de la CA, tamaño de las claves, etc.

Las extensiones más comunes usadas para los certificados son:

- *.pem:* codificación en base 64[20] de un formato DER[21]. Pueden contener la clave privada o no, pero se prefiere que no la contenga.

- *.cer, .crt, .der:* codificación binaria con el formato DER.

- *.p7b, .p7c:* pueden contener un certificado o una cadena de certificados, pero no puede contener la clave privada. Utiliza codificación en base 64.

- *.pfx, .p12:* creados para contener certificados de servidores, certificados intermedios y claves privadas. Utilizan un formato binario.

4.3.1.1. Uso de certificados en el sistema de registro.

Para establecer las conexiones entre los servidores, estos necesitan tener cierta confianza entre sí. Las conexiones que se establecen son conexiones SSL/TLS que hacen que los mensajes que se trasmiten estén cifrados. Para el establecimiento de las conexiones, ambos extremos de la comunicación comprueban que pueden confiar en el certificado del otro extremo.

Cada extremo de la comunicación tiene un keystore y un truststore. En el keystore se almacenan las claves privadas y en el truststore se almacenan los certificados, tanto del propio servidor, como de los servidores en los que se confía, de forma que si el servidor de la CA confía en el servidor de SMS guardará en su truststore el certificado del servidor de SMS, además del suyo y en el truststore del servidor de SMS se guardará el certificado del servidor de la CA y su propio certificado. En ambos servidores se almacena el certificado de la CA en el truststore.

Mientras que el certificado del otro extremo de la conexión esté guardado en el almacén de claves, no es demasiado importante que sean certificados autofirmados o estén firmados por una CA, pues almacenar el certificado en el almacén de claves significa que se confía en el certificado que se almacena. Aunque siempre es mejor que el certificado esté firmado por una CA de confianza. En la siguiente tabla se especifica cómo se distribuyen las claves en los almacenes de claves para los servidores implicados en el proceso de registro y el usuario:

[20] Base 64: es un sistema de numeración posicional que usa 64 como base. Es la mayor potencia de 2 que puede ser representada usando únicamente los caracteres de ASCII
[21] Formato DER: Es un formato binario usado principalmente en Java y en plataforma Macintosh

	Usuario	Servidor de la CA	Servidor de SMS
Keystore	-Clave privada usuario -Certificado usuario	-Clave privada del servidor de la CA	-Clave privada del servidor de SMS
Truststore	-Certificado del servidor de la CA -Certificado del servidor de SMS	- Certificado de la CA - Certificado del servidor de la CA - Certificado del servidor de SMS	- Certificado de la CA - Certificado del servidor de SMS - Certificado del servidor de la CA

Tabla 4.1. Configuración de los almacenes de claves para el proceso de registro.

En los siguientes apartados se explica cómo se crea una autoridad certificadora y como se establece la confianza entre los certificados.

4.3.1.1.1. Creación de una Autoridad Certificadora.

La finalidad de la aplicación que se ha desarrollado en este proyecto es obtener una identidad digital. Para obtener una identidad digital, un usuario debe enviar un CSR a una CA y obtener un certificado digital firmado por la misma CA. En el caso de este proyecto, la CA también es la que otorga cierta confianza al servidor de cara al usuario de la aplicación, ya que el certificado que usa este servidor también está firmado por ella.

Para la creación de la CA, se usa la herramienta Openssl, con la cual se genera una clave privada y un certificado autofirmado. El comando que se utiliza es el siguiente:

```
openssl req -x509 -newkey rsa:4096 -keyout CAkey.key -days 3650 -out CAcert.pem
```

Este comando indica que se crea una clave privada de tamaño 4096 bits con el algoritmo RSA. Esta clave se almacena en el fichero CAkey.key. El certificado que se crea se almacena en el fichero CAcert.pem, tiene una validez de 3650 días (10 años) y es un certificado autofirmado.

4.3.1.1.2. Conexión entre el Servidor de la CA y el Servidor de SMS.

La comunicación que se produce entre el llamado servidor de la CA y el servidor de SMS es una conexión en la que existe tanto autenticación de cliente como autenticación de servidor, en donde el papel de cliente es jugado por el servidor de la CA y el papel de servidor es jugado por el servidor de SMS.

En una conexión SSL la autenticación de servidor es obligatoria y la autenticación de cliente es opcional.

La autenticación de servidor consiste en que el cliente (en este caso el servidor de la CA) tiene almacenado el certificado del servidor de SMS en su truststore y cuando se establece la conexión, se comprueba durante el handshake del protocolo SSL/TLS (apartado *2.3.6*) de forma transparente al desarrollador, que esto se cumple y que el certificado almacenado es el del servidor de SMS.

De igual forma, la autenticación de cliente consiste en que el servidor (en este caso el servidor de la CA) tiene almacenado en su truststore el certificado del cliente

Configuración de los almacenes de claves en el servidor de la CA. (Generación de la identidad digital del servidor de la CA)

1) Creación de un par de claves dentro de una keystore:

```
keytool -genkey -keyalg RSA -keysize 4096 -alias servalias -keystore ksServidorCA.jks
```

Esto indica que se crea una clave privada con el algoritmo criptográfico RSA, con un tamaño de 4096 bits. La entrada de las claves en el keystore corresponde al alias "servalias" y el keystore que se crea tiene como nombre ksServidor.jks.

2) Generación del CSR del servidor:

```
keytool -keystore ksServidorCA.jks -certreq -alias servalias -file CSRservidor.csr
```

Se obtiene del keystore un CSR que será firmado por la CA.

3) Firma del CSR por parte de la CA:

```
openssl x509 -req -days 3650 -in CSRservidor.csr -CA CAcert.pem -CAkey CAkey.key -Cacreateserial -out CertServidor.pem
```

Como resultado se obtiene un certificado *"CertServidor.pem"* con una validez de 3650 días firmado por la CA, lo que de cara a un usuario da confianza al servidor.

4) Importar el certificado del servidor y de la CA al truststore:

```
keytool -import -alias servalias -keystore tsServidorCA.jks -trustcacerts -file CertServidor.pem

keytool -import -alias caalias -keystore tsServidorCA.jks -trustcacerts -file CAcert.pem
```

Configuración de los almacenes de claves en el servidor de SMS. (Generación de la identidad digital del servidor de SMS)

Para la configuración de la seguridad en este servidor, se ha hecho de una forma algo diferente en relación a la configuración del servidor de CA, si bien, se podría haber

configurado de la misma forma, pues se ha comprobado que habría funcionado correctamente.

En este caso, el servidor de SMS utiliza un certificado autofirmado. Esto podría restar seguridad, sin embargo, si se importa el certificado del servidor de SMS en el truststore del servidor de la CA, se está indicando que el servidor de la CA confía en el servidor de SMS.

1) Creación del keystore y truststore del servidor de SMS:

```
keytool -genkeypair -keyalg RSA -keysize 4096 -dname "CN=servSMS, C=ES" –
alias aliasservidorsms -keypass password -storepass password -validity 3650 -
keystore ksServidorSMS.jks
```

Este comando indica que se genera un par de claves, usando el algoritmo RSA, con un tamaño de 4096 bits. El Common Name del servidor es *"servSMS"*, el país (Country o C) es España (ES). La contraseña tanto del keystore como de la clave privada es *"password"*, tiene una validez de 3650 días y el almacén de claves se llama *"ksServidorSMS.jks"*.

2) Exportación del certificado del servidor:

```
keytool -export -alias aliasservidorsms -keystore ksServidorSMS.jks -rfc –
file CertServidorSMS.pem
```

3) Importar el certificado del servidor de SMS en el truststore del servidor de la CA
4) Importar el certificado del servidor de la CA en el truststore del servidor de SMS

Estos dos pasos se hacen como en el paso 4 del caso anterior.

4.3.1.1.3. Conexión entre el Servidor de la CA y el usuario.

En este caso, en la comunicación que se produce sólo existe autenticación de servidor, ya que el usuario no tiene certificado que le autentique (pues entre otras cosas obtener un certificado es una de las finalidades de este proyecto).

El servidor ya se encuentra configurado pues ya se han creado las claves, los certificados y los almacenes de claves tal y como se ha mostrado en apartados anteriores.

Para la configuración en la parte del cliente sólo se necesita crear un truststore que contenga el certificado del servidor de la CA. Para ello se puede hacer de dos formas:

1) Mediante la herramienta keytool:

```
keytool -importcert -v -trustcacerts -file CertServidor.pem -alias servalias -
keystore kscliente.bks –provider
```

```
org.bouncycastle.jce.provider.BouncyCastleProvider -providerpath bcprov-
jdk15on-148.jar" -storetype BKS -storepass password
```

2) Mediante la API Portecle:

File-> New Keystore -> BKS

Tools -> Import Trusted Certificate

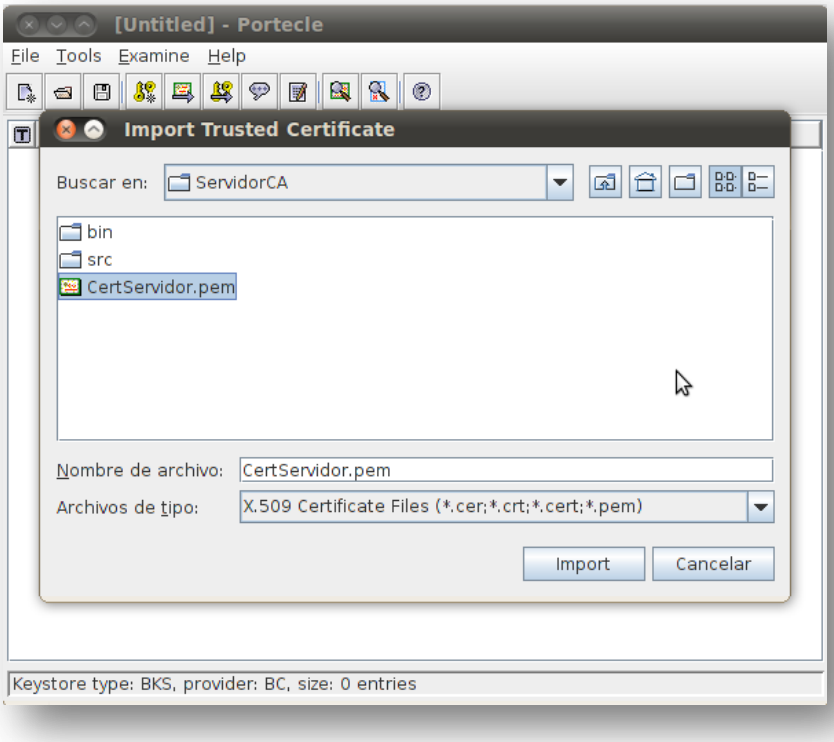


Figura 4.6. Imagen de la API Portecle. Portecle permite importar y exportar certificados, crear keystores y truststores de diferentes formatos, por ejemplo, .jks y .bks, etc.

4.3.1.1.4. Conexión entre el Servidor de SMS y el usuario.

En este caso, dado que se simula la recepción de mensajes mediante mensajes de texto, no se configura ningún elemento de seguridad, pues en un escenario real, está comunicación no se produciría a través de internet, sino que la seguridad se realiza con elementos propios de comunicaciones móviles y de sus protocolos, que como se ha comentado en el apartado *4.3.1* es muy similar a la seguridad implementada en un sistema informático, pues la autenticación de un usuario depende de una cifra aleatoria y una firma sobre la misma.

4.3.1.1.5. Firma del certificado del usuario.

La aplicación que se desarrolla en este proyecto, está configurada de tal forma, que es la propia aplicación la que genera el par de claves privada y pública y un CSR.

Este CSR se envía al programa que hace las veces de servidor de CA. Este, permite la ejecución de un script *(firmar.sh)* que hace que la CA firme el CSR de forma automática.

Se crea un script porque este facilita la ejecución de un comando OpenSSL desde un programa escrito en lenguaje de programación Java. En un entorno real, la realización de la firma, estaría implementada de tal forma que su ejecución sería más automática y no requeriría la implementación de dicho script. Esto se ha llevado a cabo de esta forma, porque esta implementación y el uso de OpenSSL facilita una tarea que permite la consecución de los objetivos principales del proyecto, que como ya se ha dicho son la creación de una aplicación para la obtención de identidades digitales y la implementación de un protocolo del que haga uso la aplicación.

El comando que se ejecuta en el script es el siguiente:

```
#!/bin/bash
openssl x509 -req -in ./CSRusuarios/$1.csr -CA CAcert.pem -CAkey CAkey.key -
passin pass:password -CAcreateserial -out CertUsuarios/$1.pem -days 3650
exit
```

$1 representa el parámetro que se pasa a este script. En este caso, este parámetro es el número de teléfono enviado por el propio usuario. Es decir, el CSR enviado por el usuario y el certificado que se le envía, se guardan asignándolos con el número de teléfono, de forma que son fácilmente legibles por el programa para volver a envíaselo al usuario.

4.3.1.1.6. Almacenamiento del certificado obtenido.

Cuando el usuario recibe el certificado firmado, la aplicación solicitará al usuario darle un nombre.

El certificado se almacena en el dispositivo de dos formas:

1) Como un archivo con extensión *".pem"* dentro de un directorio creado por la aplicación para almacenar los certificados.
2) Dentro de una keystore (*ksusuario.bks*) que se crea por la aplicación a la vez que el archivo *".pem"*. Para almacenar un archivo en una keystore, se requiere que cada entrada se guarde con un alias, en la aplicación cada entrada usa como alias el nombre que el usuario da al certificado.

Por ejemplo, si un usuario llama a su certificado *"MiCert"* se creará un directorio (si éste no existe), en donde se almacenará el certificado *"MiCert.pem"* y en el directorio dedicado al almacenamiento del keystore del usuario se creará una keystore (si esta no existe), que guarda el keystore del usuario *ksusuario.bks*. Este archivo *ksusuario.bks*, almacenará una entrada con la clave privada del usuario y el certificado, usando como alias *"MiCert"*. Esto se debe a que en las primeras versiones de la aplicación no se pensaba incluir el uso de una keystore por parte del usuario, sino que fue algo que surgió a medida que el proyecto avanzaba. Una futura implementación o mejora, puede ser no hacer uso de archivos para almacenar los certificados y sólo utilizar un keystore.

Es importante dejar claro que, aunque normalmente las claves públicas o certificados se almacenan en los truststores, en este caso el certificado se guarda en el keystore. Como se ha explicado en otros capítulos, un keystore puede almacenar tanto claves públicas como privadas. Almacenar el certificado del usuario en el keystore se debe a que el truststore del usuario se ha creado de forma previa con la herramienta Keytool y se ha guardado en la carpeta de recursos de la aplicación, que es una carpeta sólo de lectura y por tanto, no puede ser modificada desde código, ya que al igual que el resto de la aplicación está en la memoria RAM. Sin embargo, las claves del usuario y el certificado se almacenan desde código por lo que no se pueden almacenar en el truststore. Almacenar el certificado en el keystore es indiferente en este caso.

El código de la aplicación y la forma en la que se crea una keystore desde código se encuentra en el *anexo F*.

4.3.2. Autenticación usando la identidad digital generada

La siguiente acción principal de la aplicación es la autenticación. La aplicación desarrollada tiene la capacidad de que el usuario se autentique en una plataforma. Esto se produce cuando el usuario elige la opción de iniciar sesión, la cual permite la posibilidad de autenticarse usando un certificado. Una vez que el usuario elige un certificado se inicia una serie de acciones transparentes al usuario que le permiten autenticarse. Al igual que anteriormente es una acción simulada, pues el usuario no accede realmente a una plataforma, si no que accede al programa creado para hacer las veces de servidor.

La forma en la que el usuario se autentica al iniciar sesión, es mediante la generación desde la aplicación de una firma digital generada por el certificado que ha sido obtenido por el usuario durante el proceso de registro. Esta firma digital se ha producido al realizar una firma sobre un nonce generado en la aplicación.

Posteriormente, el número aleatorio junto a la firma y al certificado, es enviado al servidor.

El servidor, como paso principal, debe comprobar que el certificado digital enviado por el usuario ha sido emitido por la CA. Posteriormente, haciendo uso de la firma y el número aleatorio recibidos se verifica la firma. Si se verifican tanto el certificado como la firma recibidas, se le da acceso al usuario. Si por el contrario, si uno de los dos no se puede verificar, la identidad del usuario no se puede afirmar y el servidor no da acceso a esa persona.

La firma se realiza en la aplicación usando la clave privada del certificado del usuario. Por tanto, para que el servidor verifique la firma del usuario tiene que descifrar la firma usando la clave pública del usuario, la cual extrae del certificado recibido. Una vez descifrada la firma, compara el resultado con el nonce enviado y si son iguales, se ha verificado el certificado.

Para verificar que el certificado enviado por la aplicación y recibido en el servidor es válido, es decir, que ha sido emitido por la CA, el servidor descifra la firma incluida en el certificado con la clave pública de la CA y comprueba que el resultado es igual al contenido del certificado.

Se muestra a continuación un esquema del proceso:

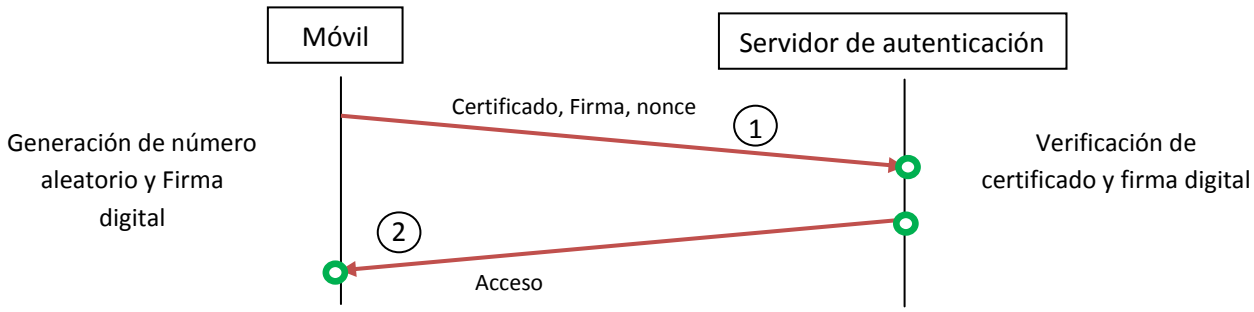


Figura 4.7. Diagrama de secuencias del proceso de autenticación. El usuario envía el certificado, firma y nonce y el servidor comprueba que el certificado ha sido realmente emitido por una CA y que la firma realmente está hecha con tal certificado.

La conexión que se establece entre la aplicación y el servidor, se establece mediante una conexión realizada mediante sockets y el uso del protocolo SSL/TLS con autenticación de servidor pero no de cliente, ya que la autenticación de cliente no se produce hasta que no se verifican los datos del usuario.

Dado que la conexión que se establece entre el usuario y el servidor es como el que se produce entre el usuario y el servidor de la CA, la forma de configurar los almacenes de

claves y la generación de claves para el servidor se realiza de la misma forma que la especificada en el apartado *4.3.1.1.3*.

En cualquier caso, se muestra a continuación una tabla que muestra como se distribuyen los certificados en los almacenes de claves en esta conexión:

	Usuario	Servidor Autenticación
Keystore	-Clave privada usuario -Certificado usuario	-Clave privada del servidor de autenticación
Truststore	-Certificado del servidor de autenticación	- Certificado de la CA - Certificado del servidor de autenticación

Tabla 4.2. Configuración de los almacenes de claves para la autenticación.

4.3.2.1. Generación y verificación de firma y de certificado para la autenticación del usuario.

El siguiente esquema muestra el proceso de autenticación de un usuario:

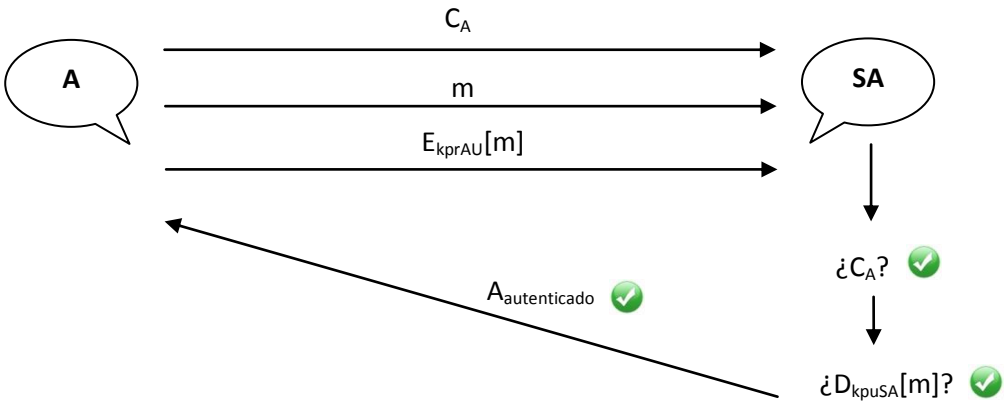


Figura 4.8. Esquema de autenticación. El usuario envía el certificado, un mensaje (en este caso un nonce) y la firma digital. El servidor verifica que el certificado ha sido emitido por la CA y la firma.

Donde:

- A es el usuario.
- C_A es el certificado del usuario.
- m es el mensaje, en este caso es un nonce.
- $E_{kprAU}[m]$ es la firma digital.
- DkpuSA[m] representa la verificación o descifrado del certificado.

Acciones realizadas por la aplicación (A):

- Generación de un número aleatorio.

- Firma digital E_{kprU} de ese número aleatorio (m) usando para ello la clave privada del certificado elegido por el usuario. La firma digital se ha realizado sobre un número aleatorio pero podría haberse realizado sobre cualquier otro tipo de mensaje.
- Envío del certificado del usuario (C_A), envío del mensaje y envió de la firma digital.

Acciones realizadas por el servidor (SA):

- Recibe el certificado del usuario (C_A), el mensaje y la firma.
- Primero verificar que el certificado ha sido emitido por la CA, para ello se comprueba la firma del certificado usando la clave pública de la CA.
- Después, haciendo uso del número aleatorio recibido, se verifica la firma mediante la clave pública del certificado del usuario.
- Una vez verificados el certificado del usuario y la firma, se da acceso al usuario.

4.3.3. Eliminación de identidades digitales.

La última acción que permite la aplicación es la eliminación de los certificados solicitados.

Si el usuario pulsa sobre esta opción se encontrará con que tiene una lista de certificados entre los que puede elegir cual quiere eliminar.

Esta opción es la que ha presentado un desarrollo más sencillo de entre las tres que presenta la aplicación, pues no requiere una conexión con ningún servidor, sino que simplemente los certificados son eliminados tanto del directorio en el que se almacenan cuando se realiza el proceso de registro, como de la keystore del usuario *“ksusuario.bks”*.

4.4. ARQUITECTURA SOFTWARE DEL SISTEMA

En este apartado se pretende exponer como es la arquitectura del software que compone el sistema.

Siguiendo la estructura que se ha llevado a lo largo del capítulo, se mostrará cada una de las partes que componen la aplicación: registro, autenticación y eliminación de certificados.

En la parte de registro, se mostrará las clases y los métodos que componen la aplicación, así como los que componen los servidores de la CA y de SMS. También se relacionarán con el diagrama de secuencias de la *figura 4.2*.

A continuación se hará lo propio con la arquitectura del sistema de autenticación y con la arquitectura del sistema para eliminar los certificados.

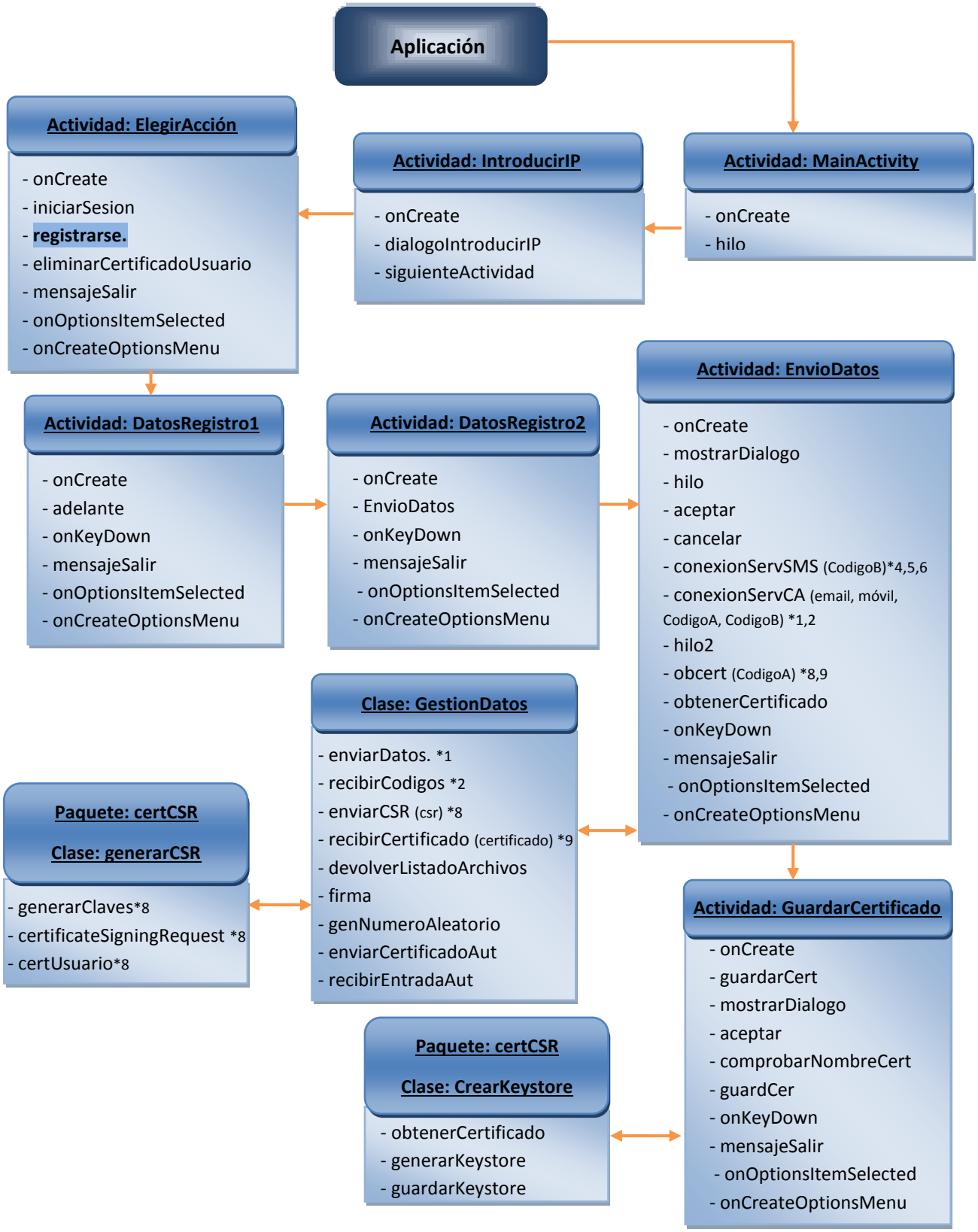
4.4.1. Arquitectura del sistema de registro

Se muestra a continuación la arquitectura para el sistema de registro.

En azul están las clases o actividades que componen la aplicación, indicando los métodos de cada una de las actividades, así como los tipos de datos relacionados con cada método y mostrando su relación con el diagrama de la *figura 4.2*.

En rojo las clases, métodos y tipos de datos principales relacionados con el servidor de SMS. En verde, están las clases y métodos del servidor de la CA.

4.4.1.1. Arquitectura de la aplicación para el registro



En el gráfico anterior se muestra la estructura de la aplicación para la parte de registro. Al lado de algunos métodos se muestran unos números que corresponden a la parte

en la que se usan en la *figura 4.3*. También entre paréntesis se muestran los atributos más importantes que son usados en relación a la *figura 4.3.*

Definición de las clases y métodos usados

Clase/Actividad MainActivity: Actividad principal de la aplicación, que muestra la portada de la aplicación. Consta de los siguientes métodos:

- *onCreate:* método que inicializa la actividad.
- *hilo:* método que permite ejecutar en un hilo aparte un tiempo la aplicación para que se muestra la portada durante un par de segundos.

Clase/Actividad IntroducirIP: actividad que solicita al usuario que está utilizando la aplicación la IP con la que funciona la aplicación. Contiene los siguientes métodos:

- *onCreate*: método que inicializa la cualquier actividad.
- *dialogoIntroducirIP*: método al que se llama desde *onCreate* y muestra un diálogo (mirar *anexo D*) en el que se le solicita introducir la IP a la que se quiere conectar.
- *siguienteActividad*: método al que se llama después de que el usuario haya introducido la IP y haya pulsado sobre *"Aceptar"*. Inicializa la siguiente actividad.

Clase/Actividad ElegirAccion: esta actividad es la que permite al usuario interactuar y elegir que acción realizar. Consta de los siguientes métodos:

- *onCreate*: método que inicializa la actividad.
- *iniciarSesion*: método que se activa al pulsar sobre el botón *"Iniciar sesión"* y pasa a la siguiente actividad. A partir de este método surge una serie de acciones encaminadas a realizar la autenticación del usuario.
- *registrarse*: método que se activa al pulsar sobre el botón *"REGISTRARSE"* y pasa a la siguientes actividades orientadas a realizar el proceso de registro de un usuario.
- *elimiarCertificadoUsuario*: método que se activa al pulsar el usuario sobre la opción *"Eliminar certificado"*. Cuando se pulsa el botón la aplicación manda al usuario a la siguiente actividad que le permite eliminar un certificado.
- *mensajeSalir*: es un diálogo que se muestra al usuario cuando elige en las opciones de menú salir de la aplicación. Si el usuario pulsa sobre la opción afirmativa, la aplicación se cierra. Si el usuario pulsa sobre la opción negativa, la aplicación no hace nada.
- *onOptionsItemSelected*: muestra las opciones que puede tener el menú para la actual actividad.
- *onCreateOptionsMenu*: inicializa el menú para mostrarlo por pantalla.

Clase/Actividad DatosRegistro1: esta actividad muestra al usuario una serie de recuadros que el usuario debe rellenar con sus datos. Esta actividad consta de los siguientes métodos:

- *onCreate*: método que inicializa la actividad.
- *adelante*: inicializa la siguiente actividad cuando el usuario pulsa sobre el botón que indica que se quiere continuar con el proceso.
- *onKeyDown*: reacciona al pulsar en el botón de ir hacia atrás del móvil y vuelve a la actividad anterior.
- mensajeSalir, onOptionsItemSelected y onCreateOptionsMenu realizan las mismas acciones que las que se han explicado en la Actividad anterior.

Clase/Actividad DatosRegistro2: Al igual que la clase anterior muestra al usuario una serie de casillas que deben ser rellenadas y un botón cuando ha terminado de introducir los datos. Consta de los siguientes métodos:

- *envioDatos*: obtiene los datos rellenados por el usuario y los pasa a la siguiente actividad.
- onCreate, onKeyDown, mensajeSalir, onOptionsItemSelected y onCreateOptionsMenu realizan las mismas acciones que en la actividad anterior.

Clase/Actividad EnvioDatos: esta clase es la encargada de conectarse con el servidor de la CA y con el servidor de SMS de la forma anteriormente comentada. Este método hace uso de la clase GestionDatos. Contiene los siguientes métodos:

- *mostrarDialogo*: muestra un diálogo de alerta que informa al usuario de que va a recibir un código que le dará acceso para poder recibir un certificado.
- *hilo*: método que ejecuta una actividad en otro hilo diferente al principal. Mientras en el método principal se muestra un diálogo de progreso (ver *anexo D*), en este hilo se ejecutará la conexión con el servidor de la CA.
- *aceptar*: método al que se le llama desde el método mostrarDialogo y permite la ejecución del hilo secundario que establece la conexión con el servidor de la CA.
- *cancelar*: método al que se le llama desde el método mostrarDialogo y que manda al usuario a la actividad ElegirAccion.
- *conexionServSMS*: método que establece la conexión con el servidor de SMS y manda y recibe datos de él.
- *conexionServCA*: método que establece la conexión con el servidor de la CA. Le envía el email y número de teléfono y recibe los códigos.
- *obcert*: envía el CSR al servidor de la CA y recibe el certificado del usuario.
- *obtenerCertificado*: muestra un diálogo de progreso mientras llama al método *obcert* para obtener el certificado y cierra la conexión.

- onCreate, onKeyDown, mensajeSalir, onOptionsItemSelected y onCreateOptionsMenu realizan las mismas acciones que en el resto de actividades.

Clase/Actividad GuardarCertificado: esta actividad solicita al usuario un nombre para un certificado que ha obtenido y guarda el certificado en memoria y en una Keystore. Contiene los siguientes métodos:

- *guardarCert*: método que guarda el certificado recibido en un archivo.
- *mostrarDialogo*: método que muestra diálogo en el que se le informa al usuario de que se ha creado correctamente el certificado recibido.
- *aceptar*: método que envía al usuario a la actividad ElegirAccion cuando el usuario pulsa sobre el botón *"Aceptar"* del diálogo.
- *comprobarNombreCert*: método que comprueba que no existe ningún certificado con el nombre especificado por el usuario.
- *guardCert*: método que llama al método *comprobarNombreCert* y si no existe el nombre llama al método *guardarCert* y también guarda el certificado en un keystore.
- El resto de métodos son iguales a los que se repiten en las actividades anteriores.

Clase GestionDatos: es una clase adicional de la que hacen uso muchas de las actividades. Se encarga de tareas más secundarias que permiten gestionar los datos que se reciben y se envían de los servidores. Hace uso del paquete certCSR. Contiene los siguientes métodos:

- *enviarDatos*: método que envía los datos (email y número de teléfono) al servidor de la CA.
- *recibirCodigos*: método que recibe los códigos A y B del servidor de la CA.
- *enviarCSR*: método que envía el CSR al servidor de la CA.
- *recibirCertificado*: método que se encarga de recibir el certificado obtenido.
- *devolverListadoArchivos*: método que devuelve una lista de archivos de un directorio si se le pasa la ruta del directorio.
- *firma*: método que realiza una firma digital y la devuelve al método que lo ha solicitado.
- *genNumeroAleatorio*: método que devuelve un número aleatorio.
- *enviarCertificadoAu*: envía el certificado que el usuario ha elegido al servidor que realiza la autenticación (ver apartado *4.4.2*).
- *recibirEntradaAut*: método que recibe la confirmación del servidor para que el usuario acceda o no (apartado *4.4.2*)

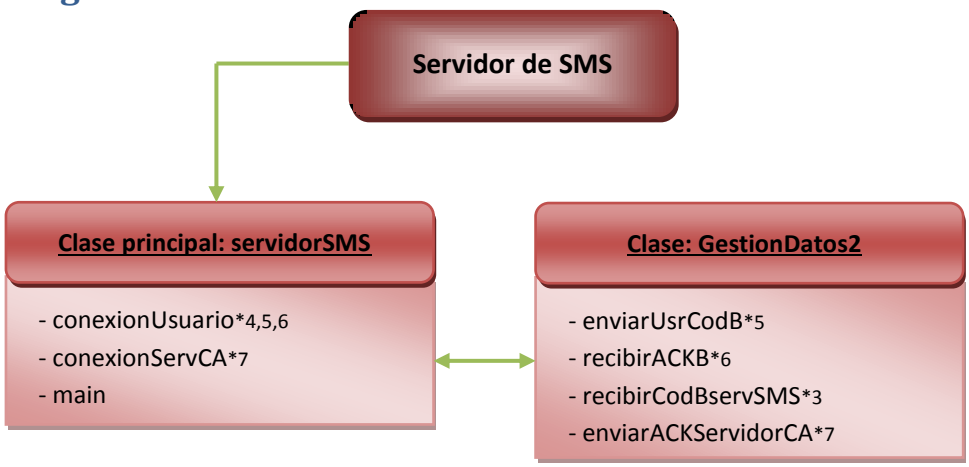
Paquete certCSR/Clase GenerarCSR: clase que permite la creación y generación de las claves privadas y públicas del usuario y el CSR que el usuario envía al servidor de la CA. Consta de los siguientes métodos:

- *generarClaves*: método que recibe un entero indicando el número de bits que van a tener las claves y va a devolver un par de claves en una variable de tipo KeyPair.
- *certificateSigningRequest*: método que devuelve un CSR.
- *certUsuario*: método que llama al método *certificateSigningRequest* y recibe un CSR que convierte a un String en formato PEM y lo devuelve al método que haya llamado a este.

Paquete certCSR/Clase CrearKeystore: clase que mediante el uso de una serie de métodos, crea un keystore.

- *obtenerCertificado*: lee el certificado del directorio en el que está almacenado y lo devuelve al método *generarKeystore*.
- *generarKeystore*: genera un keystore a partir del certificado del usuario y sus claves.
- *crearKeystore*: método que comprueba si existe el archivo en el que está almacenado el keystore y si no existe lo crea. Posteriormente llama al método *obtenerCertificado*.

4.4.1.2. Arquitectura del servidor de SMS para el registro.



Definición de las clases y métodos usados

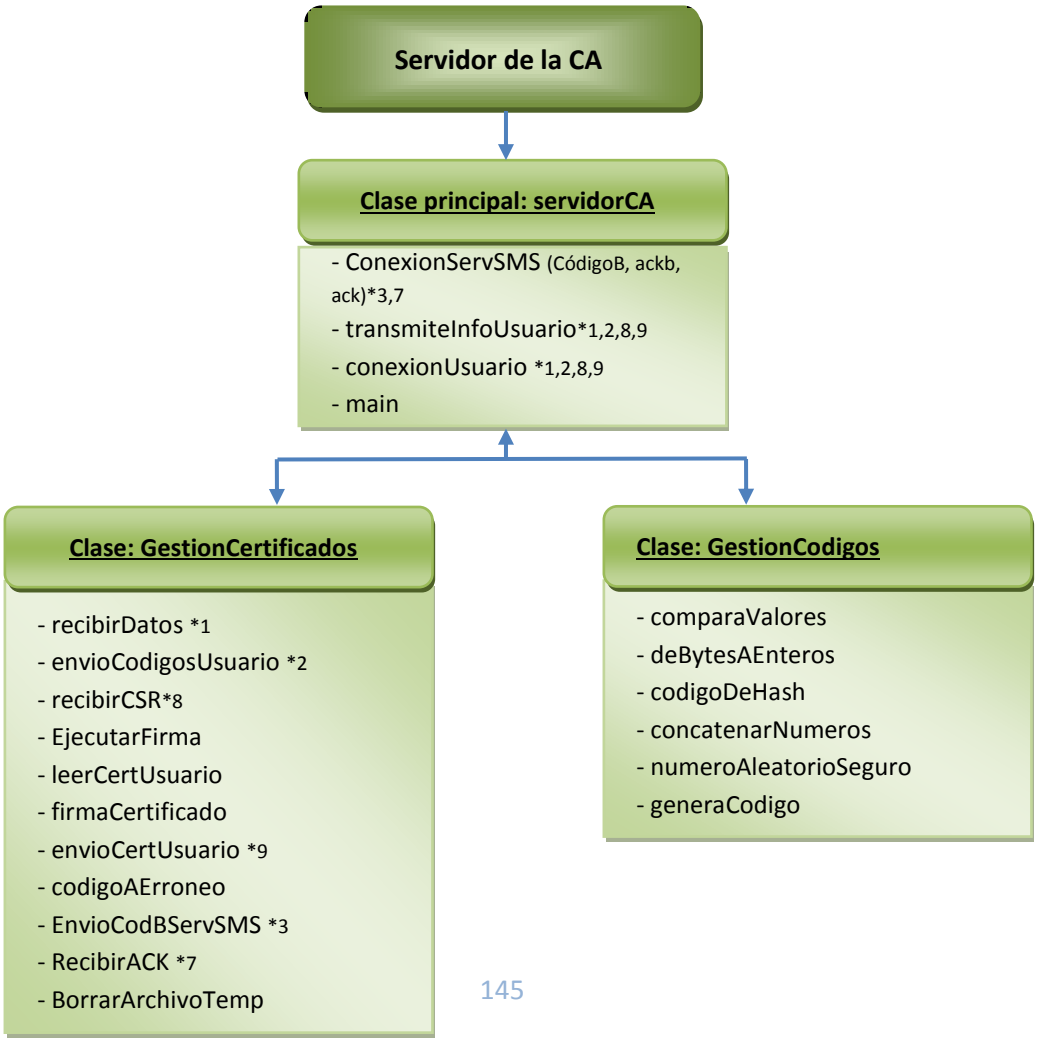
Clase ServidorSMS (Clase Principal): clase que realiza las acciones más importantes del servidor. Consta de los siguientes métodos:

- *conexionUsuario*: método que realiza la conexión del servidor con el usuario. No se realiza ni autenticación de cliente ni de servidor.
- *conexionServCA*: método que realiza la conexión entre los servidores. El servidor de la CA actua como cliente en esta ocasión y existen tanto autenticación de cliente como de servidor.
- *main*: método que llama a los otros dos métodos anteriores.

Clase GestionDatos2: clase que realiza las gestiones necesarios para realizar la implementación del protocolo que anteriormente explicado. Consta de los siguientes métodos:

- *enviarUsrCodB*: método que envía el código B (ticket SSL_B) al usuario.
- *recibirACKB*: método que recibe el ACK del usuario.
- *recibirCodBServSMS*: método que recibe el código B desde el servidor de la CA antes de enviárselo al usuario.
- *enviarACKServidorCA*: método que envía el ACK recibido del usuario, al servidor de la CA.

4.4.1.3. Arquitectura del servidor de la Autoridad Certificadora para el registro.



Definición de las clases y métodos usados

Clase servidorCA (Clase principal): Clase que permite la conexión con el usuario y con el servidor de SMS y realiza las funciones principales del servidor. Consta de los siguientes métodos:

- *ConexionServSMS*: método que permite la conexión con el servidor de SMS, existe autenticación tanto de cliente como de servidor y en este caso, el servidor de la CA actúa como cliente.
- *transmiteInfoUsuario*: método que llama a las funciones que permite la interacción del servidor con el usuario.
- *conexionUsuario*: método que permite la conexión con el usuario y llama al método *transmiteInfoUsuario.*

Clase GestionCertificados: Clase que permite gestionar las acciones que se van a realizar con los certificados de cara al usuario. Consta de los siguientes métodos:

- *recibirDatos*: método que recibe el email y el número de teléfono por parte del usuario.
- *envioCodigosUsuarios*: método que envía los códigos aleatorios generados al usuario.
- *recibirCSR*: método que recibe el CSR enviado por el usuario.
- *EjecutarFirma*: método que ejecuta un script que permite realizar la firma por parte de la CA del CSR recibido del usuario.
- *leerCertUsuario*: método que lee el archivo en el que está almacenado el certificado del usuario y devuelve un String que lo contiene.
- *firmaCertificado*: método que escribe el CSR del usuario en un archivo y llama al método *EjecutarFirma* para realizar la firma sobre el CSR.
- *envioCertUsuario*: método que envía el certificado firmado por la CA al usuario.
- *codigoAErroneo*: método que comprueba que el codigoA recibido por el usuario es igual al que se le envió.
- *envioCodBServSMS*: método que envía el Codigo B generado al servidor de SMS.
- *RecibirACK*: método que recibe el ACK desde el servidor de SMS.
- *BorrarArchivoTemp*: método que permite una vez que el certificado ya ha sido firmado y enviado al usuario, eliminarlo para desaprovechar recursos.

Clase GestionCodigos: Clase que realiza una serie de operaciones para crear y gestionar los códigos que se le van a enviar al usuario.

- *comparaValores*: método que compara si el código A recibido en el servidor es igual al código A que se envío previamente al usuario.

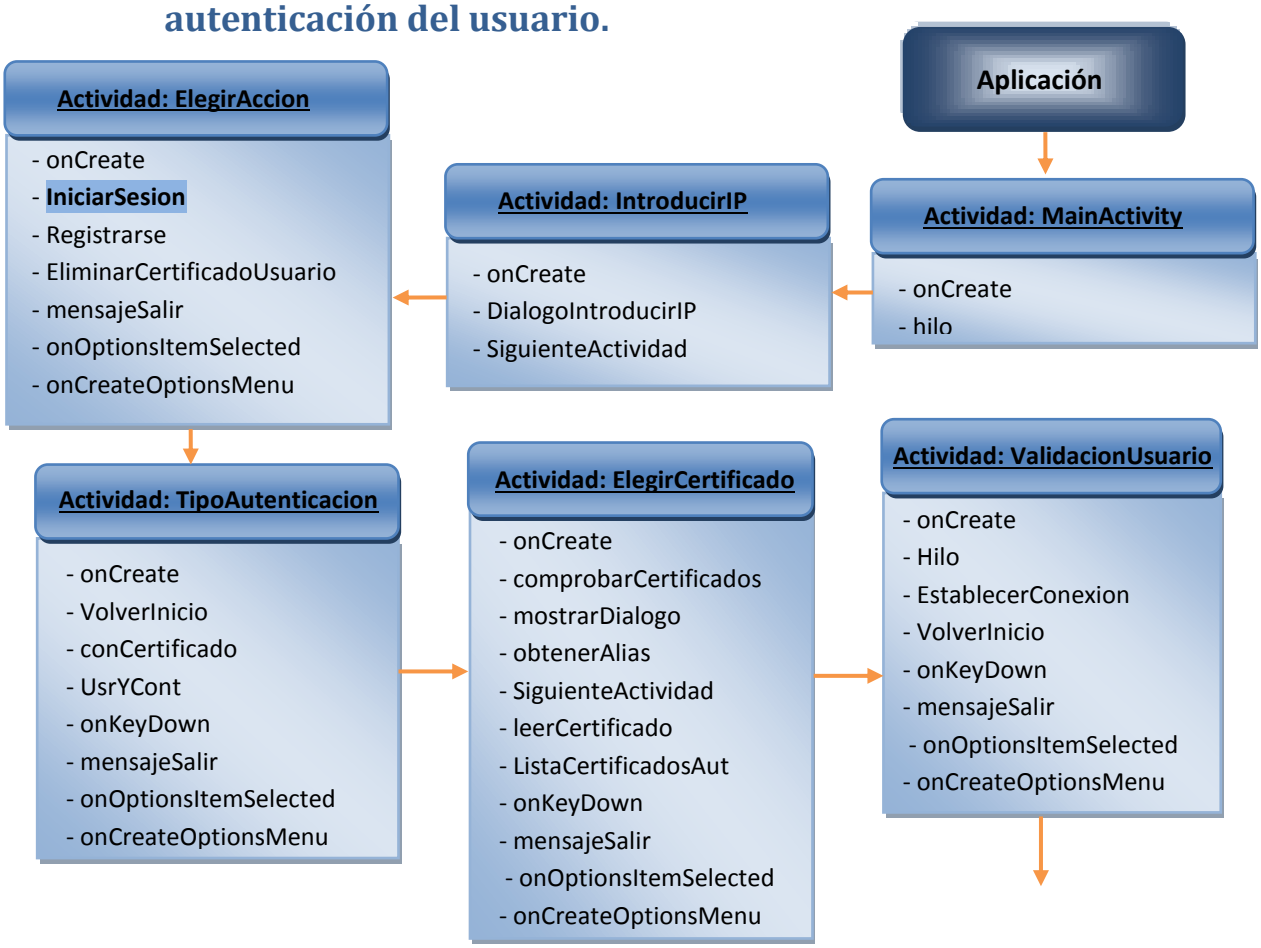
- *deBytesAEnteros*: método que convierte una cadena de Bytes a un número entero.
- *codigoDeHash*: método que realiza una hash a un número recibido.
- *concatenarNumeros*: método que concatena el número de teléfono enviado por el usuario a un número aleatorio generado.
- *numeroAleatorioSeguro*: método que genera un número aleatorio a partir del número recibido del usuario.
- *generaCodigo*: método que llama a una serie de métodos anteriormente mencionados para generar un número aleatorio.

4.4.2. Arquitectura del sistema para la autenticación

En este apartado se va a realizar un análisis de las clases y métodos utilizados tanto por la aplicación como por el servidor para realizar con éxito la autenticación de un usuario.

Como en el apartado anterior, se muestran en azul las clases y métodos de la aplicación y en morado la clase y métodos del servidor.

4.4.2.1. Arquitectura de la aplicación para la autenticación del usuario.



Definición de las clases y métodos usados

Las actividades MainActivity, IntroducirIP y ElegirAccion, se han definido en el apartado sobre la arquitectura de la aplicación para registro.

Clase/Actividad TipoAutenticacion: Clase que da al usuario la posibilidad de autenticarse usando un usuario y una contraseña o usando un certificado del usuario. Consta de los siguientes métodos:

- *VolverInicio*: método al que se le llama cuando el usuario ha introducido un usuario y contraseña y ha presionado sobre *"Aceptar"*, en ese momento la aplicación dirige al usuario a la actividad ElegirAccion, ya que en realidad la autenticación usando un usuario y una contraseña no es el objetivo de la aplicación.
- *conCertificado*: método que se activa cuando el usuario elige autenticarse usando un certificado, para lo cual pulsa sobre el botón dedicado a ello.
- *UsrYCont*: método que se activa cuando el usuario elige autenticarse usando un usuario y una contraseña. Muestra un diálogo indicando al usuario que debe introducir un usuario y una contraseña.
- onCreate, onKeyDown, mensajeSalir, onOptionsItemSelected y onCreateOptionsMenu se definen de igual forma que en las actividades dedicadas al registro.

Clase/Actividad ElegirCertificado: Clase que muestra una lista de certificados entre los que el usuario puede elegir cual quiere usar para autenticarse en el sistema. Consta de los siguientes métodos:

- *comprobarCertificados:* método que comprueba que existe una lista de certificados, si no existe vuelve a la actividad MainActivitiy2 e informa al usuario de que no tiene certificados almacenados.

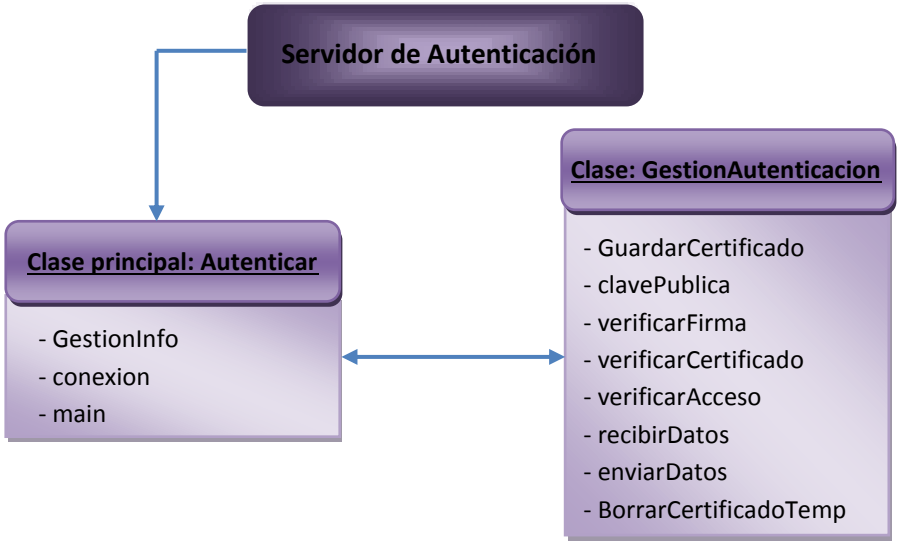
- *mostrarDialogo*: método que muestra un diálogo para informar al usuario de que tiene que elegir un certificado.
- *obtenerAlias*: método que obtiene a partir del nombre del certificado elegido por el usuario el alias con el que se ha almacenado en el keystore.
- *SiguienteActividad*: método que pasa a la siguiente actividad cuando es llamado.
- *leerCertificado*: método que lee el certificado elegido y lo almacena en un String.
- *listaCertificadosAut*: método que muestra al usuario una lista de certificados solicitados por el usuario.
- onCreate, onKeyDown, mensajeSalir, onOptionsItemSelected y onCreateOptionsMenu se definen de igual forma que en la actividad anterior.

Clase/Actividad ValidacionUsuario: Clase que realiza la conexión con el servidor que realiza la autenticación sobre el certificado enviado por el usuario. Contiene los siguientes métodos:

- *Hilo*: método que permite la conexión en un hilo diferente al principal mientras que en el hilo principal se muestra un dialogo de progreso.
- *EstablecerConexion*: método que establecer la conexión con el servidor que realiza la autenticación.
- *VolverInicio*: método que vuelve a la actividad ElegirAccion cuando es llamado.

Clase GestionDatos: en la parte que realiza la autenticación, también se hace uso de esta clase que se ha definido previamente en el apartado de registro.

4.4.2.2. Arquitectura del servidor que autentica al usuario.



Definición de las clases y métodos usados

Clase Autenticar (Clase principal): Clase que realiza la conexión con el usuario y recibe de él su certificado, la firma digital realizada con el certificado y un número aleatorio enviado por el mismo. Contiene los siguientes métodos:

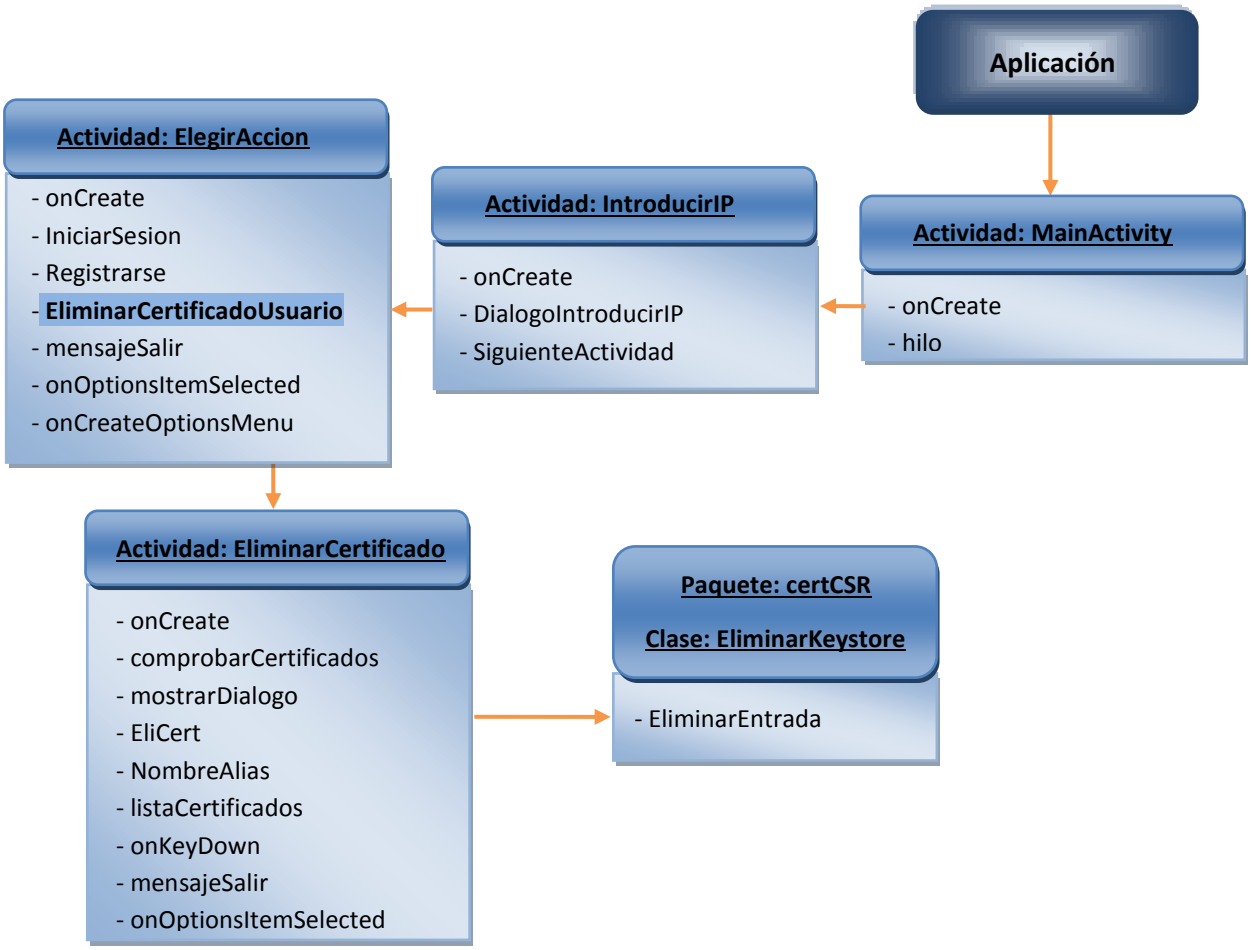
- *GestionInfo*: método que permite gestionar la información que se recibe y se envía al usuario.
- *conexión*: método que permite la conexión con el usuario. Solamente realiza autenticación de servidor.
- *main*: método principal que llama al método que establece la conexión.

Clase GestionAutenticacion: clase que se encarga de recibir la información del usuario, comprueba los datos recibidos y autentica al usuario. Contiene los siguientes métodos:

- *GuardarCertificado*: método que guarda el certificado recibido en un archivo temporal.
- *clavePública*: método que obtiene la cláve pública del certificado de la CA contenida en el truststore del servidor.
- *verificarFirma*: método que comprueba y verifica que la firma digital enviada por el usuario procede del mismo.
- *verificarCertificado*: método que verifica que el certificado enviado por el usuario está realmente firmado con la clave pública de la CA.
- *verificarAcceso*: método que llama a los anteriores y devuelve true si se ha confirmado tanto la firma, como el certificado.
- *recibirDatos*: método que recibe los datos enviados por el usuario, es decir, su certificado.
- *enviarDatos*: método que envía la confirmación de acceso del usuario si realmente se ha verificado su identidad.
- *BorrarCertificadoTemp*: método que borra el certificado previamente guardado, una vez que se ha verificado la identidad del usuario de forma que se ahorran recursos.

4.4.3. Arquitectura del sistema para la eliminación de certificados

Esta parte de la aplicación no interactúa con ningún servidor. Esta parte, simplemente muestra al usuario una lista de certificados entre los que el usuario debe elegir y una vez seleccionado, se elimina.



Definición de las clases y métodos usados

Las actividades MainActivity, IntroducirIP y ElegirAccion son las mismas actividades descritas en el apartado de registro.

Clase/Actividad EliminarCertificado: clase que permite una serie de tareas encaminadas a la eliminación de certificado seleccionado por el usuario. Consta de los siguientes métodos:

- *comprobarCertificados*: método que comprueba si el directorio donde se almacena los certificados está vacío o no. Si está vacío envía al usuario a la clase ElegirAccion.
- *mostrarDialogo*: muestra un diálogo al usuario indicándole que tiene que seleccionar un certificado para que este sea eliminado.
- *EliCert*: método que elimina el certificado de seleccionado por el usuario y vuelve a la actividad ElegirAccion, a la vez que informa al usuario de que el certificado ha sido eliminado.
- *NombreAlias*: método que obtiene el alias de un certificado a partir del nombre del certificado.

- *listaCertificados*: método que muestra al usuario una lista de certificados.
- onCreate, onKeyDown, mensajeSalir, onOptionsItemSelected y onCreateOptionsMenu se definen de igual forma que en el resto de actividades.

Paquete certCSR/Clase EliminarKeystore: clase que se usa para eliminar una entrada de la keystore. Sólo contiene un método:

- *EliminarEntrada*: método que elimina una entrada del keystore una vez que se le ha pasado el alias de la entrada que lo contiene.

Capítulo 5:

Resultados

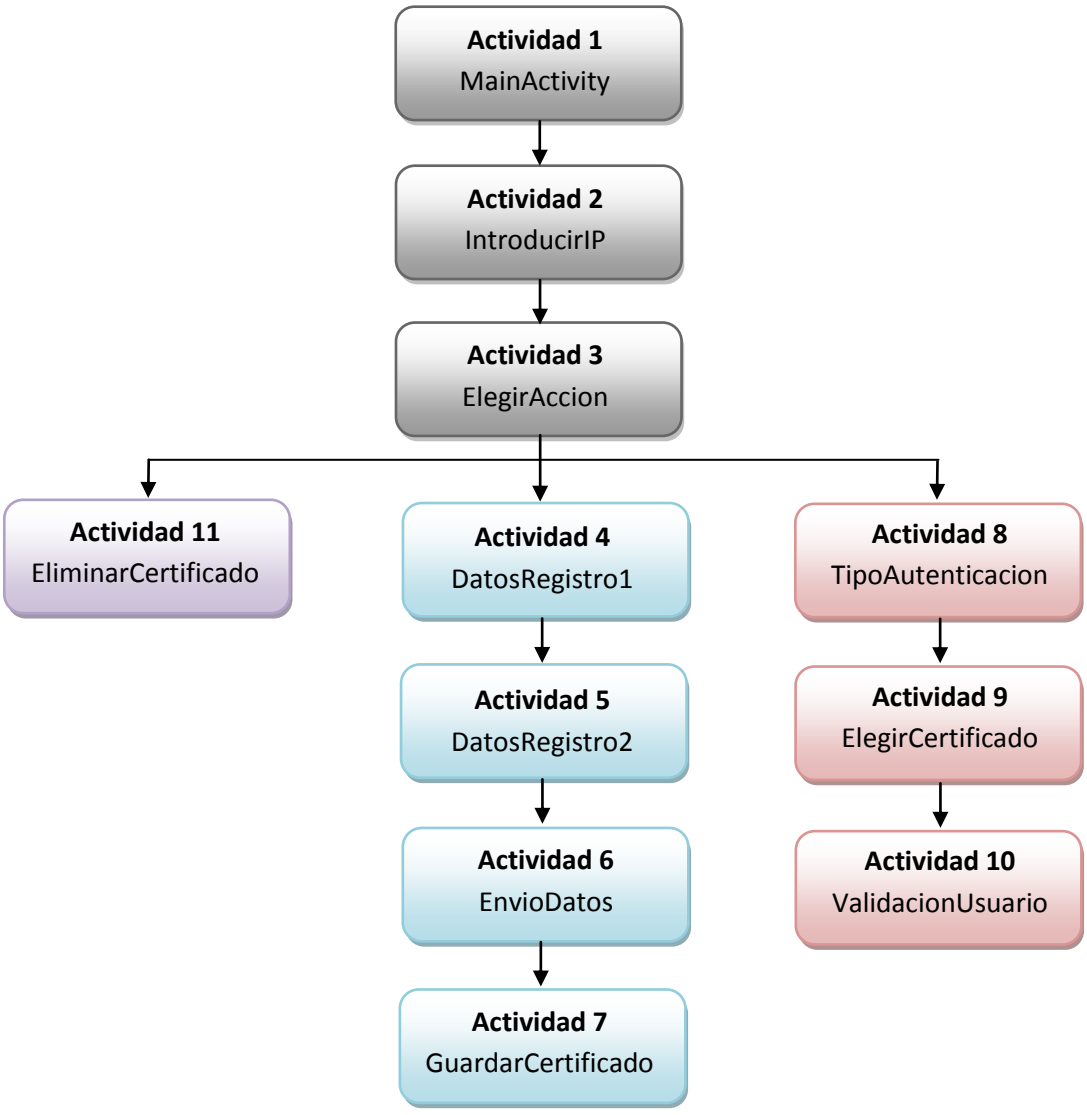
5. RESULTADOS

5.1. INTRODUCCIÓN

En este capítulo se muestra la aplicación resultante. Como en capítulos anteriores, se analiza primeramente la función principal de la aplicación que es el proceso de registro y en segundo lugar, las funciones que permiten al usuario gestionar y utilizar esos certificados para autenticarse o le permite eliminarlos si lo desea.

También se estudian los tiempos de respuesta de la aplicación y del protocolo de registro dependiendo de la acción realizada por el usuario y de otra serie de factores que se detallarán posteriormente.

5.2. DIAGRAMA RESUMEN DE LA APLICACIÓN FINAL



5.3. ESTUDIO DE LA APLICACIÓN RESULTANTE

Actividad 1: MainActivity

Esta primera actividad, simplemente muestra al usuario la portada con el icono de la aplicación. Esta pantalla se muestra dos segundos y posteriormente se pasa a la siguiente actividad:

Figura 5.1. Primera actividad con el icono de la aplicación. En esta actividad se muestra una pantalla con el icono de la aplicación como portada de la aplicación

Actividad 2: IntroducirIP

En esta actividad, se le muestra al usuario un mensaje para que introduzca la IP a la que se desea conectar. Esto permite que posteriormente la aplicación se conecte a los servidores para obtener un certificado digital y autenticarse en un sistema.

Esto se ha implementado así, debido a que los programas de los servidores no se han implementado en un equipo que disponga de una IP fija, por tanto, es más cómodo que la IP sea introducida pues puede modificarse cada vez que el usuario inicia la aplicación y no requiere establecer una IP fija en el equipo. De cara a un usuario y una implementación real del sistema, la IP que se introduce no es accesible al usuario y es transparente a él, por lo que esta actividad no sería necesaria. Es decir, se configuraría

internamente la aplicación para tener acceso a cada uno de los servidores, cada uno con su IP y su puerto, pero estos no serían conocidos por el usuario.

Esta segunda pantalla de la aplicación, correspondiente a la segunda actividad especificada en el esquema del apartado anterior, es la siguiente:

Figura 5.2. Segunda actividad: IntroducirIP. En esta pantalla se muestra un diálogo que informa al usuario de que debe introducir la IP a la que se desea conectar.

Actividad 3: ElegirAccion

La tercera actividad, muestra de cara al usuario, las acciones más importantes que puede llevar a cabo la aplicación. El usuario tiene tres opciones:

- Registrarse: el usuario, al pulsar sobre el botón inicia el proceso de registro que le permite obtener un certificado digital para acceder al sistema. Esta es la parte principal y más importante de la aplicación, ya que el resto de las acciones posibles no se pueden realizar sin haber obtenido un certificado en primer lugar.

- Iniciar sesión: al pulsar sobre esta opción, se le permite al usuario realizar una elección entre entrar al sistema con certificado o con un usuario y una contraseña, pero esta opción está inhabilitada pues no forma parte del objetivo de este proyecto. Si el usuario elige entrar con un certificado, este, mediante la elección de un certificado puede acceder a la plataforma digital.
- Eliminar certificado: permite al usuario eliminar certificados que ya no necesita o no le hagan falta, es decir, de alguna forma le permite gestionar los certificados.

Se muestra una imagen de esta tercera actividad:

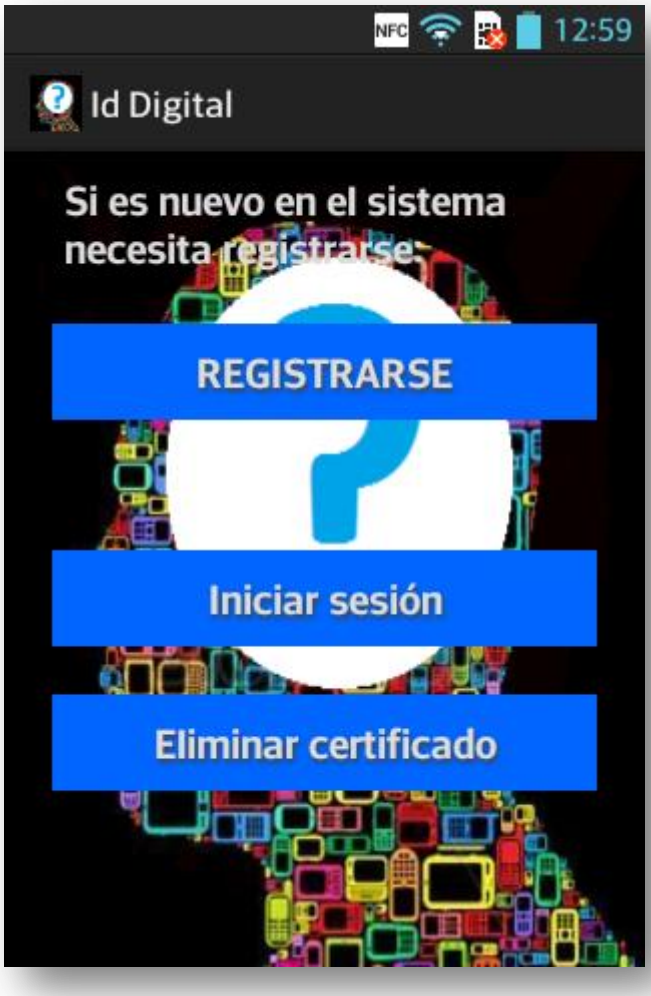


Figura 5.3. Tercera actividad: ElegirAccion. Esta actividad permite al usuario elegir que acción desea realizar. No es posible eliminar certificado o iniciar sesión, sin que previamente el usuario el usuario haya solicitado un certificado y lo tenga almacenado.

En esta actividad el usuario tiene dos opciones de menú:

- Volver a introducir IP si el usuario se ha equivocado al escribir la IP a la que se quiere conectar.

- Salir de la aplicación. También se puede salir de la aplicación pulsando sobre el botón atrás. 

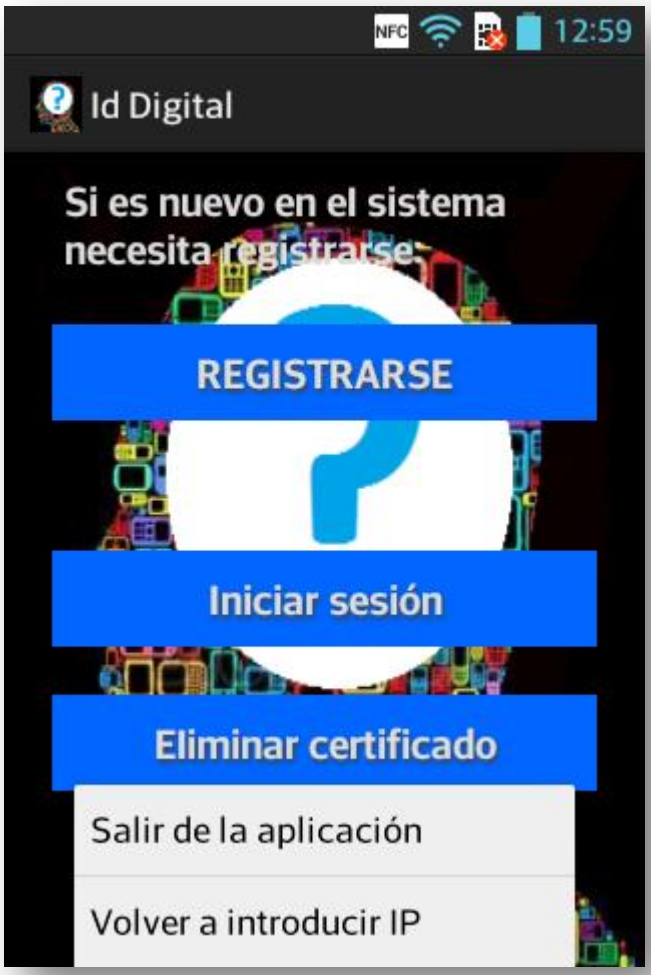


Figura 5.4. Opciones de menú de la tercera actividad. El usuario puede elegir salir de la aplicación o volver a introducir la IP correcta, si se ha confundido al introducirla la primera vez.

5.3.1. Registro

Al pulsar el botón *"REGISTRARSE"*, el usuario accede a una serie de actividades que permiten guardar sus datos personales para ser enviados a un servidor y después obtener un certificado digital que le acredite.

Actividad 4: DatosRegistro1

En esta actividad se le solicita al usuario algunos datos personales, los cuales serán incluidos en el CSR que se le envía al servidor para ser firmado por la CA.

En esta actividad se introducen los datos que serán añadidos al CSR, por tanto, está relacionado con la flecha número 8 de la *figura 4.3*.

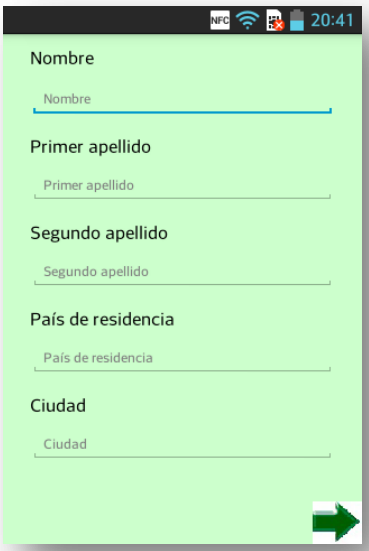


Figura 5.5. Actividad de registro. El usuario debe indicar su nombre, primer apellido, segundo apellido, país y ciudad en la que reside.

Antes de pasar a la siguiente actividad, la aplicación realiza una serie de comprobaciones:

1. Que el usuario no ha dejado todos los campos vacios.
2. Que no se haya dejado alguno de los apartados por separado vacios.

Esto se muestra al usuario con una notificación:

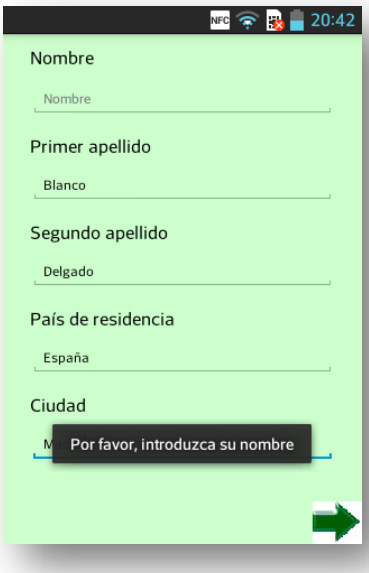


Figura 5.6. Ejemplo de notificación para el usuario. Si el usuario no ha introducido algún dato se le informa.

Las otras notificaciones que el usuario ve cuando deja algún dato sin meter son las siguientes:

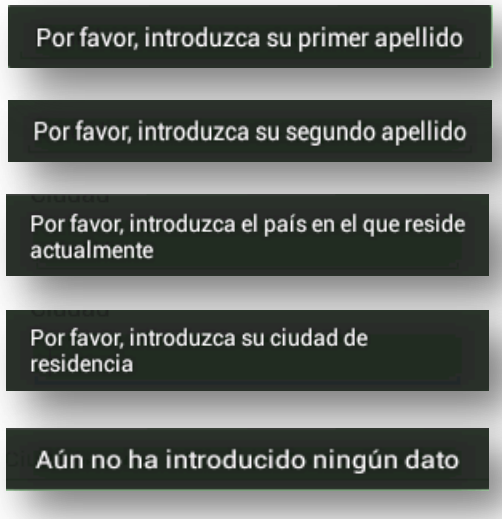


Figura 5.7. Notificaciones que se muestran al usuario en la actividad DatosRegistro1. Al usuario se informa si no ha introducido algún dato o si no ha introducido ninguno de ellos

Actividad 5: DatosRegistro2

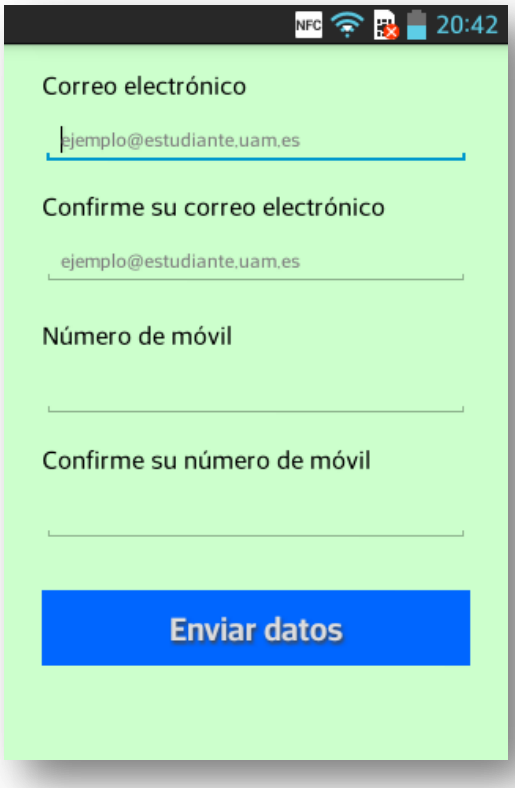


Figura 5.8. Segunda actividad de registro. Se le solicita al usuario su email y número de teléfono y la confirmación de ambos.

Esta actividad es la continuación del registro realizado en la primera actividad de registro. Se le pide al usuario su correo electrónico y su teléfono. Estos no se incluyen en el CSR, sino que serán enviados al servidor de la CA en el primer paso (flecha 1) de la *figura 4.3*.

Las comprobaciones que se realizan en esta actividad son las siguientes:

1. Que no están vacios todos los campos.
2. Que no se ha dejado alguno de los campos por separado vacios.
3. Que los correos electrónicos coinciden.
4. Que los teléfonos móviles coinciden.
5. Que el número de teléfono que se introduce tiene 9 dígitos como suelen tener los números de teléfono.

A continuación se presentan dos de las notificaciones que se muestran en ésta actividad y como vería un usuario otras dos de estas notificaciones:

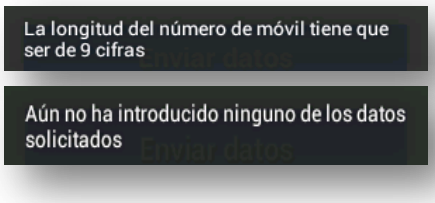
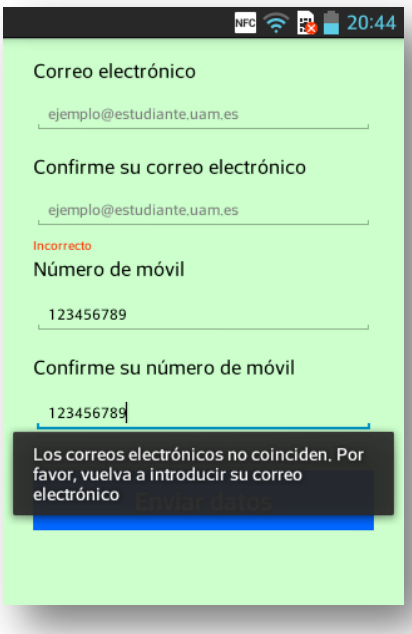
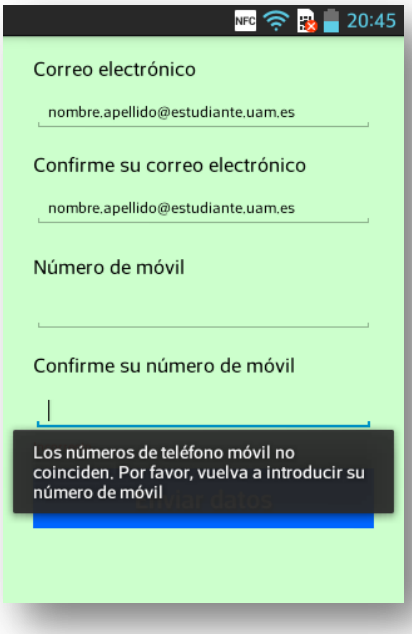


Figura 5.9. Dos de las cuatro notificaciones que se muestran al usuario en esta actividad. Se informa al usuario de que el número de teléfono notiene la longitud estándar o que no ha introducido ninguno de los datos





Figuras 5.10 y 5.11. Ejemplos de cómo el usuario ve las notificaciones. Si el usuario se ha equivocado al introducir el número de teléfono o el email, se le informa.

Además, tanto en la actividad anterior como en esta actual, el usuario tiene dos opciones de menú:

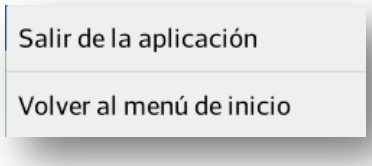


Figura 5.12. Opciones de menú para las actividades DatosRegistro1 y DatosRegistro2. El usuario puede elegir salir de la aplicación si no desea seguir realizando ninguna acción o volver al menú de inicio si quiere realizar otra acción.

Actividad 6: EnvioDatos

Esta quinta actividad se encarga de enviar la solicitud de un certificado al servidor.

Se realizan cuatro acciones:

1. Informar al usuario sobre la obtención de un código.
2. Obtención del código.
3. Solicitud del certificado.
4. Obtención del certificado.

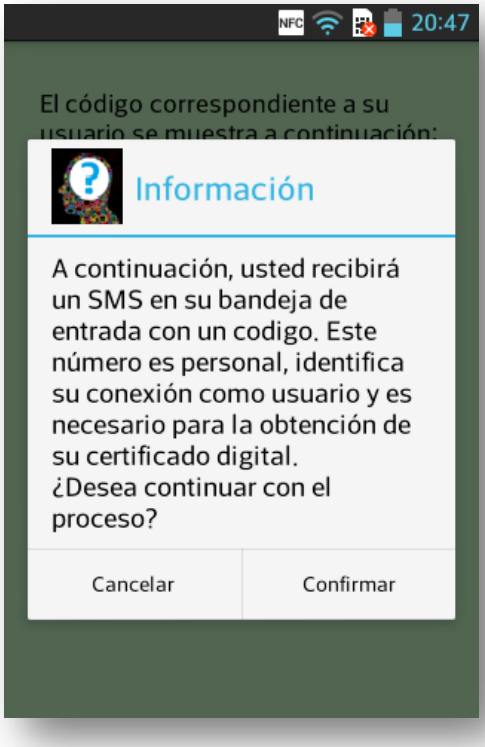


Figura 5.13. Dialogo de alerta o información. Se informa al usuario de que va a recibir un SMS en su bandeja de entrada que habilita la posibilidad de obtener un certificado.

La imagen muestra un dialogo de alerta o notificación que informa al usuario de que va a obtener un código a través de un mensaje de texto en su bandeja de entrada.

En realidad, este comportamiento simula una situación real, pues la aplicación desarrollada en este proyecto no obtiene ningún dato a partir de un mensaje de texto, ya que las comunicaciones se realizan a través de internet. Sin embargo, en una aplicación real, debería ser posible recibir el mensaje de texto mencionado.

Cuando el usuario pulsa sobre *"Cancelar"*, la aplicación entiende que el usuario no desea obtener un certificado digital y le envía a la actividad de inicio.

Cuando el usuario pulsa sobre *"Confirmar"* se envía al servidor el número de teléfono del usuario y la dirección de correo electrónico.

En este último caso, se establece la conexión con el servidor de la CA y el servidor de SMS. En particular, se siguen los pasos definidos por las flechas 1, 2, 3, 4 y 5 de la *figura 4.3*.

Estos pasos son transparentes al usuario y en la aplicación se muestra la siguiente imagen:

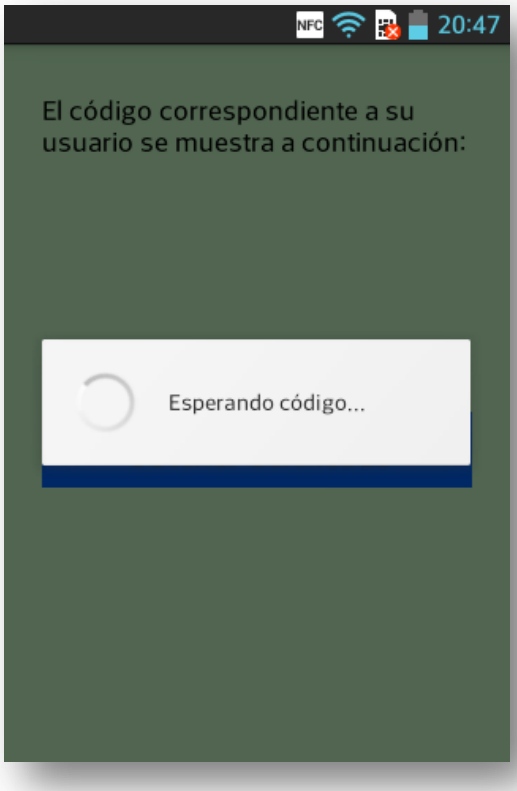


Figura 5.14. Dialogo de progreso. Estos diálogos se muestran al usuario cuando una acción requiere la espera por parte del usuario, en este caso, el usuario espera a recibir el código que le da acceso a poder obtener la identidad digital.

Como muestra la imagen anterior, mientras el usuario espera para recibir el código que le permita obtener un certificado digital, se muestra un dialogo de progreso que a la vez avisa al usuario de que está esperando el código. Una vez que el usuario recibe el código se muestra lo siguiente:

Figura 5.15. Obtención de código y solicitud de Certificado. Este código da acceso a un usuario a la obtención de un certificado.

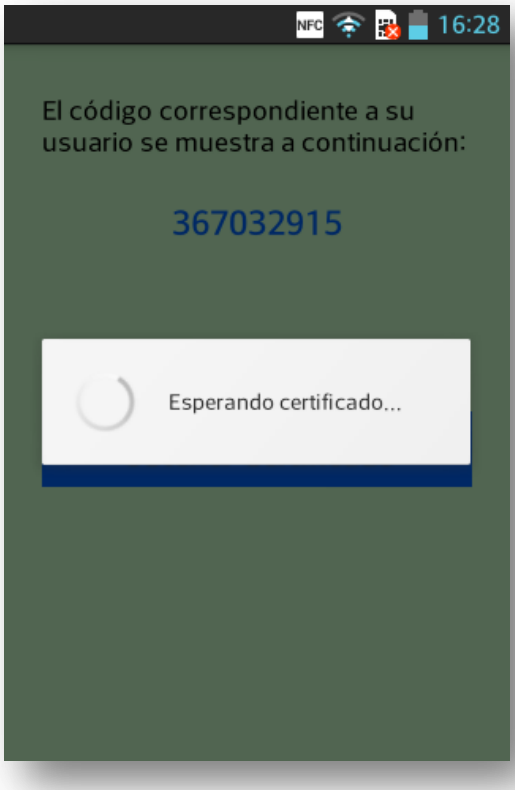


Figura 5.16. Dialogo de progreso para recibir un certificado. Se informa al usuario de que está en el proceso de recibir su certificado.

Lo que muestran las imágenes es que en esta parte de la actividad, el usuario ya ha recibido el código y para obtener el certificado deberá pulsar sobre el botón *"Obtener Certificado"*, de forma que el CSR será enviado al servidor.

Una vez que se ha pulsado el botón, se muestra al usuario otro dialogo de progreso que le informa de que va a obtener el certificado.

Esta parte de la aplicación, está relacionada con los pasos indicados por las flechas 6, 7, 8 y 9 de la *figura 4.13*.

En todo este proceso, una vez que se ha pulsado sobre *"Confirmar"* (*Figura 5.13*), si no se ha podido establecer conexión o se ha perdido, tanto en el proceso de envio de número de móvil y recepción de código (Figuras *5.14* y *5.15*), así como en el proceso de

envío de CSR y recepción de certificado (*Figura 5.16*), se informará al usuario de la siguiente forma:

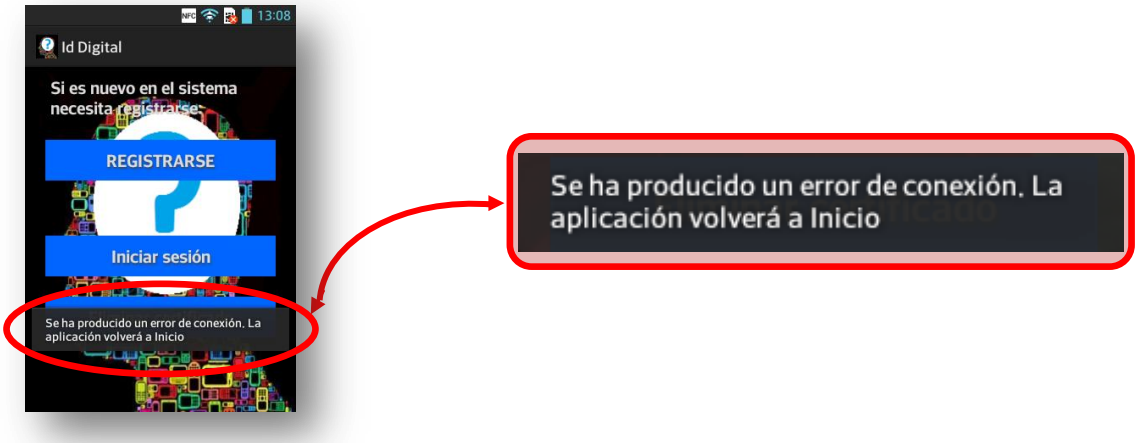


Figura 5.17. Notificación que informa al usuario si no se ha podido establecer una conexión. Si hay un problema en la conexión, se vuelve a la actividad que permite al usuario elegir una acción y se informa al usuario.

Actividad 7: GuardarCertificado

En esta actividad, se le solicita al usuario que introduzca un nombre con el que se guardará el certificado del usuario.

Además, se avisará al usuario de que se ha guardado correctamente su certificado con un dialogo de alerta.

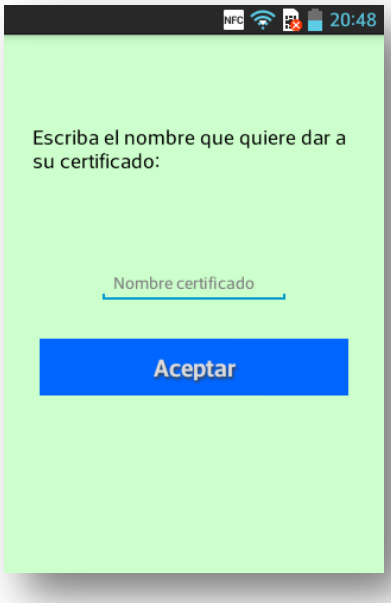


Figura 5.18. Actividad 7: GuardarCertificado. Se le solicita al usuario que introduzca un nombre para el certificado.

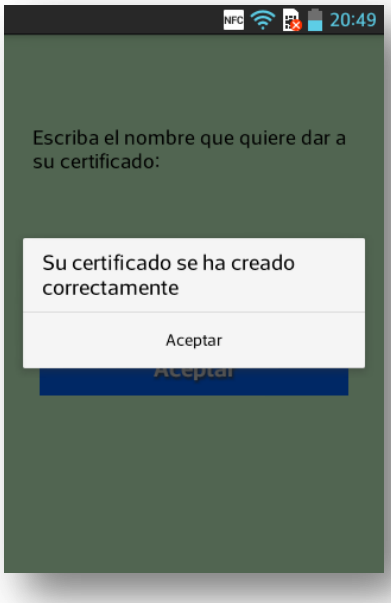


Figura 5.19. Actividad 7: GuardarCertificado. Se informa al usuario de que se ha creado correctamente su certificado.

También se comprueba si ya existe un certificado con ese nombre. Si existe se muestra al usuario un mensaje avisándolo, de forma que el usuario deberá introducir otro nombre.

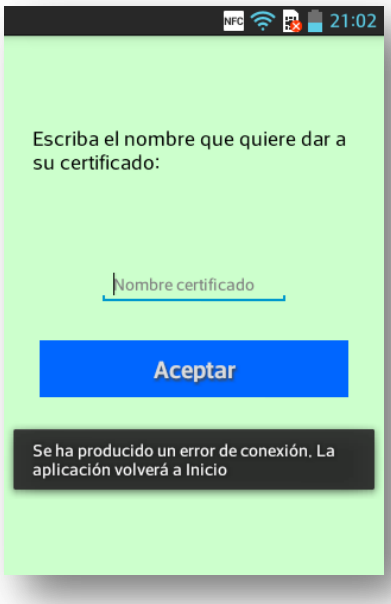


Figura 5.20. Notificación para informar al usuario de que ya existe un certificado con ese nombre. Dos o más certificados no pueden tener el mismo nombre

Las opciones de menú para esta actividad y la anterior son iguales. El usuario puede elegir salir de la aplicación sin guardar el certificado o volver al menú de inicio. En cualquiera de los dos casos, el usuario abandonaría la actividad sin guardar el certificado.

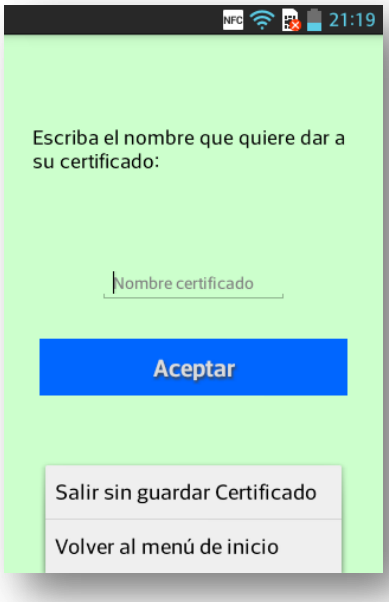


Figura 5.21. Opciones de menú que se muestra al usuario en esta actividad. El usuario puede salir sin guardar el certificado o volver al menú.

5.3.2. Iniciar sesión (Autenticación)

Al pulsar el botón *"Iniciar sesión"* el usuario accede a una actividad que le da la posibilidad de autenticarse usando un certificado o un usuario y una contraseña (esto último no hace nada, ya que no es el objetivo del proyecto y la aplicación vuelve a inicio).

Si el usuario elige autenticarse con un certificado, se le mostrará una lista de certificados y tras elegir un certificado, se establece una conexión. De forma transparente al usuario, la aplicación envía el certificado elegido, un *nonce* y una firma digital sobre tal *nonce*. El servidor verifica estos datos y da acceso al usuario.

Actividad 8: TipoAutenticacion

En esta actividad, se le da al usuario la opción de autenticarse en el sistema usando un certificado o mediante el uso de un usuario y una contraseña. La opción de autenticarse en el sistema usando un usuario y una contraseña no está operativa pues no es el objetivo de este proyecto. Se muestra a continuación una imagen de la actividad y una imagen del dialogo que se muestra al usuario si pide autenticarse usando un usuario y una contraseña:

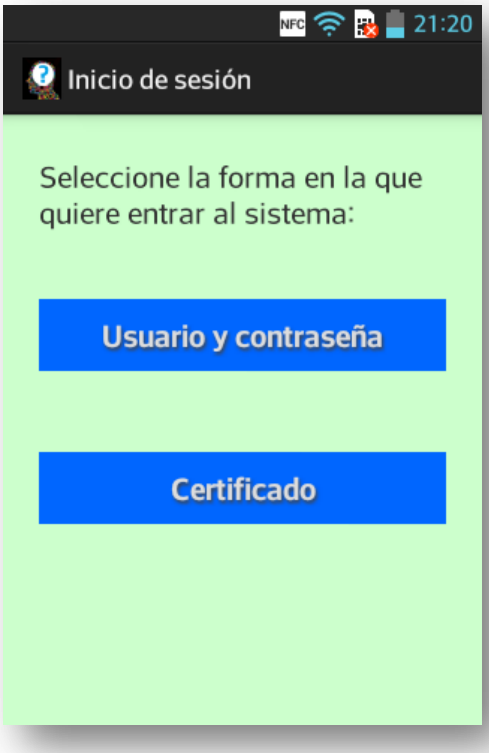


Figura 5.22. Actividad TipoAutenticacion. Da al usuario una opción para autenticarse.

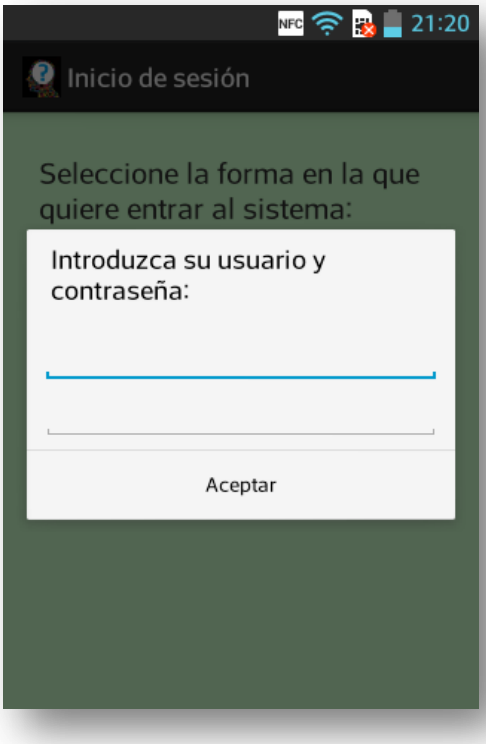


Figura 5.23. Dialogo que se muestra al usuario para que introduzca un usuario y una contraseña. Opción deshabilitada.

Aunque se muestra un dialogo para que el usuario introduzca sus datos, cuando el usuario pulse sobre *"Aceptar"*, la aplicación dirigirá al usuario al menú inicial, ya que esta opción no está operativa, pues no forma parte del objetivo de este proyecto.

Las opciones de menú para esta actividad son las mismas que para las actividades *DatosRegistro1* y *DatosRegistro2*, es decir, el usuario puede elegir salir de la aplicación o volver al menú inicial.

Actividad 9: ElegirCertificado

En esta actividad se muestra al usuario un diálogo que le informa de que para entrar al sistema necesita elegir un certificado. El diálogo que se muestra obliga al usuario a leerlo antes de continuar. Tras pulsar sobre el botón *"Aceptar"* el usuario ve una lista de los certificados que ha solicitado previamente, y al pulsar sobre uno de ellos se pasa a la siguiente actividad que permite el establecimiento de una conexión segura con el servidor.

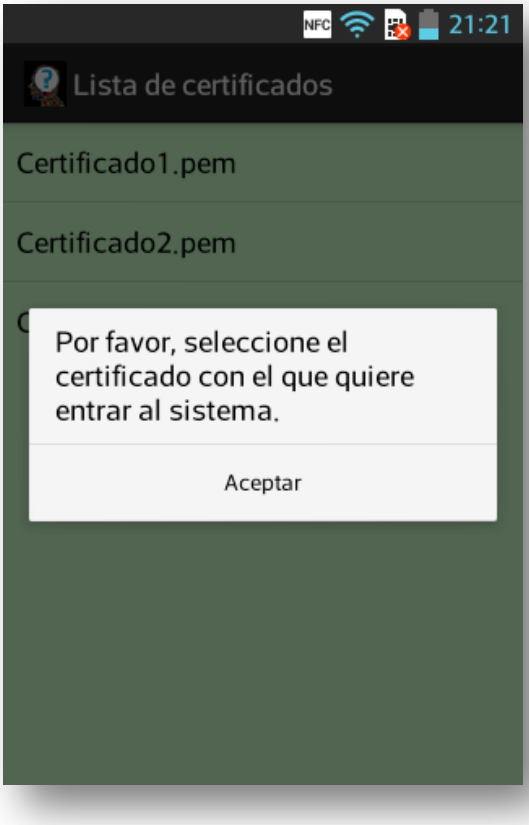


Figura 5.24. Notificación para informar al usuario de que debe elegir un certificado de la lista para poder entrar al sistema. Al tener un botón, está notificación obliga al usuario a leerla.

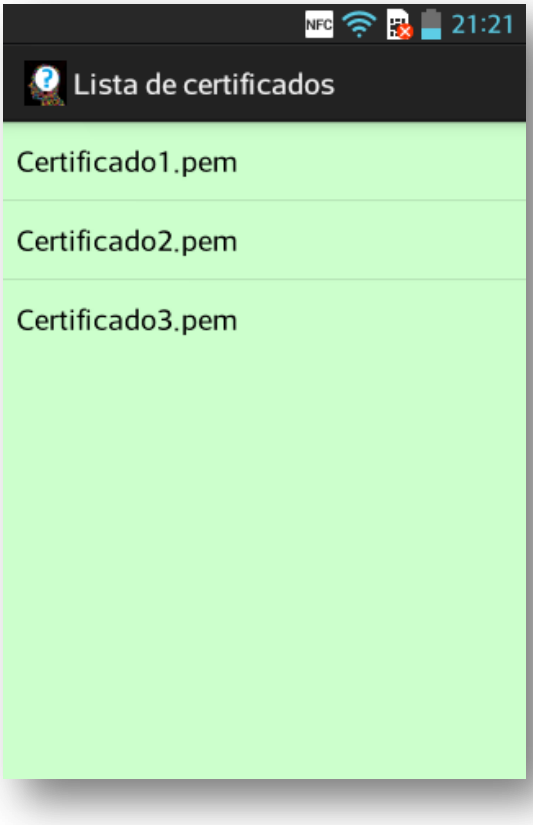


Figura 5.25. Lista de certificados que se muestran al usuario para que elija con cual quiere acceder al sistema. El usuario debe elegir con qué certificado quiere autenticarse.

Si no existe ningún certificado guardado, se informa al usuario de ello con una notificación mientras se vuelve a la actividad principal:

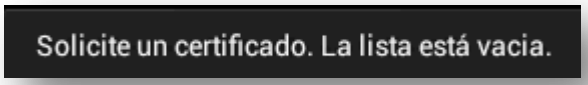


Figura 5.26. Notificación que avisa al usuario si no hay ningún certificado guardado. Si la lista está vacia, se redirige al usuario directamente al menú inicial.

En este caso existen más opciones de menú. El usuario además de poder elegir entre salir del sistema o volver al menú tiene la opción de volver a la actividad anterior y cambiar la forma de autenticarse en el sistema.

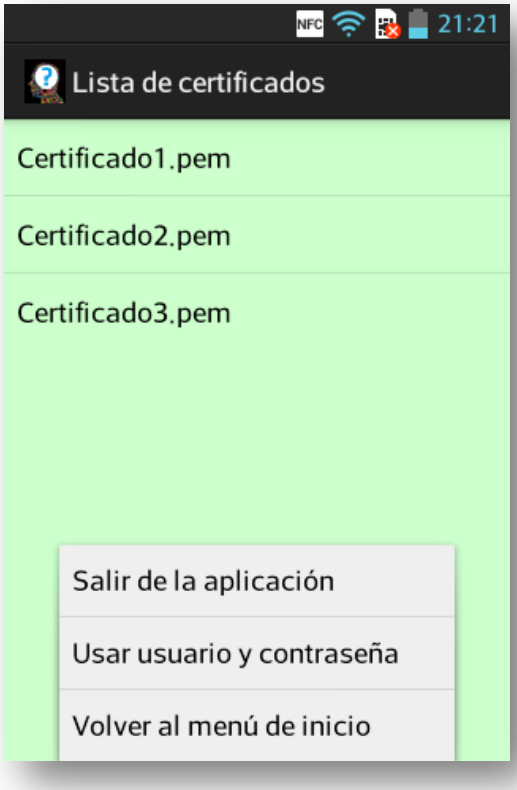


Figura 5.27. Opciones de menú para la actividad ValidacionUsuario1. El usuario puede salir de la aplicación, volver al menú de inicio o usar usuario y contraseña para autenticarse.

Actividad 10: ValidacionUsuario

Mientras se establece la conexión con el servidor, se comprueba la identidad del usuario mediante la verificación de la firma y del certificado y se recibe la confirmación

del acceso, se muestra al usuario un dialogo de progreso que informa de que se está comprobando sus datos:

Figura 5.28. Dialogo de progreso mientras se establece una conexión con el servidor. El usuario debe esperar mientras se confirma que tiene acceso al sistema.

Si se confirma que el certificado del usuario contiene el mismo correo electrónico que el que está almacenado en la base de datos, se le da acceso al usuario al sistema:

Figura 5.29. Acceso al sistema. Esta imagen muestra si el usuario tiene acceso al sistema, se le muestra el mensaje de bienvenida.

Estas dos imágenes anteriores están relacionadas con la *figura 4.7*. La *figura 5.28* establece relación con la flecha 1 y la *figura 5.29* con la flecha 2.

Si el usuario no tiene acceso al sistema se le muestra una notificación para comunicárselo y se vuelve directamente a la página de inicio:

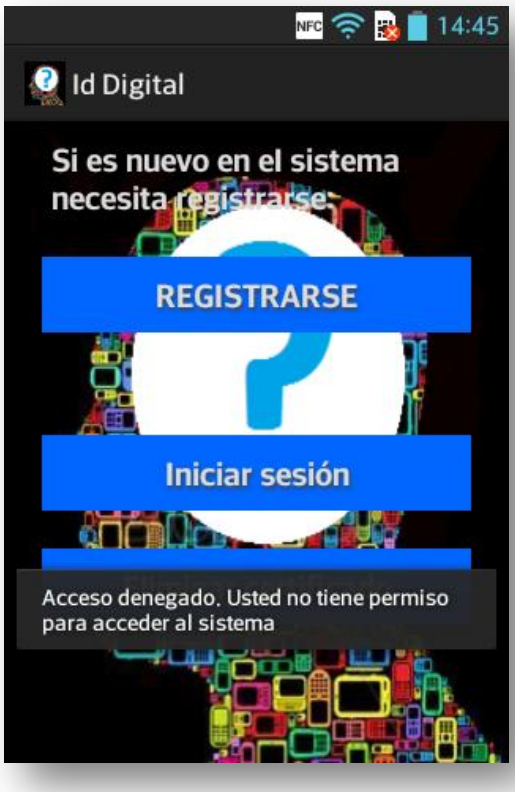


Figura 5.30. Notificación que se muestra al usuario si no tiene acceso. Si después de comprobar los datos, estos no se verifican, se deniega el acceso al usuario.

Al igual que en otros casos, las opciones de menú permiten al usuario elegir entre las opciones *"salir de la aplicación"* o *"volver al menú de inicio"*.

5.3.3. Eliminar certificado (Gestión de certificados)

Esta opción es la más sencilla de todas. Al pulsar sobre el botón *"Eliminar certificado"*, el usuario accede a una lista de certificados y al presionar sobre uno de ellos, este se eliminará tanto de la carpeta en la que está almacenado como de la keystore, de forma que si elimina todos los certificados, no existe forma de que el usuario tenga acceso al sistema a menos que solicitase un nuevo certificado.

Actividad 11: EliminarCertificado

En esta actividad se muestra una notificación que informa al usuario de que debe de elegir de entre una lista de certificados, el certificado que desea eliminar. Una vez que el usuario pulsa sobre uno de los certificados, este se elimina y se vuelve a la pantalla de inicio.

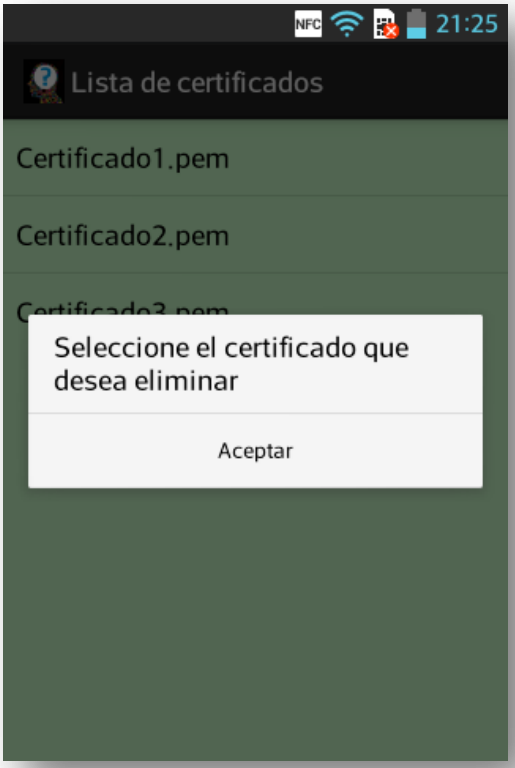


Figura 5.31. Notificación para informar al usuario de que debe elegir un certificado que quiera eliminar de la lista. El botón sirve para que el usuario tenga cuidad y no elimine un certificado por error.

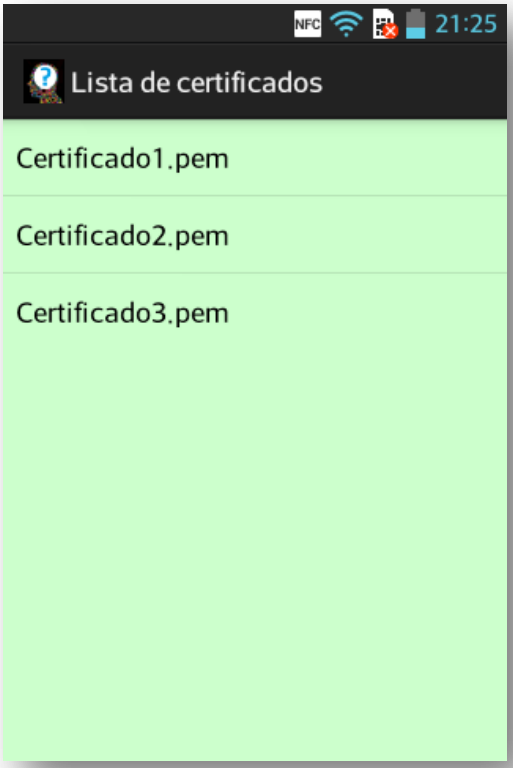


Figura 5.32. Lista de certificados que se muestran al usuario para que elija cual quiere eliminar. Al pulsar sobre un certificado, se elimina y se vuelve a inicio.

Si el certificado se elimina correctamente se informa al usuario y se vuelve a la actividad de inicio:

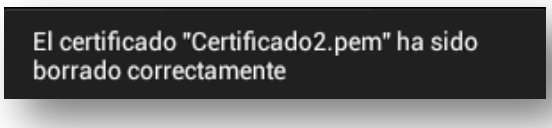


Figura 5.33. Notificación de certificado borrado correctamente. Se informa al usuario de que su certificado se ha eliminado y se vuelve al menú inicial.

Si cuando el usuario accede a la lista de certificados esta está vacía, se muestra una notificación al usuario:

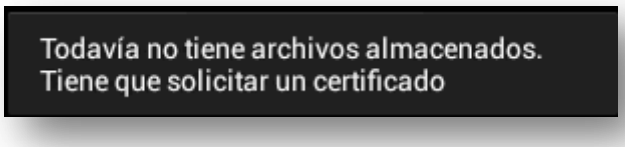


Figura 5.34. Notificación de que la lista está vacía. Si el usuario no tiene certificados almacenados, se le mostrará al usuario esta notificación.

Todas las actividades, como se ha visto en muchas de las imágenes anteriores, tienen una opción de menú para salir de la aplicación. Si un usuario pulsa sobre esta opción, se le mostrará un diálogo como el que sigue:

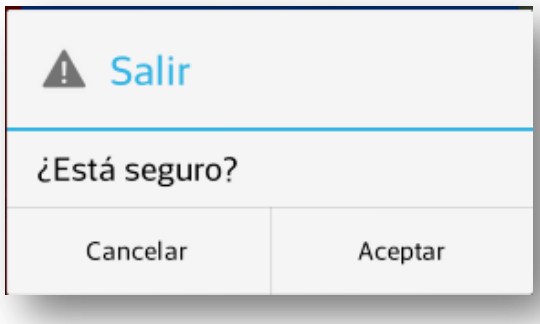


Figura 5.35. Diálogo para preguntar al usuario si desea salir de la aplicación. Si el usuario desea salir de la aplicación pulsará sobre *"Aceptar"* y saldrá, en cambio si pulsa sobre *"Cancelar"* no sucede nada.

Se ha realizado un video que muestra el resultado de la aplicación usando el emulador de Android y se puede consultar en la siguiente dirección de internet:

https://www.youtube.com/watch?v=27dYxJ6dRAE

5.4. RENDIMIENTO DEL SISTEMA

Para realizar las pruebas sobre la aplicación y cuánto tarda el sistema en responder, se ha usado un móvil de la marca LG Optimus L5 E610.

Las características más destacables de este móvil en relación a la memoria y la capacidad son:

- Procesador de 800 MHz de 32 bits y un único núcleo.
- SO Android 4.0.3 Ice Cream Sandwich actualizable a la versión 4.0.4.
- Memoria RAM de 512 MB.
- Memoria interna de 4 GB.
- Puede hacer uso de una memoria extraíble de 32 GB (pero no se ha empleado en la implementación de la aplicación ni en las pruebas).

Para establecer los servidores se ha usado un ordenador de sobremesa con el sistema operativo Linux, específicamente la versión Ubuntu 10.04.1 LTS. Este ordenador posee una memoria RAM de 2,5 GB aproximadamente.

La conectividad de los equipos se muestra en la siguiente imagen:

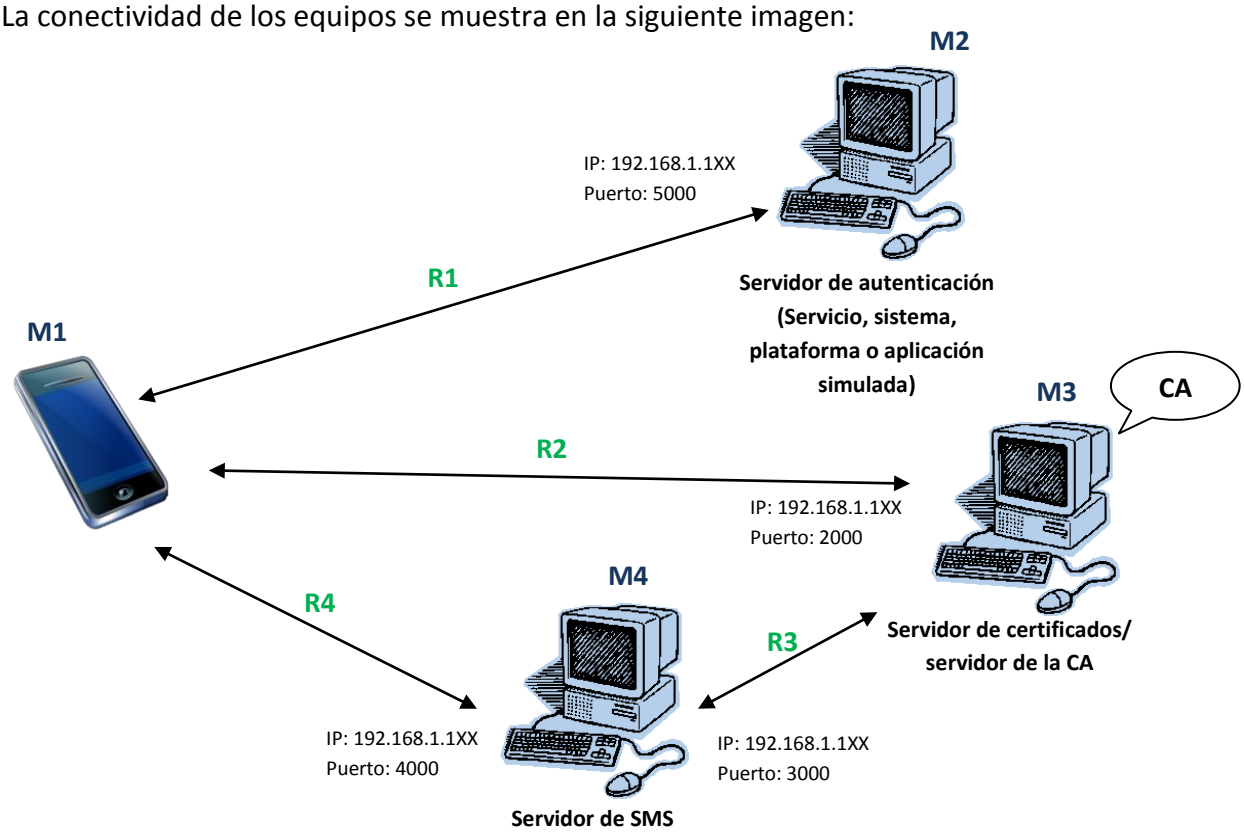


Figura 5.36. Conectividad de los diferentes elementos del proyecto. El esquema muestra como se conectan los elementos que componen el sistema. La IP usada es una IP local, cuyos dos últimos dígitos varían, los puertos varían según el servidor que se use. R1, R2, R3, R4: Redes 1, 2, 3 y 4; M1, M2, M3, M4: Máquinas 1, 2, 3 y 4.

Como se muestra en la imagen, a cada servidor se le asigna un puerto que se usará para establecer las conexiones. En realidad, aunque la imagen representa tres ordenadores, la implementación del sistema se ha llevado a cabo en un sólo ordenador, por lo que la IP en este caso, es la misma para cada conexión, siendo esta una IP local que varía cada vez que se enciende el ordenador, sobre todo si hay otros dispositivos conectados a la red. No obstante, al igual que muestra la imagen, el sistema podría funcionar perfectamente con servidores independientes situados en diferentes máquinas.

Para comprobar el rendimiento del sistema impelementado, se han realizado una serie de pruebas consistentes en realizar un conjunto de iteraciones tomando como muestra en cada iteración el tiempo que tarda en realizarse la acción:

A) El primer conjunto de pruebas ha consistido en la medida del tiempo medio que tarda la aplicación en realizarse, desde que el usuario envía el número de teléfono y el correo electrónico al servidor de la CA hasta que se recibe el certificado firmado y su desviación típica. Para ello se han realizado un conjunto de 100, 500 y 1000 iteraciones.

B) El segundo conjunto de pruebas ha consistido en introducir un tiempo aleatorio entre iteración e iteración para comprobar cuál es el comportamiento del sistema, es decir, si mejora o empeora con respecto a un conjunto de iteraciones seguidas. Al igual que en el caso anterior, se ha medido el tiempo medio y la desviación típica para 100, 500 y 1000 iteraciones.

C) En tercer lugar, se ha comprobado únicamente el tiempo que tarda la aplicación desde que se genera el CSR, hasta que se recibe el certificado. Tanto realizando 100, 500 y 1000 iteraciones seguidas, como introduciendo un tiempo aleatorio entre cada iteración.

A los valores obtenidos después de estas iteraciones se les ha calculado la media:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{N}$$

Y también se cálcula la desviación típica:

$$s = \left(\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2 \right)^2$$

5.4.1. Estadísticas sobre el tiempo de ejcución del protocolo de registro.

Para realizar estas pruebas, se ha medido todo el tiempo que tarda el sistema para realizar el protocolo de la *figura 4.3*. Para ello, se ha medido el tiempo inicial antes de establecer cualquier conexión y posteriormente el tiempo final, una vez que se ha obtenido el certificado. Esta operación sólo implica a la actividad "*EnvioDatos.java*" explicada anteriormente.

La instrucción que se usa para medir el tiempo inicial es el siguiente:

```
long tiempoInicio = System.currentTimeMillis();                    (1)
```

Para saber el tiempo total que tarda en ejecutarse, se resta el tiempo actual menos el tiempo inicial, usando la siguiente instrucción:

```
long totalTiempo = System.currentTimeMillis() - tiempoInicio;      (2)
```

Los tiempos, se miden en milisegundo, por lo que para saber el tiempo en segundos, se aplica la siguiente operación:

```
tiempo = totalTiempo/1000.000;                                     (3)
```

Entre la instrucción *1* y la instrucción *2*, se ejecuta el método que permite la realización del protocolo: *ConexionServCA();* (código de la aplicación en *Anexo F*).

La instrucción *1*, el método *ConexionServCA()* y las instrucciones *2* y *3*, se engloban en un bucle que realiza el número de iteraciones requeridos. Se debe requerir que se incluya el establecimiento de la conexión, es decir, medir desde antes de la flecha 1 en la *figura 4.3*, hasta después de que se cierre la última conexión, es decir, después de la flecha 9 en la *figura 4.3*.

En estas pruebas intervienen las máquinas M1, M3 y M4 y las redes R2, R3 y R4 de la *figura 5.36*. Al estar implementado el sistema en un mismo ordenador es posible que el tiempo usado por la red R2 sea menor que el tiempo real que se utilizaría.

A continuación, se muestran los resultados de las diferentes pruebas en este apartado.

Media y dispersión del tiempo total para 100 iteraciones medidas en **segundos**:

	Tamaño de las claves			
	1024	**2048**	**3072**	**4096**
Media	6.1401	14.3977	29.1074	64.1300
Desviación	4.5975	9.3188	14.9723	34.1106

Tabla 5.1. Tiempo medio y dispersión para la ejecución del protocolo, para 100 iteraciones y tamaño de claves: 1024, 2048, 3072 y 4096

Media y dispersión del tiempo total para 500 iteraciones medidas en **segundos**:

	Tamaño de las claves			
	1024	**2048**	**3072**	**4096**
Media	5.8276	13.3859	31.8114	65.9824
Desviación	3.2791	6.9773	17.0740	38.0009

Tabla 5.2. Tiempo medio y dispersión para la ejecución del protocolo, para 500 iteraciones y tamaño de claves: 1024, 2048, 3072 y 4096

Media y dispersión del tiempo total para 1000 iteraciones medidas en **segundos**:

	Tamaño de las claves			
	1024	**2048**	**3072**	**4096**
Media	5.7245	13.0730	32.5464	68.5815
Desviación	3.1367	6.8721	17.6298	39.8677

Tabla 5.3. Tiempo medio y dispersión para la ejecución del protocolo, para 1000 iteraciones y tamaño de claves: 1024, 2048, 3072 y 4096

Los resultados muestran que, como es normal, a mayor tamaño de clave, mayor es el tiempo medio de ejecución del protocolo.

Como se ha comentado, el tiempo medido incluye el tiempo implicado en las máquinas M1, M3 y M4 y las redes R2, R3 y R4. Pero también, se incluye en el tiempo medido, el tiempo que tarda el móvil en generar las claves privada y pública y el CSR.

La siguiente gráfica muestra los valores de tiempo medio para los diferentes tamaños de clave, según el número de iteraciones:

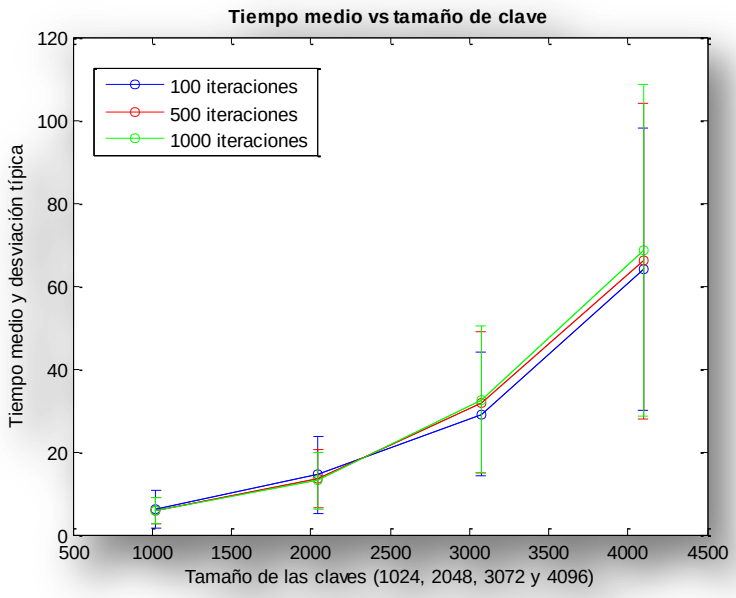


Figura 5.37. Ejecución del protocolo: comparación de los tiempos medios de ejecución del protocolo para los distintos tamaños de claves. En esta imagen cada número de iteraciones viene representada por una línea y representa los tiempos medios y la desviación típica según el tamaño de clave

La imagen muestra un resultado esperado: a mayor tamaño de clave, mayor trabajo computacional de la aplicación y por tanto, mayor tiempo medio.

También el error es mayor para un tamaño mayor de claves, lo que quiere decir, que podría darse el caso de obtener una clave en muy poco tiempo y tardar mucho más en la siguiente vez que se intente.

En la siguiente gráfica se comparan los tiempos medios para los distintos tamaños de las claves generadas. Cada una de las líneas representa el tiempo medio para los distintos tamaños de claves:

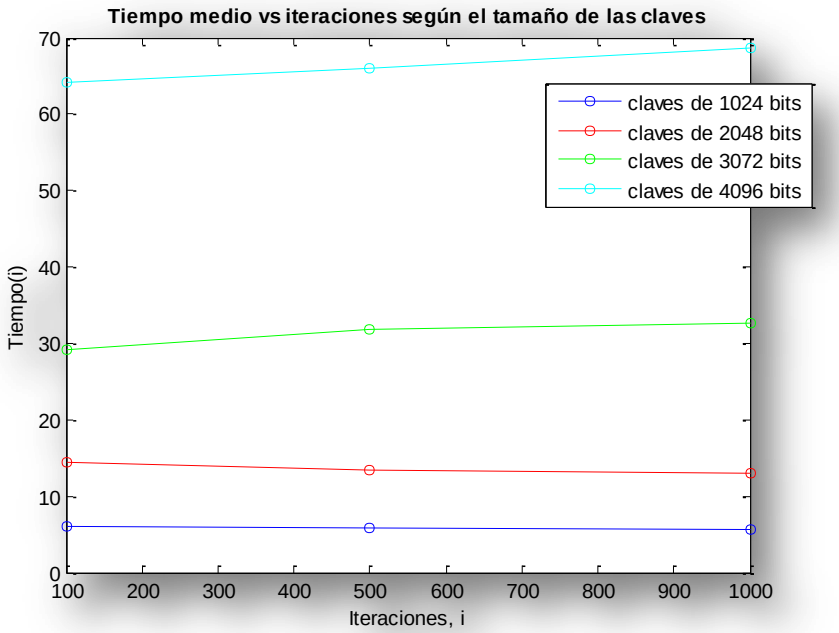


Figura 5.38. Ejecución del protocolo: comparación de los tiempos medios para las distintas iteraciones. Cada línea representa un tamaño de clave, el eje de abcisas representa el número de iteraciones y el eje de ordenadas los tiempos medios.

Como se observa en la figura, a mayor tamaño de clave, mayor es el tiempo medio en realizar el protocolo, pero el tiempo se mantiene prácticamente constante (la variación es mínima entre el tiempo inicial y final). Con el móvil usado para realizar las pruebas, el tiempo medio para la ejecución del protocolo para un tamaño de clave de 4096 bits es un tiempo poco práctico a la hora de que el usuario solicite un certificado, ya que es demasiado elevado y este podría pensar que la aplicación no funciona. Es decir, la experiencia de usuario no sería muy buena si el usuario tiene que experar más de 60 segundos para obtener un certificado.

En el caso de de un tamaño de 3072 bits, la experiencia de usuario sería mejor, pues tarda la mitad de tiempo que para un tamaño de 4096 bits, sin embargo podría ser aún demasiado elevada.

Por tanto, dependiendo del móvil usado, de su calidad y capacidad habría que informar al usuario de que posiblemente la aplicación tarda mayor tiempo en obtener un certificado.

Se propone por tanto para este móvil, como una relación adecuada entre el tiempo de ejecución de la aplicación y el tamaño de claves, el caso de 2048 bits, ya que este, todavía se considera un tamaño adecuado para que no se rompa la seguridad del sistema. Sin embargo, dentro de unos años la seguridad para este tamaño de clave puede ser fácilmente rompible, a no ser que se mejore también la capacidad de los dispositivos móviles, tal y como se espera, aunque no se sabe si podrá ser al mismo ritmo que la capacidad de romper la seguridad para una clave de 2048 bits. Para mayor información respecto al tamaño de las claves, es recomendable visitar la página http://www.keylength.com/ .

Para ver con más detalle como el error (desviación típica) afecta a cada una de de las muestras según el tamaño de clave, se va a mostrar a continuación la representación del tiempo medio y desviación para cada uno de los tamaños de clave:

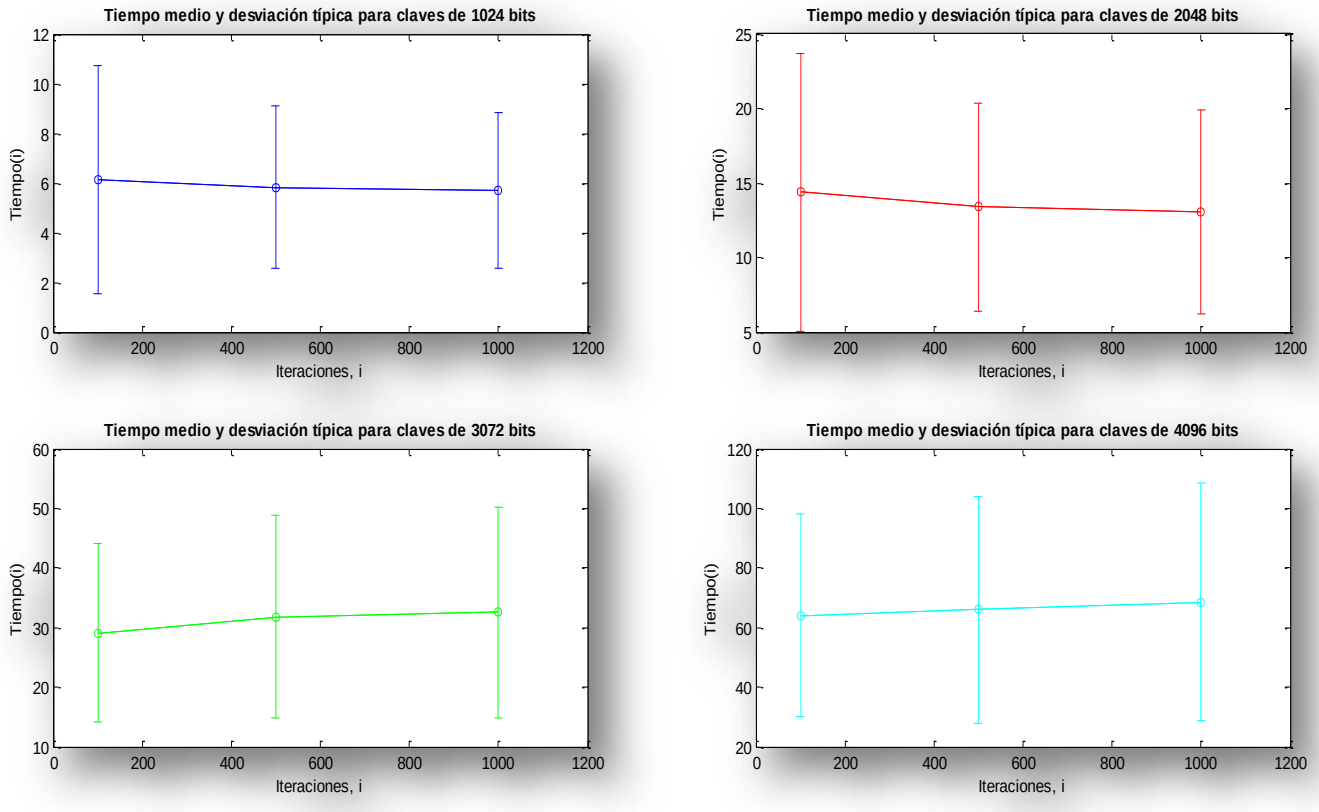


Figura 5.39. Ejecución del protocolo: representación del tiempo medio y desviación típica para los distintos tamaños de claves. La primera gráfica (azul) representa el tiempo medio y desviación típica para un tamaño de 1024 bits; la segunda (roja) los resultados para un tamaño de 2048 bits; la tercera (verde) los resultados para un tamaño de 3072 bits; la última (cían) resultados para un tamaño de 4096 bits

En la imagen anterior, al igual que se ha comentado anteriormente, se observa que a mayor número de iteraciones menor es el tiempo medio y el error en las dos primeras gráficas.

Con tiempo aleatorio entre muestra

Para realizar esta medida, se ha generado un tiempo aleatorio que detiene la aplicación brevemente antes de la introducción de las instrucciones *1*.

Para la generación de un tiempo aleatorio se ha creado un método que devuelve un número (double):

```
private double tiempoAleatorio()
{
        double aleatorio;
        int DESDE =T_MIN;
        int HASTA = T_MAX;
        aleatorio=(Math.random()*(HASTA-DESDE+1)+DESDE);
        return aleatorio;
}
```

T_MIN siempre es 0 y *T_MAX* es el doble del valor medio (redondeado hacia arriba) obtenido en los resultados anteriores para 1000 iteraciones, es decir:

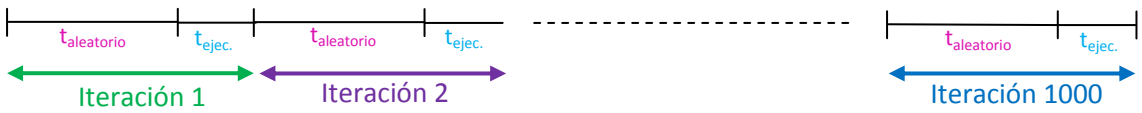
Tamaño de clave (bits)	1024	2048	3072	4096
T_MAX	12	28	66	138

Para que la aplicación pare brevemente en cada iteración antes de realizar el protocolo, se han introducido las siguientes instrucciones:

```
tiemposleep=(int)(tiempoAl*1000);
Thread.sleep((long)tiemposleep);
```

Donde *tiempoAl* es el valor devuelto en el método *tiempoAleatorio()*. Se multiplica por 1000 porque en estas instrucciones el tiempo se mide en *milisegundos*.

De esta forma, la distribución de tiempos se realiza de la siguiente manera:



Media y dispersión del tiempo total para 100 iteraciones medidas en **segundos**:

	Tamaño de las claves			
	1024	**2048**	**3072**	**4096**
Media	8.8607	12.0800	32.1689	68.9593
Desviación	6.0108	6.3070	17.1029	36.8560

Tabla 5.4. Tiempo medio y dispersión para la ejecución del protocolo, para 100 iteraciones, tiempo aleatorio entre muestras y tamaño de claves: 1024, 2048, 3072 y 4096

Media y dispersión del tiempo total para 500 iteraciones medidas en **segundos**:

	Tamaño de las claves			
	1024	**2048**	**3072**	**4096**
Media	7.6344	13.2076	33.0127	70.4872
Desviación	4.5209	6.3943	17.2249	37.6150

Tabla 5.5. Tiempo medio y dispersión para la ejecución del protocolo, para 500 iteraciones, tiempo aleatorio entre muestras y tamaño de claves: 1024, 2048, 3072 y 4096

Media y dispersión del tiempo total para 1000 iteraciones medidas en **segundos**:

	Tamaño de las claves			
	1024	**2048**	**3072**	**4096**
Media	7.7193	13.6963	32.9017	71.0582
Desviación	4.7955	6.2847	17.2006	38.0108

Tabla 5.6. Tiempo medio y dispersión para la ejecución del protocolo, para 1000 iteraciones, tiempo aleatorio entre muestras y tamaño de claves: 1024, 2048, 3072 y 4096

Como se ve en las tablas, como es natural, al aumentar el tamaño en bits de las claves, también aumenta el tiempo medio para la obtención del certificado.

Al incluir un tiempo aleatorio entre las muestras, se puede observar que los tiempos medios en general son algo peores que para el caso en el que no se mete tiempo medio entre muestras. Sin embargo, para un tamaño de clave de 2048 bits, el tiempo medio y la desviación mejoran levemente, aunque para el resto de tamaños de claves empeora.

La representación de los valores de las tablas anterior se muestra a continuación:

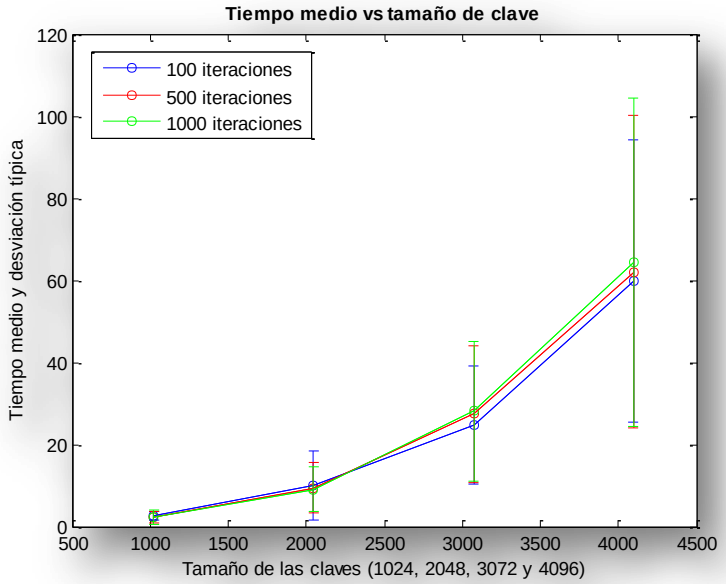


Figura 5.40. Ejecución del protocolo: comparación de los tiempos medios de ejecución del protocolo para los distintos tamaños de claves introduciendo un tiempo aleatorio entre iteración. En esta imagen cada número de iteraciones viene representada por una línea y representa los tiempos medios y la desviación típica según el tamaño de clave

La gráfica anterior representa el tiempo medio según el tamaño de las claves.

Para comprar los tiempos medios de cada uno de los tamaños de clave según el número de iteraciones, se ha realizado la siguiente gráfica:

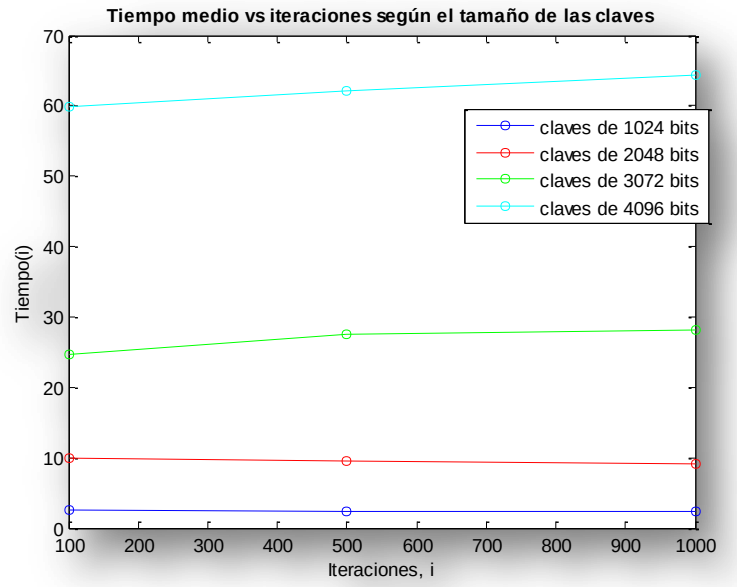


Figura 5.41. Ejecución del protocolo: comparación de los tiempos medios para las distintas iteraciones introduciendo un tiempo aleatorio entre iteración. Cada línea representa un tamaño de clave, el eje de abcisas representa el número de iteraciones y el eje de ordenadas los tiempos medios.

Para ver con detalle la desviación de las muestras se representa una gráfica con cada una de las medias:

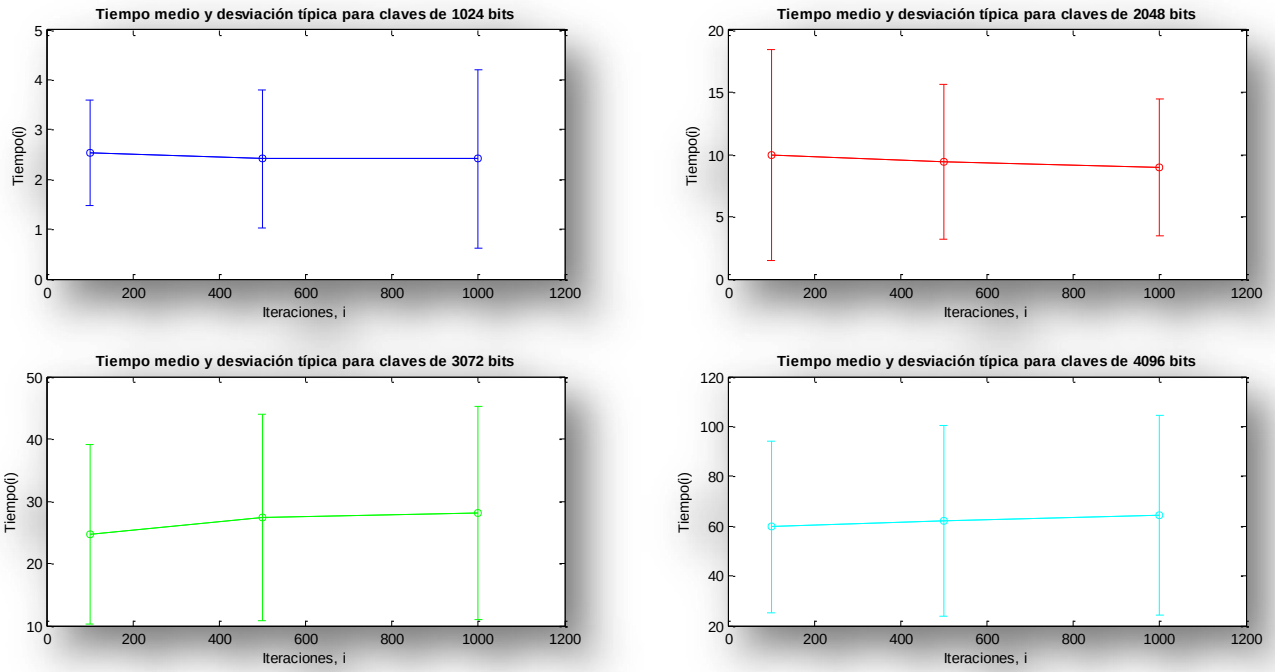


Figura 5.42. Ejecución del protocolo: representación del tiempo medio y desviación típica para los distintos tamaños de claves con tiempo aleatorio entre muestras. La primera gráfica (azul) representa el tiempo medio y desviación típica para un tamaño de 1024 bits; la segunda (roja) los resultados para un tamaño de 2048 bits; la tercera (verde) los resultados para un tamaño de 3072 bits; la última (cían) resultados para un tamaño de 4096 bits

5.4.2. Estadísticas sobre el tiempo de obtención de una identidad digital

Para la realización de las medidas en este caso, se introduce la instrucción *1* antes de la llamada al método que genera el CSR y el que establece la conexión con el servidor de la CA (*enviarCSR*). Este método primero llama al método que genera el CSR (*certUsuario*) y luego lo envía al servidor. Las instrucciones *2* y *3* se introduce una vez que se cierra el socket. (ver código en el *anexo F*).

El tiempo que se mide en este caso, incluye las flechas 8 y 9 de la *figura 4.3.* Intervienen las máquinas M1 y M3 y la red R2 de la *figura 5.36.* Además, se incluye en en estas estadísticas, el tiempo que tarda la aplicación en generar las claves y el CSR.

Media y dispersión del tiempo total para 100 iteraciones medidas en **segundos**:

	Tamaño de las claves			
	1024	**2048**	**3072**	**4096**
Media	2.5373	9.9463	24.6493	59.7945
Desviación	1.0513	8.4782	14.4172	34.5145

Tabla 5.7. Tiempo medio y dispersión para la obtención del certificado, 100 iteraciones y tamaño de claves: 1024, 2048, 3072 y 4096

Media y dispersión del tiempo total para 500 iteraciones medidas en **segundos**:

	Tamaño de las claves			
	1024	**2048**	**3072**	**4096**
Media	2.4116	9.4285	27.4287	62.0588
Dispersión	1.3851	6.1877	16.6171	38.1460

Tabla 5.8. Tiempo medio y dispersión para la obtención del certificado, 500 iteraciones y tamaño de claves: 1024, 2048, 3072 y 4096

Media y dispersión del tiempo total para 1000 iteraciones medidas en **segundos**:

	Tamaño de las claves			
	1024	**2048**	**3072**	**4096**
Media	2.4102	9.0076	28.1094	64.4286
Dispersión	1.7997	5.4956	17.1103	39.9366

Tabla 5.9. Tiempo medio y dispersión para la obtención del certificado, 1000 iteraciones y tamaño de claves: 1024, 2048, 3072 y 4096

Se comprueba, que el tiempo en el que la aplicación tarda en generar el CSR y obtener el certificado es bastante menor que el tiempo total de ejecución del prótocolo para los casos de un tamaño de clave de 1024 bits, pero a medida que aumenta el tamaño de claves, la diferencia no es tanta. Esto puede ser debido a una de las instrucciones que se usan para generar el *nonce* que se le envía al usuario: *SecureRandom.* Esta instrucción es un poco lenta y hace que para un tamaño de clave de 1024 bits la aplicación sea un poco más lenta que si se utilizase otro tipo de algoritmo más sencillo para obtener un número aleatorio. Sin embargo, *SecureRandom*, permite obtener un número aleatorio seguro.

Los resultados se reprsentan de forma gráfica en las siguientes imágenes:

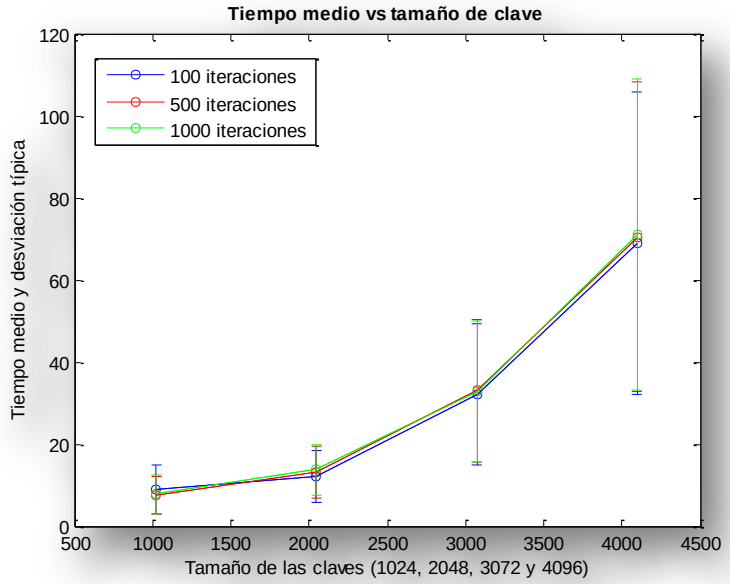


Figura 5.43. Obtención del certificado: comparación de los tiempos medios de obtención de un certificado para los distintos tamaños de claves. En esta imagen cada número de iteraciones viene representada por una línea y representa los tiempos medios y la desviación típica según el tamaño de clave

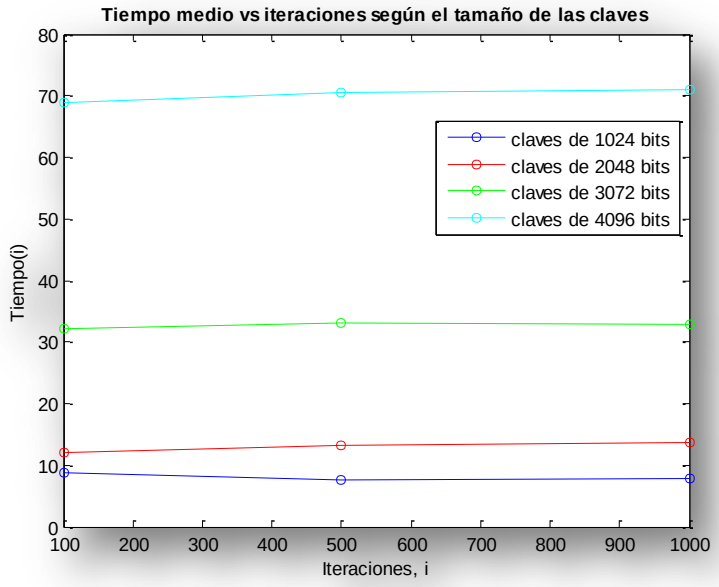


Figura 5.44. Obtención del certificado: comparación de los tiempos medios para las distintas iteraciones. Cada línea representa un tamaño de clave, el eje de abcisas representa el número de iteraciones y el eje de ordenadas los tiempos medios.

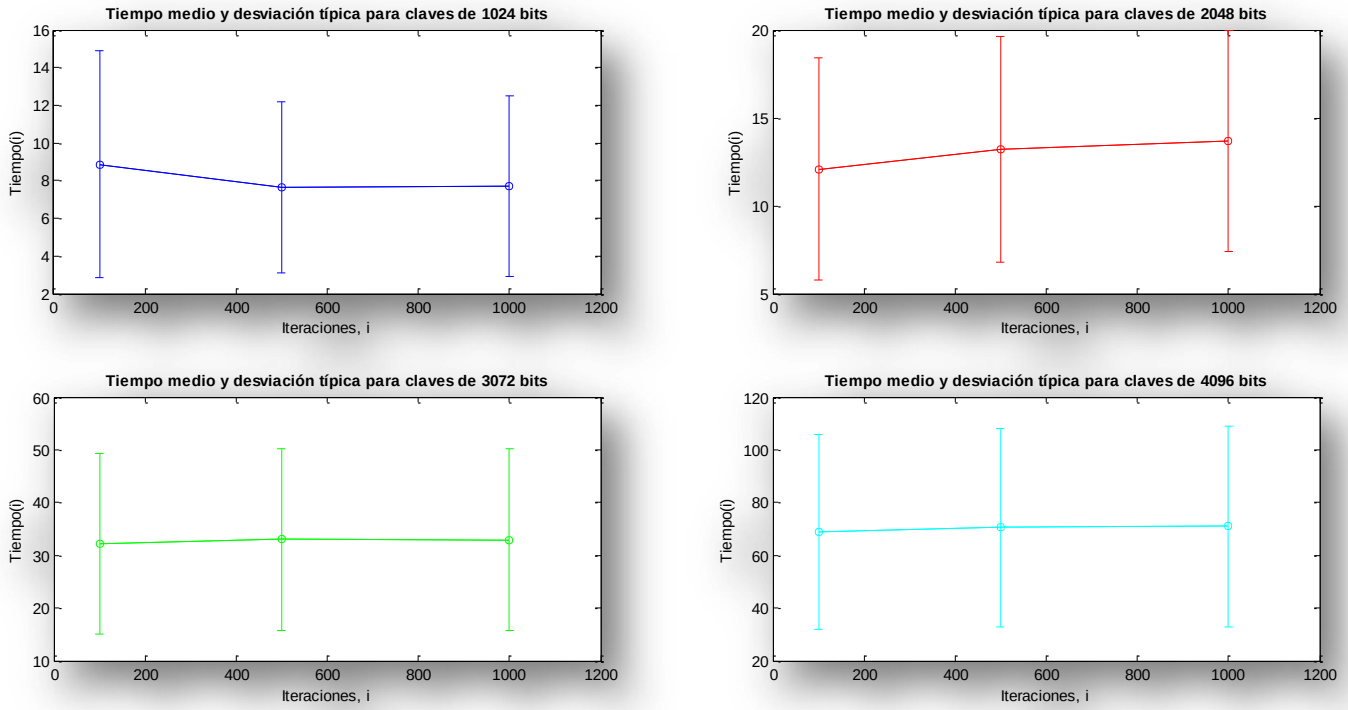


Figura 5.45. Obtención del certificado: representación del tiempo medio y desviación típica para los distintos tamaños de claves. La primera gráfica (azul) representa el tiempo medio y desviación típica para un tamaño de 1024 bits; la segunda (roja) los resultados para un tamaño de 2048 bits; la tercera (verde) los resultados para un tamaño de 3072 bits; la última (cían) resultados para un tamaño de 4096 bits

Lo que muestran los resultados, es que, naturalmente el tiempo medio empeora a medida que aumenta el tamaño de las claves, debido a la complejidad computacional que conlleva para el móvil.

Por otro lado, se puede observar que el tiempo medio se mantiene constante en todas las iteraciones para el mismo tamaño de clave y que este tiempo como es normal es mayor para un tamaño de clave mayor.

Con tiempo aleatorio entre muestra

Para la obtención de esta medida se siguen los mismos pasos que en la segunda parte del apartado anterior. Es decir, se para la aplicación durante un tiempo aleatorio generado con el método antes dicho, de forma que se intercala un tiempo aleatorio con el tiempo de ejecución. Los valores de *T_MAX* son los mismos que en el apartado anterior.

Media y dispersión del tiempo total para 100 iteraciones medidas en **segundos**:

	Tamaño de las claves			
	1024	**2048**	**3072**	**4096**
Media	3.0329	8.2455	27.4657	62.1493
Dispersión	2.3815	4.7579	17.1129	36.5980

Tabla 5.10. Tiempo medio y dispersión para la obtención del certificado, tiempo aleatorio entre muestra, 100 iteraciones y tamaño de claves: 1024, 2048, 3072 y 4096

Media y dispersión del tiempo total para 500 iteraciones medidas en **segundos**:

	Tamaño de las claves			
	1024	**2048**	**3072**	**4096**
Media	2.7476	8.8134	27.8942	64.6440
Dispersión	2.4220	5.1975	16.6914	37.2081

Tabla 5.11. Tiempo medio y dispersión para la obtención del certificado, tiempo aleatorio entre muestra, 500 iteraciones y tamaño de claves: 1024, 2048, 3072 y 4096

Media y dispersión del tiempo total para 1000 iteraciones medidas en segundos:

	Tamaño de las claves			
	1024	**2048**	**3072**	**4096**
Media	2.7515	8.9366	27.5758	65.2418
Dispersión	2.6561	5.0601	16.4548	37.6159

Tabla 5.12. Tiempo medio y dispersión para la obtención del certificado, tiempo aleatorio entre muestra, 1000 iteraciones y tamaño de claves: 1024, 2048, 3072 y 4096

A continuación, se muestran los resultados en las siguientes gráficas:

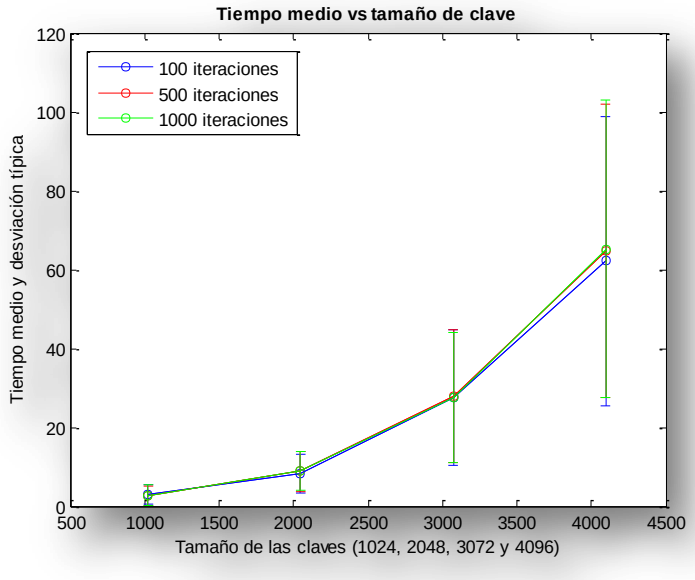


Figura 5.46. Obtención del certificado: comparación de los tiempos medios para los distintos tamaños de claves introduciendo un tiempo aleatorio entre iteración. En esta imagen cada número de iteraciones viene representada por una línea y representa los tiempos medios y la desviación típica según el tamaño de clave

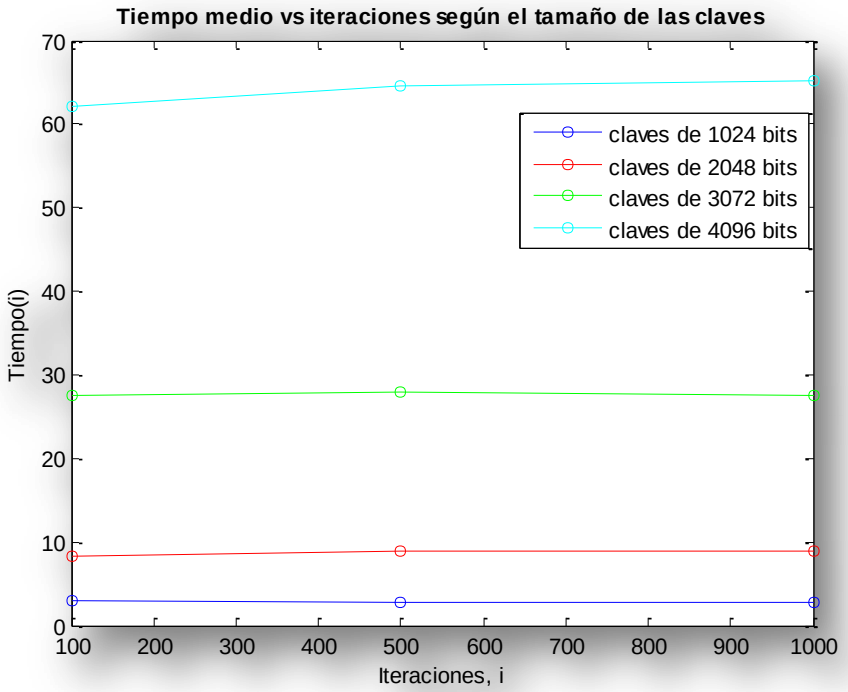


Figura 5.47. Obtención del certificado: comparación de los tiempos medios para las distintas iteraciones introduciendo un tiempo aleatorio entre iteración. Cada línea representa un tamaño de clave, el eje de abcisas representa el número de iteraciones y el eje de ordenadas los tiempos medios.

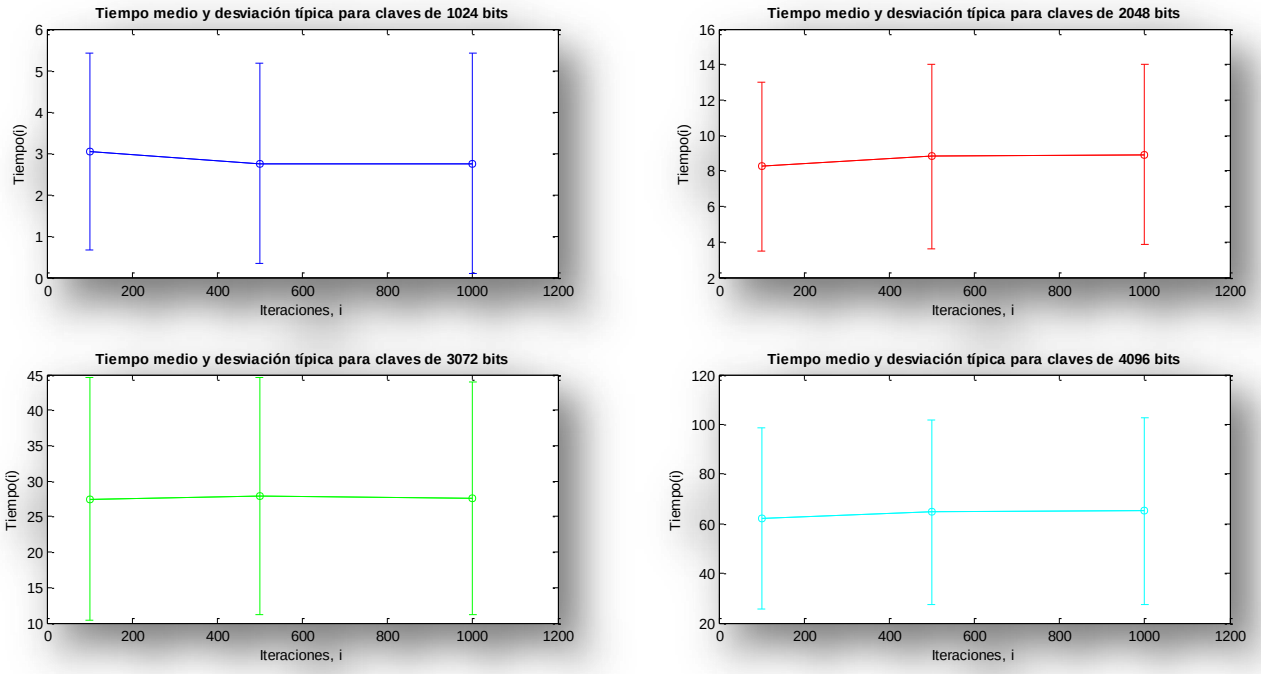


Figura 5.48. Ejecución del protocolo: representación del tiempo medio y desviación típica para los distintos tamaños de claves con tiempo aleatorio entre muestras. La primera gráfica (azul) representa el tiempo medio y desviación típica para un tamaño de 1024 bits; la segunda (roja) los resultados para un tamaño de 2048 bits; la tercera (verde) los resultados para un tamaño de 3072 bits; la última (cían) resultados para un tamaño de 4096 bits

En este caso, los valores medios de las muestras se observan muy equilibrados en relación al caso en el que las iteraciones se realizan seguidas.

El tiempo medio mejora en el caso de un tamaño de clave de 2048 bits y en general la desviación típica es menor en este caso que para la realización de las iteraciones seguidas.

Por tanto, si en el caso de la ejecución del protocolo el tiempo medio es peor, podría deberse no al tiempo de ejecución de las firmas, si no al tiempo de obtención de los *nonces* que se envían al usuario.

5.4.3. Estadísticas sobre el tiempo necesario para autenticarse.

Para la realización de estas medidas, se han seguido los mismos pasos que en el apartado *5.4.1.* , usando las instrucciones *1*, *2* y *3.* En este caso, el método al que se le llama entre las instrucciones *1* y *2* es *EstablecerConexion().* Intervienen las máquinas M1, M2 y la red R1 de la *figura 5.36*, así como las flechas 1 y 2 de la *figura 4.7.* Además, en el dispositivo móvil se realiza la generación del *nonce* que se firma, así como la firma, mientras que en el servidor se verifica esta firma a partir del *nonce* y el certificado enviado por el usuario.

Media y dispersión del tiempo total para 100 iteraciones medidas en **segundos**:

	Tamaño de las claves			
	1024	**2048**	**3072**	**4096**
Media	1.2991	1.1571	2.2805	1.4558
Desviación	1.0955	0.8819	1.7549	0.4518

Tabla 5.13. Tiempo medio y dispersión para autenticarse, 100 iteraciones y tamaño de claves: 1024, 2048, 3072 y 4096

Media y dispersión del tiempo total para 500 iteraciones medidas en **segundos**:

	Tamaño de las claves			
	1024	**2048**	**3072**	**4096**
Media	1.5041	1.5044	2.2688	1.6653
Dispersión	2.3737	2.2846	2.7868	1.5950

Tabla 5.14. Tiempo medio y dispersión para autenticarse, 500 iteraciones y tamaño de claves: 1024, 2048, 3072 y 4096

Media y dispersión del tiempo total para 1000 iteraciones medidas en **segundos**:

	Tamaño de las claves			
	1024	**2048**	**3072**	**4096**
Media	1.5433	1.5951	2.0601	2.1213
Dispersión	2.3870	2.2718	2.7660	2.0752

Tabla 5.15. Tiempo medio y dispersión para autenticarse, 1000 iteraciones y tamaño de claves: 1024, 2048, 3072 y 4096

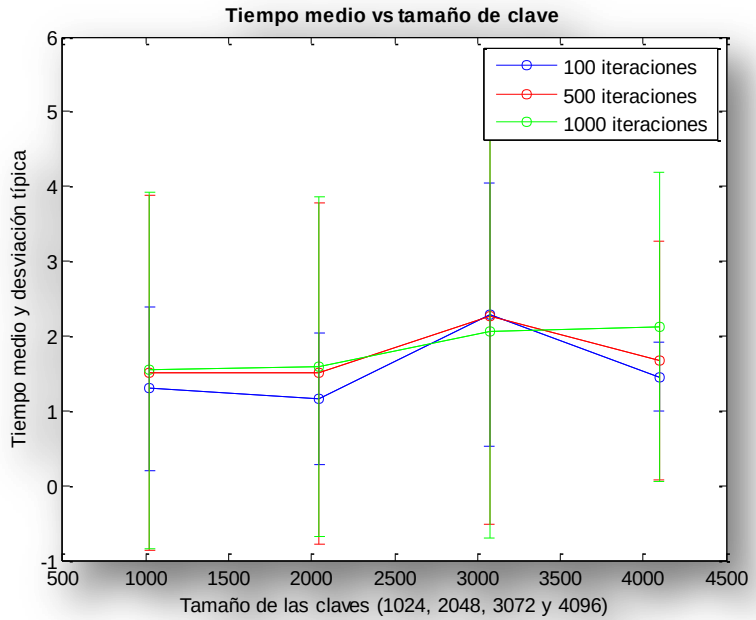


Figura 5.49. Autenticación: comparación de los tiempos medios de ejecución del protocolo para los distintos tamaños de claves. En esta imagen cada número de iteraciones viene representada por una línea y representa los tiempos medios y la desviación típica según el tamaño de clave

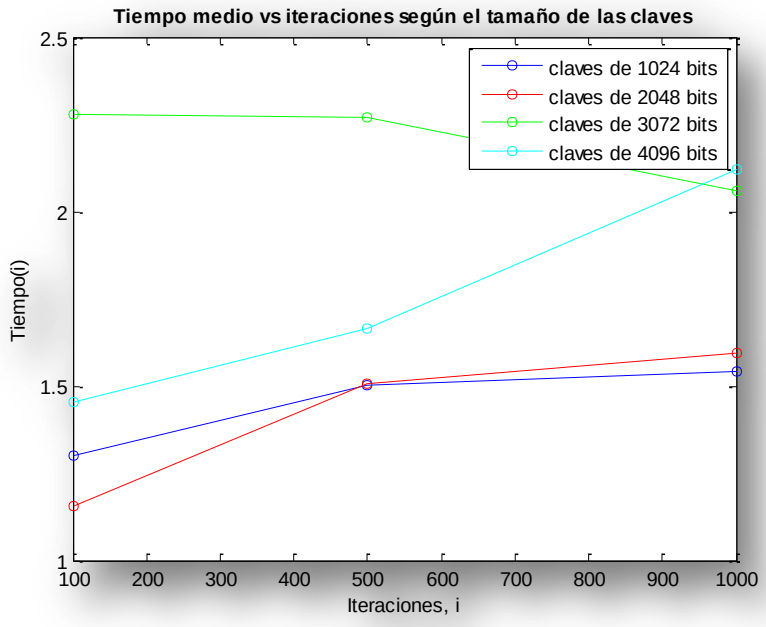


Figura 5.50. Autenticación: comparación de los tiempos medios para las distintas iteraciones. Cada línea representa un tamaño de clave, el eje de abcisas representa el número de iteraciones y el eje de ordenadas los tiempos medios.

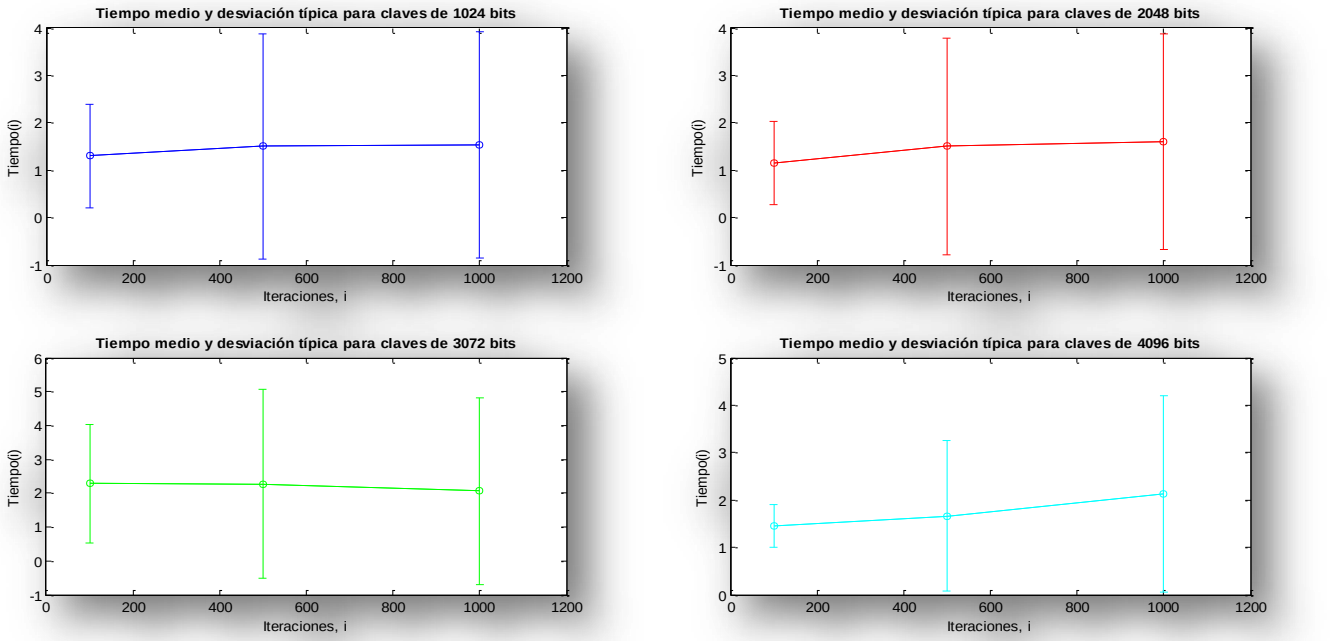


Figura 5.51. Autenticación: representación del tiempo medio y desviación típica para los distintos tamaños de claves. La primera gráfica (azul) representa el tiempo medio y desviación típica para un tamaño de 1024 bits; la segunda (roja) los resultados para un tamaño de 2048 bits; la tercera (verde) los resultados para un tamaño de 3072 bits; la última (cían) resultados para un tamaño de 4096 bits.

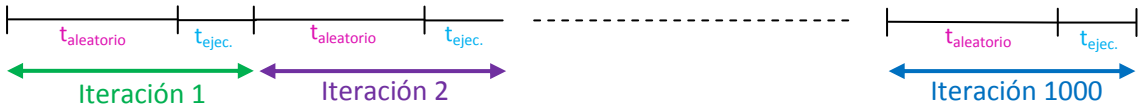
En este caso, se observa que los tiempos para realizar la autenticación del usuario son mínimos. Se observa que a medida que aumenta el tamaño de clave, también aumenta el tiempo medio, lo que es normal.

La escala de tiempo en la que se mantiene la media de los resultados, es menor que para los tiempos de registro, por lo que en la gráfica los cambios parecen mayores. Sin embargo, todos los resultados se mantienen en una media que está en un rango de 1 a 2,5 segundos.

Pero en general, las diferencias de tiempo no son demasiado notables dependiendo del tamaño de clave y esto quizá es debido a que el *nonce* que se genera para ser firmado, es de un tamaño pequeño, por lo que a la aplicación no le cuesta mucho generar la firma.

Con tiempo aleatorio entre muestra

Para la obtención de estos resultados, se ha realizado la misma implementación que las comentadas en los apartados anteriores, usando las mismas instrucciones, de modo que la distribución del tiempo aletorio y el tiempo de ejecución de la aplicación es la siguiente:



Media y dispersión del tiempo total para 100 iteraciones medidas en **segundos**:

	Tamaño de las claves			
	1024	**2048**	**3072**	**4096**
Media	1.5932	1.9959	2.0226	2.0386
Dispersión	1.0981	2.7582	2.9728	1.2177

Tabla 5.16. Tiempo medio y dispersión para autenticarse introduciendo tiempo aleatorio entre muestra, 100 iteraciones y tamaño de claves: 1024, 2048, 3072 y 4096

Media y dispersión del tiempo total para 500 iteraciones medidas en **segundos**:

	Tamaño de las claves			
	1024	**2048**	**3072**	**4096**
Media	1.4892	1.8862	2.0609	2.3386
Dispersión	1.4705	2.2686	2.4414	1.9406

Tabla 5.17. Tiempo medio y dispersión para autenticarse introduciendo tiempo aleatorio entre muestra, 500 iteraciones y tamaño de claves: 1024, 2048, 3072 y 4096

Media y dispersión del tiempo total para 1000 iteraciones medidas en segundos:

	Tamaño de las claves			
	1024	**2048**	**3072**	**4096**
Media	1.5396	1.8492	1.9371	2.4022
Dispersión	1.9298	2.3106	2.1259	2.4965

Tabla 5.18. Tiempo medio y dispersión para autenticarse introduciendo tiempo aleatorio entre muestra, 1000 iteraciones y tamaño de claves: 1024, 2048, 3072 y 4096

Los resultados se muestran gráficamente en las siguientes imágenes:

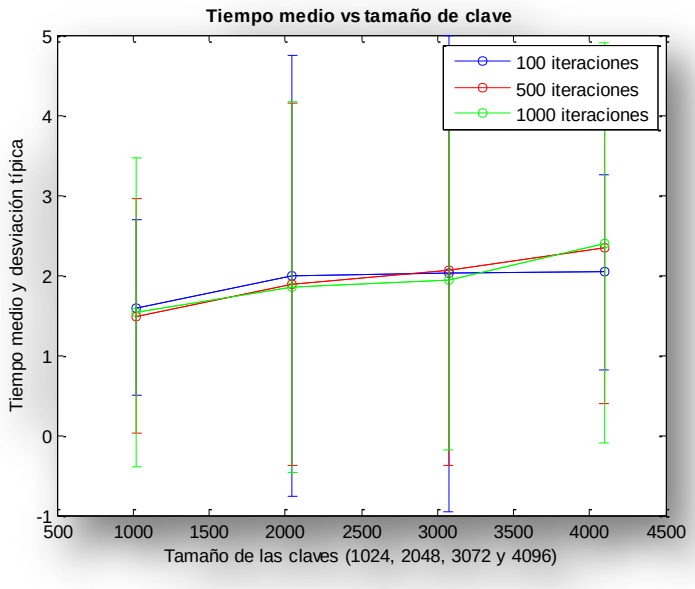


Figura 5.52. Autenticación: comparación de los tiempos medios para los distintos tamaños de claves introduciendo un tiempo aleatorio entre iteración. En esta imagen cada número de iteraciones viene representada por una línea y representa los tiempos medios y la desviación típica según el tamaño de clave

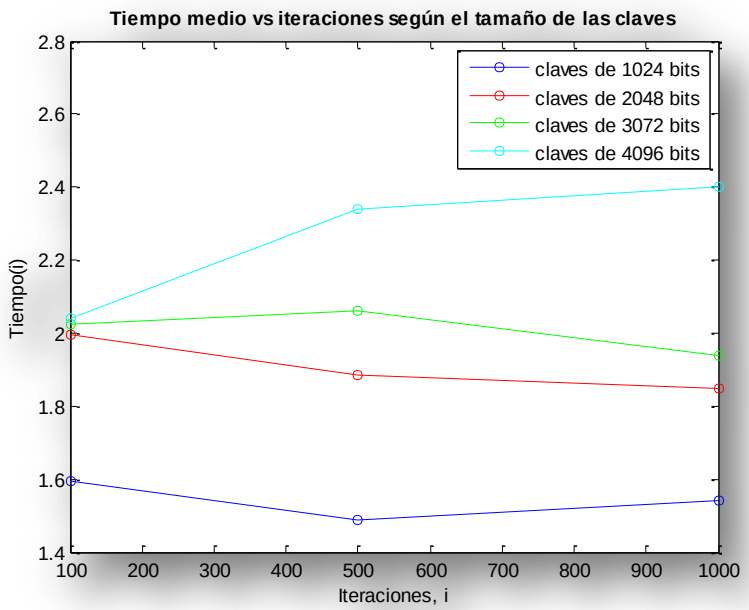


Figura 5.53. Autenticación: comparación de los tiempos medios para las distintas iteraciones introduciendo un tiempo aleatorio entre iteración. Cada línea representa un tamaño de clave, el eje de abcisas representa el número de iteraciones y el eje de ordenadas los tiempos medios.

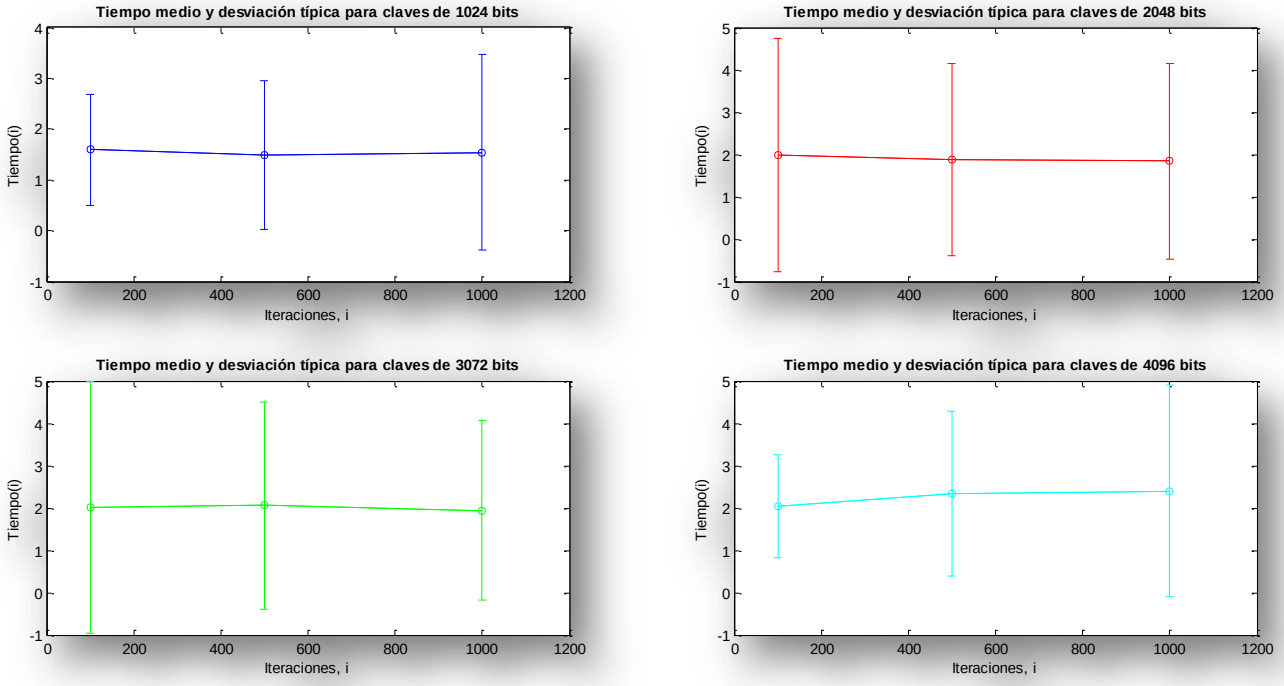


Figura 5.54. Autenticación: representación del tiempo medio y desviación típica para los distintos tamaños de claves con tiempo aleatorio entre muestras. La primera gráfica (azul) representa el tiempo medio y desviación típica para un tamaño de 1024 bits; la segunda (roja) los resultados para un tamaño de 2048 bits; la tercera (verde) los resultados para un tamaño de 3072 bits; la última (cían) resultados para un tamaño de 4096 bits

Otra vez la diferencia entre los tiempos medios para la autenticación dependiendo de los distintos tamaños de claves es mínima y está en un rango de valores entre 1 y 2,5 segundos.

Como es normal, a mayor tamaño de clave mayor es el tiempo que se tarda en realizar la autenticación.

En esta operación intervienen las máquinas M1 y M2, así como la red R1 de la *figura 5.36*.

5.5. TEST DE USABILIDAD

Para completar el apartado de resultados, se ha llevado a cabo una encuesta sobre un grupo de usuarios compuesto por 8 personas. En esta encuesta se explica al usuario de que forma se divide la aplicación y se le da una serie de opciones a elegir según se adapten mejor a su opinión. Debajo de estas opciones se muestra el número de usuarios que han elegito dicha opción. La encuesta es como sigue:

El objetivo de esta encuesta es saber la opinión de los usuarios acerca de la aplicación desarrollada.

La aplicación consta de tres partes diferenciadas: registro, autenticación (iniciar sesión en un sistema) y eliminar certificado (gestión de certificados).

Con esta encuesta se pretende saber la opinión de los usuarios sobre la parte fundamental de la aplicación (registro) y su aplicación (autenticación) y también una opinión general sobre el uso de la aplicación.

Las opciones que el usuario puede elegir van desde un nivel más desfavorable (1) hasta un nivel más favorable (5).

Redondee las siguientes cuestiones según su opinión sobre el proceso de **registro :**

A.- El proceso de registro me ha parecido sencillo

Complicado	1	2	3	4	5	**Sencillo**
Nº votos	0	0	1	0	7	

B.- El proceso de registro me ha parecido rápido

Lento	1	2	3	4	5	**Rápido**
Nº votos	0	1	3	2	2	

C.- El proceso de registro me ha parecido seguro

Inseguro	1	2	3	4	5	**Seguro**
Nº votos	0	0	2	3	3	

D.- El proceso de registro me ha parecido bien desarrollado.

Mal desarrollado	1	2	3	4	5	**Bien desarrollado**
Nº votos	0	0	0	1	7	

De igual forma indique su opinión acerca del proceso de **autenticación**.

E.- El proceso de autenticación me ha parecido sencillo

Complicado	1	2	3	4	5	**Sencillo**
Nº votos	0	0	2	0	6	

F.- El proceso de autenticación me ha parecido rápido

Lento	1	2	3	4	5	**Rápido**
Nº votos	0	1	4	1	2	

G.- El proceso de autenticación me ha parecido seguro

Inseguro	1	2	3	4	5	**Seguro**
Nº votos	0	0	1	2	5	

H.- El proceso de autenticación me ha parecido bien desarrollado.

Mal desarrollado	1	2	3	4	5	**Bien desarrollado**
Nº votos	0	0	1	1	6	

Por último, conteste estas dos últimas preguntas **generales** acerca de la aplicación:

I.- El uso de la aplicación me parece intuitiva

Poco intuitiva	1	2	3	4	5	**Muy intuitiva**
Nº votos	0	0	1	0	7	

J.- Me parece que la aplicación está bien desarrollada

Mal desarrollada	1	2	3	4	5	**Bien desarrollada**
Nº votos	0	0	3	1	4	

Respecto a la parte de la aplicación que se encarga del registro, los resultados son como siguen:

- En la pregunta A, sobre la opinión de los usuarios acerca del proceso de registro, el 87,5 % de personas opinan que el proceso de registro es bastante sencillo, mientras que el 12,5 % opina que es normal.
- En la pregunta B, acerca de la rapidez del registro, el 25 % de personas opinan que el proceso de registro es bastante rápido, otro 25 % opina que es rápido, el 37,5 % opina que es normal y el 12,5 % opina que es lento.

- En la pregunta C el 37,5 % de personas opinan que el proceso de registro es bastante seguro, otro 37,5 % opina que es seguro y el 25 % opina que es normal.

- En la pregunta D el 87,5 % de personas opinan que el proceso de registro esta bastante bien desarrollado y el 12,5 % opina que está bien desarrollado.

Respecto de la parte de la aplicación que se encarga de la autenticación:

- En la pregunta E el 75 % de personas opinan que el proceso de autenticación es bastante sencillo, el 25 % opina que el proceso es normal.

- En la pregunta F el 25 % de personas opinan que el proceso de autenticación es bastante rápido, el 12,5 % opina que es rápido, el 50 % opina que es normal y el 12,5 % opina que es lento.

- En la pregunta G el 62,5 % de personas opinan que el proceso de autenticación es bastante seguro, el 25 % opina que es seguro y el 12,5 % opina que es normal.

- En la pregunta H el 75 % de personas opinan que el proceso de autenticación esta bastante bien desarrollado, el 12,5 % opinan que está bien desarrollado y el otro 12,5 % opina que es normal.

Respecto de las preguntas generales, las opiniones son las siguientes:

- En la pregunta I el 87,5 % de personas opinan que la aplicación es bastante intuitiva y el 12,5 % opina que es normal.

- En la pregunta J el 50 % de personas opinan que la aplicación está bastante bien desarrollada, el 12,5 % opina que está bien desarrollada y el 37,5 % opina que es normal.

En un recuento final del número de votos, se puede comprobar que para una puntuación de 1 no se ha recibido ningún voto; para una puntuación de 2 sólo se han recibido 2 votos; para una puntuación de 3 se ha recibido 18 votos; para una puntuación de 4 puntos se ha recibido 11 votos y finalmente para una puntuación de 5 se ha recibido 49 votos.

Por tanto, como conclusión final se puede decir, que la opinión de los usuarios en relación a la usabilidad de la aplicación se puede considerar positiva, pues la mayoría de los votos recibidos son para la máxima puntuación y no hay ningún voto totalmente negativo.

Capítulo 6:

Conclusiones y trabajo futuro

6. Conclusiones y trabajo futuro

6.1. CONCLUSIONES

En este proyecto, se ha llevado a cabo el desarrollo de una aplicación Android que permite obtener una identidad digital, hacer uso de ella para autenticarse en un sistema y gestionarla. Además, se ha implementado un protocolo que permite que la aplicación haga uso de este para la obtención de dicha identidad digital [1][2][3]. Por tanto, este proyecto sirve como prueba de concepto para demostrar que es posible la implementación de un sistema con cierto nivel de seguridad para la obtención de una identidad digital en una plataforma tan extendida como lo es Android.

La ventaja que tiene la aplicación y el protocolo desarrollado en este proyecto en relación a otras aplicaciones similares existentes, es que permite a un usuario obtener su identidad digital sin necesidad de desplazarse a una entidad de registro. Esto puede ser útil en sistemas de registro cerrado, como por ejemplo, permitir el acceso a un servicio a los alumnos de la EPS, de la UAM, de una empresa, etc., es decir, a un grupo de personas con un correo electrónico reconocible y cuya identidad pueda comprobarse mediante este.

El diseño del proyecto se centró en el objetivo principal que era obtener una identidad digital. Para ello, el desarrollo de la aplicación partió de la generación de un CSR desde la aplicación móvil a partir de unos datos de registro que el usuario debía introducir y que están contenidos en el propio CSR.

Al usuario se le pide también que introduzca su número de teléfono. Este número no está contenido en el CSR, sino que se envía a un servidor (servidor de la CA) para la generación de un "*nonce*" a partir de él. El *nonce* es devuelto al usuario por dos vías: a través de la propia conexión del servidor y a través de la conexión del servidor con otro servidor (servidor de SMS) que se encarga de enviar el nonce al usuario. En la aplicación ambos códigos se comparan, si no ha recibido ningún ataque, ambos números tienen que ser iguales. Si lo son se envía un *ack* a través del servidor de SMS hacia el servidor de la CA, que confirma al servidor de la CA que el usuario recibió correctamente el código. Una vez hecho esto, el protocolo está diseñado para que la aplicación envie al servidor de la CA el CSR, el cuál firma el CSR y le devuelve al usuario un certificado.

Una vez que se consiguió la realización del principal objetivo, se vió necesario demostrar la utilidad de la consecución por parte de un usuario de una identidad digital. Para ello, se implementó un sistema de autenticación en una plataforma

simulada. La autenticación se consiguió mediante la realización de una firma digital sobre un *nonce*, que se envía junto con la firma y el certificado del usuario a un servidor. Este servidor generará la firma con el certificado recibido sobre el nonce y comparará la firma con la recibida. Además, el servidor comprueba que el certificado fue firmado por la CA de confianza. Si esto se cumple se le da acceso al usuario.

También se le añadió a la aplicación la capacidad de gestionar los certificados, permitiendo a los usuarios eliminar los certificados de los que él no quisiera hacer uso.

Por último, se han realizado una serie de pruebas consistente en un conjunto de muestras que miden el rendimiento del sistema en su conjunto, así como el rendimiento del sistema si sólo se tiene presente el tiempo que transcurre desde que se genera el CSR hasta que se recibe el certificado en la aplciación. Además, estas muestran no sólo se han tomado con un conjunto de iteraciones una detrás de otra, sino que se ha querido comprobar cuál sería el rendimiento del sistema con un tiempo aleatorio entre una muestra y otra.

Como resultado de este análisis, se ha llegado a la conclusión que, en general, la mejor relación entre un tamaño de clave seguro y el tiempo medio que requiere la aplicación para la autenticación se produce para un tamaño de clave de 2048 bits. Este tamaño de clave requiere más tiempo para la obtención de una identidad digital y también de su uso, pero la calidad de usuario no empeora mucho en relación a un tamaño de clave de 1024 bits. Sin embargo, para un tamaño de clave de 3072 y 4096 bits, la obtención de un certificado requiere un mayor tiempo aunque su uso implicase más seguridad, por lo que para mejorar la experiencia del usuario habría que informarle de que la aplicación requiere más tiempo para la obtención del certificado. En cualquier caso, esta propuesta se hace conforme al móvil usado para las pruebas, pues los tiempos pueden mejorar o empeorar dependiendo de la calidad del móvil usado.

Podría considerarse también, que la generación de los *nonces* implica que se envían al usuario y que implican un tiempo extra, podría realizarse con algunas instrucciones diferentes que podrían agilizar y mejorar el tiempo medio para la obtención de la identidad digital.

Si bien se ha demostrado que la implementación de un protocolo de seguridad es posible para ser usado por una aplicación Android para la obtención de identidades digitales, puede haber una brecha de seguridad si el móvil es sustraído a una persona o dicha persona lo pierde, pues la persona que sustrae o encuentra el móvil, podría usarla de una forma poco adecuada, obteniendo un certificado a nombre de otra persona. Por ello, se considera que como se ha comentado anteriormente, la aplicación puede ser muy útil para sistemas de registro cerrado, pero quizá se debería evitar su uso para instituciones oficiales que requieren realmente la identificación de un usuario mediante el DNI.

6.2. TRABAJO FUTURO

Este proyecto sirve como prueba de concepto, lo que significa que es una primera versión para demostrar que es posible una implementación real en un futuro, por ello, a continuación se exponen un conjunto de propuestas para ser usadas en una implementación futura:

- Desarrollo e implementación del protocolo en un entorno real. Para ello se propone que:
    - El servidor de la CA sea implementado en un servidor web en lugar de usar un servidor programado en Java.
    - El servidor de SMS sea implementado en un servidor web real que haga uso de servicios telefónicos de mensajería o bien la implementación tradicional usando correo electrónico, enviando un enlace a la bandeja de entrada del usuario.
- En relación al anterior punto, el sistema podría mejorar bastante si los servidores tuviesen la capacidad de atender a la vez las peticiones de varios usuarios.
- Uso de una CA oficial, en lugar de una CA creada por la herramienta OpenSSL.
- Recepción de SMS reales en lugar de la simulación implementada en el proyecto.
- Inclusión de nuevos servicios y capacidades a la apliación aquí implementada, como uso de los certificados para firmar o cifrar correos electrónicos.
- Recepción del SMS con el nonce en background.
- Almacenamiento de los certificados únicamente en el keystore del usuario para ahorrar recursos de memoria del dispositivo móvil.
- Posibilitar a un usuario autenticarse en ciertos servicios utilizando usuario y contraseña en lugar de certificado, es decir, darle una elección real del modo de autenticarse.
- A medida que la tecnología vaya mejorando, mejoraran las capacidades de los dispositivos móviles pero también será posible romper más fácilmente la seguridad, por lo que se propone que en un futuro las claves generadas en la aplicación sean de un tamaño superior a 2048 bits.
- Extender la aplicación para hacer uso de firmas grupales.

Es necesario comentar, que dentro del trabajo futuro aplicable a la aplicación y a este proyecto, se está realizando de forma adicional un artículo que se presentará en una revista especializada y que recogerá el trabajo realizado en el proyecto.

Referencias

REFERENCIAS

[1] Jesús Diaz, David Arroyo, and Francisco B. Rodriguez. "On securing online registration protocols: Formal verification of a new proposal." *Knowledge-Based Systems* 59 (2014): 149-158.

[2] Jesús Diaz, David Arroyo, and Francisco B. Rodriguez. "An approach for adapting moodle into a secure infrastructure." *Computational Intelligence in Security for Information Systems*. Springer Berlin Heidelberg, 2011. 214-221.

[3] Diaz, Jesus, David Arroyo, and Francisco B. Rodriguez. "A formal methodology for integral security design and verification of network protocols." *Journal of Systems and Software* (2013).

[4] Shashi Kiran, Patricia Lareau, Steve Lloyd; "PKI Basics – A technical perspective".

[5] William Stallings; "Network Security Essentials: Applications and Standars"

[6] Mobile Commerce (M-COMM); Mobile Signatures; Business and Functional Requirements.

[7] Marko Gargenta; "Learning Android: Building Applications for the Android Market"

[8] Guía Android para el desarrollador: http://developer.android.com/guide/components/fundamentals.html; Consultado por última vez el 11/02/2014

[9] Jorge E. Carballo, David D. Arjani Arjani; "Desarrollo de aplicaciones móviles en Android".

[10] http://es.wikipedia.org/wiki/Anexo:Historial_de_versiones_de_Android; Consultado por última vez el 11/02/2014

[11] David Seal; "Architecture Reference Manual. Second Edition"

[12] Andrew Hoog; "Android Forensics: Investigation, Analysis and Mobile Security for Google Android"

[13] Google remotely wipes apps off Android phones; http://news.cnet.com/8301-27080_3-20008922-245.html; Consultado por última vez el 11/02/2014

[14] Jeff Six; "Aplication security for the Android Platform".

[15] Symantec; "Protección del mercado de aplicaciones móviles"

[16] http://adeideas.com/wordpress/?p=673; Consultado por última vez el 29/09/2013

[17] http://www.genbeta.com/seguridad/en-que-consiste-la-vulnerabilidad-de-android-y-como-podemos-protegernos; Consultado por última vez el 29/09/2013

[18] http://www.elandroidelibre.com/2013/07/android-tiene-menos-vulnerabilidades-que-ios-pero-es-mas-atacado-por-malware.html; Consultado por última vez el 29/09/2013

[19] Sheran A. Gunasekera; "Android Apps Security"

[20] Javier Areitio; "Seguridad de la información: Redes, informática y sistemas de información"

[21] John R. Vacca; "Public Key Infraestructure. Building Trusted Aplications and Web Service"

[22] Carlisle Adams y Steve Lloyd; "Understanding PKI: Concepts, Standards, and Deployment Considerations".

[23] Manuel José Lucena López; "Criptografía y seguridad en computadores"

[24] Principios de Kerckhoffs; http://es.wikipedia.org/wiki/Principios_de_Kerckhoffs; Consultado por última vez el 11/02/2014

[25] Alfred J. Menezes, Paul C. Van Oorschot, Scott A. Vanstone; "Handbook of Applied Cryptography"

[26] Andrew S. Tanenbaum; "Redes de computadoras"

[27] Eva Fernández Gómez; "Conocimientos y aplicaciones tecnológicas para la dirección comercial"

[28] RFC 2692; http://datatracker.ietf.org/doc/rfc2692/?include_text=1; Consultado por última vez el 01/04/2014

[29] RFC 2693; http://datatracker.ietf.org/doc/rfc2693/; Consultado por última vez el 01/04/2014

[30] Simson Garfinkel; "PGP: Pretty Good Privacy"

[31] ITU-T X.509 (11/2008); "Series X: Data Networks, Open system communications and security; Information technology – Open Systems interconnection – The Directory: Public-key and attribute certifícate frameworks"

[32] RFC 5280; "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile"; http://tools.ietf.org/html/rfc5280; Consultado por última vez el 01/04/2014

[33] Man Young Rhee; "Internet Security: Cryptographic principles, algorithms and protocols"

[34] RFC 2986; Certificate Request Syntax Specification; http://tools.ietf.org/html/rfc2986; Consultado por última vez el 01/04/2014

[35] Benjamín Ramos Álvarez, Arturo Ribagorda Garnacho; "Avances en Criptografía y seguridad de la información"

[36] F. Maciá, F. J. Mora, J. A. Gil, V. Gilart, D. Marcos, J. V. Berná, J. C. Monllor, H. Ramos, A. Albadalejo, A. Hernandez; "Administración de servicios de internet: De la teoría a la práctica"

[37] RFC 6101; http://tools.ietf.org/html/rfc6101; Consultado por última vez el 01/04/2014

[38] Hackers break SSL encryption used by millions of sites; http://www.theregister.co.uk/2011/09/19/beast_exploits_paypal_ssl/; Consultado por última vez el 01/04/2014

[39] Soluciones móviles de Safelayer S.A; http://www.safelayer.com/es/soluciones/mobile-pki; Consultado por última vez el 01/04/2014

[40] Página del desarrollador de Android; http://developer.android.com/index.html; Consultado por última vez el 01/04/2014

[41] Scott McCracken; "Android: Curso de desarrollo de aplicaciones"

[42] Java SE 7 Security Documentation; http://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html; Consultado por última vez el 01/04/2014

[43] Jamie Jaworski, Paul J. Perrone; "Seguridad en Java"

[44] Artículo de la wikipedia sobre Bouncy Castle; http://en.wikipedia.org/wiki/Bouncy_Castle_(cryptography); Consultado por última vez el 01/04/2014

[45] Página oficial de la Legión de Bouncy Castle; http://www.bouncycastle.org/java.html; Consultado por última vez el 01/04/2014

[46] Documentación de Oracle sobre Keytool; http://docs.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html#Changes; Consultado por última vez el 01/04/2014

[47] Página oficial de OpenSSL; http://www.openssl.org/docs/apps/openssl.html; Consultado por última vez el 01/04/2014

[48] Página para los desarrolladores de Android; https://developer.android.com/about/versions/kitkat.html; Consultado por última vez el 11/02/2014

[49] Internet Security thread report 2013; http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v18_2012_21291018.en-us.pdf; Consultado por última vez el 12/02/2014

[50] William Stallings; "Crytography and Network Security: Principles and practices".

[51] Wei-Meng Lee; "Beggining Android 4: Aplication Development".

[52] Página de OpenSSL; https://www.openssl.org/; Consultado por última vez el 02/04/2014

[53] Documentación de Portecle; http://portecle.sourceforge.net/; Consultado por última vez el 01/04/2014

[54] FIPS PUB 46-3; "Data Encryption Standard (DES)"

[55] NIST; Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher.

[56] FIPS PUB 197; "Specifications for the Advanced Encryption Standards (AES)"

[57] Ronald L. Rivest; "The RC5 Encryption Algorithm"

[58] R. L. Rivest, A. Shamir, L. Adleman; "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems"

[59] Página de ayuda del desarrollador de Android; https://support.google.com/googleplay/android-developer/answer/113469?hl=es ; Consultado por última vez el 11/03/2014

[60] FIPS PUB 180-4; "Secure Hash Standard (SHS)"

[61] FIPS PUB 186-3; "Digital Signature Standard (DSS)"

[62] RFC 5246; "The Transport Layer Security (TLS) Protocol Version 1.2"; http://tools.ietf.org/html/rfc5246; Consultado por última vez el 01/04/2014

[63] Página de la FNMT para la obtención de certificados digitales; https://www.sede.fnmt.gob.es/certificados/persona-fisica; Consultada por última vez el 02/04/2014

[64] Artículo de wikipedia sobre la empresa Safelayer; http://es.wikipedia.org/wiki/Safelayer_Secure_Communications; Consultado por última vez el 02/04/2014

[65] Identidad móvil: aplicación idBCN; http://www.mobileid.cat/ca#acreditarse; Consultado por última vez el 02/04/2014

[66] Como obtener una identidad digital usando la aplicación idBCN; http://www.mobileid.cat/ca/com-es-pot-obtenir-la-mobileid#acreditar-se; Consultado por última vez el 02/04/2014

[67] Ivan Ristic; "OpenSSL CookBook; A guide to the most frequently used OpenSSL, features and commands"

[68] Artículo de INTECO sobre fallo en Android; http://www.osi.es/es/actualidad/avisos/2014/03/fallo-de-seguridad-critico-en-android; Consultado por última vez el 09/04/2014

[69] Luyi Xing, Xiaorui Pan, Rui Wang, Kan Yuan y Xiao Feng Wang; "Upgrading your Android, Elevating my Malware: Privilege Escalation Through Mobile OS Updating"; http://www.informatics.indiana.edu/xw7/papers/privilegescalationthroughandroidupdating.pdf ; Consultado por última vez el 09/04/2014

Anexo A

Tabla con las diferentes versiones de Android y sus respectivas características

ANEXO A.

TABLA CON LAS DIFERENTES VERSIONES DE ANDROID Y SUS RESPECTIVAS CARACTERÍSTICAS.

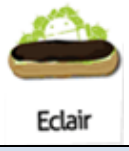
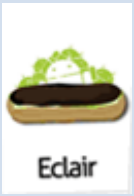
Lanzamiento	Versión	Características
23 de Sept. 2008	Android 1.0 (Nivel de API 1)	- Fue la primera release que llegó a los teléfonos comerciales. - Soporte para Wi-Fi y Bluetooth - Navegador web. - Soporte para cámara básica. - Acceso a servidores de correo electrónico. - Reproductor de audio y video. - Tuvo muchos errores y problemas de optimización, todos ellos pospuestos hasta la versión 1.1. - Definió muchos de los estándares existentes en la actualidad (las `Activity`, las notificaciones, los `intents` y los `Broadcast`…)
9 de Feb. 2009	Android 1.1 (Nivel de API 2)	- Fue la primera versión del SDK[22] en permitir definir "versión mínima". - Nace el Android Market, hoy conocido como Google Play Store. - Corrigió muchos errores en las aplicaciones instaladas por defecto, y fallos del único terminal disponible, el T-Mobile G1 o HTC Dream.
Abril 2009	Android 1.5 (Cupcake) (Nivel de API 3) 	- Acceso a múltiples emuladores con distintas configuraciones. - Se incorpora el teclado táctil. Teclados con predicción de texto y diccionario. - Grabación y reproducción en formato MPEG-4 y 3 GP. - A partir de esta versión se pueden hacer widgets propios. - Bluetooth A2DP. - Recepción de cambios de localización a través del `LocationManager`. - Mejor framework para OpenGL.[23]

[22] SDK es un conjunto de herramientas de desarrollo de software que le permite a un programador crear aplicaciones para un sistema concreto (p. e. Android SDK)

		- Librerías de reconocimiento de voz.
Sept. 2009	Android 1.6 (Donut) (Nivel de API 4)  	- Primera versión en soportar tanto GSM como CDMA. - Pantalla que indica el consumo de batería por aplicación. - Soporta pantallas con múltiples densidades de píxeles distintas. - Síntesis del habla para expresar cadenas de caracteres. - Se puede especificar que configuraciones se soportan en el *Android Manifest*. - Sistema adaptado al dispositivo de búsqueda de recursos (la carpeta `res` inteligente de la que disponemos hoy en día). - Cada usuario puede compartir el contenido de sus aplicaciones con el sistema de búsqueda integrado *Quick Search Box.* - Introducción de la librería de detección de Gestos. - Integración del `zipalign` (herramienta que optimiza los APKs[24]).
26 de Nov. 2009	Android 2.0 (Éclair) (Nivel de API 5)  	- API de administración de cuentas sincronizadas de servicios de terceros. - Nueva API para Contactos. - Navegador HTML5. - API de Bluetooth. - Nuevas características para la cámara: soporte flash, zoom digital, balance de blancos, efecto de colores y enfoque macro. - Mejora en la velocidad de tipeo. - Soporte para más tamaños de pantalla y resoluciones. - Librería de detección de movimientos en la pantalla.
3 de Dic. 2009	Android 2.0.1 (Éclair) (Nivel de API 6)	- Pequeños cambios en la API. - Corrección de errores. - Cambios en el framework.

[23]OpenGL (Open Graphics Library) es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.

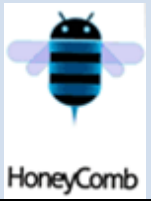
[24] **APK: A**pplication **P**ac**k**age File. Los archivos con extensión .apk es un paquete para el sistema operativo Android. Es una variante del formato JAR de Java. Es básicamente un archivo comprimido ZIP con diferente extensión.

		
12 de Enero 2010	Android 2.1 (Éclair) (Nivel de API 7) 	- Inclusión de los "Live Wallpapers" que utilizan una versión primitiva de *Renderscript*[25]. - API para hacer Live Wallpapers. - Pequeños cambios de API de telefonía y *views*. - Aparición de Nexus One. - Corrección de errores.
20 de Mayo 2010	Android 2.2 (Froyo) (Nivel de API 8) 	- Optimización de la velocidad, memoria y rendimiento. - Introducción de un compilador de tiempo real a la máquina virtual Dalvik. - Introducción de las notificaciones push desde la C2DM (Android Cloud to Device Messaging) de Google (`Intents` a través de Internet). - API de copia de seguridad de aplicaciones unido a la cuenta de usuario ("App Backup API"). - Aplicaciones disponibles ahora desde la tarjeta SD para liberar espacio. - Actualización automática de aplicaciones del Android Market si no cambian de permisos. - Opción para deshabilitar acceso de datos sobre red móvil. - Soporte para contraseñas numéricas y alfanuméricas. - Introducción de OpenGL ES 2.0
18 de Enero 2011	Android 2.2.1 (Froyo) (Nivel de API 8) 	- Corrección de errores. - Actualizaciones de seguridad. - Mejoras de rendimiento.
22 de Enero 2011	Android 2.2.2 (Froyo) (Nivel de API 8)	- Arreglo de fallos menores, como el routeo de SMS que afectaba al Nexus One.

[25] Renderscript es un componente del sistema operativo Android para dispositivos móviles. Es una API de bajo nivel para computación intensiva usando computación heterogénea.

		
21 de Nov. 2011	Android 2.2.3 (Froyo) (Nivel de API 8) 	- Dos parches de seguridad.
6 de Dic. 2010	Android 2.3 (Gingerbread) (Nivel de API 9) 	- Introducción del NDK (Native Development Kit) que permite introducir código C/C++ en las aplicaciones desarrolladas por un desarrollador. - El JNI o Java Native Interface hace de puente entre el SDK y el NDK (no se puede hacer aplicaciones sólo con el SDK). - APIs Dalvik para C/C++. - Recolector de basura concurrente. - API que facilita la construcción de servicios VoIP (SIP-based VoIP). - Soporte y API para NFC (Near Field Communications). - Soporte para giroscopio y múltiples cámaras a nivel de API. - Soporte para tamaños y resoluciones de pantalla extra-grande. - Cambios desde YAFFS a ext 4 en dispositivos nuevos. - Mejoras en la administración de la energía. - Nueva API para todo tipo de Media. - API que permite a las aplicaciones solicitar al SO que realice una descarga. - Soporte para dispositivos sin tarjeta SD.
9 de Feb. 2011	Android 2.3.3 (Gingerbread) (Nivel de API 10) 	- Mejoras y arreglos del API.
28 de Abril 2011	Android 2.3.4 (Gingerbread) (Nivel de API 10) 	- Soporte de chat de video o voz. - Soporte a la biblioteca Open Accesory.

25 de Julio 2011	Android 2.3.5 (Gingerbread) (Nivel de API 10) 	- Mejoras en el sistema. - Mejoras a la aplicación del correo electrónico. - Mejoras en el software de la cámara. - Mejorada la eficiencia de la batería.
2 de Sept. 2011	Android 2.3.6 (Gingerbread) (Nivel de API 10) 	- Arreglado fallo en la búsqueda por voz.
21 de Sept. 2011	Android 2.3.7 (Gingerbread) (Nivel de API 10) 	- Soporte de Google Wallet para el Nexus S 4G.
Feb. 2011	Android 3.0 (HoneyComb) (Nivel de API 11)  HoneyComb	- Introducción de los Fragments. - Soporte exclusivo para *tablets* (diseño primero horizontal). - Soporte para procesadores multi-núcleo. - Añade un portapapeles a nivel de sistema. - API mucho más sencilla para operaciones *Drag&Drop*. - Nuevos widgets. - Interfaz simplificada y más intuitiva para copiar/pegar. - Utilización de pestañas múltiples. - Nueva interfaz de contactos. - Nuevas notificaciones. - "Loader Framework". - Framework de animaciones totalmente nuevo. - Tema holográfico. - Aceleración 2D para aplicaciones. - Introducción Oficial de Renderscript. - Aceleración de hardware. - Framework para DRM, soporte de dispositivos USB para transferencia de datos. - Habilidad para cifrar todos los datos del usuario. - Mejoras en el uso de HTTPS con Server Name Indication (SNI)
10 de Mayo	Android 3.1	- Continúa siendo exclusivo para *tablets.*

2011	(HoneyComb) (Nivel de API 12) 	- Nuevas APIs USB que permiten utilizar distintos tipos de accesorios, incluidos mandos para jugar. - Refinamiento de la interfaz de usuario. - Conectividad para accesorios USB. - Widgets redimensionado en la pantalla de inicio, es decir, son de tamaño adaptable. - Soporte para dispositivos externos como teclados externos, joysticks, etc. - Soporte para proxy HTTP para cada punto de Wi-Fi conectado.
15 de Julio 2011	Android 3.2 (HoneyComb) (Nivel de API 13) 	- Mejoras de soporte hardware. - Compatibilidad para aplicaciones que no han sido diseñadas para tablets. - Nuevas funciones de soporte de pantallas. - Nuevos protocolos y APIs para acceder a cámaras y otros dispositivos que soporten el PTP o Picture Transfer Protocol. - API para comunicaciones en tiempo real (RTP). - Mejoras en distintos frameworks. - Caché LRU disponible para desarrolladores. - Mejoras de velocidad.
20 de Sept. 2011	Android 3.2.1 (HoneyComb) (Nivel de API 13) 	- Corrección de errores menores y mejoras de seguridad. - Mejoras de estabilidad y Wi-Fi. - Actualizaciones del Android Market. - Mejoras en el soporte de Adobe Flash del navegador.
Oct. 2011	Android 4.0/4.0.3 (Ice-Cream Sandwich) (Nivel de API 14 y 15) 	- Todos los cambios de las versiones 3.x llegan a los teléfonos. - Por primera vez, existe una versión que unifica la experiencia de usuario (UX) a lo largo de toda la gama de dispositivos posibles. - Notificaciones más elegantes. - API Android Beam para transferir información utilizando NFC como medio. - Extensión a la API de Contactos. - API de acceso al calendario. - Nuevo framework para añadir efectos a fotos y vídeos. - API de control remoto de la multimedia que esté en “play” en el dispositivo.

		- API para hacer streaming multimedia a través de Internet. - API de detección de caras para aplicaciones que utilicen la cámara. - Soporte para conexiones Wi-Fi directas y Bluetooth 4.0. - Sensores de temperatura y humedad. - Mejor integración de voz y dictado de texto en tiempo real continuo. - Nuevo navegador web que permite tener abiertas hasta 15 pestañas. - Mejoras de estabilidad. - Mejoras en el rendimiento de la cámara. - Mejoras en el reconocimiento de números de teléfono.
27 de Jun. 2012	Android 4.1 (Jelly Bean) (Nivel de API 16) 	- Mejora drásticamente la experiencia de usuario a través del "Project Butter" (mejoras en el tiempo de respuesta, mejor sincronización con los ciclos de respuesta de la pantalla y *triple-buffering*). - Notificaciones expandibles y con acciones integradas. - Reconocimiento de voz *offline.* - Nuevas APIs para obtener avisos de alto consumo de memoria en el dispositivo. - Nueva API para la navegación "Up". - Mejoras generales a lo largo de la API. - Nuevos estilos de fuente disponibles. - El navegador de serie de Android es remplazado por la versión móvil de Google Chrome.
23 de Julio y 9 de Octubre 2012	Android 4.1.1/4.1.2 (Jelly Bean) (Nivel de API 16)  	- Arreglo de fallos y mejoras de rendimiento - Soporte de rotación en la pantalla principal. - Notificaciones expansión/contracción con un dedo.
29 de Oct. 2012	Android 4.2 (Jelly Bean) (Nivel de API 17)	- Introducción de los "Daydreams", o screensavers (protectores de pantalla) interactivos. - API para activar Daydreams desde nuestra aplicación. - Soporte para aplicaciones secundarias (ver videos, mostrar presentaciones) sobre Wi-

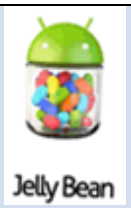
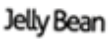
		Fi. - Widgets en la pantalla de bloqueo. - Entorno con soporte para múltiples usuarios. - Soporte para pantallas inhalambricas. - API para diferenciar preferencias entre las del usuario y las generales. - Soporte para layouts que automáticamente se adaptan a lenguajes que se escriben de derecha a izquierda. - Soporte para Fragments embebidos (Nested Fragments).
24 de Julio de 2013	Android 4.3 (Jelly Bean) (Nivel de API 18)  	- Soporte para Bluetooth de Baja Energía. - OpenGL ES 3.0 - Modo de perfiles con acceso restringido. - DRM APIs de mayor calidad - Mejora en la escritura - Cambio de usuarios más rápida - Soporte para Hebreo y Árabe - Locación de WiFi en segundo plano - Añadido el soporte para 5 idiomas más. - Opciones para creadores de Apps. - Mejoras en la seguridad
31 de Oct. 2013	Android 4.4 (KitKat) (Nivel de API 19) 	- Realización de muchas tareas por medio de comandos de voz. - Identificador de llamadas que busca la localización de números desconocidos en Google Maps. - Impresión de documentos a través mediante impresoras HP conectada por WiFi. - Nuevo marco de acceso a almacenamiento. - Aplicaciones para uso deportivo como cuentakilómetros, etc. - Un nuevo proveedor de SMS. - Un sistema que permite crear videos de las aplicaciones tipo video tutoriales.

Tabla A.1. Versiones de Android [9], [10] y [48]

Anexo B

Algoritmos de criptografía simétrica

ANEXO B.

ALGORITMOS DE CRIPTOGRAFÍA SIMÉTRICA

DES (Data Encription Standard)

Este tipo de cifrado fue escogido en 1977 como un Estándar de Procesamiento de Información Federal (Federal Information Processing Standard 46 o FIPS PUB 46) por el Instituto Nacional de Estándares y Tecnología (National Institute of Standards and Tecnology o NIST). Al algoritmo en si se le conoce como DEA y fue creado por IBM.

En la actualidad este algoritmo no es muy usado y su uso se ha sustituido por el algoritmo AES. Esto es debido a que ha conseguido ser descifrado en un tiempo menor a 24 horas. El algoritmo Triple DES, que es un derivado de DES, es algo más seguro pues no ha sido roto en la práctica, sin embargo, teóricamente si se ha podido romper. [50]

Las claves DES consisten en 64 dígitos binarios o bits(“0” o “1”). De estos 64 bits, 56 bits son generados aleatoriamente. Los 8 bits restantes son usados para detección de errores, por ejemplo, cada bit de estos 8 bits detectan la paridad de un bloque de 1 byte.

Si el receptor no tiene la misma clave, es imposible descifrar la información, pues para descifrarla es necesario tener la misma, sin embargo, si se puede obtener la información usando la fuerza bruta, es decir, comprobando todas las posibles claves una a una. Una persona puede tardar bastante en obtener la información, pero los ordenadores actuales tienen la suficiente capacidad como para obtener la información en relativamente poco tiempo.

El algoritmo de DES está diseñado para cifrar y descifrar bloques de datos consistentes en 64 bits bajo el control de claves de 64 bits, como se ha comentado anteriormente, numerados de izquierda a derecha, siendo el primer bit por la izquierda el bit 1. El descifrado se debe hacer con la misma clave con la que se ha cifrado pero con la colocación de la lista de los bits alterada de forma que el descifrado sea el proceso inverso al cifrado. Al bloque al que se va a cifrar se le somete a una permutación inicial (*IP*), después a una computación compleja dependiente de la clave y finalmente a una permutación que es la inversa de la permutación inicial (IP^{-1}). A la función dependiente de la clave (*f*), se le llama función de cifrado, a la función KS, se le llama Key Schedule. [54]

Funcionamiento del cifrado utilizando el algoritmo DES:

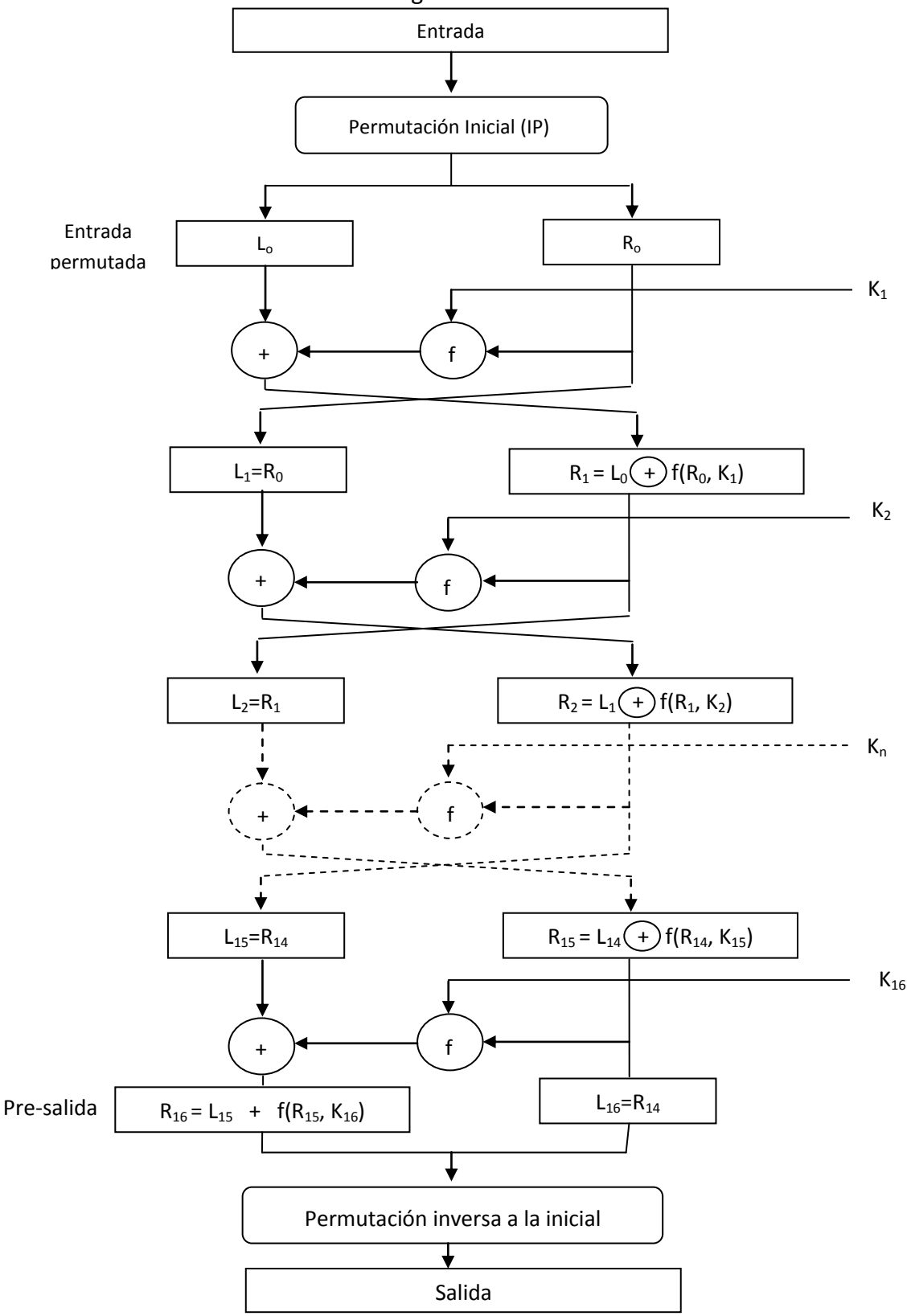


Figura B.1. Proceso de cifrado DES

El anterior esquema se explica de la siguiente forma:

Cifrado

Los 64 bits de entrada se someten a la siguiente permutación[26] incial:

IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Tabla B.1. Permutación inicial

El bit 58 sería el primero, el 50 el segundo y así sucesivamente. El bloque de entrada permutado, se convierte en la entrada de un cálculo dependiente de la clave que se describirá más adelante. La salida de este cálculo se somete a otra permutación que es la inversa de la primera:

IP^{-1}

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Tabla B.2. Permutación inversa

El cálculo que usa el bloque de entrada permutado como su entrada para producir el bloque de pre-salida consiste en un cálculo de 16 iteraciones que se describe más adelante en función de la función de cifrado *f* que opera en dos bloques, uno de 32 bits y otro de 48 bits, dando un bloque de 32 bits de salida.

El bloque de entrada consiste en un grupo L de 32 bits y otro grupo R de otros 32 bits. El bloque de entrada se puede expresar como LR.

El bloque k se compone de los primeros 48 bits de la clave de 64 bits. La salida L'R' para una entrada LR se define como:

[26] La permutación cambia el orden de los bit según se indica en la matriz, por ejemplo, en la primera matriz se coloca en la primera posición el bit de la posición 58 del texto original.

$$L' = R$$

$$R' = L \oplus f(R, K)$$

Donde $\oplus$ denota una suma bit a bit módulo 2.

Si L'R' es la salida de la decimo sexta iteración R'L' es el bloque de pre-salida. A cada iteración un bloque diferente de K bits de la clave se elige de la clave de 64 bits, designada por la palabra KEY en la formulación matemática.

Sea KS una función que toma como entero *n* en un rango de 1 a 16 y un bloque KEY de 64 bits como la entrada y obtiene como salida un bloque k_n de salida que es una selección permutada de bits de KEY:

$$k_n = KS(n, KEY)$$

Con K_n determinado por los bits en 48 posiciones de bit distintas de KEY. A KS se le llama key schedule donde el bloque K usado en la iteración n-ésima es el bloque K_n determinado por $k_n = KS(n, KEY)$

De esta forma:

$$L_n = R_{n-1}$$

$$R_n = L_{n-1} \oplus f(R_{n-1}, k_n)$$

La función *f* se calcula según el siguiente esquema:

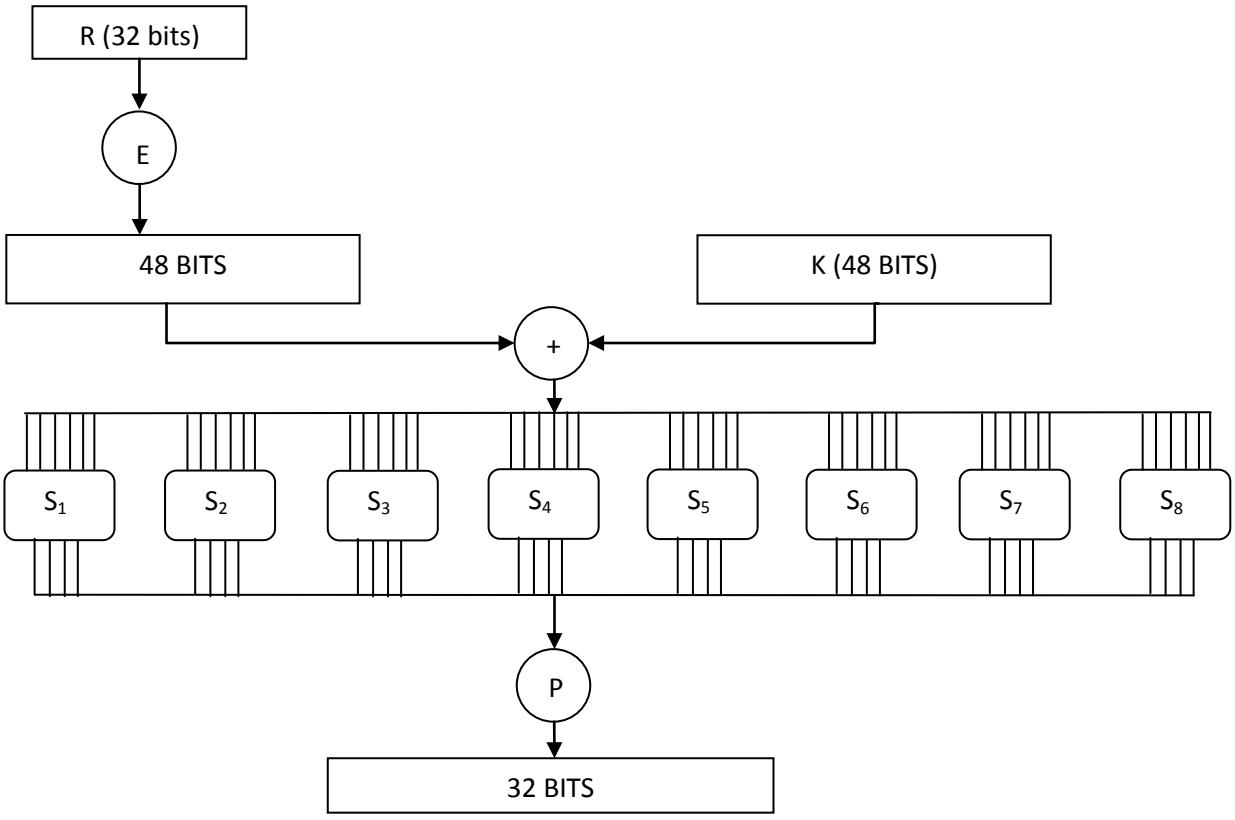


Figura B.2. Cálculo de *f(R,K)*

E es una función que toma como entrada 32 bits y obtiene a la salida 48 bits. La función *E* son 8 bloques con 6 bits cada bloque:

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Tabla B.3. Tabla de E

Para determinar la salida de los bloques S_i. Para un número binario de entrada de 6 bits, se elige el primer bit y el último como fila y los valores intermedios como el número de columna. Por ejemplo, si el valor de entrada es 011101, la fila sería 01 que en decimal es 1 y la columna sería 1110, que en decimal es la columna 14. Las funciones S_i se muestran a continuación:

S_1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Tabla B.4. Tabla de la función S_1

S_2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Tabla B.5. Tabla de la función S_2

S_3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Tabla B.6. Tabla de la función S_3

S_4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Tabla B.7. Tabla de la función S_4

S_5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Tabla B.8. Tabla de la función S_5

S_6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

Tabla B.9. Tabla de la función S_6

S_7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

Tabla B.10. Tabla de la función S_7

S_8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Tabla B.11. Tabla de la función S_8

La función primitiva P es:

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Tabla B.12. Tabla de la función primitiva P

Kn se calcula como se muestra en la siguiente figura:

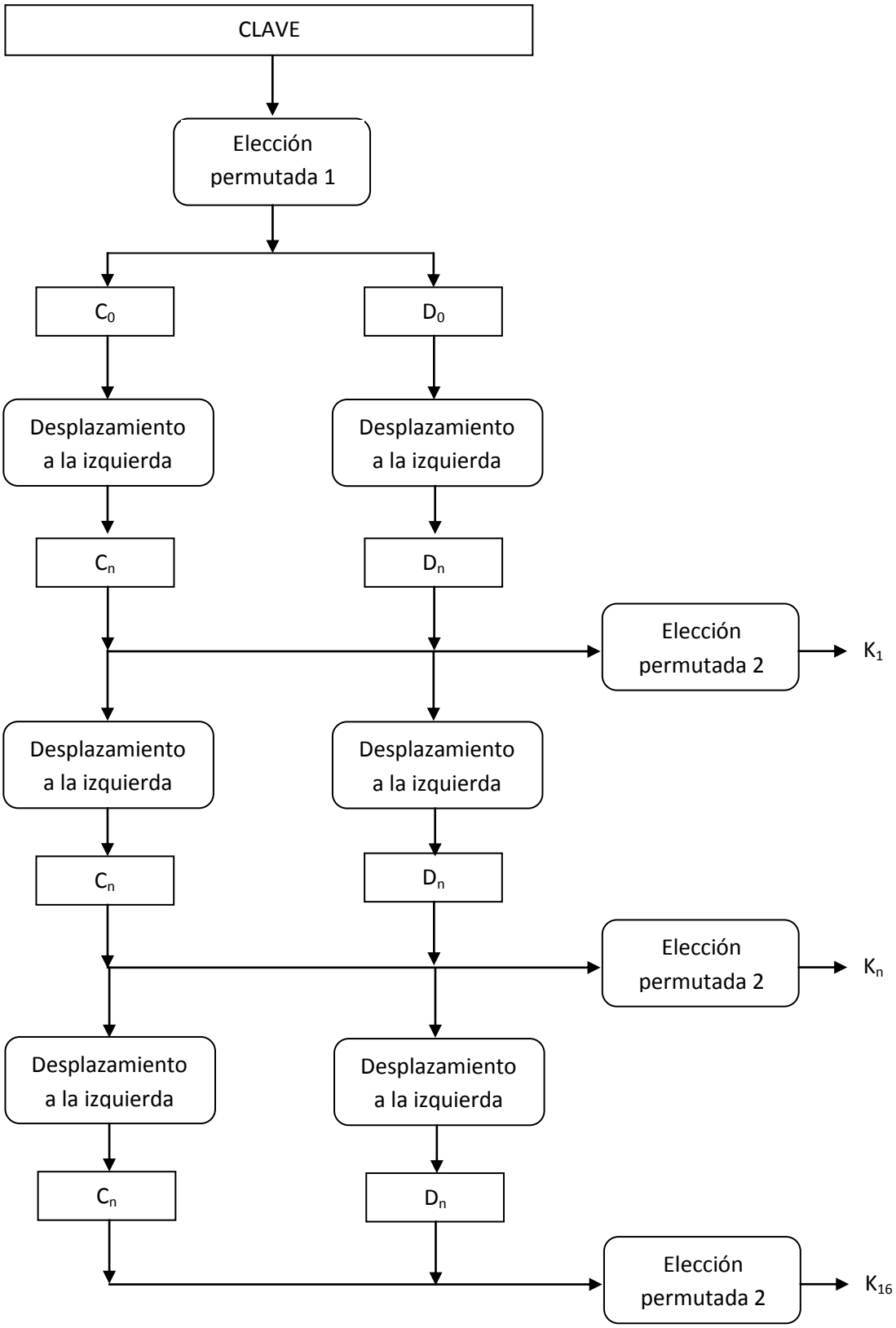


Figura B.3. Cálculo de K_n

Para la elección de la permutación 1 se utiliza:

***PC*-1**

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Tabla B.13. Tabla de elección de la permutación 1.

La elección de la permutación 2 está determinada por la siguiente tabla:

***PC*-2**

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Tabla B.14. Tabla de elección de la permutación 2.

Y por último, para saber el número de desplazamientos a la izquierda que hay que hacer se utiliza la siguiente tabla:

Número de Iteración	Número de desplazamientos
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2

15	2
16	1

Tabla B.15. Tabla de desplazamientos.

Descifrado

La permutación IP-1 aplicada al bloque de pre-salida es la inversa de la permutación inicial IP aplicada a la entrada.

Por tanto, para el descifrado sólo es necesario aplicar el mismo algoritmo del bloque del mensaje cifrado, comprobando que los mismos bits del bloque de clave son usados para el descifrado de la misma forma que fueron utilizados para el cifrado.

Y la ecuación utilizada, sigue siendo la misma:

$$L' = R$$

$$R' = L \oplus f(R, K)$$

Triple DES (Data Encription Standard)

Este cifrado se creó para subsanar los problemas de seguridad del cifrado DES. Consiste en la aplicación sucesiva durante tres iteraciones del algoritmo DES. Tiene una longitud efectiva de clave de 168 bits al usar tres veces una clave con longitud de 56 bits.

Sean $E_k(I)$ y $D_k(I)$ la representación de un cifrado y descifrado DES de una entra *I* y *k* su clave, cada operación de cifrado/descifrado del algoritmo Triple DES, se compone de operaciones de cifrado y descifrado DES siguiendo las siguientes operaciones:

1. Operación de cifrado del algoritmo TDES: la transformación de un bloque de entrada *I* de 64 bits en un bloque de salida *O,* también de 64 bits, que se define como sigue:

$$O = D_{k1}(E_{k2}(D_{k3}(I)))$$

Figura B.4. Proceso de cifrado de un algoritmo TDES

2. Operación de descifrado del algoritmo TDES: la transformación de un bloque de entrada *I* de 64 bits en un bloque de salida *O* de salida también de 64 bits:

$$O = D_{k1}(E_{k2}(D_{k3}(I)))$$

Figura B.5. Proceso de descifrado de un algoritmo TDES

El estándar TDES especifica las siguientes opciones para la utilización de las claves (K_1, K_2, K_3):

1. K_1, K_2 y K_3 son claves independientes.
2. K_1 y K_2 son claves independientes y $K_3 = K_1$.
3. $K_1 = K_2 = K_3$. Esta opción no es segura y es mejor que no se dé este caso.

A veces un mensaje cifrado con TDES puede ser descifrado por DES y viceversa. [54]

Los requisitos que deben de cumplir las claves son los siguientes:

a. Se deben mantener en secreto.
b. Deben ser generadas usando un método adecuado que se basa en la salida de un generador aleatorio de bits adecuado.
c. Deben ser independientes de otro conjunto de claves.
d. Se debe mantener la integridad de forma que cada clave del conjunto de claves no haya sido alterada de una forma no autorizada desde el tiempo en el que fueron generadas, transmitidas o almacenadas por una entidad autorizada.
e. Deben ser utilizadas en el orden apropiado según el modo particular que se haya especificado.
f. Una clave no puede ser manipulada dejando las otras dos claves sin cambios.

A continuación se muestran un conjunto de claves que se consideran inseguras en el algoritmo DES y que deben ser evitadas, por tanto, también en TDES:

- 01010101 01010101
- FEFEFEFE FEFEFEFE
- E0E0E0E0 F1F1F1F1
- 1F1F1F1F 0E0E0E0E

Algunos pares de claves cifran el texto obteniendo como resultado el mismo texto de origen y por tanto, también deben ser evitados:

- 011F011F010E010E y 1F011F010E010E01
- 01E001E001F101F1 y E001E001F101F101
- 01FE01FE01FE01FE y FE01FE01FE01FE01
- 1FE01FE00EF10EF1 y E01FE01FF10EF10E
- 1FFE1FFE0EFE0EFE y FE1FFE1FFE0EFE0E
- E0FEE0FEF1FEF1FE y FEE0FEE0FEF1FEF1 [55]

AES (Advanced Encryption Standard)

El estándar AES fue publicado por el Instituto Nacional de Estándares y Tecnología (NIST) en 2001. AES es un cifrado simétrico por bloques con el que se intenta remplazar a DES como el estándar aprobado por un amplio rango de aplicaciones. AES tiene una estructura compleja a diferencia de lo que ocurre con cifrados asimétricos como RSA. [54]

Cada entrada y salida del algoritmo AES consiste en 128 bits (dígitos con valores de "0" o "1"). Estas secuencias se conocen como bloques y el número de bits se refiere como su longitud. La clave de cifrado para el algoritmo AES es una secuencia de 128, 192 o 256 bits. No se permiten otras longitudes de entradas, salidas y claves. [56]

Los bits de estas secuencias se numeran de *0* a *n-1*, siendo *n* la longitud de la clave.

La unidad básica en la que funciona el algoritmo AES es el byte, conjunto de 8 bits. Tanto las entradas, salidas, como las claves se dividen en bytes. Para representar las entradas se utiliza la notación a_n. Donde *n* se encuentra entre los siguientes valores:

- Longitud de clave = 128 bits, $0 \leq n < 16$;
- Longitud de clave = 192 bits, $0 \leq n < 24$;
- Longitud de clave = 256 bits, $0 \leq n < 32$;

Por tanto, un array de bytes se representarán como $a_0\ a_1\ a_2 \ldots\ a_{15}$, *donde:*

$a_0 = \{entrada_1, entrada_2, \ldots, entrada_7\}$

$a_1 = \{entrada_8, entrada_9, \ldots, entrada_{15}\}$

…

$a_{15} = \{entrada_{121}, entrada_{122}, \ldots, entrada_{128}\}$

Donde cada una de las entradas representa un bit.

Todos los valores de byte en AES se representan como la concatenación de sus valores de bit individuales en el siguiente orden: b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0. Estos bytes se interpretan como un campo finito de elementos usando una representación polinomial:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_i x^i$$

Por ejemplo, {01100011} identifica al elemento $x^6 + x^5 + x + 1$.

Además conviene representar los bytes en numeración hexadecimal, de esta forma, el anterior número 01100011 se representaría con el número 63 en hexadecimal.

Internamente, las operaciones del algoritmo AES se realizan en dos arrays de dimensiones de bytes llamadas States. Un State consiste en cuatro filas de bytes, cada uno de los cuales contienen *Nb* Bytes, donde *Nb* es la longitud del bloque dividida por 32. En el array de State denotado por el símbolo s, cada byte individual tiene dos índices, con su número de fila en el rango de 0 ≤ r < 4 y su número de columna c en el rango de 0 ≤ c < Nb. Esto permite denotar a cada byte individual del State como $s_{r,k}$

Por tanto, los vectores de entrada (in_i), se convierten en los de la salida según el siguiente esquema:

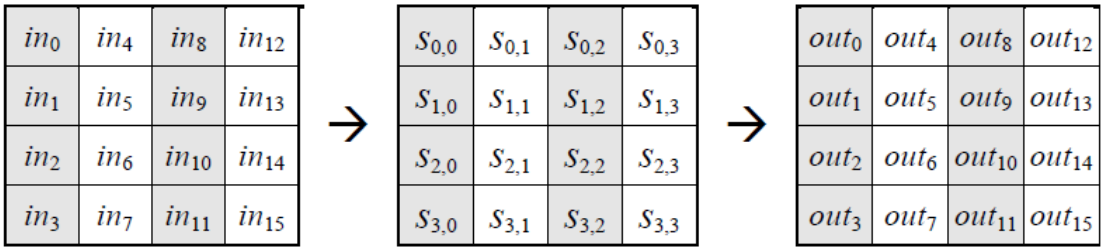


Figura B.6. Array de State, de entrada y de salida.

El vector de entrada se convierte en el vector de State siguiendo la siguiente fórmula:

s[r, c] = in[r + 4c] para 0 ≤ r < 4 y 0 ≤ c < Nb

y el vector de State se copia en el vector de la salida (out) de la siguiente forma:

out[r + 4c] = s[r, c] para 0 ≤ r < 4 y 0 ≤ c < Nb

Los cuatro bytes de cada columna del vector de State forman palabras de 32 bits, donde el índice *r* de las filas, representa también un índice para cada palabra de la siguiente forma:

$W_0=s_{0,0}\ s_{1,0}\ s_{2,0}\ s_{3,0}$ $W_2=s_{0,2}\ s_{1,2}\ s_{2,2}\ s_{3,2}$

$W_1=s_{0,1}\ s_{1,1}\ s_{2,1}\ s_{3,1}$ $W_3=s_{0,3}\ s_{1,3}\ s_{2,3}\ s_{3,3}$

En AES, los bloques de entrada son de 128 bits y por tanto, Nb =4.

La longitud de las claves de cifrado, K, son de 128, 192 y 256 bits, por lo que la longitud de la clave de cifrado es de Nk = 4, 6 y 8, que representa el número de palabras de 32 bits dentro de la clave de cifrado.

El número de rondas que se realizan durante la ejecución del algoritmo depende del tamaño de clave. El número de rondas se representa por Nr. Se muestra a

continuación una tabla con las correspondencias entre Nk, Nb y Nr según el tamaño de clave que se utilice:

	Longitud de clave (Nk)	**Tamaño de bloque (Nb)**	**Número de rondas (Nr)**
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Tabla B.16. Array de State, de entrada y de salida.

Tanto para el cifrado como para el cifrado inverso, el algoritmo AES usa una función de ronda que se compone de cuatro transformaciones orientadas a byte diferentes:

1) Sustitución de byte usando una tabla de sustitución (S-box)
2) Filas de desplazamiento del vector de State con diferentes offsets.
3) Mezcla de los datos de cada columna del vector de State.
4) Suma de una subclave al State.

Cifrado

Al principio del cifrado, la entrada se copia al array de State de la forma en la que se ha comentado anteriormente. Después de la suma inicial de la subclave, el vector de State es transformado mediante la implementación de una función 10, 12 o 14 veces (dependiendo de la longitud de la clave) cuya ronda final difiere ligeramente de las primeras *Nr-1* rondas. El State final se copia entonces a la salida como se ha comentado anteriormente.

Cada función de ronda se parametriza usando una key schedule[27] que consiste en un array de una dimensión de palabras de cuatro bytes derivados del uso de una rutina de expansión de clave.

[27] Una key Schedule es un algoritmo que, dado una clave, calcula las subclaves para esa ronda.

El cifrado se describe en el siguiente pseudo-código:

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)]) begin
byte state[4,Nb]

      state = in
      AddRoundKey(state, w[0, Nb-1])

      for round = 1 step 1 to Nr-1
      SubBytes(state)
      MixColumns(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      end for

      SubBytes(state)
      ShiftRows(state)
      AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

out = state end
```

Figura B.7. Pseudo-código de cifrado AES.

Todas las rondas *Nr* son iguales excepto la última que no incluye la transformación *MixColumns()*.

Transformación SubBytes().

Esta transformación es una sustitución de byte no linear que opera independientemente en cada byte del State usando una tabla de sustitución (S-box):

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Tabla B.17. Tabla (S-box) con los valores de sustitución x e y (expresados en hexadecimal)

Esta tabla es invertible, se construye realizando dos transformaciones:

1. Tomando una inversa multiplicativa en el campo finito GF(2^8). El elemento {00} se mapea a sí mismo.

Para cualquier polinomio binario distinto de cero b(x) con grado menor a 8, el inverso multiplicativo de b(x), denotado por $b^{-1}(x)$, se puede encontrar utilizando el algoritmo Euclídeo extendido, que se usa para calcular polinomios *a(x)* y *c(x)* tales que:

$b(x)a(x) + m(x)c(x) = 1$

Así pues $a(x) \bullet b(x) mod(m(x)) = 1$, lo que significa que:

$$b^{-1}(x) = a(x) \bmod m(x)$$

Una multiplicación en GF(2^8) (denotado por ●) corresponde con la multiplicación de módulos polinomiales con un polinomio irreducible de grado 8. Este polinomio irreducible (m(x)), es el siguiente:

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

2. Aplicar la siguiente transformación afín (sobre GF(2)):

$$b_i' = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

Para $0 \leq i < 8$, donde b_i es el i-ésimo bit del byte y c_i es el i-ésimo bit de un byte c con el valor {63} o {01100011}.

En forma de matriz, el elemento de la transformación afín de S-box puede ser expresado como:

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Los efectos de la transformación subBytes() se puede ver en la siguiente imagen:

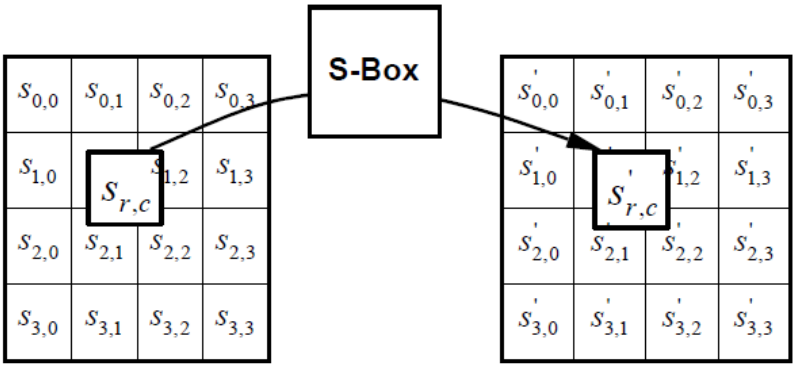


Figura B.8. Transformación subBytes() a cada byte del State.

Por ejemplo si buscamos el elemento $S_{1,1}$={53}, entonces el valor de sustitución corresponde con la intersección de la fila 5 con la columna 3, que daría como resultado el valor {ed} (Tabla C.2)

Transformación ShiftRows()

En esta transformación los bytes en las últimas tres filas del State son cíclicamente desplazados sobre diferentes números de bytes (offsets). La primera fila, *r=0,* no es desplazada. Específicamente, la transformación ShiftRows() funciona de la siguiente forma:

$$S'_{r,c} = S_{r,(c+shift(r,Nb)) \bmod Nb}$$ Para 0 < r < 4 y 0 ≤ c < Nb

Donde el valor de desplazamiento *shift (r, Nb)*, depende en el número de fila, *r*, de la siguiente forma (Nb = 4):

- Shift (1, 4) = 1;
- Shift (2, 4) = 2;
- Shift (3,4) = 3;

Esto tiene el efecto de mover bytes de posiciones más bajas en la fila, mientras que los valores más bajos suben a posiciones más altas en la fila. Esto se muestra en la siguiente figura:

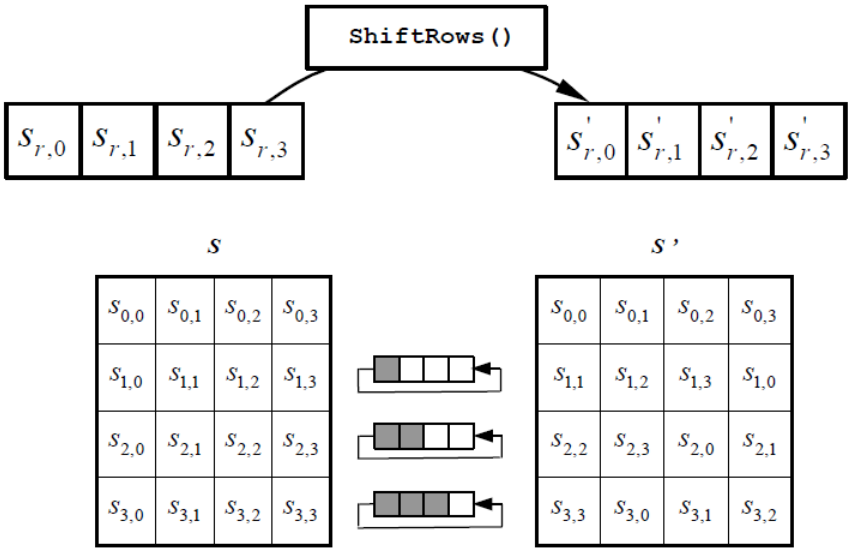


Figura B.9. Transformación ShiftRows()

Transformación MixColumns()

Esta transformación opera en el State columna a columna, tratando cada columna como un polinomio de cuatro términos. Las columnas son consideradas como polinomios sobre GF(2^8) y multiplicado por el módulo de x^4+1 con un polinomio fijo *a(x)*, dado por:

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

Esto se puede escribir como una multiplicación de matrices. Sea $s'(x)= a(x) \otimes s(x)$:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \qquad \text{Para } 0 \le c < Nb$$

Como resultado de esta multiplicación, los cuatro bytes de una columna se remplazan por los siguientes resultados:

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

La siguiente imagen muestra de forma gráfica el comportamiento de la transformación MixColumns:

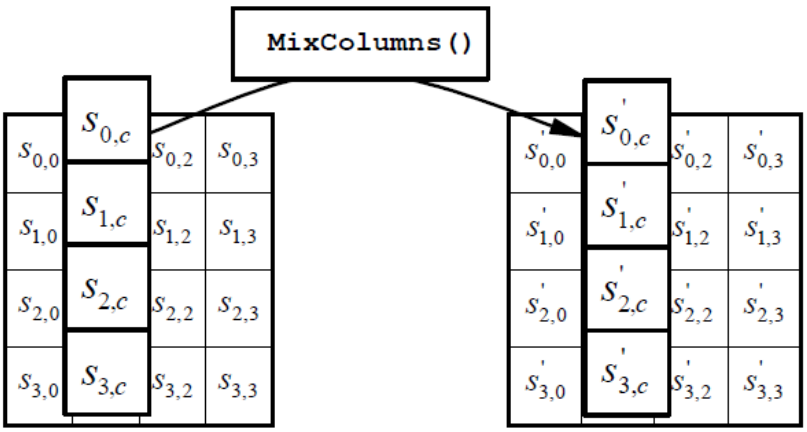


Figura B.10. Transformación MixColumns().

Transformación AddRoundKey()

En esta transformación una subclave se suma al State por una simple operación XOR bit a bit. Cada subclave consiste en *Nb* palabras de la key schedule. Éstas *Nb* palabras son sumadas a las columnas de State de la siguiente forma:

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{round*Nb+c}]$$ Para 0 ≤ c < Nb

Donde $[w_i]$ son las palabras de las key schedules que se describen más adelante y round es un valor en el rango de 0 ≤ round ≤ Nr. En el cifrado, la suma inicial de la subclave ocurre cuando round = 0, antes de la aplicación de la función de ronda. La aplicación de la transformación AddRoundKey() a la ronda Nr del cifrado ocurre cuando 1 ≤ round ≤ Nr.

El funcionamiento de esta transformación (donde *l = round*Nb*) se muestra en la siguiente imagen:

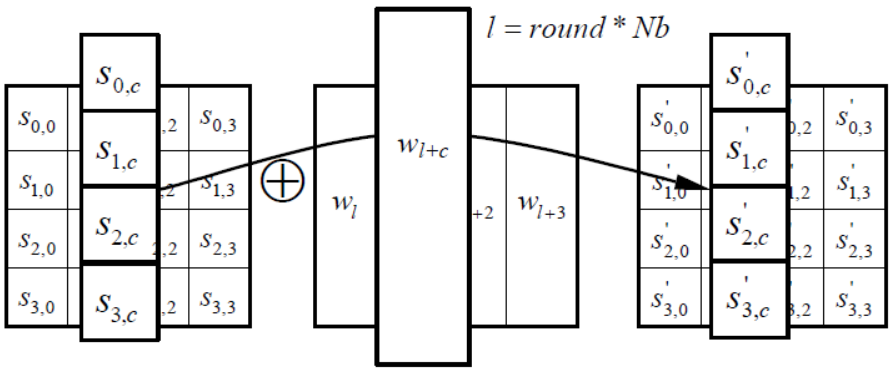


Figura B.11. Transformación AddRoundKey()

Expansión de clave

El algoritmo AES toma la clave de cifrado, K, y realiza una rutina de expansión de clave para generar una key Schedule. La expansión de la clave genera un total de *Nb(Nr + 1)* palabras: el algoritmo requiere un conjunto inicial de *Nb* palabras, y cada uno de las Nr rondas requiere *Nb* palabras de datos de clave. La key schedule resultante consiste en un array linear de palabras de 4 bytes, denotadas por $[w_i]$, con *i* en el rango de $0 \leq i < Nb(Nr + 1)$.

La expansión de la clave de entrada en el key Schedule funciona de acuerdo al siguiente pseudo-código:

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)]) begin byte
state[4,Nb]

      state = in

      AddRoundKey(state, w[0, Nb-1])

      for round = 1 step 1 to Nr-1
      SubBytes(state)
      ShiftRows(state)
      MixColumns(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      end for

      SubBytes(state) ShiftRows(state) AddRoundKey(state, w[Nr*Nb,
      (Nr+1)*Nb-1])

out = state end
```

Figura B.12. Pseudo-código de expansión de clave.

SubWord() es una función que toma una palabra de entrada de 4 bytes y aplica la S-box a cada uno de los cuatro bytes para producir una palabra de salida. La función RotWord() toma una palabra $[a_0, a_1, a_2, a_3]$ como entrada, realiza una permutación cíclica y devuelve la palabra $[a_1, a_2, a_3, a_0]$. *Rcon[i]*, contiene los valores dados por $[x^{i-1}, \{00\}, \{00\}, \{00\}]$ con x^{i-1} siendo potencias de *x* (*x* se denota como {02}) en el campo GF(2^8).

Las primeras *Nk* palabras de la clave expandida son rellenadas con la clave de cifrado. Todas las siguientes palabra, w[[i]], es igual al XOR de las palabras previas, w[[i-1]], y la palabra *Nk* posiciones anteriores, w[[i-Nk]]. Para palabras en posiciones que son un múltiplo de *Nk*, se aplica una transformación a w[[i-1]] antes de realizar la XOR, seguida de una XOR con una constante round, *Rcon[i].* Esta transformación consiste en un desplazamiento cíclico de los bytes en una palabra (RotWord ()), seguido de la aplicación de una tabla de lookup a los cuatro bytes de la palabra (SubWord()).

La rutina de expansión de clave para claves de cifrado de 256 bits (Nk = 8) es ligeramente diferente que para claves de cifrado de 128 y 192 bits. Si Nk = 8 e i-4 es un múltiplo de Nk, entonces SubWord() se aplica a w[[i-1]] previamente al XOR.

Descifrado

La transformación hecha por el cifrado, puede ser invertido e implementado en orden inverso para producir un cifrado inverso sencillo para el algoritmo AES.

El cifrado inverso se describe en el siguiente pseudo-código:

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)]) begin
byte state[4,Nb]

     state = in

     AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

     for round = Nr-1 step -1 downto 1
     InvShiftRows(state)
     InvSubBytes(state)
     AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
     InvMixColumns(state)
     end for

     InvShiftRows(state) InvSubBytes(state) AddRoundKey(state, w[0,
     Nb-1])

out = state end
```

Figura B.13. Pseudo-código de descifrado.

Las funciones InvShiftRows(), InvSubBytes(), AddRoundKey() e InvMixColumns() realizan las funciones inversas a las funciones SubBytes(), ShiftRows(), MixColumns(), AddRoundKey().

RC5

RC5 es un cifrado por bloques que fue creado por Ronald Rivest en 1994 y fue diseñado de forma que debería de cumplir los siguientes objetivos:

- Debería ser un cifrado simétrico por bloques. La misma clave es usada tanto para el cifrado como para descifrado. Tanto el mensaje plano como el mensaje cifrado tienen una secuencia de longitud fija de bits.
- Debe servir tanto para hardware como para software.
- Debe ser rápido. Lo que viene a indicar que debe ser orientado a palabra: las operaciones básicas computacionales deberían ser operadores que trabajan con palabras completas de datos.
- Debe ser adaptables a procesadores de diferentes longitudes de palabras.

- Debe tener una estructura iterativa, con un número variable de rondas. Lo que permite elegir entre una velocidad más rápida o una seguridad mayor.
- Debe tener una clave criptográfica de longitud variable.
- Debe ser simple y fácil de implementar.
- Debe necesitar poca cantidad de memoria, de forma que sea fácil de implementar en smart cards u otros dispositivos con restricciones de memoria.
- Debe proporcionar alta seguridad. [57]

RC5 se compone de tres elementos: expansión de clave y algoritmo de cifrado y descifrado. El mensaje de entrada a RC5 consiste en dos palabras (A y B) de una longitud de *ω* bits. Utiliza una tabla de clave expandida, S[0…t-1] consistente en *t=2(r+1)* palabras de *ω* bits. El algoritmo de expansión de clave inicializa *S* a partir de un parámetro *K*, clave secreta de un usuario. Los bloques de entrada/salida se expresa en little-endian, es decir, el primer byte ocupa las posición más baja de los registros.

Cifrado

Se asume que el bloque de entrada se da en dos registros (A y B) de *ω* bits. Se asume también que la expansión de clave ya se ha realizado, de tal forma que el vector S[0…t-1] ya ha sido calculado. El pseudo-código que realiza el cifrado es el siguiente:

$$
\begin{aligned}
&A = A + S[0];\\
&B = B + S[1];\\
&\textbf{for } i = 1 \textbf{ to } r \textbf{ do}\\
&\qquad A = ((A \oplus B) \ll B) + S[2 * i];\\
&\qquad B = ((B \oplus A) \ll A) + S[2 * i + 1];
\end{aligned}
$$

Cada ronda del algoritmo RC5 actualiza los registros A y B donde se almacena la salida de los mensajes cifrados.

Descifrado

La rutina de descifrado es fácilmente derivable de la rutina de cifrado:

$$
\begin{aligned}
&\textbf{for } i = r \textbf{ downto } 1 \textbf{ do}\\
&\qquad B = ((B - S[2 * i + 1]) \gg A) \oplus A;\\
&\qquad A = ((A - S[2 * i]) \gg B) \oplus B;\\
&B = B - S[1];\\
&A = A - S[0];
\end{aligned}
$$

Expansión de clave

La rutina de expansión de clave, expande la clave secreta *K* del usuario para rellenar el array de claves expandidas *S,* de forma que *S* parece un array de *t=2(r+1)* palabras

aleatorias binarias determinadas por *K.* El algoritmo de expansión de claves usa dos "constantes mágicas" que consisten en tres partes algorítmicas simples.

Definición de la constante mágica

El algoritmo de expansión de clave usa constantes binarias de tamaño de palabra P_ω y Q_ω. Se definen con una ω arbitraria como sigue:

$$P_\omega = Odd((e-2)2^\omega)$$

$$Q_\omega = Odd((\emptyset-2)2^\omega)$$

Donde

$$\mathrm{e} = 2.718281828459\ldots$$

$$\emptyset = 1.618033988749\ \ldots$$

Y donde Odd(*x*) es el entero impar más próximo a *x* (redondeado hacia arriba si *x* es un entero par). Para ω = 16, 32 y 64 se muestran a continuación las constantes P y Q en binario y hexadecimal:

P_{16} = 1011011111100001 = b7e1
Q_{16} = 1001111000110111 = 9e37

P_{16} = 10110111111000010101000101100011= b7e15163
Q_{16} = 10011110001101110111100110111001 = 9e3779b9

P_{16} =
10110111111000010101000101100010110000101000101011101101001010100110101
11 = b7e151628aed2a6b

Q_{16} =
10011110001101110111100110111001101110010111111101001010011111000001010
101 = 9e3779b97f4a7c15

Convertir la clave secreta de bytes a palabras.

El primer paso del algoritmo de expansión de clave es copiar la clave secreta *K[0…b - 1]* a un array *L[0…c - 1]* de *c =[b/u]* palabras, donde *u=ω/8* es el número de bytes/palabra. Esta operación se hace de una forma natural, usando *u* bytes de la clave *K* consecutivos para rellenar cada una de las palabras sucesivas en *L,* de un byte de orden menor a un byte de mayor orden. Cualquier posición de byte de *L* que no esté relleno, se rellena con un cero. En el caso en el que *b = c = 0* se reinicia *c* a 1 y se inicializa *L[0]* a cero.

El siguiente pseudo-código muestra el comportamiento que tiene que tener esta parte del algoritmo, suponiendo que todos los bytes son sin signo y que el array *L* está inicializado a cero.

$$c = \lceil \max(b, 1)/u \rceil$$
for $i = b - 1$ **downto** 0 **do**
$$L[i/u] = (L[i/u] \lll 8) + K[i];$$

Inicialización del Array S.

El segundo paso del algoritmo de la expansión de clave es inicializar el array *S* a un patrón de bit pseudo-aleatorio particular (independiente de clave), usando una progresión aritmética de módulo 2^{ω} determinado por las "constantes mágicas" P_{ω} y Q_{ω}. Puesto que Q_{ω} es impar, la progresión aritmética tiene un periodo de 2^{ω}:

$$S[0] = P_w;$$
for $i = 1$ **to** $t - 1$ **do**
$$S[i] = S[i-1] + Q_w;$$

Maximización de la clave secreta

El tercer paso del algoritmo de expansión de clave es mezclar en la clave secreta del usuario en tres pasadas los arrays *S* y *L*. Más precisamente, debido a los tamaños potencialmente diferentes de *S* y *L*, el array más grande será procesado en tres veces, y el otro puede ser realizado más veces.

$$i = j = 0;$$
$$A = B = 0;$$
do $3 * \max(t, c)$ **times:**
$$A = S[i] = (S[i] + A + B) \lll 3;$$
$$B = L[j] = (L[j] + A + B) \lll (A + B);$$
$$i = (i + 1) \bmod (t);$$
$$j = (j + 1) \bmod (c);$$

IDEA (International Data Encryption Algorithm)

Es un cifrado simétrico por bloques desarrollado por XuejiaLai y James Massey del Instituto Federal Suizo de Tecnología en 1991 y fue pensado como remplazo del algoritmo DES. Se suele utilizar en el protocolo PGP, del cual se habla en el apartado *2.3.4*.

IDEA usa una clave de 128 bits. Difiere de otros algoritmos al no utilizar S-box, sino tres operaciones matemáticas: XOR, suma binaria de enteros de 16 bits (suma módulo 2^{16}) y multiplicación de enteros de 16 bits (multiplicación módulo $2^{16}+1$), es decir, no hay operaciones a nivel de bit, lo que además facilita su programación en alto nivel. El algoritmo de generación de subclaves recae solamente en el uso de desplazamientos

circulares pero los usa de una forma compleja de forma que se generan un total de seis subclaves para cada una de las ocho rondas que realiza IDEA. En cada ronda se modifican todos los bits de bloque y no solamente la mitad como ocurre en DES.

El funcionamiento de IDEA se muestra en la siguiente imagen:

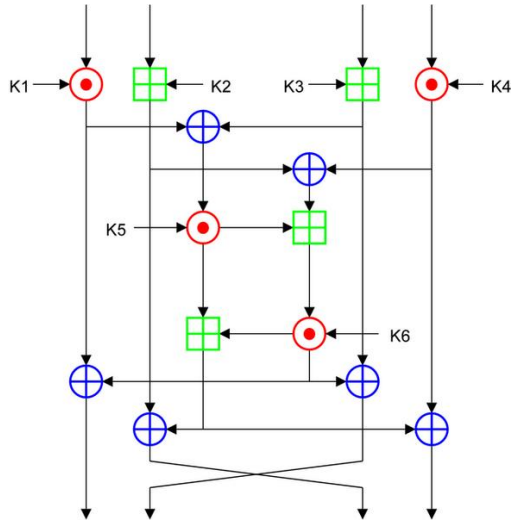


Figura B.14. Ronda de cifrado del algoritmo IDEA

En donde:

- ⊙ es un multiplicador $2^{16} - 1$
- ⊞ es el sumador módulo 2^{16}
- ⊕ XOR bit a bit.

Blowfish

Blowfish es un cifrado por bloques de 64 bits que fue desarrollado en 1993 por Bruce Schneier y pronto se convirtió en una alternativa a DES. Fue diseñado para ser fácil de implementar y para tener una alta velocidad de ejecución. También es un algoritmo muy compacto,º lo que permite que pueda ejecutarse en menos de 5 K de memoria. Otra característica es que la longitud de la clave puede ser variable llegando a los 448 bits. Pero normalmente se usan claves de 128 bits. El algoritmo realiza 16 iteraciones.

Este algoritmo usa S-boxes y la función XOR al igual que hace DES, pero también usa sumas binarias. A diferencia de DES que usa S-boxes[28] fijas, Blowfish usa S-boxes dinámicas que se generan como una función de la clave. Las subclaves y las S-boxes se generan repitiendo el algoritmo de Blowfish a la propia clave. Un total de 521 ejecuciones del algoritmo de cifrado de Blowfish se necesitan para producir las

[28] Las S-boxes son unas tablas de sustitución, a veces con formas de tablas de lookup, que permiten hacer que sea más difícil de comprender la relación entre un mensaje plano y el mensaje cifrado. En los anexos B, C y D se habla de las S-boxes

subclaves y las S-box. Blowfish no es recomendable en aplicaciones que cambian frecuentemente la clave secreta. [4][22]

En la figura X se muestra el funcionamiento de Blowfish. P es un array de 18 entradas y 4 S-box de 256 entradas. Una entrada del array P es usada cada ronda, después de la ronda final, a cada mitad del bloque de datos se le aplica una XOR con una de las dos entradas del array P que no han sido utilizadas. Cada array de P contiene r + 2 número de subclaves de 32 bits, donde r es el número de rondas.

La función divide las entradas de 32 bits en 4 bloques de 8 bits, y usa los bloques como entradas para las S-box. Las salidas deben estar en módulo 2^{32} y se les aplica un Xor para producir una salida final de 32 bits.

Las siguientes imágenes muestran el funcionamiento de Blowfish y de la función interna de Blowfish:

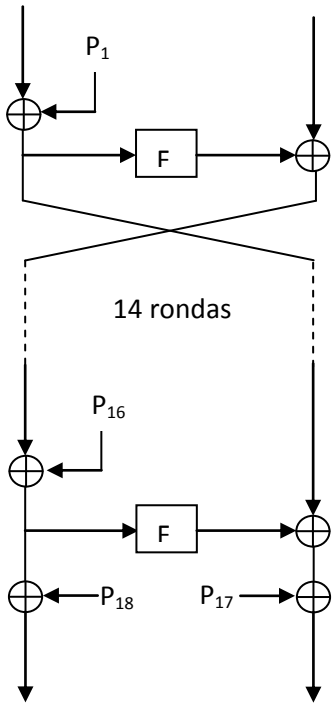


Figura B.15. Cifrado Blowfish

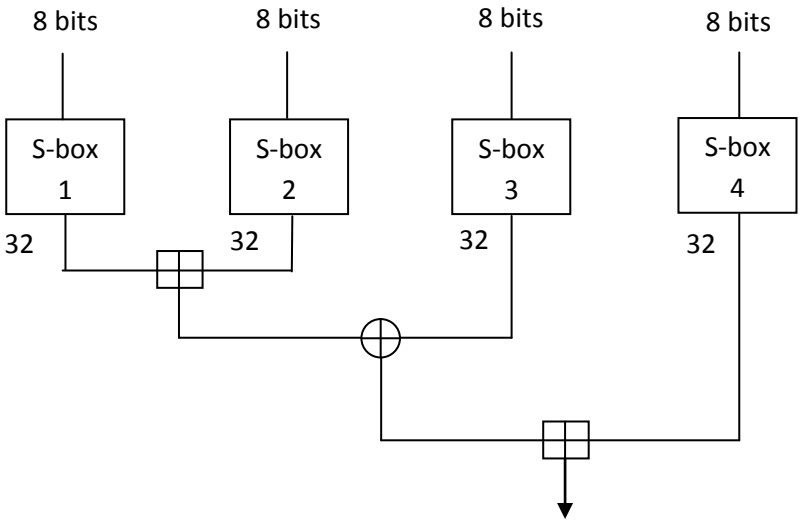


Figura B.16. Función de ronda dependiente de clave

Resumen de los métodos de cifrado simétrico

A continuación se muestra una tabla resumen que contiene las principales características de los métodos de cifrado simétrico aquí expuestos:

Algoritmo	Tamaño de clave (bits)	Tamaño de bloques (bits)	Número de rondas
DES	56	64	16
Triple DES	112 o 168	64	48
AES	128, 192 o 256	128	10,12 o 14
IDEA	128	64	8
Blowfish	Variable hasta 448	64	16
RC5	Variable hasta 2048	64	Variable hasta 255

Tabla B.18. Resumen de cifrados simétricos [4]

Existen más métodos de cifrado simétrico de los que aquí se han comentado, sin embargo, estos son los más utilizados actualmente.

Anexo C

Algoritmos de criptografía asimétrica y algoritmos de firma

ANEXO C.

ALGORITMOS DE CRIPTOGRAFÍA ASIMÉTRICA Y ALGORITMOS DE FIRMA.

Diffie-Hellman

Fue el primer algoritmo de clave asimétrica que apareció. Se emplea para acordar una clave común entre dos interlocutores que no han tenido contacto previo, a través de un canal de comunicación inseguro y de manera anónima. No son necesarias claves públicas en el sentido estricto.

Se emplea como medio para acordar claves simétricas que serán empleadas para el cifrado de una sesión.

Una de sus vulnerabilidades radica en la posibilidad de que una tercera persona intercepte la información. [5][23]

El algoritmo Diffie-Hellman depende de la dificultad de calcular logaritmos discretos. Si *a* es una raíz primitiva[29] de un número primo *p,* entonces los números:

$$a \bmod p,\ a^2 \bmod p,\ \ldots,\ a^{p-1} \bmod p$$

son distintos y consisten en enteros en el rango de *1* a *p-1*.

Para cualquier entero *b* y una raíz primitiva *a* de un número primo *p,* podemos encontrar un único exponente *i* tal que:

$$b \equiv a^i (mod\ p) \qquad donde\ 0 \leq i \leq (p-1)$$

El exponente *i* es al que llamamos logaritmo discreto de *b* para la base *a,* mod *p.* Se expresa este valor como:

$$d \cdot log_{a,p}(b)$$

Para el esquema de funcionamiento de este algoritmo, existen dos números conocidos públicamente: un número primo q y un entero α que es la raíz primitiva de q. Si el emisor (A) quiere intercambiar una clave con el receptor (B). El usuario A selecciona un entero aleatorio tal que $X_A < q$ y calcula:

$$Y_A = \alpha^{X_A} (mod\ q).$$

[29] Se dice que *a* es una raíz primitiva módulo *n* si es capaz de generar un conjunto de números invertibles. Una raíz primitiva es aquella que cumple $a^g \equiv 1 (mod\ n)$, si mcd(a, n)=1 y si a≠n y n es un número primo.

De la misma forma, el usuario B selecciona independientemente un entero aleatorio $X_B<q$ y calcula:

$$Y_B = \alpha^{X_B}(mod\ q).$$

Cada parte de la comunicación mantiene la parte de X secreta y mantiene pública la parte Y.

El usuario A calcula la clave como:

$$K = (Y_B)^{X_A}\ mod\ q$$

El usuario B calcula la clave de igual forma:

$$K = (Y_A)^{X_B}\ mod\ q$$

Estás dos fórmulas dan el mismo resultado.

La seguridad de este algoritmo recae en que aunque es relativamente fácil calcular exponentes módulo de un número primo, es muy difícil calcular logaritmos discretos, por lo que es impracticable para números primos grandes, ya que una persona que quisiera averiguar la clave privada de otra para descifrar el mensaje, tendría que calcular:

$$X = d \cdot log_{\alpha,q}(Y)$$

RSA (Rivest, Shamir, Adleman)

Este algoritmo fue desarrollado en 1977 por Ron Rivest, Adi Shamir y Len Adleman en el MIT.

El esquema RSA es un cifrado de bloques en el que el mensaje plano y el mensaje cifrado son enteros entre *0* y *n-1* dado un valor de *n*. Un tamaño típico para *n* es 1024 bits o 309 dígitos decimales.

Las claves obtenidas a partir de RSA sirven tanto para codificar como para autenticar. Es uno de los algoritmos asimétricos más seguros. Se basa en la dificultad para factorizar grandes números. Sin embargo, existen ciertos casos para los cuales el algoritmo RSA deja el mensaje original tal cual. Las claves pública y privada se calculan a partir de un número que se obtiene como producto de dos números primos grandes.

Actualmente se considera segura una clave RSA con una longitud de al menos 1024 bits, si bien se recomienda el uso de claves no inferiores a 2048 bits.

Con el algoritmo RSA nunca se debe firmar un mensaje después de codificarlo. Existen ataques que permiten manipular con éxito mensajes primero codificados y luego firmados, aunque se empleen primero funciones resumen. [23][58][61]

A continuación se explica, la forma en la que RSA funciona a la hora de cifrar un mensaje (M), usando un cifrado de clave pública (*e, n*)[30] y sus fundamentos matemáticos.

El mensaje se representa como un entero entre *0* y *n-1.* Se divide un mensaje largo en una serie de bloques y se representa cada bloque como tal entero.

Para cifrar el mensaje (*M*) se eleva a la potencia de e módulo n. El resultado (el mensaje cifrado *C*) es el resto cuando M^e se divide por *n.* El algoritmo de cifrado (*E*) es el siguiente:

$$C = E(M) = M^e (mod\ n)$$

Para descifrar el mensaje cifrado, se debe elevar a otra potencia *d* módulo *n*, como muestra el algoritmo de descifrado (*D*):

$$D(C) = C^d (mod\ n)$$

El cifrado no aumenta el tamaño de un mensaje, tanto el mensaje como el texto cifrado son enteros en el rango de *0* a *n-1.*

Así, la clave de cifrado es el par de enteros positivos (*e, n*) y la clave de descifrado es el par de enteros positivos (*d, n*). Cada usuario hace su clave de cifrado pública y mantiene su correspondiente clave de descifrado de forma privada.

Se calcula *n* como el producto de dos números primos *p* y *q:*

$$n = p \cdot q$$

p y *q* son dos números primos aleatorios. Aunque *n* se hace público, los factores *p* y *q* serán mantenidos de forma privada debido a que dificulta la factorización de *n*. Esto oculta la forma en la que *d* deriva de *e.*

Se debe escoger un entero *d* que sea grande y aleatorio y que satisfaga que:

$$m.c.d\left(d, (p-1)\cdot(q-1)\right) = 1$$

Donde m.c.d es el máximo común divisor.

El entero *e* se calcula a partir de *p, q* y *d* para que sea el multiplicativo inverso de *d* módulo *(p-1)·(q-1).* Así pues tenemos que:

$$e \cdot d \equiv 1 (mod (p-1)\cdot(q-1))$$

[30] e y n hacen referencia a un par de enteros positivos.

Fundamentos matemáticos

Se demuestra que es correcto el uso del algoritmo de descifrado usando una identidad de Euler y Fermat.

Para cualquier entero (mensaje) M que es relativamente primo a *n:*

$$M^{\varphi(n)} \equiv 1 \ (mod\ n) \qquad (1)$$

φ(n) es la función φ de Euler o la función indicatriz de Euler que da el número de enteros positivos menores a *n* que son relativamente primos a *n.* Para números primos *p*:

$$\varphi(p) = p - 1$$

Por propiedades elementales de la función de Euler:

$$\varphi(n) = \ \varphi(p) \cdot \varphi(q) = (p-1) \cdot (q-1) = n - (p+1) + 1$$

Puesto que *d* es relativamente prima a φ(n), tiene un multiplicador inverso *e* entorno a enteros módulo φ(n):

$$e \cdot d \equiv 1 (mod\ \varphi(n))$$

La comprobación de que el descifrado funciona correctamente si *e* y *d* se eligen como se ha explicado, se muestra a continuación:

$$D\big(E(M)\big) \equiv \big(E(M)\big)^d \equiv (M^e)^d \equiv M^{e \cdot d} (mod\ n)$$

$$E\big(D(M)\big) \equiv \big(D(M)\big)^e \equiv (M^d)^e \equiv M^{e \cdot d} (mod\ n)$$

y

$$M^{e \cdot d} \equiv M^{k \cdot \varphi(n)+1} (mod\ n) \quad para\ un\ entero\ k$$

De (1) observamos que para todos los M tal que *p* no divida a M:

$$M^{p-1} \equiv 1 \ (mod\ p)$$

Y puesto que *(p-1)* divide a φ(n)

$$M^{k \cdot \varphi(n)+1} \equiv M (mod\ p)$$

Esto es cierto cuando M ≡ 0 (mod *p*), así esta igualdad realmente se cumple para todos los *M*. De manera similar, para *q*:

$$M^{k \cdot \varphi(n)+1} \equiv M (mod\ q)$$

Estas dos últimas ecuaciones implican que para todo M:

$$M^{e \cdot d} \equiv M^{k \cdot \varphi(n)+1} \equiv M (mod\ n)$$

Esto demuestra el funcionamiento de cifrado y descifrado para todo M, 0 ≤ M < n.

Algoritmo

Cifrado

Calcular $M^e\ (mod\ n)$ requiere al menos $2 \cdot log_2(e)$ multiplicaciones y el mismo número de divisiones usando el siguiente procedimiento:

1) Sea $e_k e_{k-1} \ldots e_1 e_0$ una representación binaria de *e.*
2) Inicializar el valor de *C* a 1.
3) Repetir los pasos a) y b) para *i=k, k-1, …,0:*
    a. Inicializar C al resto de C^2 cuando se divide por *n.*
    b. Si $e_i = 1$, inicializar C al resto de C·M cuando es dividido por *n.*
4) C ahora es la forma cifrada de M.

Este procedimiento se conoce como "exponenciación por repetición del cuadrado y multiplicación".

El hecho de que el cifrado y el descifrado se realicen de idéntica forma, permite una implementación sencilla.

DSA (Digital Signature Algorithm)

Este algoritmo es usado por el estándar DSS (Digital Signature Standar) [61] para firmar las funciones hash, al igual que otros algoritmos como RSA.

Sea *L* un múltiplo de 64 en el rango de $512 \leq L \leq 1024$, *p* un número primo de *L* bits, esto es $2^{L-1} < p < 2^L$ y sea *q* un número primo de 160 bits que divide a *p-1*. Sea *h* una raíz primitiva módulo *p* en el intervalo $1 < h < p-1$ y $g = h^{(p-1)/q}\ mod\ p$, entonces *g* tiene orden *q* módulo *p.*

Al igual que en Diffie-Hellman, DSA asume que los logaritmos discretos módulo *p* son difíciles de calcular.

Los usuarios eligen una clave privada que mantienen en secreto *x* en el rango entre $1 < k < q$ y hace pública su clave pública $y = g^x\ mod\ p$. Cada vez que un usuario quiere firmar un mensaje *M*, elige un número aleatorio secreto *k* en el rango entre $1 < k < q$ y calcula una hash de *M, h(M)*.

El usuario A firma un mensaje *M* con el par *r, s,* donde $r = (g^k\ mod\ p)\ mod\ q$, y $s = [k^{-1}(h(M)+xr)] mod\ q$.

Si el usuario B recibe el mensaje *M'* con firma *r', s'* de Alice, el verifica su firma calculando:

$$\omega = (s')^{-1} \, mod \, q$$

$$u_1 = [h(M')\omega] \, mod \, q$$

$$u_2 = (r')\omega \, mod \, q$$

$$v = [(g^{u_1} y^{u_2}) mod \, p] mod \, q$$

y es la clave pública del usuario A.

Si se cumple que *v = r'*, entonces el usuario B acepta que *M'=M* es realmente un mensaje enviado a él por el usuario A.

ElGamal Digital Signature

Los elementos más importantes que utiliza este algoritmo son un número primo *q* y α que es la raíz primitiva de *q*.

El emisor A genera un par de claves pública y privada de la siguiente forma:

1. Se genera un entero aleatorio X_A, tal que $1 < X_A < q-1$.
2. Se calcula $Y_A = \alpha^{X_A} \, mod \, q$
3. La clave privada de A es X_A; la clave pública de A es {*q*, α, Y_A }

Para firmar un mensaje *M,* el usuario *A* primero calcula la hash m=H(M), tal que *m* es un entero en el rago de $0 \leq m \leq q-1$. Entonces A realizar una firma digital de la siguiente forma:

1. Se elige un entero aleatorio *K* tal que $1 \leq K \leq q-1$ y *mcd(K, q-1)=1.* Esto es, *K* es relativamente primo a *q-1.*
2. Se calcula $S_1 = \alpha^K \text{ mod } q$.
3. Se calcula el inverso de *K* módulo *q -1* ($K^{-1} mod(q-1)$)
4. Se calcula $S_2 = K^{-1}(m - X_A S_1) \text{mod}(q-1)$
5. La firma consiste en el par (S_1, S_2).

El usuario B puede verificar la firma de la siguiente forma:

1. Calcula $V_1 = \alpha^m \, mod \, q$
2. Calcula $V_2 = (Y_A)^{S1} (S_1)^{S2} \, mod \, q$.

La firma es válida si $V_1 = V_2$. [50]

Anexo D

Detalles técnicos

ANEXO D.

DETALLES TÉCNICOS

AndroidManifest.xml de este proyecto.

A continuación se muestra como se ha configurado el *AndroidManifest.xml* de este proyecto final de carrera:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.id_digital_pfc"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <activity
            android:name="com.id_digital_pfc.MainActivity"
            android:label="@string/app_name"
            android:theme="@style/ActivMain" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="com.id_digital_pfc.MainActivity2"
            android:label="@string/app_name"
            android:parentActivityName="com.id_digital_pfc.MainActivity"
            android:theme="@style/AppBaseTheme" >
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.id_digital_pfc.MainActivity" />
        </activity>
        <activity
            android:name="com.id_digital_pfc.DatosRegistro1"
            android:label="@string/title_activity_datos_registro1"
            android:parentActivityName="com.id_digital_pfc.MainActivity2"
            android:theme="@style/Registro1" >
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.id_digital_pfc.MainActivity2" />
        </activity>
        <activity
            android:name="com.id_digital_pfc.DatosRegistro2"
            android:label="@string/title_activity_datos_registro2"
            android:parentActivityName="com.id_digital_pfc.DatosRegistro1"
            android:theme="@style/Registro1" >
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.id_digital_pfc.DatosRegistro1" />
        </activity>
```

```
        <activity
            android:name="com.id_digital_pfc.EnvioDatos"
            android:label="@string/title_activity_envio_datos"
            android:parentActivityName="com.id_digital_pfc.DatosRegistro2"
            android:theme="@style/Registro1" >
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.id_digital_pfc.DatosRegistro2" />
        </activity>
        <activity
            android:name="com.id_digital_pfc.GuardarCertificado"
            android:label="@string/title_activity_guardar_certificado"
            android:parentActivityName="com.id_digital_pfc.EnvioDatos"
            android:theme="@style/Registro1" >
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.id_digital_pfc.EnvioDatos" />
        </activity>
        <activity
            android:name="com.id_digital_pfc.EliminarCertificado"
            android:label="@string/lista_certs"
            android:parentActivityName="com.id_digital_pfc.MainActivity2"
            android:theme="@style/AppBaseTheme" >
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.id_digital_pfc.MainActivity2" />
        </activity>
        <activity
            android:name="com.id_digital_pfc.TipoAutenticacion"
            android:parentActivityName="com.id_digital_pfc.MainActivity2"
            android:label="@string/inicio_sesion"
            android:theme="@style/AppBaseTheme" >
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.id_digital_pfc.MainActivity2" />
        </activity>
        <activity
            android:name="com.id_digital_pfc.ElegirCertificado"
            android:parentActivityName="com.id_digital_pfc.ValidacionUsuario"
            android:label="@string/lista_certs"
            android:theme="@style/AppBaseTheme" >
             <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.id_digital_pfc.TipoAutenticacion" />
        </activity>
        <activity
            android:name="com.id_digital_pfc.ValidacionUsuario"
            android:label="@string/pag_ini"
            android:parentActivityName="com.id_digital_pfc.ValidacionUsuario1"
            android:theme="@style/AppBaseTheme" >
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.id_digital_pfc.ElegirCertificado" />
        </activity>

    </application>

</manifest>
```

Figura D.1.AndroidManifest.xml de la aplicación.

Se ha sombreado de diferentes colores cada una de las partes del manifest, que se describen en la siguiente leyenda:

- Paquete, versión y nombre de la versión.
- Configuración del sdk
- Permisos de la aplicación
- Definición de la aplicación: icono, etiqueta, etc.
- Definición de las actividades.

Definir permisos en AndroidManifest.xml

Como se ha comentado en capítulos anteriores, una de las características de Android en relación a la seguridad es el uso de permisos. Por ejemplo, para el uso de internet por parte de una aplicación, el desarrollador tiene que definir este permiso en *AndroidManifest.xml*.

Esta definición de permisos se hace de la siguiente forma:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Este tipo de permisos, son permisos creados por el sistema y se informa al usuario cuando este se quiere instalar la aplicación de que se va a hacer uso de este servicio. El desarrollador también puede crear sus propios permisos, pero la instrucción sería diferente.

Los permisos se declaran antes de la declaración de aplicación.

Otra forma de declarar un permiso es pulsando *Permissions* y añadiendo las características que se deseen:

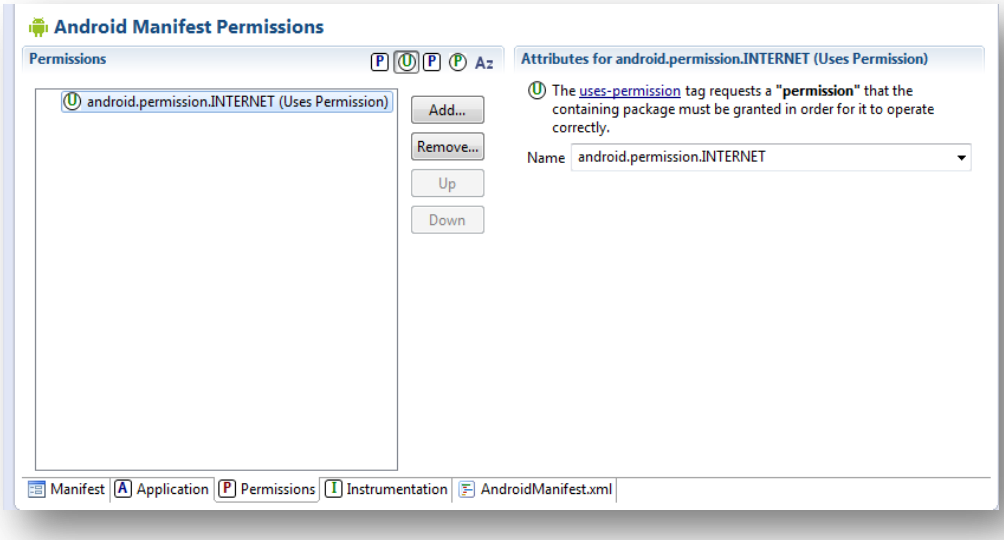


Figura D.2. Declaración de permisos en AndroidManifest.xml

Cómo importar librerías externas

Para la realización del proyecto se han utilizado librerías diferentes a las librerías nativas de Java y de Android, que son necesarias importarlas desde fuera.

En primer lugar, se tienen que descargar las librerías desde la página de últimas realizaciones de BouncyCastle (http://www.bouncycastle.org/latest_releases.html).

Estas librerías se eligen en función de la versión del JDK de Java que se haya instalado en el equipo.

Las librerías descargadas para este proyecto son:

- bcpkix-jdk15on-148.
- bcprov-jdk15on-148.

Para importarlas al proyecto en Eclipse, se tiene que pulsar con el botón derecho del ratón sobre el proyecto. Después se pulsa sobre *build path-> Configure Build Path…*

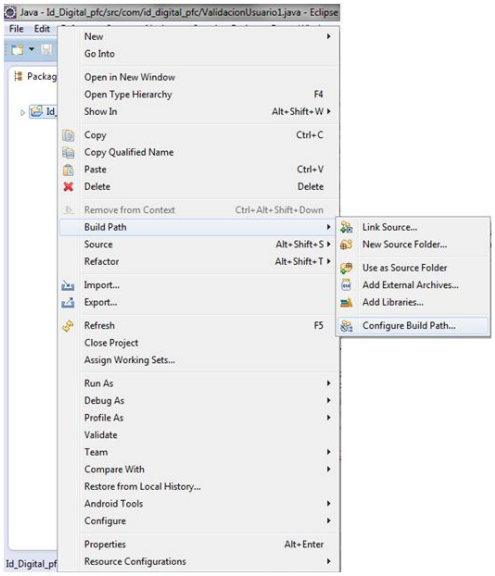


Figura D.3. Primer paso para importar una librería externa.

Una vez que realizado el paso anterior, se debe pulsar sobre Add External Jars y buscar las librerías en el directorio en el que están almacenadas.

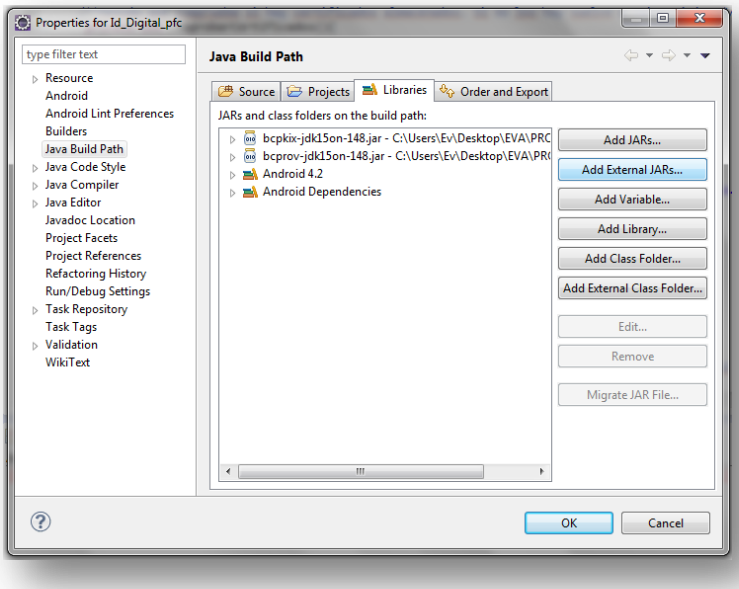


Figura D.4. Segundo paso para importar una librería externa.

Aún así es posible que pueda dar problemas a la hora de ejecutar el programa. Para que no haya problemas y las librerías importadas hagan sus funciones correctamente, hay que añadir las librerías a la carpeta *"libs"* del proyecto.

Introducir una imagen como fondo de pantalla.

Para introducir una imagen como fondo de pantalla se guarda una imagen en una de las carpeta drawable (*hdpi, ldpi, mdpi, xhdpi* dependiendo de la resolución del dispositivo que se haya elegido en el emulador). Cuando se crea un emulador, hay que fijarse en la versión y en los detalles en los que te indica la resolución de cada dispositivo que se simula con el emulador. Para el proyecto, he utilizado la mayor parte del tiempo el mismo tipo de dispositivo con una resolución de mpdi.

El archivo *.xml* que define el layout que se va a utilizar, tiene que incluir la siguiente linea:

```
android:background="@drawable/nombre_foto"
```

Donde nombre_foto se tiene que sustituir por el nombre que le hayamos dado a la la foto que queremos incluir como fondo de pantalla.

Para cambiar el color del fondo de pantalla, se hace de la misma forma pero se cambia drawable por color, de la siguiente forma:

```
android:background="@color/verde"
```

Diseñar botones con imágenes

Para diseñar botones con imágenes se guardan las imágenes en las carpetas que incluye cada proyecto para ello. Según el tipo de emulador que se use se guarda en una carpeta u otra. Una vez que las imágenes se guardan, se utiliza el control ImageButton en al archivo *.xml* de la actividad en la que se incluye. Por ejemplo, para este proyecto final de carrera, se ha creado un botón con la imagen de una flecha y se ha definido de la siguiente forma:

```
<ImageButton
        android:id="@+id/fl_d"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/NomCiudad"
        android:layout_alignRight="@+id/ApellidoUsuario2"
        android:layout_marginRight="24dp"
        android:layout_marginTop="20dp"
        android:background="@color/blanco"
        android:clickable="true"
```

```
        android:contentDescription="@string/boton_derecha"
        android:onClick="adelante"
        android:scaleType="centerInside"
        android:src="@drawable/flecha_dcha" />
```

La fuente en donde se encuentra la imagen viene indicado, en este caso por la última linea.

Los formatos que soporta Android para el uso de imágenes en las aplicaciones son *.PNG, .JPEG* y *.GIF*, aunque este último rara vez se usa.

Modificar una actividad para elimiar el ActionBar.

El ActionBar es la parte de una Actividad que le da nombre, el icono de la aplicación y muestra una serie de opciones, dependiendo de como se haya implementado la aplicación. Para verlo más claro se muestra la imagen del ActionBar en una de actividades del proyecto:

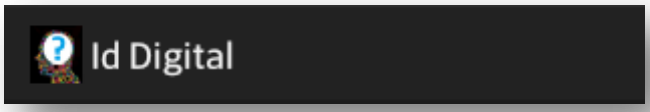


Figura D.5. ActionBar de la primera actividad de la aplicación.

Existen varias formas de eliminar el ActionBar de una aplicación:

Modificando o creando un nuevo estilo en el archivo styles.xml

Cuando se crea un proyecto en Android, se define el tipo de estilo que van a tener sus actividades y este se añade de forma automática en *AndroidManifest.xml*. Esto suele venir indicado por la siguiente linea que aparece encima de todo el conjunto de actividades:

```
android:theme="@style/AppTheme"
```

Además del tema (estilo de la actividad), aparecen controles con el nombre de la aplicación y el icono del lanzador que hemos creado.

Para este proyecto se ha querido que no apareciese ActionBar en las actividades de registro que requieren la introducción de varios datos por parte del usuario, para ello, a parte del estilo que viene por defecto, al cual hemos cambiado de nombre por *ActivMain* (sustituyendo el nombre de *AppTheme*), hemos creado otro estilo en el que hemos eliminado el nombre de la actividad y el ActionBar. Para ver mejor como se ha hecho se muestra a continuación un extracto del archivo *styles.xml*:

```
<!-- Application theme. -->
    <style name="ActivMain" parent="AppBaseTheme">
```

```
        <!-- Se pueden añadir modificaciones no pertenecientes a ningun nivel
API aqui debajo. -->
    </style>
    <style name="SinActionBar" parent="AppBaseTheme">
         <!-- Eliminación del ActionBar y del titulo en la parte superior. --
>
        <item name="android:windowActionBar">false</item>
      <item name="android:windowNoTitle">true</item>
    </style>
```

El primer estilo (*ActivMain*) es el que viene por defecto, sólo se ha cambiado el nombre y el segundo estilo (*SinActionBar*) es el que se ha creado para el resto de las actividades. En el AndroidManifest.xml, poniendo como ejemplo las dos primeras actividades que se han creado, esto queda reflejado de la siguiente forma:

```
<application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
                                          >
        <activity
            android:name="com.id_digital_pfc.MainActivity"
            android:label="@string/app_name"
            android:theme="@style/ActivMain" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="com.id_digital_pfc.DatosRegistro1"
            android:label="@string/title_activity_datos_registro1"
            android:theme="@style/SinActionBar" >
        </activity>
```

La primera línea sombreada refleja la eliminación del estilo que se aplicaba de forma general y que luego se particulariza para cada una de las actividades.

La segunda línea sombreada, indica el uso del estilo que se genera por defecto y al cual hemos modificado el nombre, particularizado para la segunda actividad, actividad en la que se muestran las opciones entre las que el usuario puede elegir.

La tercera línea sombreada refleja, en el *AndroidManifest.xml*, el uso del estilo creado para el resto de las actividades.

Mediante código en las actividades en las que se quiere eliminar el ActionBar

Otra forma de eliminar el ActionBar es mediante un par de líneas de código que se añaden en el método que inicializa la actividad (`onCreate()`). Como ejemplo se muestra las siguientes líneas de código:

```
@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);

		ActionBar actionBar = getActionBar();
		actionBar.hide();

	}
```

Una vez que se ha introducido este par de líneas (sombreadas), si un usuario quiere volver a mostrarlas después de haber realizado cierta acción, sólo tiene que añadir otra línea:

```
actionBar.show();
```

Modificándolo directamente en el AndroidManifest.xml

Para ello, simplemente se debe añadir la siguiente línea a la parte dedicada a la actividad a la que queremos eliminar el *ActionBar*:

```
android:style/Theme.Holo.NoActionBar
```

En este proyecto se ha eliminado el ActionBar de la primera forma, ya que el resto de métodos se han aprendido tiempo después de que la aplicación fuese finalizada, sin embargo, quizá el método más sencillo y eficaz es el segundo método.

Pasar parámetros de una actividad a otra.

Para pasar un parámetro desde la actividad actual a la siguiente se utiliza las siguientes instrucciones:

```
intent.putExtra("String", valor);
```

En “String” se debe poner un nombre con el que se va a nombrar al valor que se va a pasar, y valor es el valor que se pasa, que puede ser un string, un doublé, float etc.

La actividad que recoge los valores, necesita implementar las siguientes instrucciones:

```
Bundle bundle=getIntent().getExtras();
Str = bundle.getString("String");
```

Se debe crear una variable del mismo tipo que se recoge, es decir, si el valor que se va a recoger es un String, la variable que se crea tiene que ser un string también, es decir, en el ejemplo anterior, *Str* también es un String.

Cabe destacar que estas modificaciones se hacen en los archivos .java de cada actividad.

Un ejemplo de cómo se realiza esto, es el paso de la primera actividad de registro de una serie de datos hacia la segunda actividad de registro:

```
Intent intent = new Intent(this, DatosRegistro2.class);
                    intent.putExtra("NOMBRE", nom);
                    intent.putExtra("APELLIDO1", ape1);
                    intent.putExtra("APELLIDO2", ape2);
                    intent.putExtra("PAIS", pais);
                    intent.putExtra("CIUDAD", ciu);
                    intent.putExtra("DIR_IP", IP);
                    startActivity(intent);
                    this.finish();
```

Y a continuación se muestra, como estos datos son recibidos por la segunda actividad de registro en el método onCreate():

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_datos_registro2);

        //Se reciben todos los parámetros de la actividad anterior
        Bundle bundle=getIntent().getExtras();
        nom = bundle.getString("NOMBRE");
        ape1 = bundle.getString("APELLIDO1");
        ape2 = bundle.getString("APELLIDO2");
        pais = bundle.getString("PAIS");
        ciu = bundle.getString("CIUDAD");
        IP = bundle.getString("DIR_IP");

    }
```

Creación de una Keystore[31]

En relación a la seguridad existe una diferencia en la keystore entre Java y Android. Esta diferencia es debida a que las Keystores que se crean para su uso en aplicaciones de Java (formato *.jks* o *Java ks*) tienen diferente formato a las aplicaciones que se crean con Android (formato *.bks*). Esto es importante mencionarlo en este apartado porque a la hora de implementar una aplicación de seguridad como de la que trata este proyecto, puede suponer una pérdida muy importante de recursos y de tiempo no conocer de antemano este detalle que diferencia a Android de Java.

Existen dos formas de crear una Keystore:

1. Mediante una línea de comandos usando la herramienta Keytool de la cual se hablará más adelante en este capítulo:

[31] Una keystore es un almacén de claves que se utilizan para guardar certificados digitales y claves privadas. Más adelante en este mismo capítulo se hablará sobre los almacenes de claves.

```
keytool -import -file cert.cer -keypass genericPassword -keystore
keystore.bks -storetype BKS -storepass genericPassword -providerClass
org.bouncycastle.jce.provider.BouncyCastleProvider -providerpath
/home/user1/lib/bcprov-jdk16-146.jar -alias cert1
```

Para poder realizar esta operación, además de la herramienta Keytool debemos tener descargada la librería de BouncyCastle que se especifica en el comando cuando se va a crear la Keystore (No necesariamente tiene que ser la versión que se menciona anteriormente, pues en el momento de empezar este proyecto la última versión era la bcprov-jdk16-146.jar, pero la última versión en el momento en el que se escribió esto es la bcprov-jdk16-150.jar).

2. Mediante el uso de una interfaz gráfica de usuario de java llamada Portecle y que circula de forma gratuita por la red, pulsando sobre *File->New Keystore->BKS*

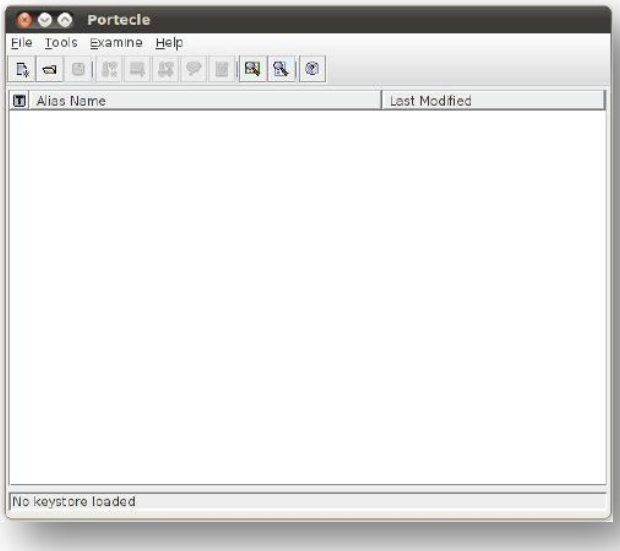


Figura D.6. API Portecle para la creación y gestión de claves, Kesytores y truststores.

Cualquiera de estas dos formas permite crear una keystore con formato *.bks* y se ha comprobado que de cualquiera de las dos formas funciona.

Creación de notificaciones *Toast*

Las notificaciones *Toast*, son mensajes con fondo negro y algo traslucido, que se muestran brevemente en la parte inferior de la pantalla de un móvil. Sirven para dar información muy breve y poco comprometedora que no requiere la interacción del usuario.

Un ejemplo de *Toast* usado en la aplicación desarrollada en este proyecto es el siguiente:

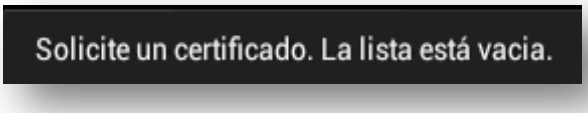


Figura D.7. Ejemplo de notificación tipo *Toast*.

Se utiliza la siguiente línea de código:

```
Toast.makeText(getApplicationContext(), "Mensaje que se va a mostrar",
Toast.LENGTH_LONG).show();
```

Los parámetros que se introducen son el contexto, la cadena de caracteres que se quieren mostrar y el tiempo que se quiere mostrar el mensaje, que puede ser: *LENGTH_LONG* para indicar que se muestra más tiempo y *LENGTH_SHORT* para indicar que se muestra menos tiempo.

Estos mensajes se pueden customizar, pero en este proyecto se ha usado el modelo estándar.

Diálogo de alerta

Este tipo de diálogo se usan para tres usos: mostrar un mensaje, dar una confirmación o dar al usuario a elegir entre dos opciones.

Los diálogos se suelen implementar en el método existente *onCreateDialog()*, este método devuelve el constructor *builder* y para mostrar el diálogo se utiliza la expresión *showDialog()*. En el método *onCreateDialog()* se pueden implementar varios casos según el mensaje que se quiera mostrar usando un *switch*, en tal caso, si el usuario quiere elegir uno u otro mensaje se pasa un argumento mediante el método *showDialog(int)* con la elección que se quiera mostrar.

La forma en la que se ha implementado un diálogo en este proyecto, es usándolo en un método cualquiera. Por ejemplo, el método *mostrarDialogo()* del proyecto, es como sigue:

```
public void mostrarDialogo(){

                // String que se mostrará al usuario
                mensajeDialogo="A continuación, usted recibirá un SMS en su bandeja de
entrada con un codigo. Este número es personal, identifica su conexión como usuario y es
necesario para la obtención de su certificado digital.\n¿Desea continuar con el proceso?";

                //Creación del diálogo
                AlertDialog.Builder dialogo = new AlertDialog.Builder(this);
                dialogo.setIcon(R.drawable.ic_launcher); //Muestra el icono de la
aplicación
                dialogo.setTitle("Información");
                dialogo.setMessage(mensajeDialogo);
```

```
                dialogo.setCancelable(false);

                //Si se pulsa aceptar se llama al método aceptar()
                dialogo.setPositiveButton("Confirmar", new
DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialogo, int id) {
                                dialogo.dismiss();
                                        aceptar();
                        }
                });
                //Si se pulsa el botón "cancelar", se llama al método cancelar
                dialogo.setNegativeButton("Cancelar", new DialogInterface.OnClickListener()
{

                        public void onClick(DialogInterface dialogo, int id) {
                                dialogo.dismiss();
                                cancelar();
                                }
                        });
                dialogo.show();
        }
```

Cuando el usuario pulsa sobre *"Cancelar"* se llama al método *cancelar()*. Cuando el usuario pulsa sobre *"Confirmar"* se llama al método *aceptar()*.

El aspecto que muestra este código en el proyecto es el siguiente:

Figura D.8. Diálogo de alerta o información con dos opciones.

También se puede implementar de tal forma que sólo haya un botón, en ese caso el código podría ser como sigue:

```
private void mostrarDialogo(){

                // String que se mostrará al usuario
                String mensajeDialogo="Su certificado se ha creado correctamente";

                //Creación del diálogo
                AlertDialog.Builder dialogo = new AlertDialog.Builder(this);
```

```
                dialogo.setMessage(mensajeDialogo);
                dialogo.setCancelable(false);
                dialogo.setNeutralButton("Aceptar", new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialogo, int id) {
                                dialogo.dismiss();
                                //Cuando se pulsa el botón aceptar se accede
                                aceptar2();
                        }
                });
                //Se muestra el diálogo
                dialogo.show();
        }
```

Se muestra sombreado las líneas en las que se diferencian principalmente (*setNeutralButton, setPositiveButton* y *setNegativeButton*).

El aspecto del ejemplo implementado en ese código es el siguiente:

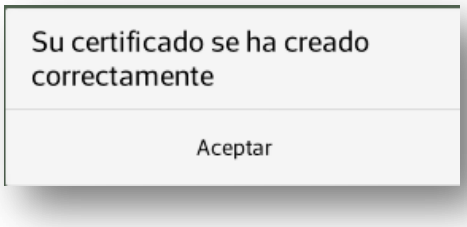


Figura D.9. Diálogo de alerta o información con una sola opción.

Si no se quiere mostrar ninguna opción simplemente habría que eliminar las líneas sombreadas.

Diálogo de progreso

Los diálogos de progreso son notificaciones que indican al usuario que debe esperar mientras se realiza cierta acción que tarda bastante tiempo. Normalmente la acción que tarda en realizarse se ejecuta en otro hilo. Por ejemplo, en la aplicación de este proyecto, la conexión que se realiza con el servidor, se hace desde un hilo:

```
protected void Hilo(ProgressDialog progreso)
        {
                new Thread(new Runnable() {
                        public void run() {
                                conexionServCA();

                                //Acción que se muestra en el hilo principal
                                runOnUiThread(new Runnable() {
                                 public void run() {
                                        CodigoUser =(TextView) findViewById(R.id.Codigo);
                                                CodigoUser.setText(codigoB);

                                 }
                            });
                    }
                }).start();
```

```
        }
```

Mientras, en el hilo principal se implementará el diálogo:

```
        ProgressDialog progreso =ProgressDialog.show(EnvioDatos.this,"", "Esperando
código...", true);
        progreso.setProgressStyle(ProgressDialog.STYLE_SPINNER);
        progreso.setCancelable(true);
        Hilo(progreso);
```

Para detener el diálogo se usará la siguiente línea:

```
progreso.dismiss();
```

El código mostrado como ejemplo, queda en la aplicación de la siguiente forma:

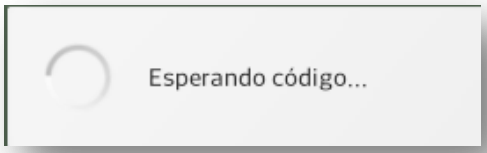


Figura D.10. Diálogo de progreso.

Creación de menús

Para la creación de menús, hay que configurar el archivo XML correspondiente a la actividad en la que se quiera incluir el menú. Este archivo, se encuentra en el directorio "*menú*", dentro del directorio "*res*". Usando como ejemplo, se muestra uno de los archivos XML del proyecto:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

<item
        android:id="@+id/menu_salir5"
        android:orderInCategory="100"
        android:showAsAction="never"
        android:title="@string/Salir"/>
    <item
        android:id="@+id/menu_entrar_contrasena"
        android:orderInCategory="100"
        android:showAsAction="never"
        android:title="@string/usar_usuario_y_contrasena"/>
    <item
        android:id="@+id/menu_volver_inicio4"
        android:orderInCategory="100"
        android:showAsAction="never"
        android:title="@string/otra_opcion"/>

</menu>
```

Implementar un menú también requiere hacer cambios en el archivo Java del proyecto. Para ello, se necesita añadir este código:

```
@Override
```

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_entrar_contrasena:
        // Acción que se realizará al pulsar sobre esta opción
           return true;
        case R.id.menu_volver_inicio4:
       // Acción que se realizará al pulsar sobre esta opción
            return true;
        case R.id.menu_salir5:
        // Acción que se realizará al pulsar sobre esta opción
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

/**Muestra una serie de opciones en el menu al usuario*/
@Override
public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.activity_elegir_certificado, menu);
        return super.onCreateOptionsMenu(menu);
}
```

Código de ejemplo para la recepción de SMS

A continuación, se muestra un ejemplo de código para la recepción del mensaje sms:

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsMessage;
import android.widget.Toast;

public class ReceptorSMS extends BroadcastReceiver
{
       @Override
       public void onReceive(Context context, Intent intent){
              Bundle bundle = intent.getExtras();
              SmsMessage[] msgs = null;
              String str = "";
              if (bundle != null)
              {
                     Object[] pdus = (Object[]) bundle.get("pdus");
                     msgs = new SmsMessage[pdus.length];

                     for (int i=0; i<msgs.length; i++){

                        msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
                        str += "SMS de " + msgs[i].getOriginatingAddress();
                        str += ":";
                        str += msgs[i].getMessageBody().toString();
                        str += "\n";
                        Toast.makeText(context, str, Toast.LENGTH_LONG).show();

                            }
                     }
              }
       }
```

El funcionamiento de este código es el siguiente:

- Se crea una clase heredada de BroadcastReceiver a la que se le incluye un método llamado onReceive.
- Se crea un objeto Bundle para recibir los extras y un objeto SmsMessage que es el que se encargará de recibir el mensaje.Los mensajes son mandados y recibidos en un formato llamado PDU (Protocol Data Unit).
- Se crea un String *"str"* en el que se almacena la información del sms, empezando por la persona que lo ha enviado y continuando con el mensaje recibido.
- Por último se declara la actividad en el AndroidManifest y el permiso para recibir SMS, lo que además permitiría no interrumpir la actividad que estuviésemos haciendo en ese momento con el móvil.

Anexo E

Configuraciones necesarias

ANEXO E.

CONFIGURACIONES NECESARIAS

Configuración del router: abrir puertos en el router.

Los pasos para abrir un puerto en el router son:

1. Se comprueba en el ordenador las direcciones que vamos a usar, para ello utilizamos el comando ifconfig si lo hacemos desde Linux y ipconfig desde Windows, como en el siguiente ejemplo:

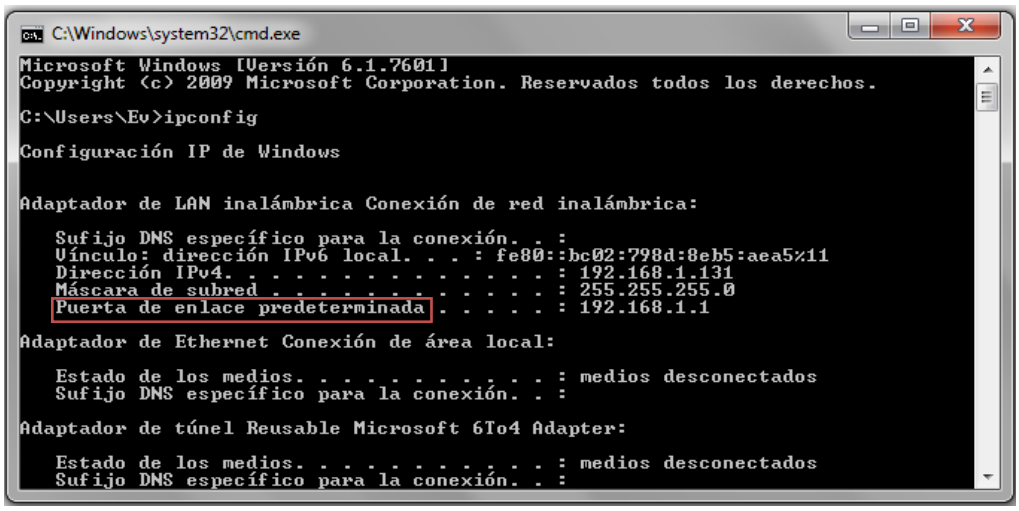


Figura E.1. Captura de pantalla con las direcciones que se usan.

2. Se utiliza la dirección de enlace predeterminada, que es la que da acceso al router y a todos sus datos. Esta dirección se pone en el navegador y nos piden una contraseña:

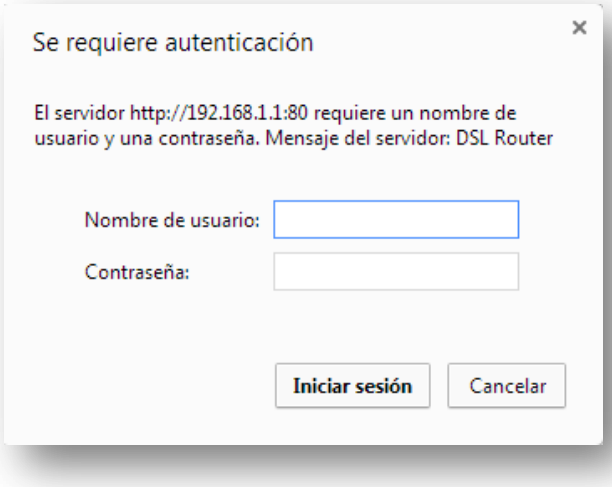


Figura E.2. El usuario tiene que introducir un usuario y contraseña.

Normalmente las compañías de telefonía utilizan las mismas contraseñas y nombre de usuario por defecto para los routers, que suelen ser: *"1234"* para routers de Telefónica y *"admin"* para routers de Jazztel y en general, la mayoría utiliza una de estas dos contraseñas.

3. Al meter la dirección del router, hay una serie de pestañas que nos permiten configurar el router según lo que deseemos hacer. Para abrir un puerto, hay que cambiar datos de la *NAT* (*Network Address Translation* o Traducción de la dirección de red):

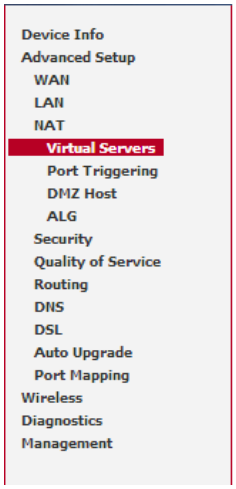


Figura E.3. Configuración del router.

Después de pulsar sobre Virtual Servers, se pulsa sobre *"Add"* y se añaden los datos del puerto que vamos a abrir, por ejemplo, si se abre el puerto 4000 y también se introduce la dirección IPv4 que se marca en la primera figura:

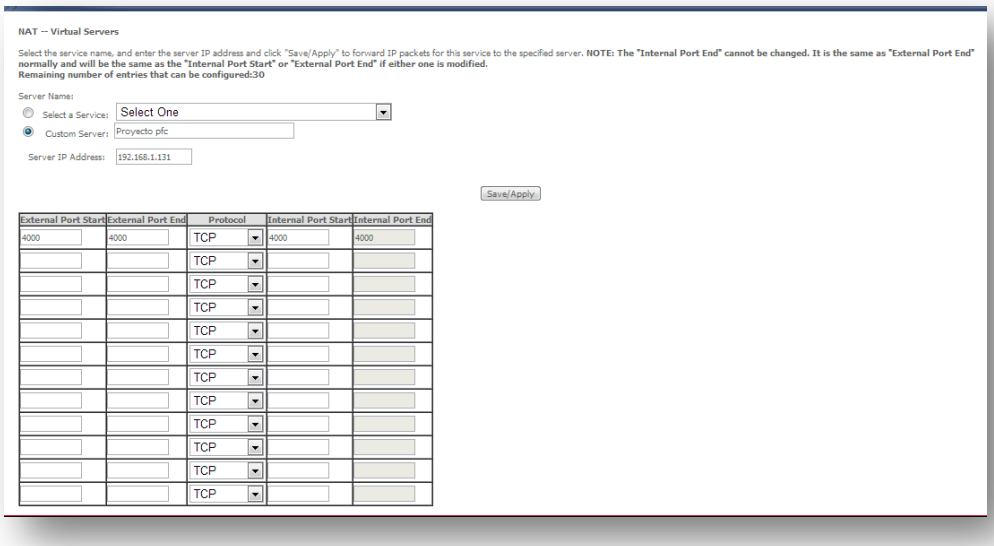


Figura E.4. Captura de pantalla con los datos que se introducen en el router.

Con ello, tras pulsar sobre *"Save/Apply"* e indicar el uso o nombre que se le va a dar al puerto, ya se ha podido abrir el puerto 4000.

Anexo F

Codigo del proyecto

Anexo F

CÓDIGO DEL PROYECTO

Código del servidor de la CA

servidorCA.java

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.InvalidKeyException;
import java.security.KeyManagementException;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.SignatureException;
import java.security.UnrecoverableKeyException;
import java.security.cert.CertificateException;
import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSession;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.TrustManagerFactory;

/**Clase que permite la conexion y gestion de los datos que recibe
 * del usuario para enviarle un certificado digital
 * @author: Eva M. Blanco Delgado
 * @version: 18/01/2014_v0.0*/
public class servidorCA {
// Declaración de variables
ServerSocket Servidor;
int Puerto = 2000;
int Puerto2 = 3000;
String IP = "192.168.1.129";
String TimeStamp;
Object prueba;
String certificado="";
String ksName = "ksServidorCA.jks";
String tsName = "tsServidorCA.jks";
int numMovil;
/**Entero en el que se almacena el codigo A generado
 * @see #transmiteInfoUsuario(SSLSocket)*/
static int codigoA;
/**Entero en el que se almacena el codigo B generado
 * @see #transmiteInfoUsuario(SSLSocket)*/
static int codigoB;
String cert="";
char ksPass[] = "109638".toCharArray();
char ctPass[] = "109638".toCharArray();
int ack;
int valorAck;
int ackb;

servidorCA() throws ClassNotFoundException, UnrecoverableKeyException, KeyStoreException,
NoSuchAlgorithmException, CertificateException, KeyManagementException, InvalidKeyException,
IllegalStateException, NoSuchProviderException, SignatureException {

        try {
                conexionUsuario();
```

```
		} catch (IOException e) {
			e.printStackTrace();
		}
}

/**Metodo que permite una conexion segura entre el servidor de SMS y el servidor
 * de la CA. Tambien permite enviar el codigo B y recibir el ack por parte del
 * usuario
 * @return <code>ackb</code>: variable entera con el ACK del servidor de sms
 * @see GestionCertificados#RecibirACK(Socket)
 * @see GestionCertificados#EnvioCodBServSMS(Socket, String)*/
public int conexionServSMS(SSLSocket client) throws NoSuchAlgorithmException, CertificateException,
FileNotFoundException, IOException, KeyStoreException, UnrecoverableKeyException,
KeyManagementException, ClassNotFoundException{                //Trustore del servidor de SMS (con
el certificado del servidor de la CA)                    KeyStore ts = KeyStore.getInstance("JKS");
		ts.load(new FileInputStream(tsName), ksPass);
		TrustManagerFactory tmf =
TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
		tmf.init(ts);

		//keystore del servidor de SMS(con los certificados del servidor de SMS)
		KeyStore ks = KeyStore.getInstance("JKS");
		ks.load(new FileInputStream(ksName), ksPass);
		KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
		kmf.init(ks, ctPass);

		SSLContext sslcontext = SSLContext.getInstance("TLS");
		sslcontext.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
		Socket socket = sslcontext.getSocketFactory().createSocket(IP, Puerto2);

		// Se instancia un objeto de la clase GestionCertificados
		GestionCertificados gc = new GestionCertificados();

		if(socket.isConnected())
		{
			String cB = String.valueOf(codigoB);
			gc.EnvioCodBServSMS(socket, cB);
			// Se envían los códgigos al usuario
			gc.envioCodigosUsuario(client, codigoA, codigoB);
			//DESPUES DE LA CONEXIÓN CON EL SERVIDOR DE SMS
			ackb=gc.RecibirACK(socket);
		}
		socket.close();
		return ackb;
}

/**Metodo que permite la transmision de informacion entre el servidor de la CA y
 * el cliente y la gestion de los datos y los certificados
 * @throws SignatureException
 * @throws NoSuchProviderException
 * @throws IllegalStateException
 * @throws InvalidKeyException
 * @param client Un socket seguro SSL que permite la conexiÃ³n segura con el usuario
 * @see GestionCertificados#recibirDatos(SSLSocket)
 * @see GestionCertificados#envioCodigosUsuario(SSLSocket, int, int)
 * @see GestionCertificados#recibirCSR(SSLSocket)
 * @see GestionCertificados#firmaCertificado(String, int)
 * @see GestionCertificados#leerCertUsuario(int)
 * @see GestionCertificados#envioCertUsuario(SSLSocket, String)
 * @see GestionCertificados#GuardarCertEnTruststore(String, int)
 * @see GestionCertificados#codigoAErroneo(SSLSocket)
 * @see GestionCodigos#generaCodigo(int)
 * @see GestionCodigos#comparaValores(int, int)*/
public void transmiteInfoUsuario(SSLSocket client) throws NoSuchAlgorithmException, IOException,
ClassNotFoundException, UnrecoverableKeyException, CertificateException, KeyStoreException,
KeyManagementException, InvalidKeyException, IllegalStateException, NoSuchProviderException,
SignatureException
{
		// Se instancia un objeto de la clase GestionCertificados y un objeto de la
		// clase GestionCodigos
		GestionCertificados gc = new GestionCertificados();
		GestionCodigos gcod = new GestionCodigos();
		// Se obtiene el nÃºmero de mÃ³vil por parte del usuario
		numMovil = gc.recibirDatos(client);
```

```
		// Se generan los cÃ³digos A y B para enviar al usuario
		codigoA=gcod.generaCodigo(numMovil);
		//System.out.println("Codigo A:"+codigoA);
		codigoB=gcod.generaCodigo(numMovil);
		System.out.println("Codigo B:"+codigoB);

		/*CONEXIÓN CON EL SERVIDOR DE SMS*/
		// TambiÃ©n se le envía al servidor de SMS el código B y se espera del mismo
		// un ACK que nos permita comprobar que no ha habido fallos de seguridad
		// en la conexión
		ack = conexionServSMS(client);
		if(ack==1)
		{
			// Si el ACK es 1 significa que no hay fallos y se recibe el CSR por parte
			// del cliente
			String[] datos=gc.recibirCSR(client);

			// Se compara que el código A que recibimos del usuario es igual al
			// código A que previamente habiamos enviado, lo cual demostraría que no ha
			// habido fallos de seguridad
			if(gcod.comparaValores(codigoA, Integer.parseInt(datos[0]))==true)
			{
				// Se llama al método que firma el certificado y se le pasa el CSR y
				// el número de movil
				gc.firmaCertificado(datos[1], numMovil);
				// Se lee la cadena de caracteres del certificado firmado ya por la CA.
				String certificadoUsr=gc.leerCertUsuario(numMovil);
				// Se envía el certificado ya firmado al usuario.
				gc.envioCertUsuario(client, certificadoUsr);
			}
			else
			{
				// Si el código enviado por el usuario no coincide con el código
				// que el usuario ha recibido, se informa al mismo
				gc.codigoAErroneo(client);
			}
		}
		else  // Si el ACK no es el correcto se avisa al usuario
			gc.codigoAErroneo(client);

		//Una vez se ha enviado el certificado al usuario se elimina de memoria el certificado
		//y el CSR del usuario.
		gc.BorrarArchivoTemp("/home/eva/trabajoEclipse/ServidorCA/CSRusuarios/"+numMovil+".csr");
		gc.BorrarArchivoTemp("/home/eva/trabajoEclipse/ServidorCA/CertUsuarios/"+numMovil+".pem");
}

/**Metodo que permite la conexion segura entre el usuario y el servidor de la CA
 * @throws SignatureException
 * @throws NoSuchProviderException
 * @throws IllegalStateException
 * @throws InvalidKeyException
 * @see #transmiteInfoUsuario(SSLSocket) */
public void conexionUsuario() throws KeyStoreException, NoSuchAlgorithmException,
CertificateException, FileNotFoundException, IOException, UnrecoverableKeyException,
KeyManagementException, ClassNotFoundException, InvalidKeyException, IllegalStateException,
NoSuchProviderException, SignatureException{

KeyStore ks = KeyStore.getInstance("JKS");
ks.load(new FileInputStream(ksName), ksPass);
KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
kmf.init(ks, ctPass);
SSLContext sc = SSLContext.getInstance("TLS");
sc.init(kmf.getKeyManagers(), null, null);
SSLServerSocketFactory ssf = sc.getServerSocketFactory();
SSLServerSocket servidor = (SSLServerSocket) ssf.createServerSocket(Puerto);
servidor.setNeedClientAuth(false);
SSLSocket client = (SSLSocket)servidor.accept();
SSLSession ss = client.getSession();
client.startHandshake();
System.out.println("Cipher suite"+ss.getCipherSuite());
System.out.println("Cipher protocol"+ss.getProtocol());
```

```
// Se llama al método que permite la transmisión de información entre el
// servidor de la CA y el usuario.
        transmiteInfoUsuario(client);
        client.close();
        servidor.close();
}

/**Metodo principal del programa
 * @param args argumentos que se le pasan al programa
 * @throws ClassNotFoundException
 * @throws CertificateException
 * @throws NoSuchAlgorithmException
 * @throws KeyStoreException
 * @throws KeyManagementException
 * @throws UnrecoverableKeyException
 * @throws SignatureException
 * @throws NoSuchProviderException
 * @throws IllegalStateException
 * @throws InvalidKeyException
 * @throws IOException
 * @see #servidorCA()
 */
public static void main(String[] args) throws ClassNotFoundException, UnrecoverableKeyException,
KeyManagementException, KeyStoreException, NoSuchAlgorithmException, CertificateException,
InvalidKeyException, IllegalStateException, NoSuchProviderException, SignatureException,
IOException {
        while(true){
        new servidorCA();
                }
        }
}
```

GestionCertificados.java

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.PrintWriter;
import java.net.Socket;
import java.security.NoSuchAlgorithmException;
import javax.net.ssl.SSLSocket;

/**Clase que permite el envio y recepcion de datos por parte del
 * usuario a la vez que permite la gestion de esos datos para la obtencion de un certificado por
 * parte del usuario.
 * @author: Eva M. Blanco Delgado
 * @version: 18/01/2014_v0.0*/
public class GestionCertificados{
String cert="";
/**String con la direccion de email recibido por parte del usuario
 * @see #recibirDatos(SSLSocket)*/
String [] emails = null;

/**Metodo que recibe datos del cliente. En particular, el numero de movil
 * y la direccion de correo electronico
 * @param client variable que contiene el socket Seguro SSL
 * @return <code>numMovil</code>: Entero con el numero de movil introducido por el usuario y
recibido por el servidor
 * @see servidorCA#transmiteInfoUsuario(SSLSocket)*/
public int recibirDatos(SSLSocket client) throws IOException, ClassNotFoundException,
NoSuchAlgorithmException{

        ObjectInputStream email=new ObjectInputStream(client.getInputStream());
        ObjectInputStream movil=new ObjectInputStream(client.getInputStream());

        String inputEmail=(String) email.readObject();
        String inputMovil=(String) movil.readObject();
```

```
        // Se convierte el número de movil a int para que la función devuelva un entero
        // y sea más cómodo trabajar con el número de teléfono
        int numMovil=Integer.parseInt(inputMovil);

        return numMovil;
}
/**Metodo que una vez que se han generado los codigos, los envia
* @param client variable que contiene el socket seguro SSL para el establecimiento de la conexion
* @param codigoA entero que contiene el codigo A que se le envia al usuario
* @param codigoB entero que contiene el codigo B que se le envia al usuario
 * @see servidorCA#transmiteInfoUsuario(SSLSocket)*/
public void envioCodigosUsuario(SSLSocket client, int codigoA, int codigoB) throws IOException{
        // Se convierten los códigos de enteros a Strings para poder transmitirlos al usuario
        String cA = String.valueOf(codigoA);
        String cB = String.valueOf(codigoB);
        ObjectOutputStream codA=new ObjectOutputStream(client.getOutputStream());
        ObjectOutputStream codB=new ObjectOutputStream(client.getOutputStream());
        codA.writeObject(cA);
        codB.writeObject(cB);
        codA.flush();
        codB.flush();
}
/**Metodo para recibir el codigo A y el CSR por parte del usuario
* @param client variable que contiene el socket seguro SSL para el establecimiento de la conexion
* @return datos cadena de Strings que contiene el codigo A y el codigo B recibido por el usuario
* @see servidorCA#transmiteInfoUsuario(SSLSocket)*/
public String [] recibirCSR(SSLSocket client) throws IOException, ClassNotFoundException{

        ObjectInputStream codigoA=new ObjectInputStream(client.getInputStream());
        ObjectInputStream csr=new ObjectInputStream(client.getInputStream());
        String inputcA=(String) codigoA.readObject();
        String inputCSR=(String) csr.readObject();

        String[] datos = {inputcA, inputCSR};
        return datos;
}

/**Metodo que permite la ejecucion de un script, donde se encuentran las ordenes
* de openssl para firmar el CSR recibido por el usuario
* @param numM entero con el numero de movil del usuario
* @see #firmaCertificado(String, int)*/
private void EjecutarFirma(int numM)
{
        String s = null;
        try {
                String cmd = "./firmar.sh "+numM;
                Process p = Runtime.getRuntime().exec(cmd);

                BufferedReader stdInput = new BufferedReader(new
                InputStreamReader(p.getInputStream()));

                BufferedReader stdError = new BufferedReader(new
                InputStreamReader(p.getErrorStream()));

                while ((s = stdInput.readLine()) != null) {
                          System.out.println(s);
              }

              // Si se produce algún error se muestra en la consola
              System.out.println("Posibles errores:\n");
              while ((s = stdError.readLine()) != null) {
                    System.out.println(s);
              }
        } catch (IOException ioe) {
                System.out.println (ioe);
                }
}
/**Metodo que permite leer del directorio especificado, el certificado ya firmado
* del usuario y que finalmente se le enviara al mismo
* @param numM entero con el numero de movil del usuario
* @return <code>cert</code>: String con el certificado ya firmado del usuario y en formato PEM*/
public String leerCertUsuario(int numM) throws IOException{
```

```
        String path ="/home/eva/trabajoEclipse/ServidorCA/CertUsuarios/"+numM+".pem";
        File f1 = new File(path);
        if(f1.exists())
        {
                System.out.println("Existe el archivo");
        }
        else
                System.out.println("No existe el archivo");

        BufferedReader br = new BufferedReader(new FileReader(new File(path)));
        String linea= br.readLine();
        while(linea!=null) {
                cert = cert+linea+System.getProperty("line.separator");
                linea= br.readLine();
                }
        br.close();
        return cert;
}
/**Metodo que lee el CSR recibido por parte de un usuario y que se ha almacenado
* en un archivo dentro de una carpeta del proyecto y que llama al metodo
* EjecutarFirma para poder obtener el certificado firmado por el usuario
* @param CSR String con el CSR recibido del usuario
* @param numM entero que contiene el nÃºmero de móvil del usuario
* @see #EjecutarFirma(int)
* @see servidorCA#transmiteInfoUsuario(SSLSocket)*/
public void firmaCertificado(String CSR, int numM) throws IOException{

String path ="/home/eva/trabajoEclipse/ServidorCA/CSRusuarios/"+numM+".csr";
BufferedWriter bw= new BufferedWriter(new FileWriter(new File(path)));
PrintWriter wr= new PrintWriter(bw);
wr.write(CSR);
wr.close();
bw.close();
EjecutarFirma(numM);

}
/**Metodo para el envio del certificado al usuario
* @param client variable que contiene el socket seguro SSL para establecer la conexion con el usuario
* @param certificado String que contiene el certificado firmado para ser enviado al usuario*/
public void envioCertUsuario(SSLSocket client, String certificado) throws IOException
{
        ObjectOutputStream cert=new ObjectOutputStream(client.getOutputStream());
        cert.writeObject(certificado);
        cert.flush();
}
/**Metodo que envÃ-a un mensaje informando de que ha habido un error
* @param client variable con el socket seguro SSL que permite establecer una conexion con el usuario
* @see servidorCA#transmiteInfoUsuario(SSLSocket)*/
public void codigoAErroneo(SSLSocket client) throws IOException
{
        ObjectOutputStream cert=new ObjectOutputStream(client.getOutputStream());
        cert.writeObject("ErrCodigoA");
}
/**Metodo que envia el codigo B al servidor de SMS
* @param socket variable que contiene un socket que permite establecer una conexion con el usuario
* @param codigoB String que contiene el codigo B que se le envia al usuario
* @see servidorCA#conexionServSMS()*/
public void EnvioCodBServSMS(Socket socket, String codigoB) throws IOException
{
        ObjectOutputStream codB=new ObjectOutputStream(socket.getOutputStream());
        codB.writeObject(codigoB);
        codB.flush();
}
/**Metodo que recibe el ACK por parte del servidor de SMS y comprueba que
* el ack es 1
* @param socket variable que contiene un socket que permite establecer una conexion con el usuario
* @return <code>0</code> o <code>1</code> en funcion de si se recibe un ACK correcto
* @see servidorCA#conexionServSMS()*/
public int RecibirACK(Socket socket) throws IOException, ClassNotFoundException
{
        ObjectInputStream ackb=new ObjectInputStream(socket.getInputStream());
        String ackB=(String) ackb.readObject();
        if(ackB.equals("AckB_Correcto"))
```

```
        {
                System.out.println(ackB);
                socket.close();
                return 1;
        }
        return 0;
}

/**Metodo que permite la eliminacion de archivos temporales que se usan para almacenar
* temporalmente el CSR y el certificado del usuario. Una vez se ha firmado el certificado
* se elimina al igual que el CSR recibido para evitar uno uso abusivo de los recursos
* @see servidorCA#transmiteInfoUsuario(SSLSocket)*/
public void BorrarArchivoTemp(String PATH)
{
        File f = new File(PATH);
        if(f.delete())
                System.out.println("\nEL CERTIFICADO HA SIDO ELIMINADO");
        else
                System.out.println("\nEL CERTIFICADO NO SE PUEDE ELIMINAR");
}
}
```

GestionCodigo.java

```
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;

/**Clase que permite la generacion y gestion de los codigos que el servidor
 * envia al usuario mediante una serie de operaciones
 * @author: Eva M. Blanco Delgado
 * @version: 18/01/2014_v0.0*/
public class GestionCodigos {
// Declaración de variables
int nRandom;
int aleatorio;
String nconcat;
String concatenados;
int entero;
int ent;
int codigo;
/**Metodo que compara dos valores: el codigo A generado con el recibido por parte del usuario.
 * @param A1 Entero que contiene el entero generado por el servidor.
 * @param A2 Entero recibido por parte del usuario.
 * @return <code>false</false> o <code>true</false> dependiendo de si los codigos son iguales o no.
 * @see servidorCA#transmiteInfoUsuario(javax.net.ssl.SSLSocket)*/
public boolean comparaValores(int A1, int A2){
        if(A1==A2)
                return true;
        else
                return false;
        }
/**Metodo que convierte una cadena de bytes a un numero entero, si el numero entero es
* negativo se multiplica por (-1) para convertirlo en su valor absoluto
* @param bytes Una cadena de bytes
* @return <code>entero</code>: el valor de los bytes recibidos en enteros.
* @see #codigoDeHash(String)*/
private int deBytesAEnteros(byte[] bytes){
        entero= new BigInteger(bytes).intValue();
        if(entero<0)
        {
                entero = entero*(-1);
        }
        return entero;
}
/**Metodo que obtiene un codigo haciendo uso de una funcion resumen o Hash.
* @param nconcat String que contiene el numero de telefono concatenado a un valor aleatorio.
* @return <code>ent</code>: Un entero con el numero aleatorio generado.
* @see #deBytesAEnteros(byte[])
* @see #generaCodigo(int)*/
private int codigoDeHash(String nconcat) throws NoSuchAlgorithmException
```

```
{
        MessageDigest miMD = MessageDigest.getInstance("SHA-1");
        miMD.reset();
        byte[] bytes = nconcat.getBytes();
        miMD.update(bytes);
        ent=deBytesAEnteros(miMD.digest());
        return ent;
}
/**Metodo que concatena un numero aleatorio con el numero enviado por el usuario
* @param numMovil Entero que contiene el numero de movil del usuario.
* @param nRandom Entero con el numero aleatorio generado.
* @return <code>concatenados</code>: String que concatena el numero de movil con el numero
* aleatorio generado
* @see #generaCodigo(int)*/
private String concatenarNumeros(int numMovil, int nRandom)
{
        String a = String.valueOf(numMovil);
        String b = String.valueOf(nRandom);
        concatenados = a+b;
        return concatenados;
}
/**Metodo para la generacion de un nÃºmero aleatorio seguro.
* @return <code>aleatorio</code>: entero que contiene un numero aleatorio.
* @see #generaCodigo(int)*/
private int numeroAleatorioSeguro() throws NoSuchAlgorithmException{
        int ndesde=0;
        int nhasta=1000000000;
        SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
        int valor = random.nextInt((nhasta+1)-ndesde);
        aleatorio=ndesde+valor;
        return aleatorio;
}

/**Metodo que llama a los diferentes metodos a partir de los cuales se genera el codigo que
* se envia al usuario
* @param numMovil Entero que contiene el numero de telefono del usuario.
* @return <code>codigo</code>: entero con el numero aleatorio generado.
* @see #numeroAleatorioSeguro()
* @see #concatenarNumeros(int, int)
* @see #codigoDeHash(String)
* @see servidorCA#transmiteInfoUsuario(javax.net.ssl.SSLSocket)*/
public int generaCodigo(int numMovil) throws NoSuchAlgorithmException{
        nRandom=numeroAleatorioSeguro();
        nconcat=concatenarNumeros(numMovil, nRandom);
        codigo=codigoDeHash(nconcat);
        return codigo;
        }
}
```

Código del servidor de SMS

servidorSMS.java

```
package servidorsms;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.KeyManagementException;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.UnrecoverableKeyException;
import java.security.cert.CertificateException;
import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSession;
import javax.net.ssl.SSLSocket;
```

```
import javax.net.ssl.TrustManagerFactory;

/**Clase que permite la conexion entre el servidor de SMS y el usuario y
 * la conexion entre el servidor de SMS y el servidor de la CA
 * @author: Eva M. Blanco Delgado
 * @version: 18/01/2014_v0.0*/
public class servidorSMS {
//DeclaraciÃ³n de variables
String ksName = "ksServidorSMS.jks";
String tsName = "tsServidorSMS.jks";
char ksPass[] = "109638".toCharArray();
char ctPass[] = "109638".toCharArray();
int Puerto2 = 3000;
int Puerto3=4000;
ServerSocket Servidor;
static int ack;
static int ackdeusr;
String respuesta;

/**Instancia de la clase para la conexion del servidor de la CA.
* @throws InterruptedException */
servidorSMS() throws UnrecoverableKeyException, KeyManagementException, KeyStoreException,
NoSuchAlgorithmException, CertificateException, FileNotFoundException, IOException,
ClassNotFoundException, InterruptedException {
conexionServCA();
}
/**Metodo que permite la conexion del Servidor de SMS con el usuario
* @return <code>ackdeusr</code>: devuelve un entero que indica si se ha recibido un
* ACK por parte del usuario.
* @throws InterruptedException
* @see #conexionServCA()
* @see GestionDatos2#EnviarUsrCodB(Socket)
* @see GestionDatos2#RecibirACKB(Socket)*/
public int ConexionUsuario() throws IOException, ClassNotFoundException, InterruptedException{
        Servidor = new ServerSocket(Puerto3);
        Socket client=Servidor.accept();
        // Se instancia un objeto de la clase GestionDatos2
        GestionDatos2 gdusuario = new GestionDatos2();
        // Llamada al método que permite enviar el código B desde el servidor de la
        // CA al usuario, teniendo como intermediario al servidor de SMS
        gdusuario.EnviarUsrCodB(client);
        // Si el código B enviado al usuario desde el servidor de la CA, coincide
        // con el código B enviado desde el servidor de SMS, el usuario envia un
        // ack que es recibido por el mÃ©todo RecibirACKB
        ackdeusr = gdusuario.RecibirACKB(client);
        client.close();
        Servidor.close();
        return ackdeusr;
}
/**Metodo que permite la conexion del servidor de SMS con el servidor de la CA
* @throws InterruptedException
* @see #ConexionUsuario()
* @see GestionDatos2#RecibirCodBServSMS(SSLSocket)
* @see GestionDatos2#EnviarACKServidorCA(SSLSocket, String)*/
public void conexionServCA() throws KeyStoreException, NoSuchAlgorithmException,
CertificateException, FileNotFoundException, IOException, UnrecoverableKeyException,
KeyManagementException, ClassNotFoundException, InterruptedException{
        int i=0;
        while(true){
                //Trustore del servidor de SMS (con el certificado del servidor de la CA)
                KeyStore ts = KeyStore.getInstance("JKS");
                ts.load(new FileInputStream(tsName), ksPass);
                TrustManagerFactory tmf =
TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
                tmf.init(ts);
                //keystore del servidor de SMS(con los certificados del servidor de SMS)
                KeyStore ks = KeyStore.getInstance("JKS");
                ks.load(new FileInputStream(ksName), ksPass);
                KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
                kmf.init(ks, ctPass);
                //Se establece una conexión TLS
                SSLContext sc = SSLContext.getInstance("TLS");
                //Se inicializan los parÃ¡metros de la conexiÃ³n
```

```
                sc.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
                SSLServerSocketFactory ssf = sc.getServerSocketFactory();
                SSLServerSocket servidor = (SSLServerSocket) ssf.createServerSocket(Puerto2);
                servidor.setNeedClientAuth(true);
                SSLSocket client = (SSLSocket)servidor.accept();
                SSLSession ss = client.getSession();
                //System.out.println("Cipher suite"+ss.getCipherSuite());
                //System.out.println("Cipher protocol"+ss.getProtocol());

                // Se instancia un objeto de la clase GestionDatos2
                GestionDatos2 gdservSMS = new GestionDatos2();
                // Se llama al método que permite recibir el código B del servidor de la CA.
                gdservSMS.RecibirCodBServSMS(client);

                // Llamada al método que permite establecer la conexión entre el servidor de SMS
                // y el usuario. Este método devuelve un valor que indica si se ha recibido
                // correctamente por parte del usuario el codigo B, según se haya recibido
                // correctamente o no, se informa al servidor de la CA.
                ack=ConexionUsuario();
                ack = 1;

                if(ack==1)
                {
                        respuesta="AckB_Correcto";
                }
                else
                {
                        respuesta = "AckB_Incorrecto";
                }
                // Se llama al método que se encarga de informar sobre que ACK se ha recibido.
                // Si es el correcto o no
                gdservSMS.EnviarACKServidorCA(client, respuesta);
                client.close();
                servidor.close();
                i++;
                System.out.println(i);
        }
}

/**Metodo principal del programa
* @param args argumentos que se le pasan al programa principal
* @throws IOException
* @throws FileNotFoundException
* @throws CertificateException
* @throws NoSuchAlgorithmException
* @throws KeyStoreException
* @throws KeyManagementException
* @throws UnrecoverableKeyException
* @throws ClassNotFoundException
* @throws InterruptedException
*/
public static void main(String[] args) throws UnrecoverableKeyException, KeyManagementException,
KeyStoreException, NoSuchAlgorithmException, CertificateException, FileNotFoundException,
IOException, ClassNotFoundException, InterruptedException {

        new servidorSMS();
        }
}
```

GestionDatos2.java

```
package servidorsms;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import javax.net.ssl.SSLSocket;

/**Clase que gestiona los datos que se envian al servidor de SMS
 * @author: Eva M. Blanco Delgado
 * @version: 18/01/2014_v0.0*/
```

```
public class GestionDatos2 {
//Inicialización de las variables que se van a usar
/**String que recoge el codigo B que se le envia al servidor de SMS desde el servidor
* de la CA y que posteriormente es enviado desde el servidor de SMS hasta el usuario.
* @see #EnviarUsrCodB(Socket)*/
static String codigoB;

/**Metodo que envia al usuario el codigo generado. Simula el funcionamiento que tendria
* en un escenario real en el que se enviaria un sms
* @see servidorSMS#ConexionUsuario()*/
public void EnviarUsrCodB(Socket client) throws IOException, ClassNotFoundException{
        ObjectOutputStream out=new ObjectOutputStream(client.getOutputStream());
        out.writeObject(codigoB);
        out.flush();
}
/**Metodo que recibe un ack por parte del usuario si el cÃ³digo enviado via servidor sms
* y servidor CA, son el mismo
* @param client variable que contiene el socket que permite establecer una conexion con el usuario
* @return <code>0</code> o <code>1</code> dependiendo de si se recibe un ACK correcto desde
* el usuario o no.
* @see servidorSMS#ConexionUsuario()*/
public int RecibirACKB(Socket client) throws  ClassNotFoundException
{
        try
        {
                ObjectInputStream in=new ObjectInputStream(client.getInputStream());
                String input=(String) in.readObject();
                //System.out.println("Client says : ");
                System.out.println(input);
                if(!input.equals(null))
                {
                        if(input.equals("codB_OK"))
                                return 1;
                }
                else
                        return 0;

        }catch(IOException e)
        {
                System.out.println("Error de IO");
        }
        return 0;
}
/**Metodo que recibe el codigo B por parte del servidor de la CA
* @param client variable en la que se almacena un socket seguro SSL que permite establecer la
conexion
* entre un el servidor de SMS y el servidor de la CA
* @see servidorSMS#conexionServCA()*/
public void RecibirCodBServSMS(SSLSocket client) throws IOException, ClassNotFoundException{
        ObjectInputStream codB=new ObjectInputStream(client.getInputStream());
        codigoB=(String) codB.readObject();
        System.out.println("Codigo B: "+codigoB);
}
/**Metodo que envia un ack al servidor de la CA
* @param client variable en la que se almacena un socket seguro SSL que permite establecer la conexion.
* @param respuesta String que se envia al usuario que almacena una cadena de caracteres que
* informa al usuario
* si se ha recibido correctamente el ACK por parte del servidor de SMS.
* @see servidorSMS#conexionServCA() */
public void EnviarACKServidorCA(SSLSocket client, String respuesta) throws IOException
{
        ObjectOutputStream ack=new ObjectOutputStream(client.getOutputStream());
        ack.writeObject(respuesta);
        ack.flush();
        ack.close();
}
}
```

Código del servidor que realiza la Autenticación

Autenticar.java

```
import java.io.FileInputStream;
import java.io.IOException;
import java.security.InvalidKeyException;
import java.security.KeyManagementException;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.SignatureException;
import java.security.UnrecoverableKeyException;
import java.security.cert.CertificateException;
import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSession;
import javax.net.ssl.SSLSocket;

/**Clase que simula la conexion a una plataforma, autenticando al usuario, dependiendo de
 * si la firma enviada por el usuario es correcta y de si su certificado ha sido firmado por
 * una CA.
 * @author: Eva M. Blanco Delgado
 * @version: 18/01/2014_v0.0*/
public class Autenticar {
//Inicialización de las variables que se utilizan
String ksName = "ksServidorAut.jks";
char ksPass[] = "109638".toCharArray();
char ctPass[] = "109638".toCharArray();
int Puerto2 = 5000;
String email;

/**Instancia de la clase que pone en funcionamiento el servidor*/
Autenticar() throws UnrecoverableKeyException, KeyManagementException, KeyStoreException,
NoSuchAlgorithmException, CertificateException, ClassNotFoundException, IOException,
InvalidKeyException, IllegalStateException, NoSuchProviderException, SignatureException {

                conexion();
}

/**Metodo que se encarga de la gestion (llamadas y recibir datos) de los distintos metodos
* del programa.
* @param servidor variable que contiene el socket seguro SSL que permite establecer una
* conexion.
* @throws ClassNotFoundException
* @throws SignatureException
* @throws NoSuchProviderException
* @throws NoSuchAlgorithmException
* @throws CertificateException
* @throws InvalidKeyException
* @throws KeyStoreException
* @see GestionAutenticacion#recibirDatos
* @see GestionAutenticacion#enviarDatos
* @see #conexion()*/
public void GestionInfo(SSLServerSocket servidor) throws IOException, ClassNotFoundException,
InvalidKeyException, CertificateException, NoSuchAlgorithmException, NoSuchProviderException,
SignatureException, KeyStoreException
{
        String respuesta=null;
        boolean acc_verificado;
        SSLSocket client = (SSLSocket)servidor.accept();
        SSLSession ss = client.getSession();
        client.startHandshake();
        System.out.println("Cipher suite"+ss.getCipherSuite());
        System.out.println("Cipher protocol"+ss.getProtocol());

        GestionAutenticacion ga = new GestionAutenticacion();
        //Llamada al método para recibir los datos del usuario
        acc_verificado=ga.recibirDatos(client);
```

```
        if(acc_verificado==true)
        {
                //Si se verifican los datos se envía una respuesta
                respuesta="BIENVENIDO!!!!!!!!";
                ga.enviarDatos(client, respuesta);
                client.close();
        }
        else
        {
                //Si no se verifican se envía un mensaje que deniega el acceso al usuario
                respuesta="noAcceso";
                ga.enviarDatos(client, respuesta);
                client.close();
        }
}
/**Metodo que establece la conexion con el usuario, llama a los metodos que autentican al
* usuario y elimina el certificado temporal que se ha generado.
* @see #GestionInfo(SSLServerSocket)*/
private void conexion() throws KeyStoreException, NoSuchAlgorithmException, CertificateException,
UnrecoverableKeyException, KeyManagementException, ClassNotFoundException, IOException,
InvalidKeyException, IllegalStateException, NoSuchProviderException, SignatureException{

while(true){
        //Se carga el keystore del servidor donde están sus claves privadas
        KeyStore ks = KeyStore.getInstance("JKS");
        ks.load(new FileInputStream(ksName), ksPass);
        KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
        kmf.init(ks, ctPass);

        //Se establece una conexion TLS
        SSLContext sc = SSLContext.getInstance("TLS");

        //Se inicializan los parámetros de la conexion
        sc.init(kmf.getKeyManagers(), null, null);
        SSLServerSocketFactory ssf = sc.getServerSocketFactory();
        SSLServerSocket servidor = (SSLServerSocket) ssf.createServerSocket(Puerto2);
        servidor.setNeedClientAuth(false);

        /*Se llama al método que se encarga de gestionar las operaciones internas del servidor
        /para la autenticación del usuario*/
        GestionInfo(servidor);

        //Se cierra la conexion del servidor.
        servidor.close();

        GestionAutenticacion ga = new GestionAutenticacion();
        ga.BorrarCertificadoTemp();
        }
}
/**Metodo principal
* @param args argumento que se le pasa al mÃ©todo principal*/
public static void main(String[] args){
        try {
                new Autenticar();
        } catch (UnrecoverableKeyException e) {
                e.printStackTrace();
        } catch (KeyManagementException e) {
                e.printStackTrace();
        } catch (InvalidKeyException e) {
                e.printStackTrace();
        } catch (KeyStoreException e) {
                e.printStackTrace();
        } catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
        } catch (CertificateException e) {
                e.printStackTrace();
        } catch (IllegalStateException e) {
                e.printStackTrace();
        } catch (NoSuchProviderException e) {
                e.printStackTrace();
        } catch (SignatureException e) {
                e.printStackTrace();
        } catch (ClassNotFoundException e) {
```

```
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
}
}
```

GestionAutenticacion.java

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.security.InvalidKeyException;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.PublicKey;
import java.security.Security;
import java.security.Signature;
import java.security.SignatureException;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLSession;
import javax.net.ssl.SSLSocket;

import org.bouncycastle.jce.provider.BouncyCastleProvider;


/**Clase que se ocupa de gestionar las operaciones necesarias
 * para llevar a cabo la autenticacion.
 * @author: Eva M. Blanco Delgado
 * @version: 18/01/2014_v0.0*/
public class GestionAutenticacion {
//Inicialización de las variables que se utilizan
final String PATH="/home/eva/trabajoEclipse2/ServidorAutenticar/Temp/CertTemp.pem";
String tsName = "tsServidorAut.jks";
char ksPass[] = "109638".toCharArray();
/**El proveedor es BouncyCastle, permite acceder al almacen de claves para obtener la
 * clave pÃºblica del certificado de la CA */
static
{
        Security.addProvider(new BouncyCastleProvider());
}
/**Metodo que guarda el certificado enviado por el usuario en un archivo temporal
* @param certificado String que almacena el certificado que se recibe desde el usuario
* @see #recibir(SSLSocket)*/
private void GuardarCertificado(String certificado) throws IOException
{
        BufferedWriter bw= new BufferedWriter(new FileWriter(new File(PATH)));
        PrintWriter wr= new PrintWriter(bw);
        wr.write(certificado);
        wr.flush();
        wr.close();
        bw.close();
}
/**Metodo que devuelve una clave publica, en este caso, la clave publica
* del usuario
* @return <code>pk</code>: la clave pÃºblica del certificado del usuario
* @see #verificarFirma(byte[], int)*/
private PublicKey clavePublica() throws FileNotFoundException, CertificateException
```

```
{
        InputStream inStream = null;
        inStream = new FileInputStream(PATH);
        CertificateFactory cf = CertificateFactory.getInstance("X.509");
        X509Certificate cert = (X509Certificate)cf.generateCertificate(inStream);
        PublicKey pk = cert.getPublicKey();
        return pk;
}
/**Metodo que verifica que un codigo aleatorio generado por el usuario y enviado al servidor
* ha sido firmado por el certificado perteneciente al usuario.
* @param firma cadena de bytes que almacena la firma de un numero aleatorio generado por el
* certificado del usuairo
* @param numero numero aleatorio enviado por el usuario
* @return <code>true</code>: si se verifica que la firma enviada pertenece al usuario
* @return <code>false</code>: si no se verifica que la firma pertenece al usuario
* @see #clavePublica()
* @see #verificarAcceso(byte[], int)*/
private boolean verificarFirma(byte [] firma, int numero) throws FileNotFoundException,
CertificateException, NoSuchAlgorithmException, NoSuchProviderException, InvalidKeyException,
SignatureException
{
        PublicKey kp = clavePublica();
        Signature s = Signature.getInstance("SHA1withRSA", "BC");
        s.initVerify(kp);
        byte buf[]=(Integer.toString(numero)).getBytes();
        s.update(buf);
        if(s.verify(firma))
        {
                System.out.println("\nFIRMA VERIFICADA");
                return true;
        }
        else
        {
                System.out.println("\nFIRMA NO VERIFICADA");
                return false;
        }
}
/**Metodo que comprueba que el certificado enviado por el usuario ha sido firmado
* con la clave publica del certificado de la Autoridad Certificadora
* @return <code>true</code>: si se verifica que el certificado enviado por el usuario ha
* sido firmado por la CA
* @return <code>false</code>: si se verifica que el certificado enviado por el usuario
* no ha sido firmado por la CA
* @see #verificarAcceso(byte[], int)*/
private boolean verificarCertificado() throws NoSuchAlgorithmException, IOException,
KeyStoreException, InvalidKeyException, NoSuchProviderException, SignatureException
{
        try{
                KeyStore ts = KeyStore.getInstance("JKS");
                ts.load(new FileInputStream(tsName), ksPass);
                X509Certificate CAcert = (X509Certificate)ts.getCertificate("caalias");
                InputStream inStream = null;
                inStream = new FileInputStream(PATH);
                CertificateFactory cf = CertificateFactory.getInstance("X.509");
                X509Certificate cert = (X509Certificate)cf.generateCertificate(inStream);
                cert.verify(CAcert.getPublicKey());
                return true;
        }catch(CertificateException e){
                e.printStackTrace();
        return false;
        }
}
/**Metodo que comprueba que tanto la firma enviada al usuario le corresponde al usuario
* que quiere autenticarse como que el certificado enviado ha sido firmado por la CA
* @param firma cadena de bytes que contienen la firma del usuario
* @param nAleatorio numero aleatorio enviado por el usuario para poder verificar
* la firma del usuario
* @return <code>true</code>: si se ha verificado tanto la firma como el certificado.
* @return <code>false</code>: si no se ha verificado o la firma o el certificado
* @see #verificarFirma(byte[], int)
* @see #verificarCertificado() */
```

```
private boolean verificarAcceso(byte [] firma, int nAleatorio) throws InvalidKeyException,
CertificateException, NoSuchAlgorithmException, NoSuchProviderException, SignatureException,
KeyStoreException, IOException
{
	boolean veriFirma, vericert;
	//Verificar certificado
	vericert=verificarCertificado();
	//Verificar firma
	veriFirma = verificarFirma(firma, nAleatorio);
	if(vericert == true && veriFirma==true)
		return true;
	else
		return false;
}
/**Metodo que recibe el String con el certificado
* @throws IOException
* @throws ClassNotFoundException
* @throws KeyStoreException
* @throws SignatureException
* @throws NoSuchProviderException
* @throws NoSuchAlgorithmException
* @throws CertificateException
* @throws InvalidKeyException
* @see #GuardarCertificado(String)
* @see Autenticar#GestionInfo(SSLServerSocket)
* */
public boolean recibirDatos(SSLSocket client) throws IOException, ClassNotFoundException,
InvalidKeyException, CertificateException, NoSuchAlgorithmException, NoSuchProviderException,
SignatureException, KeyStoreException
{
	boolean acceso;
	ObjectInputStream cert=new ObjectInputStream(client.getInputStream());
	ObjectInputStream signature=new ObjectInputStream(client.getInputStream());
	ObjectInputStream numero=new ObjectInputStream(client.getInputStream());
	String certificado=(String) cert.readObject();
	byte [] firma=(byte[]) signature.readObject();
	int nAleatorio=numero.readInt();
	GuardarCertificado(certificado);
	acceso=verificarAcceso(firma, nAleatorio);
	if(acceso == true)
		return true;
	else
		return false;
}
/**Metodo que envia la respuesta al usuario, estableciendo si el usuario tiene
* acceso al sistema o no.
* @param servidor parametro que permite establecer una comunicacion.
* @param respuesta String que contiene la respuesta que se le va a enviar al usuario.
* @see Autenticar#GestionInfo(SSLServerSocket)
*/
public void enviarDatos(SSLSocket client, String respuesta) throws IOException
{
	ObjectOutputStream res = new ObjectOutputStream(client.getOutputStream());
	res.writeObject(respuesta);
	res.flush();
}
/**Metodo que permite eliminar el archivo temporal en el que se almacena el certificado
* del usuario para ahorrar recursos al sistema*/
public void BorrarCertificadoTemp()
{
	File f = new File(PATH);
	if(f.delete())
		System.out.println("\nEL CERTIFICADO HA SIDO ELIMINADO");
	else
		System.out.println("\nEL CERTIFICADO NO SE PUEDE ELIMINAR");
}
}
```

Código de la aplicación

MainActivity.java

```
package com.id_digital_pfc;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;

/**
 * Actividad principal de la aplicacion. Se muestra solo una unica vez cuando se inicia la
aplicacion.
 * @author: Eva M. Blanco Delgado
 * @version: 18/01/2014_v0.0
*/
public class MainActivity extends Activity {
/**Método que permite el arranque inicial de la aplicación*/
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        hilo();
}
/**Hilo que espera unos segundos permitiendo que se muestra el icono de la aplicación.*/
protected void hilo()
{
        new Thread(new Runnable() {
                public void run() {
                        try {
                                Thread.sleep(2000);
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                        runOnUiThread(new Runnable() {
                        public void run() {
                                Intent intent = new Intent(MainActivity.this, IntroducirIP.class);
                                startActivity(intent);
                                MainActivity.this.finish();
                        }
                    });
            }
        }).start();
}
}
```

IntroducirIP.java

```
package com.id_digital_pfc;

import android.os.Bundle;
import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.widget.EditText;
public class IntroducirIP extends Activity {
String IP;

/**
* Metodo que permite el arranque inicial de la actividad.
* También llama al metodo <code>DialogoIntroducirIP</code>
* @param savedInstanceState Mantiene el estado de la actividad.
* @see #DialogoIntroducirIP()
*/
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_introducir_ip);
        //Se llama al método DialogoIntroducirIP
        dialogoIntroducirIP();
}
/**
* Metodo que muestra un dialogo para que el usuario introduzca la IP que va a utilizar para
* la conexion. Tambien llama al metodo <code>SiguienteActividad</code>
* @see #SiguienteActividad(String)
*/
@SuppressLint("NewApi")
public void dialogoIntroducirIP() {
//Se construye el diálogo que informa al usuario para que introduzca la IP
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setIcon(android.R.drawable.ic_dialog_alert); //Se muestra un icono de alerta
    builder.setIconAttribute(android.R.attr.alertDialogIcon);
    builder.setTitle("Atención!"); //Se pone un titulo al diálogo
    builder.setMessage(R.string.introduzca_ip); //Se muestra el mensaje al usuario
    final EditText input = new EditText(this); //Se crea un editText para que el usuario meta la
dirección IP
    builder.setView(input);
    builder.setNeutralButton("Aceptar", new DialogInterface.OnClickListener() {
        //Al pulsar sobre el botón Aceptar se llama a la actividad SiguienteActividad
         @Override
         public void onClick(DialogInterface dialog, int id) {
                   IP=input.getText().toString();
                   siguienteActividad(IP);
        }
   });
//Se muestra el diálogo
    builder.show();
}
/**
* Metodo que inicia la siguiente actividad y finaliza la actual actividad.
* @param IP un <code>String</code> con la dirección IP que se va a utilizar.
* @see #DialogoIntroducirIP()
*/
public void siguienteActividad(String IP)
{
        //Se lanza la siguiente actividad.
        Intent intent = new Intent(this, ElegirAccion.class);
        //Se pasa la IP a la siguiente actividad.
        intent.putExtra("DIR_IP", IP);
        //Se inicia la siguiente actividad y se finaliza la actual.
        startActivity(intent);
        this.finish();
}
}
```

ElegirAccion.java

```
package com.id_digital_pfc;

import android.os.Bundle;
import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;

/**
 * Segunda actividad de la aplicacion. Muestra las tres opciones entre las que el usuario
 * puede elegir: <code>Registrarse</code>, <code>Iniciar sesión</code> y <code>Eliminar
Certificado</code>.
 * @author: Eva M. Blanco Delgado
 * @version: 18/01/2014_v0.0
*/
public class ElegirAccion extends Activity {
String IP;
/**
* Metodo que permite el arranque inicial de la actividad.
```

```
* Recupera la <code>IP</code> desde la clase anterior
* @param savedInstanceState Mantiene el estado de la actividad.
*/
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_elegir_accion);
        Bundle bundle=getIntent().getExtras();
        IP = bundle.getString("DIR_IP");
}

/** Metodo al que se le llama cuando se presiona el boton para iniciar sesion.
* LLama la siguiente actividad correspondiente a la accion de iniciar sesion y le pasa
* un String con la <code>IP </code>.
* @param view permite la vista del boton <code>Iniciar sesion </code>*/
public void iniciarSesion(View view) {
    // Al pulsar el botón se pasa a la siguiente actividad que es EliminarCertificado
        Intent intent = new Intent(this, TipoAutenticacion.class);
        intent.putExtra("DIR_IP", IP);
        startActivity(intent);
        this.finish();
}
/** Metodo al que se le llama cuando se presiona el boton para registrarse.
* LLama la siguiente actividad correspondiente a la accion de Registrarse y le pasa
* un String con la <code>IP </code>.
* @param view permite la vista del boton <code>Registrarse </code>
*/
public void registrarse(View view) {
    // Al pulsar el botón se pasa a la siguiente actividad que es DatosRegistro1
        Intent intent = new Intent(this, DatosRegistro1.class);
        intent.putExtra("DIR_IP", IP);
        startActivity(intent);
        this.finish();
}
/** Metodo al que se le llama cuando se presiona el boton para eliminar un certificado
* LLama la siguiente actividad correspondiente a la accion de eliminar un certificado y le pasa
* un String con la <code>IP </code>.
* @param view permite la vista del boton <code>Eliminar certificado </code>*/
public void eliminarCertificadoUsuario(View view) {
    // Al pulsar el botón se pasa a la siguiente actividad que es EliminarCertificado
        Intent intent = new Intent(this, EliminarCertificado.class);
        intent.putExtra("DIR_IP", IP);
        startActivity(intent);
        this.finish();
}
/**Metodo que muestra al usuario un mensaje para que elija si desea salir o no*/
@SuppressLint("NewApi")
public void mensajeSalir()
{
         new AlertDialog.Builder(this)
      .setIcon(android.R.drawable.ic_dialog_alert)
      .setIconAttribute(android.R.attr.alertDialogIcon)
      .setTitle("Salir")
      .setMessage("¿Está seguro?")
      .setNegativeButton("Cancelar", null)//sin listener
      .setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {//un listener que al
pulsar, cierre la aplicacion
        @Override
        public void onClick(DialogInterface dialog, int which){
          //Salir
          ElegirAccion.this.finish();

        }
      })
      .show();
}
/**Realiza una accion u otra dependiendo del item elegido por el usuario*/
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_meter_IP:
        Intent intent = new Intent(this, IntroducirIP.class);
                startActivity(intent);
```

```
                this.finish();
            return true;
        case R.id.menu_salir:
        mensajeSalir();
        return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
/**Muestra una serie de opciones en el menu al usuario*/
@Override
public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; Añade opciones al menú.
        getMenuInflater().inflate(R.menu.activity_elegir_accion, menu);
        return super.onCreateOptionsMenu(menu);
}
}
```

DatosRegistro1.java

```
package com.id_digital_pfc;

import android.os.Bundle;
import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

/**
 * Primera actividad dentro de la acción de registro. Muestra una serie de campos
 * que el usuario tiene que rellenar
 * @author: Eva M. Blanco Delgado
 * @version: 18/01/2014_v0.0
*/
public class DatosRegistro1 extends Activity {
//Declaración de variables
private EditText Nombre;
private EditText Apellido1;
private EditText Apellido2;
private EditText Pais;
private EditText Ciudad;
String nom;
String ape1;
String ape2;
String pais;
String ciu;
String IP;
/**
* Metodo que permite el arranque inicial de la actividad.
* Recupera la <code>IP</code> desde la clase anterior
* @param savedInstanceState Mantiene el estado de la actividad.
*/
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_datos_registro1);
        Bundle bundle=getIntent().getExtras();
        IP = bundle.getString("DIR_IP");
}
/** Metodo que obtiene los datos introducidos por un usuario y los pasa a la siguiente actividad
<code>(DatosRegistro2)</code>.
* @param view permite la vista del boton <code>Iniciar sesion </code>
* */
public void adelante(View view) {
// Al pulsar el botón se adquieren los datos y se pasan a la siguiente actividad
//se obtienen los datos especificados en el archivo .xml perteneciente a esta actividad.
```

```
        Nombre = (EditText)findViewById(R.id.NomUsuario);
        Apellido1 = (EditText)findViewById(R.id.ApellidoUsuario1);
        Apellido2 = (EditText)findViewById(R.id.ApellidoUsuario2);
        Pais = (EditText)findViewById(R.id.NomPais);
        Ciudad = (EditText)findViewById(R.id.NomCiudad);
        //Se convierten los datos obtenidos a String
        nom=Nombre.getText().toString();
        ape1=Apellido1.getText().toString();
        ape2=Apellido2.getText().toString();
        pais=Pais.getText().toString();
        ciu=Ciudad.getText().toString();

//Si los espacios a rellenar por el usuario están vacios se muestra un mensaje por pantalla
//indicándolo
        if(nom.equals("") & ape1.equals("") & ape2.equals("") & pais.equals("") & ciu.equals(""))
        {
                Toast.makeText(getApplicationContext(), "Aún no ha introducido ningún dato",
Toast.LENGTH_LONG).show();
        }
//Si el parámetro dedicado al nombre está vacio, se muestra un mensaje indicándolo
        else if(nom.equals(""))
        {
                Toast.makeText(getApplicationContext(), "Por favor, introduzca su nombre",
Toast.LENGTH_LONG).show();
        }
//Si el parámetro perteneciente al primer apellido está vacio, se muestra un mensaje indicándolo
        else if(ape1.equals(""))
        {
                Toast.makeText(getApplicationContext(), "Por favor, introduzca su primer apellido",
Toast.LENGTH_LONG).show();
        }
//Si el parámetro perteneciente al segundo apellido está vacío, se muestra un mensaje indicándolo
        else if (ape2.equals(""))
        {
                Toast.makeText(getApplicationContext(), "Por favor, introduzca su segundo
apellido", Toast.LENGTH_LONG).show();
        }
//Si el parámetro perteneciente al país está vacío, se muestra un mensaje indicándolo
        else if (pais.equals(""))
        {
                Toast.makeText(getApplicationContext(), "Por favor, introduzca el país en el que
reside actualmente", Toast.LENGTH_LONG).show();
        }
//Si el parámetro perteneciente a la ciudad está vacio, se muestra un mensaje indicándolo
        else if (ciu.equals(""))
        {
                Toast.makeText(getApplicationContext(), "Por favor, introduzca su ciudad de
residencia", Toast.LENGTH_LONG).show();
        }
//Si todos los parámetros están rellenos, se pasa los valores a la siguiente actividad
//y se inicia la misma.
        else
        {
                Intent intent = new Intent(this, DatosRegistro2.class);
                intent.putExtra("NOMBRE", nom);
                intent.putExtra("APELLIDO1", ape1);
                intent.putExtra("APELLIDO2", ape2);
                intent.putExtra("PAIS", pais);
                intent.putExtra("CIUDAD", ciu);
                intent.putExtra("DIR_IP", IP);
                startActivity(intent);
                this.finish();
        }
}
/**Metodo que permite salir de la aplicacion si el usuario pulsa el boton de atras del movil.
* @param keyCode codigo del boton de atras del movil que es transparente al usuario
* @param event detecta la accion del usuario*/
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
  if (keyCode == KeyEvent.KEYCODE_BACK) {
          Intent intent = new Intent(this, ElegirAccion.class);
          intent.putExtra("DIR_IP", IP);
          startActivity(intent);
```

```
          this.finish();
// Si el listener devuelve true, significa que el evento esta procesado, y nadie debe hacer nada
mas
        return true;
        }
//para las demas cosas, se reenvia el evento al listener habitual
  return super.onKeyDown(keyCode, event);
}
/**Metodo que muestra al usuario un mensaje para que elija si desea salir o no*/
@SuppressLint("NewApi")
public void mensajeSalir()
{
         new AlertDialog.Builder(this)
      .setIcon(android.R.drawable.ic_dialog_alert)
      .setIconAttribute(android.R.attr.alertDialogIcon)
      .setTitle("Salir")
      .setMessage("¿Está seguro?")
      .setNegativeButton("Cancelar", null)//sin listener
      .setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {//un listener que al
pulsar, cierre la aplicacion
        @Override
        public void onClick(DialogInterface dialog, int which){
                DatosRegistro1.this.finish();
                }
      })
      .show();
}
/**Realiza una accion u otra dependiendo del item elegido por el usuario*/
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_volver_inicio:
        Intent intent = new Intent(this, ElegirAccion.class);
        intent.putExtra("DIR_IP", IP);
                startActivity(intent);
                this.finish();
            return true;
        case R.id.menu_salir2:
        mensajeSalir();
        return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
/**Muestra una serie de opciones en el menu al usuario*/
@Override
public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; añade opciones al menú
        getMenuInflater().inflate(R.menu.activity_datos_registro1, menu);
        return super.onCreateOptionsMenu(menu);
}
}
```

DatosRegistro2.java

```
package com.id_digital_pfc;

import android.os.Bundle;
import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

/**
 * Segunda actividad dentro de la accion de registro. Muestra una serie de campos
```

```
 * que el usuario tiene que rellenar y un boton para el envio de datos.
 * @author: Eva M. Blanco Delgado
 * @version: 18/01/2014_v0.0
*/
public class DatosRegistro2 extends Activity {

//Declaración de variables
private EditText email1;
private EditText email2;
private TextView incorrecto;
private EditText tlfm1;
private EditText tlfm2;
private TextView incorrecto2;
String mail1;
String mail2;
String movil1;
String movil2;
String nom;
String ape1;
String ape2;
String pais;
String ciu;
String IP;
int longitudn;
/**
* Metodo que permite el arranque inicial de la actividad.
* Recupera la <code>IP</code>, <code>nombre</code>, <code>1er apellido</code>, <code>2º
apellido</code>,
* <code>localidad</code> y <code>Pais</code> del usuario desde la actividad anterior.
* @param savedInstanceState Mantiene el estado de la actividad.
*/
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_datos_registro2);
        //Se reciben todos los parámetros de la actividad anterior
        Bundle bundle=getIntent().getExtras();
        nom = bundle.getString("NOMBRE");
        ape1 = bundle.getString("APELLIDO1");
        ape2 = bundle.getString("APELLIDO2");
        pais = bundle.getString("PAIS");
        ciu = bundle.getString("CIUDAD");
        IP = bundle.getString("DIR_IP");
}
/** Metodo que obtiene los datos introducidos por un usuario y los pasa a la siguiente actividad
<code>(EnvioDatos)</code>
* al pulsar el boton <code>Enviar Datos</code>
* @param view permite la vision del boton*/
public void envioDatos(View view) {
// se obtienen los datos especificados en el archivo .xml perteneciente a esta actividad.
        email1 = (EditText)findViewById(R.id.diremail);
        email2 = (EditText)findViewById(R.id.diremail2);
        incorrecto = (TextView)findViewById(R.id.incorrecto);
        tlfm1 = (EditText)findViewById(R.id.numMov);
        tlfm2 = (EditText)findViewById(R.id.numMov2);
        incorrecto2 = (TextView)findViewById(R.id.incorrecto2);
        // Se convierten los datos obtenidos a Strings
        mail1=email1.getText().toString();
        mail2=email2.getText().toString();
        ovil1=tlfm1.getText().toString();
        movil2=tlfm2.getText().toString();
        // Se averigua la longitud del número de teléfono
        longitudn=movil1.length();
        // Si todos los parámetros están vacios se muestra un mensaje avisando
        if(mail2.equals("") & movil2.equals("") & movil1.equals("") & movil2.equals(""))
        {
                Toast.makeText(getApplicationContext(), "Aún no ha introducido ninguno de los datos
solicitados", Toast.LENGTH_LONG).show();
        }
        //Si tanto los emails como los números de teléfono no coinciden se muestra un mensaje, se
        // indica que la operación es incorrecta y se vuelven a mostrar los espacios vacios
        else if(!mail2.equals(mail1) & !movil2.equals(movil1))
        {
```

```
                Toast.makeText(getApplicationContext(), "Tanto los números de móvil, como los
correos electrónicos no coinciden. Por favor, vuelva a introducir los datos",
Toast.LENGTH_LONG).show();
                tlfm2.setText("");
                tlfm1.setText("");
                email2.setText("");
                email1.setText("");
                incorrecto.setText("Incorrecto");
                incorrecto2.setText("Incorrecto");
                return;
        }
        // Si los emails no coinciden se muestra un mensaje, se indica que la operación no es
correcta y
        // se vuelven a mostrar los espacios vacíos
        else if(!mail2.equals(mail1))
        {
                Toast.makeText(getApplicationContext(), "Los correos electrónicos no coinciden. Por
favor, vuelva a introducir su correo electrónico", Toast.LENGTH_LONG).show();
                email2.setText("");
                email1.setText("");
                incorrecto.setText("Incorrecto");
                incorrecto2.setText("");
                return;
        }
        // Si los móviles no coinciden se muestra un mensaje, se indica que la operación no es
correcta y
        // se vuelven a mostrar los espacios vacíos
        else if(!movil2.equals(movil1))
        {
                Toast.makeText(getApplicationContext(), "Los números de teléfono móvil no
coinciden. Por favor, vuelva a introducir su número de móvil", Toast.LENGTH_LONG).show();
                tlfm2.setText("");
                tlfm1.setText("");
                incorrecto2.setText("Incorrecto");
                incorrecto.setText("");
                return;
        }
        // Si la longitud del teléfono que se introduce no es 9 (tamaño normal de un número de
teléfono)
        // se avisa al usuario
        else if(longitudn!=9)
        {
                Toast.makeText(getApplicationContext(), "La longitud del número de móvil tiene que
ser de 9 cifras", Toast.LENGTH_LONG).show();
                tlfm2.setText("");
                tlfm1.setText("");
                incorrecto2.setText("Incorrecto");
                incorrecto.setText("");
                return;
        }
        // Si todos los datos se han metido correctamente, se evita que se muestren los indicadores
de
        // que se ha metido un dato incorrecto, se pasan los datos a la siguiente actividad
(EnvioDatos)
        // y se inicia la misma
        else
        {
                incorrecto.setText("");
                incorrecto2.setText("");
                Intent intent = new Intent(this, EnvioDatos.class);
                intent.putExtra("EMAIL", mail1);
                intent.putExtra("MOVIL", movil1);
                intent.putExtra("NOMBRE", nom);
                intent.putExtra("APELLIDO1", ape1);
                intent.putExtra("APELLIDO2", ape2);
                intent.putExtra("PAIS", pais);
                intent.putExtra("CIUDAD", ciu);
                intent.putExtra("DIR_IP", IP);
                startActivity(intent);
                this.finish();
        }
}

/**Metodo que permite salir de la aplicacion si el usuario pulsa el boton de atras del movil.
```

```
* @param keyCode codigo del boton de atras del movil que es transparente al usuario
* @param event detecta la accion del usuario*/
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
  if (keyCode == KeyEvent.KEYCODE_BACK) {
          Intent intent = new Intent(this, DatosRegistro1.class);
                intent.putExtra("DIR_IP", IP);
                  startActivity(intent);
                  this.finish();
    // Si el listener devuelve true, significa que el evento esta procesado, y nadie debe hacer
nada mas
    return true;
}
//para las demas cosas, se reenvia el evento al listener habitual
return super.onKeyDown(keyCode, event);
}

/**Metodo que muestra al usuario un mensaje para que elija si desea salir o no*/
@SuppressLint("NewApi")
public void mensajeSalir()
{
         new AlertDialog.Builder(this)
         .setIcon(android.R.drawable.ic_dialog_alert)
         .setIconAttribute(android.R.attr.alertDialogIcon)
         .setTitle("Salir")
         .setMessage("¿Está seguro?")
         .setNegativeButton("Cancelar", null)//sin listener
         .setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {//un listener que al
pulsar, cierre la aplicacion
         @Override
         public void onClick(DialogInterface dialog, int which){
         //Salir
          DatosRegistro2.this.finish();
          }
})
.show();
}
/**Realiza una accion u otra dependiendo del item elegido por el usuario*/
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_volver_inicio2:
        Intent intent = new Intent(this, ElegirAccion.class);
        intent.putExtra("DIR_IP", IP);
                startActivity(intent);
                this.finish();
            return true;
        case R.id.menu_salir3:
        mensajeSalir();
        return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
/**Método que muestra una serie de opciones en el menu al usuario*/
@Override
public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; Añade opciones al menú
        getMenuInflater().inflate(R.menu.activity_datos_registro2, menu);
        return super.onCreateOptionsMenu(menu);
}
}
```

EnvioDatos.java

```
package com.id_digital_pfc;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.StreamCorruptedException;
import java.net.Socket;
```

```
import java.net.UnknownHostException;
import java.security.KeyManagementException;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.cert.CertificateException;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManagerFactory;
import org.bouncycastle.operator.OperatorCreationException;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.os.StrictMode;
import android.annotation.SuppressLint;
import android.annotation.TargetApi;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.ProgressDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.res.Resources.NotFoundException;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

/**
 * Tercera actividad dentro de la accion de registro. Muestra un
 * dialogo de informacion al usuario y envia los datos
 * que el usuario tiene que rellenar y un boton para el envio de datos.
 * @author: Eva M. Blanco Delgado
 * @version: 18/01/2014_v0.0
*/
public class EnvioDatos extends Activity {
//Declaración de variables y objetos
String nom;
String ape1;
String ape2;
String pais;
String ciu;
String email;
String movil;
String mensajeDialogo;
String mensajeVolveraInicio;
String IP;
/**String en el que se guarda el codigo A recibido desde el servidor de la CA
 * @see #conexionServCA()*/
static String codigoA;
/**String en el que se guarda el codigo B recibido desde el servidor de la CA
 * @see #conexionServCA()*/
static String codigoB;
int Puerto=2000;
int Puerto2=3000;
int Puerto3=4000;
char ksPass[] = "109638".toCharArray();
final Context context=null;
private TextView CodigoUser;
int CodigoRec;
String csr;
String cert="";
final Handler mhandler = new Handler();
ProgressDialog progreso;
ProgressDialog progreso2;
int n;
int ackB=1;
Socket cliente;
/** Socket que permite el establecimiento de una conexion segura entre el
* usuario y el servidor de la CA para recibir los codigos A y B y enviar el
* CSR
* @see #conexionServCA()
```

```
* @see #Obcert()*/
static Socket socket;
/**
* Metodo que permite el arranque inicial de la actividad.
* Recupera la <code>IP</code>, <code>nombre</code>, <code>1er apellido</code>, <code>2º
apellido</code>,
* <code>localidad</code>, <code>País</code>, <code>email</code> y <code>Numero de movil</code> del
usuario desde la actividad anterior.
* Tambien llama al metodo <code>mostrarDialogo</code>
* @param savedInstanceState Mantiene el estado de la actividad.
* @see #mostrarDialogo()
*/
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_envio_datos);
        //Se reciben los datos enviados en la actividad anterior
        Bundle bundle=getIntent().getExtras();
        nom = bundle.getString("NOMBRE");
        ape1 = bundle.getString("APELLIDO1");
        ape2 = bundle.getString("APELLIDO2");
        pais = bundle.getString("PAIS");
        ciu = bundle.getString("CIUDAD");
        email = bundle.getString("EMAIL");
        movil = bundle.getString("MOVIL");
        IP = bundle.getString("DIR_IP");
        // Llamada al método mostrarDialogo
        mostrarDialogo();
    }
/**Metodo que muestra un dialogo que informa al usuario sobre los pasos que debe seguir
* @see #aceptar()
* @see #cancelar()*/
public void mostrarDialogo(){
        // String que se mostrará al usuario
        mensajeDialogo="A continuación, usted recibirá un SMS en su bandeja de entrada con un
codigo. Este número es personal, identifica su conexión como usuario y es necesario para la
obtención de su certificado digital.\n¿Desea continuar con el proceso?";
        //Creación del diálogo
        AlertDialog.Builder dialogo = new AlertDialog.Builder(this);
        dialogo.setIcon(R.drawable.ic_launcher); //Muestra el icono de la aplicación
        dialogo.setTitle("Información");
        dialogo.setMessage(mensajeDialogo);
        dialogo.setCancelable(false);
        //Si se pulsa aceptar se llama al método aceptar()
        dialogo.setPositiveButton("Confirmar", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialogo, int id) {
                dialogo.dismiss();
                try {
                        //Se llama al método aceptar y se avisa si hay algún error
                        aceptar();
                } catch (KeyManagementException e) {
                        e.printStackTrace();
                } catch (KeyStoreException e) {
                        e.printStackTrace();
                } catch (NoSuchAlgorithmException e) {
                        e.printStackTrace();
                } catch (CertificateException e) {
                        e.printStackTrace();
                } catch (NotFoundException e) {
                        e.printStackTrace();
                } catch (NoSuchProviderException e) {
                        e.printStackTrace();
                } catch (OperatorCreationException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                } catch (ClassNotFoundException e) {
                        e.printStackTrace();
                }
                }
        });
        //Si se pulsa el botón "cancelar", se llama al método cancelar
        dialogo.setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
```

```
                public void onClick(DialogInterface dialogo, int id) {
                        dialogo.dismiss();
                        cancelar();
                        }
                });
        dialogo.show();
}
/**Metodo para la utilización de un hilo en un determinado momento de la aplicacion
* el cual realiza la conexion con los servidores
* @param progreso variable para el dialogo de progreso
* @see #aceptar()
* @see #ejecutar
* */
protected void hilo(ProgressDialog progreso)
{
        new Thread(new Runnable() {
                public void run() {
                        try {
                                conexionServCA();
                        } catch (KeyManagementException e) {
                                e.printStackTrace();
                        } catch (KeyStoreException e) {
                                e.printStackTrace();
                        } catch (NoSuchAlgorithmException e) {
                                e.printStackTrace();
                        } catch (CertificateException e) {
                                e.printStackTrace();
                        } catch (NotFoundException e) {
                                e.printStackTrace();
                        } catch (NoSuchProviderException e) {
                                e.printStackTrace();
                        } catch (OperatorCreationException e) {
                                e.printStackTrace();
                        } catch (IOException e) {
                        Toast.makeText(getApplicationContext(), "Se ha producido un error de
conexión. La aplicación volverá a Inicio", Toast.LENGTH_LONG).show();
                        Intent intent = new Intent(EnvioDatos.this, ElegirAccion.class);
                        intent.putExtra("DIR_IP", IP);
                        startActivity(intent);
                        EnvioDatos.this.finish();
                                e.printStackTrace();
                        } catch (ClassNotFoundException e) {
                                e.printStackTrace();
                        }
        //Muestra el código en el hilo principal aunque la conexión se haya hecho en otro hilo
                        runOnUiThread(new Runnable() {
                        public void run() {
                                CodigoUser =(TextView) findViewById(R.id.Codigo);
                                        CodigoUser.setText(codigoB);
                        }
                    });
            }
        }).start();
}
/**Metodo al que se llama despues de haber pulsado <code>aceptar</code> en el dialogo y
*  que permite mostrar un dialogo de progreso y la ejecucion de la conexion
*  con el servidor de la CA, mediante la llamada al metodo Hilo
*  @see #Hilo(ProgressDialog)
*  @see #mostrarDialogo()*/
public void aceptar() throws KeyManagementException, KeyStoreException, NoSuchAlgorithmException,
CertificateException, NotFoundException, NoSuchProviderException, OperatorCreationException,
IOException, ClassNotFoundException{
        progreso =ProgressDialog.show(EnvioDatos.this,"", "Esperando código...", true);
        progreso.setProgressStyle(ProgressDialog.STYLE_SPINNER);
        progreso.setCancelable(false);
        hilo(progreso);
}
/**Metodo al que se llama despues de haber pulsado la opcion de <code>cancelar</code> en el
dialogo.
* Envia al usuario a la actividad principal
* @see #mostrarDialogo()*/
public void cancelar() {
        Intent intent = new Intent(this, ElegirAccion.class);
        intent.putExtra("DIR_IP", IP);
```

```
        startActivity(intent);
        this.finish();
}
/**Metodo que permite realizar la conexion con el servidor de SMS
* @param codigoB es un string con el <code>código B</code> generado aleatoreamente en el servidor.
* @see #Obcert()*/
private void conexionServSMS(String codigoB) throws ClassNotFoundException{
        try {
        // Se instancia un objeto de tipo Socket al que se le pasa la IP y el puerto con el
        // que se va a conectar
        cliente = new Socket(IP, Puerto3);
        // Si la conexión se ha establecido con éxito se recibe el código B desde el servidor
        // de SMS
        if (cliente.isConnected())
        {
        // Si la conexión se ha establecido con éxito se recibe el código B desde el servidor
        // de SMS
                if (cliente.isConnected())
                {
                        ObjectInputStream codB=new ObjectInputStream(cliente.getInputStream());
                        String cod_B=(String) codB.readObject();
                // Si el código recibido por el servidor de SMS coincide con el código recibido
                // por el servidor de la CA, se envía un ACK al servidor de SMS. Este ACK es el
                        if(!cod_B.equals(null))
                        {
                                if(cod_B.equals(codigoB))
                                {
                                        ObjectOutputStream out=new
                                ObjectOutputStream(cliente.getOutputStream());
                                        out.writeObject("codB_OK");
                                        out.flush();
                                }
                                else
                                {
                                        Toast.makeText(getApplicationContext(), "Los códigos no son
iguales", Toast.LENGTH_LONG).show();
                                        Intent intent = new Intent(this, ElegirAccion.class);
                                        intent.putExtra("DIR_IP", IP);
                                        startActivity(intent);
                                        this.finish();
                                }
                                codB.close();
                        }
                        else
                        {
                                progreso.dismiss();
                                        Toast.makeText(getApplicationContext(), "No se ha recibido
el codigo B del servidor de SMS", Toast.LENGTH_LONG).show();
                                        Intent intent = new Intent(this, ElegirAccion.class);
                                        intent.putExtra("DIR_IP", IP);
                                        startActivity(intent);
                                        this.finish();
                        }
                }
                else
                        Toast.makeText(getApplicationContext(), "No se conecta con el servidor",
Toast.LENGTH_LONG).show();

                cliente.close();

        }

        } catch (UnknownHostException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
    }

/**Metodo que permite establecer una conexion segura con el servidor de la CA y recibir los
* codigos A y B del servidor de la CA.
* @see GestionDatos#enviarDatos(Socket, String, String)
* @see GestionDatos#recibirCodigos(Socket)*/
```

```
// Para poder utilizar un nivel de API superior para este método
@TargetApi(Build.VERSION_CODES.GINGERBREAD)

public void conexionServCA() throws KeyStoreException, NoSuchAlgorithmException,
CertificateException, NotFoundException, IOException, KeyManagementException,
ClassNotFoundException, NoSuchProviderException, OperatorCreationException{
        StrictMode.ThreadPolicy policy = new
StrictMode.ThreadPolicy.Builder().permitNetwork().build();
StrictMode.setThreadPolicy(policy);

    try{
        //Instanciación de la clase keystore
        KeyStore ts = KeyStore.getInstance("BKS");
        //cargar la trustore del cliente
        ts.load(getResources().openRawResource(R.raw.tscliente),ksPass);
        TrustManagerFactory tmf =
TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
        tmf.init(ts);
        //Creación del contexto que usa el TrustManager
        SSLContext sslcontext = SSLContext.getInstance("TLS");
        sslcontext.init(null, tmf.getTrustManagers(), null);
        //Creación del socket
        socket = sslcontext.getSocketFactory().createSocket(IP, Puerto);
        // Si el socket está conectado se llama a una serie de métodos
        if(socket.isConnected())
        {
        // Se crea un objeto de la clase GestionDatos que permite realizar las operaciones
necesarias.
                GestionDatos gd = new GestionDatos();
                gd.enviarDatos(socket, email, movil);
                String[] codigos = gd.recibirCodigos(socket);
                codigoB=codigos[1];
                codigoA=codigos[0];
                progreso.dismiss();
        }
} catch(UnknownHostException e) {
// Si se produce un error se vuelve a inicio
Toast.makeText(getApplicationContext(), "Se ha producido un error de conexión. La aplicación
volverá a Inicio", Toast.LENGTH_LONG).show();
                Intent intent = new Intent(this, ElegirAccion.class);
                intent.putExtra("DIR_IP", IP);
                startActivity(intent);
                this.finish();
        }
    }
/**Metodo para la utilización de un hilo en un determinado momento de la aplicacion
* el cual realiza la conexion con los servidores, en esta ocasión para obtener el certificado
* @param progreso variable para el dialogo de progreso
* @see #Obcert()
* @see #ObtenerCertificado(View)
* */
protected void hilo2(ProgressDialog progreso2)
{
        new Thread(new Runnable() {
                public void run() {
                        try {
                                obcert();
                        } catch (StreamCorruptedException e) {
                                e.printStackTrace();
                        } catch (NoSuchAlgorithmException e) {
                                e.printStackTrace();
                        } catch (NoSuchProviderException e) {
                                e.printStackTrace();
                        } catch (OperatorCreationException e) {
                                e.printStackTrace();
                        } catch (IOException e) {
                                Toast.makeText(getApplicationContext(), "Se ha producido un error
de conexión. La aplicación volverá a Inicio", Toast.LENGTH_LONG).show();
                                Intent intent = new Intent(EnvioDatos.this, ElegirAccion.class);
                                intent.putExtra("DIR_IP", IP);
                                startActivity(intent);
                                EnvioDatos.this.finish();
                                e.printStackTrace();
                        } catch (ClassNotFoundException e) {
```

```
                                e.printStackTrace();
                            }
                            runOnUiThread(new Runnable() {
                            public void run() {
// Si desde el servidor de la CA recibimos el String ErrCodigoA, se ha producido un error,
// que hace que la aplicación vaya a la actividad de inicio, informando al usuario
// de que se ha producido un error de conexión, ya que problablemente el código enviado desde
// el usuario no coincida con el que tiene el servidor por lo que no se le puede enviar su
// Certificado
                                if(GestionDatos.certificado.equals("ErrCodigoA"))
                                {
                                    Toast.makeText(getApplicationContext(), "Se ha producido un error
de conexión. La aplicación volverá a Inicio", Toast.LENGTH_LONG).show();
                                    Intent intent = new Intent(EnvioDatos.this, ElegirAccion.class);
                                    intent.putExtra("DIR_IP", IP);
                                    startActivity(intent);
                                    EnvioDatos.this.finish();
                                }
                                else
                                {
                                    Intent intent = new Intent(EnvioDatos.this,
GuardarCertificado.class);
                                    intent.putExtra("DIR_IP", IP);
                                    startActivity(intent);
                                    EnvioDatos.this.finish();
                                }
                            }
                        });
                }
        }).start();
}
/**Metodo que se encarga de realizar las operaciones necesarias para obtener el certificado del
* servidor de la CA, llamando a los metodos que se encargan de enviar el CSR y los datos del
* usuario
* @see GestionDatos#enviarCSR(Socket, String, String, String, String, String, String, String,
String)
* @see GestionDatos#recibirCertificado(Socket)
* @see #ObtenerCertificado(View)
* @see #conexionServSMS(String)
* @see #Hilo2(ProgressDialog)*/
public void obcert() throws StreamCorruptedException, IOException, ClassNotFoundException,
NoSuchAlgorithmException, NoSuchProviderException, OperatorCreationException{
        GestionDatos gd = new GestionDatos();
        if(!codigoB.equals(null))
        {
                conexionServSMS(codigoB);
                gd.enviarCSR(socket, codigoA, movil, nom, ape1, ape2, pais, ciu, email);
                gd.recibirCertificado(socket);
        }
        else
        {
                progreso.dismiss();
                Toast.makeText(getApplicationContext(), "No se ha recibido el codigo B",
Toast.LENGTH_LONG).show();
                Intent intent = new Intent(this, ElegirAccion.class);
                intent.putExtra("DIR_IP", IP);
                startActivity(intent);
                this.finish();
        }
        socket.close();
        progreso2.dismiss();
}
/**Metodo que muestra un dialogo de progreso y llama al metodo Hilo2 que se encarga de obtener el
certificado en otro hilo.
* Las operaciones se realizan cuando el usuario pulsa sobre el boton obtener certificado.
* Posteriormente se llama a la siguiente actividad
* @param view permite la vision del boton <code>Obtener certificado</code>
* @see #Obcert()*/
public void obtenerCertificado(View view) throws NoSuchAlgorithmException, NoSuchProviderException,
OperatorCreationException, IOException, ClassNotFoundException{
        progreso2 =ProgressDialog.show(EnvioDatos.this,"", "Esperando certificado...", true);
        progreso2.setProgressStyle(ProgressDialog.STYLE_SPINNER);
        progreso2.setCancelable(false);
```

```
        hilo2(progreso2);
}

/**Metodo que permite salir de la aplicacion si el usuario pulsa el boton de atras del movil.
* @param keyCode codigo del boton de atras del movil que es transparente al usuario
* @param event detecta la accion del usuario*/
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
  if (keyCode == KeyEvent.KEYCODE_BACK) {
          Intent intent = new Intent(this, DatosRegistro2.class);
          intent.putExtra("DIR_IP", IP);
          startActivity(intent);
          this.finish();
 // Si el listener devuelve true, significa que el evento esta procesado, y nadie debe hacer nada
mas
    return true;
  }
//para las demas cosas, se reenvia el evento al listener habitual
  return super.onKeyDown(keyCode, event);
}
/**Metodo que muestra al usuario un mensaje para que elija si desea salir o no*/
@SuppressLint("NewApi")
public void mensajeSalir()
{
         new AlertDialog.Builder(this)
      .setIcon(android.R.drawable.ic_dialog_alert)
      .setIconAttribute(android.R.attr.alertDialogIcon)
      .setTitle("Salir")
      .setMessage("¿Está seguro?")
      .setNegativeButton("Cancelar", null)//sin listener
      .setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {//un listener que al
pulsar, cierre la aplicacion
                @Override
        public void onClick(DialogInterface dialog, int which){
                EnvioDatos.this.finish();
        }
      })
      .show();
}
/**Realiza una accion u otra dependiendo del item elegido por el usuario*/
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_volver_inicio7:
        Intent intent = new Intent(this, ElegirAccion.class);
        intent.putExtra("DIR_IP", IP);
                startActivity(intent);
                this.finish();
            return true;
        case R.id.menu_salir_sinGuardar:
        mensajeSalir();
        return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
/**Muestra una serie de opciones en el menu al usuario*/
@Override
public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; Añade las opciones de menú
        getMenuInflater().inflate(R.menu.activity_envio_datos, menu);
        return super.onCreateOptionsMenu(menu);
}
}
```

GuardarCertificado.java

```
package com.id_digital_pfc;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
```

```
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.cert.CertificateException;
import java.security.spec.InvalidKeySpecException;
import java.util.ArrayList;
import android.os.Bundle;
import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.util.Log;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
import certCSR.CrearKeystore;


/**
 * Cuarta actividad dentro de la accion de registro. Permite
 * guardar el certificado obtenido tanto en un archivo .pem como en una keystore.
 * @author: Eva M. Blanco Delgado
 * @version: 18/01/2014_v0.0
*/
public class GuardarCertificado extends Activity {
//Declaración de variables
private EditText nombreArchivo;
/**String que recoge el nombre que el usuario le quiere dar al archivo en el que se va a
* guardar el certificado
* @see #GuardCer(View)*/
static String nomA;
String Pk;
final String PATH="data/data/com.id_digital_pfc/files/Certificados/";
int n;
String IP;
/**
* Metodo que permite el arranque inicial de la actividad.
* Recupera la <code>IP</code> desde la clase anterior
* @param savedInstanceState Mantiene el estado de la actividad.
*/
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_guardar_certificado);
        Bundle bundle=getIntent().getExtras();
        IP = bundle.getString("DIR_IP");
}
/**Este metodo se encarga de guardar el certificado en memoria interna
* @param nombreCert String que contiene el nombre que el usuario da al certificado
* @see #GuardCer(View)*/
private void guardarCert(String nombreCert) throws IOException
{
        String certificado = nombreCert+".pem";
        //Si no existe el fichero mencionado lo crea
        File f0 = new File(PATH);
        if(!f0.exists())
        {
                f0.mkdirs();
        }
        try
        {
                // creamos el archivo en el nuevo directorio creado
                File file = new File(f0, certificado);
                FileOutputStream fout = new FileOutputStream(file);
                // Convierte un stream de caracteres en un stream de
                // bytes
                OutputStreamWriter ows = new OutputStreamWriter(fout);
                ows.write(GestionDatos.certificado);
                ows.close();
```

```
                mostrarDialogo();
        }
        catch (Exception ex)
        {
           Log.e("Ficheros", "Error al escribir fichero");
           Toast.makeText(getApplicationContext(), "El certificado "+nombreCert+".pem no se ha
podido guardar correctamente.", Toast.LENGTH_SHORT).show();
           Intent intent = new Intent(this, ElegirAccion.class);
           intent.putExtra("DIR_IP", IP);
           startActivity(intent);
           this.finish();
        }
}
/**Este metodo muestra un dialogo para informar al usuario de que su certificado se ha guardado
* correctamente. Este tipo de dialogo obliga a la persona a leerlo.
* @see #aceptar2()*/
private void mostrarDialogo(){
        // String que se mostrará al usuario
        String mensajeDialogo="Su certificado se ha creado correctamente";
        //Creación del diálogo
        AlertDialog.Builder dialogo = new AlertDialog.Builder(this);
        dialogo.setMessage(mensajeDialogo);
        dialogo.setCancelable(false);
        dialogo.setNeutralButton("Aceptar", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialogo, int id) {
                        dialogo.dismiss();
                        //Cuando se pulsa el botón aceptar se accede
                        aceptar();
                }
        });
        //Se muestra el diálogo
        dialogo.show();
}
/**Metodo que vuelve a la actividad de inicio al pulsar sobre el boton de aceptar en el dialogo
* que se muestra el usuario
* @see #mostrarDialogo()*/
private void aceptar()
{
        Intent intent = new Intent(this, ElegirAccion.class);
        intent.putExtra("DIR_IP", IP);
        startActivity(intent);
        this.finish();
}
/**Metodo que comprueba que no existe ningun otro certificado con el nombre que se le quiere
* dar en este momento
* @param nombre String con el nombre que el usuario le quiere dar al certificado y que se comprueba
* en el directorio si existe otro archivo con ese nombre
* @see #GuardCer(View)*/
public int comprobarNombreCert(String nombre)
{
        int num=0;
        int i;
        String arch=nombre+".pem";
        ArrayList<String>lista=new ArrayList<String>();
        /* Para guardar en el array todos los archivos/directorios encontrados de la
            ruta indicada sin tener en cuenta el filtro.*/
        String[] archivos = new File(PATH).list();
        if(archivos!=null)
        {
                for(i=0; i<archivos.length; i++)
                {
                        lista.add(archivos[i]);
                }
                if(lista.contains(arch))
                        num=0;
                else
                        num=1;
        }
        else
                num=2;
   return num;
}
/**Metodo que permite obtener el nombre que el usuario da al certificado y guardar el certificado
* tanto en un archivo, como en la keystore del usuario.
```

```
* @param view permite la visión del botón
* @throws NoSuchProviderException
* @throws InvalidKeySpecException
* @throws CertificateException
* @throws NoSuchAlgorithmException
* @throws KeyStoreException
* @see #ComprobarNombreCert(String)
* @see #GuardarCert(String)
* @see CrearKeystore#GuardarKeystore(String, String)
* @see GestionDatos#certificado*/
public void guardCer(View view) throws IOException, KeyStoreException, NoSuchAlgorithmException,
NoSuchProviderException, CertificateException, InvalidKeySpecException{
        nombreArchivo = (EditText)findViewById(R.id.nomCert);
        nomA=nombreArchivo.getText().toString();
        if(nomA.equals(""))
        {
                Toast.makeText(getApplicationContext(), "No ha introducido ningún nombre de
archivo", Toast.LENGTH_LONG).show();
        }
        else
        {
                n=comprobarNombreCert(nomA);
                if(n==1 || n==2)
                {
                        //Se guarda el certificado en un directorio y en un keystore
                        guardarCert(nomA);
                        CrearKeystore.guardarKeystore(GestionDatos.certificado, nomA);
                }
                else
                        Toast.makeText(getApplicationContext(), "El nombre indicado ya existe. Por
favor, vuelva a introducir un nombre", Toast.LENGTH_LONG).show();
        }
}
/**Metodo que permite salir de la aplicacion si el usuario pulsa el boton de atras del movil.
* @param keyCode codigo del boton de atras del movil que es transparente al usuario
* @param event detecta la accion del usuario*/
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_BACK) {
                  Intent intent = new Intent(this, DatosRegistro2.class);
                  intent.putExtra("DIR_IP", IP);
                  startActivity(intent);
                  this.finish();
// Si el listener devuelve true, significa que el evento esta procesado, y nadie debe hacer nada
mas
            return true;
          }
        //para las demas cosas, se reenvia el evento al listener habitual
  return super.onKeyDown(keyCode, event);
}
/**Metodo que muestra al usuario un mensaje para que elija si desea salir o no*/
@SuppressLint("NewApi")
public void mensajeSalir()
{
         new AlertDialog.Builder(this)
      .setIcon(android.R.drawable.ic_dialog_alert)
      .setIconAttribute(android.R.attr.alertDialogIcon)
      .setTitle("Salir")
      .setMessage("¿Está seguro?")
      .setNegativeButton("Cancelar", null)//sin listener
      .setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {//un listener que al
pulsar, cierre la aplicacion
        @Override
        public void onClick(DialogInterface dialog, int which){
          //Salir
        GuardarCertificado.this.finish();
        }
      })
      .show();
}
/**Realiza una accion u otra dependiendo del item elegido por el usuario*/
@Override
public boolean onOptionsItemSelected(MenuItem item) {
```

```
    switch (item.getItemId()) {
        case R.id.menu_volver_inicio8:
        Intent intent = new Intent(this, ElegirAccion.class);
        intent.putExtra("DIR_IP", IP);
                startActivity(intent);
                this.finish();
            return true;
        case R.id.menu_salir_sinGuardar2:
        mensajeSalir();
        return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
/**Muestra una serie de opciones en el menu al usuario*/
@Override
public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; Añade opciones al menú.
        getMenuInflater().inflate(R.menu.activity_guardar_certificado, menu);
        return super.onCreateOptionsMenu(menu);
}
}
```

ElegirCertificado.java

```
package com.id_digital_pfc;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.util.StringTokenizer;
import android.os.Bundle;
import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.util.Log;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;
import android.widget.AdapterView.OnItemClickListener;
/**
 * Segunda actividad dentro de la accion de iniciar sesion.
 * Permite leer el certificado seleccionado por el usuario.
 * @author Eva M. Blanco Delgado
 * @version 18/01/2014_v0.0
*/
public class ElegirCertificado extends Activity {
//Definición de variables
/**String que contiene el path en el que se encuentran los certificados
* @see #CertificadoElegido(String)
* @see #listaCertificadosAut()
* @see GestionDatos#devolverListadoArchivos(String)*/
static final String PATH="/data/data/com.id_digital_pfc/files/Certificados/";
String[] listado;
String IP;
String alias;
/**
* Metodo que permite el arranque inicial de la actividad.
* Recupera la <code>IP</code> introducida por el usuario desde la actividad anterior.
* Tambien llama a los metodos mostrarDialogo3 y listaCertificadosAut
* @param savedInstanceState Mantiene el estado de la actividad.
*/
@Override
protected void onCreate(Bundle savedInstanceState) {
        int existe;
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_elegir_certificado);
        Bundle bundle=getIntent().getExtras();
        IP = bundle.getString("DIR_IP");
        existe=comprobarCertificados();
        if(existe==1)
        {
                mostrarDialogo();
                listaCertificadosAut();
        }
}
/**Metodo que comprueba si hay certificados almacenados. Si no los hay vuelve a ElegirAccion*/
 private int comprobarCertificados(){
         int n;
         File f = new File(PATH);
         if(!f.exists())
         {
                f.mkdirs();
         }
         final String[] listacerts=GestionDatos.devolverListadoArchivos(PATH);
         if(listacerts.length==0)
         {
                 Toast.makeText(getApplicationContext(), "Solicite un certificado. La lista está
vacia.", Toast.LENGTH_LONG).show();
                 Intent intent = new Intent(this, ElegirAccion.class);
                 intent.putExtra("DIR_IP", IP);
                 startActivity(intent);
                 this.finish();
                 n=0;
         }
         else
                 n=1;
 return n;
}
/**Metodo que usa un dialogo para avisar al usuario de que debe elegir un certificado, el cual se
va a eliminar*/
public void mostrarDialogo(){
        // String que se mostrará al usuario
        String mensajeDialogo="Por favor, seleccione el certificado con el que quiere entrar al
sistema.";
        //Creación del diálogo
        AlertDialog.Builder dialogo = new AlertDialog.Builder(this);
        dialogo.setMessage(mensajeDialogo);
        dialogo.setCancelable(false);
        dialogo.setNeutralButton("Aceptar", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialogo, int id) {
                        dialogo.dismiss();
                }
        });
        dialogo.show();
}
/**Metodo que obtiene el alias con el que se ha almacenado el certificado en el keystore a partir
* del nombre del certificado
* @param nombreCertificado String que contiene el nombre del certificado que el usuario ha
* seleccionado
* @return <code>nombre</code>: variable que contiene el nombre del alias con el que se ha guardado
* el certificado en el keystore
* @see #SiguienteActividad(String, String)*/
private String obtenerAlias(String nombreCertificado)
{
        String nombre=null;
        String s2=null;
        int numTokens=0;
        StringTokenizer st = new StringTokenizer(nombreCertificado);
        while(st.hasMoreTokens())
        {
                s2=st.nextToken(".");
                if(numTokens==0)
                {
                        nombre=s2;
                }
                numTokens++;
        }
```

```
        return nombre;
}
/**Este metodo simplemente llama a la siguiente actividad y le pasa el certificado
* en forma de String
* @param certificado un String con el certificado del usuario
* @param alias String que contiene el alias con el que se ha almacenado el certificado
* en el keystore
* @see #obtenerAlias(String)*/
private void SiguienteActividad(String certificado, String alias)
{
        Intent intent = new Intent(this, ValidacionUsuario.class);
        intent.putExtra("CERTIFICADO", certificado);
        intent.putExtra("ALIAS", alias);
        intent.putExtra("DIR_IP", IP);
        startActivity(intent);
        this.finish();
}
/**Metodo que lee el certificado que ha sido elegido por el usuario y lo mete en un String
* @param nombreFichero nombre del fichero donde se encuentra almacenado el certificado
* @see #listaCertificadosAut()*/
private void leerCertificado(String nombreFichero){
         String archivo=PATH+nombreFichero;
         File f = new File(archivo);
         if(f.exists())
         {
            try
            {
                FileInputStream fis = new FileInputStream(f);
                InputStreamReader isr =new InputStreamReader(fis);
                BufferedReader br=new BufferedReader(isr);
                String linea=br.readLine();
                String cert="";
                while(linea!=null)
                {
                        cert=cert+linea+"\n";
                        linea=br.readLine();
                }
                br.close();
                isr.close();
                fis.close();
                alias = obtenerAlias(nombreFichero);
                SiguienteActividad(cert, alias);
            }
            catch (Exception ex)
            {
                Log.e("Lectura fichero", "Error al leer fichero desde memoria interna");
            }
         }
         else
         {
                 Toast.makeText(getApplicationContext(), "El archivo seleccionado no existe.",
Toast.LENGTH_LONG).show();
                 Intent intent = new Intent(this, ElegirAccion.class);
                 intent.putExtra("DIR_IP", IP);
                 startActivity(intent);
                 this.finish();
         }
}
/**Metodo para mostrar al usuario una lista de los certificados que tiene almacenados,
* entre los que el usuario debera elegir el que desea eliminar
* @see #CertificadoElegido(String)*/
public void listaCertificadosAut(){
        ListView list;
        final String[] listacerts=GestionDatos.devolverListadoArchivos(PATH);
        if(listacerts.length==0)
        {
                Toast.makeText(getApplicationContext(), "Solicite un certificado. La lista está
vacia.", Toast.LENGTH_LONG).show();
                Intent intent = new Intent(this, ElegirAccion.class);
                intent.putExtra("DIR_IP", IP);
                startActivity(intent);
                this.finish();
        }
        list = (ListView)findViewById(R.id.listview2);
```

```
        ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, listacerts);
        list.setAdapter(adaptador);
        list.setOnItemClickListener(new OnItemClickListener(){
        @Override
        public void onItemClick(AdapterView<?> arg0, View arg1, int position, long id) {
              leerCertificado(listacerts[position]);
                }
        });
}
/**Metodo que permite salir de la aplicacion si el usuario pulsa el boton de atras del movil.
* @param keyCode codigo del boton de atras del movil que es transparente al usuario
* @param event detecta la accion del usuario*/
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
  if (keyCode == KeyEvent.KEYCODE_BACK) {
          Intent intent = new Intent(this, TipoAutenticacion.class);
          intent.putExtra("DIR_IP", IP);
          startActivity(intent);
          this.finish();
    // Si el listener devuelve true, significa que el evento esta procesado, y nadie debe hacer
nada mas
    return true;
  }
//para las demas cosas, se reenvia el evento al listener habitual
  return super.onKeyDown(keyCode, event);
}
/**Metodo que muestra al usuario un mensaje para que elija si desea salir o no*/
@SuppressLint("NewApi")
public void mensajeSalir()
{
        new AlertDialog.Builder(this)
        .setIcon(android.R.drawable.ic_dialog_alert)
        .setIconAttribute(android.R.attr.alertDialogIcon)
        .setTitle("Salir")
        .setMessage("¿Está seguro?")
        .setNegativeButton("Cancelar", null)//sin listener
        .setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {//un listener que al
pulsar, cierre la aplicacion
         @Override
         public void onClick(DialogInterface dialog, int which){
         //Salir
         ElegirCertificado.this.finish();
         }
     })
      .show();
}
/**Realiza una accion u otra dependiendo del item elegido por el usuario*/
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_entrar_contrasena:
                Intent intent = new Intent(this, TipoAutenticacion.class);
                intent.putExtra("DIR_IP", IP);
                startActivity(intent);
                this.finish();
            return true;
        case R.id.menu_volver_inicio4:
                Intent intent2 = new Intent(this, ElegirAccion.class);
                intent2.putExtra("DIR_IP", IP);
                startActivity(intent2);
                this.finish();
            return true;
        case R.id.menu_salir5:
                mensajeSalir();
            return true;
        default:
            return super.onOptionsItemSelected(item);
        }
}
/**Muestra una serie de opciones en el menu al usuario*/
@Override
public boolean onCreateOptionsMenu(Menu menu) {
```

```
        // Inflate the menu; Añade opciones al menú.
        getMenuInflater().inflate(R.menu.activity_elegir_certificado, menu);
        return super.onCreateOptionsMenu(menu);
}
}
```

TipoAutenticacion.java

```
package com.id_digital_pfc;

import android.os.Bundle;
import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.text.InputType;
import android.view.Gravity;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.RelativeLayout;
import android.widget.Toast;

/**
 * Primera actividad dentro de la accion de iniciar sesion.
 * Permite utilizar dos formas de autenticacion.
 * @author: Eva M. Blanco Delgado
 * @version: 18/01/2014_v0.0
*/
public class TipoAutenticacion extends Activity {
String IP;
String Usuario;
String password;
/**
* Metodo que permite el arranque inicial de la actividad.
* Recupera la <code>IP</code> desde la clase anterior
* @param savedInstanceState Mantiene el estado de la actividad.
*/
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tipo_autenticacion);
        Bundle bundle=getIntent().getExtras();
        IP = bundle.getString("DIR_IP");
}
/**Metodo que permite volver a la segunda actividad*/
private void VolverInicio()
{
        Intent intent = new Intent(this, ElegirAccion.class);
        intent.putExtra("DIR_IP", IP);
        startActivity(intent);
        this.finish();
}
/**Metodo que permite acceder a la siguiente actividad.
* @param view permite la vista del boton*/
public void conCertificado(View view)
{
         Intent intent = new Intent(this, ElegirCertificado.class);
         intent.putExtra("DIR_IP", IP);
         startActivity(intent);
         this.finish();
}
/**Metodo que permite al usuario introducir un nombre y una contraseña. Despues de que el
* usuario introduce el nombre y la contraseña, la aplicación vuelve a la segunda actividad.
* @param view permite la vista del boton
* @see #VolverInicio()*/
@SuppressLint("NewApi")
public void UsrYCont(View view)
{
```

```
         AlertDialog.Builder builder = new AlertDialog.Builder(this);
         builder.setIcon(android.R.drawable.ic_dialog_alert);
         builder.setIconAttribute(android.R.attr.alertDialogIcon);
         builder.setMessage("Introduzca su usuario y contraseña:");
         final EditText input1 = new EditText(this);
         input1.setInputType(InputType.TYPE_CLASS_TEXT);
         input1.setText("");
         final EditText input2 = new EditText(this);
         input2.setInputType(InputType.TYPE_CLASS_TEXT | nputType.TYPE_TEXT_VARIATION_PASSWORD);
         input2.setText("");
        //Definimos parametros para luego aplicar al Layout
         LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(
         RelativeLayout.LayoutParams.WRAP_CONTENT,
         RelativeLayout.LayoutParams.WRAP_CONTENT);
         //Creamos un Layout para agrupar la entiqueta(TexView) y el campo (EditText)
         // del Usuario
         LinearLayout layoutUser = new LinearLayout(this);
         // Aplicamos los parámetros del layout a nuestros objectos
         layoutUser.setOrientation(LinearLayout.VERTICAL);
         layoutUser.setGravity(Gravity.CENTER_HORIZONTAL);
         layoutUser.setPadding(10,5,10,5);
         layoutUser.setLayoutParams(params);
         layoutUser.addView(input1);
         layoutUser.addView(input2);
         builder.setView(layoutUser);
         builder.setNeutralButton("Aceptar", new DialogInterface.OnClickListener() {
         @Override
         public void onClick(DialogInterface dialog, int id) {
                 Usuario=input1.getText().toString();
                 password=input2.getText().toString();
                 Toast.makeText(getApplicationContext(), "USUARIO: "+Usuario,
Toast.LENGTH_LONG).show();
                 VolverInicio();
                 }
         });
        builder.show();
}
/**Metodo que permite salir de la aplicacion si el usuario pulsa el boton de atras del movil.
* @param keyCode codigo del boton de atras del movil que es transparente al usuario
* @param event detecta la accion del usuario*/
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
  if (keyCode == KeyEvent.KEYCODE_BACK) {
          Intent intent = new Intent(this, ElegirAccion.class);
          intent.putExtra("DIR_IP", IP);
          startActivity(intent);
          this.finish();
    // Si el listener devuelve true, significa que el evento esta procesado, y nadie debe hacer
nada mas
    return true;
  }
//para las demas cosas, se reenvia el evento al listener habitual
 return super.onKeyDown(keyCode, event);
}
/**Metodo que muestra al usuario un mensaje para que elija si desea salir o no*/
@SuppressLint("NewApi")
public void mensajeSalir()
{
         new AlertDialog.Builder(this)
      .setIcon(android.R.drawable.ic_dialog_alert)
      .setIconAttribute(android.R.attr.alertDialogIcon)
      .setTitle("Salir")
      .setMessage("¿Está seguro?")
      .setNegativeButton("Cancelar", null)//sin listener
      .setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {//un listener que al
pulsar, cierre la aplicacion
        @Override
        public void onClick(DialogInterface dialog, int which){
        //Salir
        TipoAutenticacion.this.finish();
        }
      })
      .show();
```

```
}
/**Realiza una accion u otra dependiendo del item elegido por el usuario*/
@Override
public boolean onOptionsItemSelected(MenuItem item) {
            switch (item.getItemId()) {
                case R.id.menu_volver_inicio3:
                        Intent intent = new Intent(this, ElegirAccion.class);
                        intent.putExtra("DIR_IP", IP);
                        startActivity(intent);
                        this.finish();
                    return true;
                case R.id.menu_salir4:
                        mensajeSalir();
                    return true;
                default:
                    return super.onOptionsItemSelected(item);
            }
}
/**Muestra una serie de opciones en el menu al usuario*/
@Override
public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.activity_tipo_autenticacion, menu);
        return super.onCreateOptionsMenu(menu);
}
}
```

ValidacionUsuario.java

```
package com.id_digital_pfc;

import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;
import java.security.InvalidKeyException;
import java.security.KeyManagementException;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.SignatureException;
import java.security.UnrecoverableKeyException;
import java.security.cert.CertificateException;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManagerFactory;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.os.StrictMode;
import android.annotation.SuppressLint;
import android.annotation.TargetApi;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.ProgressDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.res.Resources.NotFoundException;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

/**
 * Tercera actividad dentro de la accion de iniciar sesion.
 * Establece una conexion entre el usuario y el servidor que autentica al usuario.
 * @author Eva M. Blanco Delgado
 * @version 18/01/2014_v0.0
*/
public class ValidacionUsuario extends Activity {
//Declaración de variables
Socket socket;
```

```
final Handler mhandler = new Handler();
final Handler mhandler2 = new Handler();
private TextView mensaje;
static ProgressDialog progreso;
int Puerto=5000;
char ksPass[] = "109638".toCharArray();
/**String con el certificado que se va a enviar al servidor que hace la autenticación
* @see #EnviarCertificado(Socket)*/
static String certificado;
/**String en el que se almacena la respuesta del servidor
* @see #RecibirEntrada(Socket)*/
static String recibir;
String IP;
String alias;
final String PATH="/data/data/com.id_digital_pfc/files/keystores/ksUsuario.bks";
String comparar=null;
/**
* Metodo que permite el arranque inicial de la actividad.
* Recupera la <code>IP</code> introducida por el usuario desde la actividad anterior.
* Tambien llama al metodo Hilo() ejecuta en paralelo la conexion con el servidor.
* @param savedInstanceState Mantiene el estado de la actividad.
*/
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_validacion_usuario);
        //Se reciben los datos enviados en la actividad anterior
        Bundle bundle=getIntent().getExtras();
        certificado = bundle.getString("CERTIFICADO");
        IP = bundle.getString("DIR_IP");
        alias=bundle.getString("ALIAS");
        progreso =ProgressDialog.show(ValidacionUsuario.this,"", "Comprobando credenciales...",
true);
        progreso.setProgressStyle(ProgressDialog.STYLE_SPINNER);
        progreso.setCancelable(false);
        Hilo(progreso);
}
/**Metodo que establece la conexión en un hilo diferente al principal.
* @param progreso variable para establecer el dialogo de progreso
* @see #EstablecerConexion()*/
protected void Hilo(ProgressDialog progreso)
{
        new Thread(new Runnable() {
            public void run() {
                try {
                        comparar=EstablecerConexion();
                        runOnUiThread(new Runnable() {
                        public void run() {
                                if(comparar!=null)
                                {
                                        mensaje =(TextView)findViewById(R.id.msg);
                                        if(!comparar.equals("noAcceso"))
                                        {
                                                mensaje.setText(comparar);
                                        }
                                        else
                                        {
                                                mensaje.setText("");
                                                Toast.makeText(getApplicationContext(), "Acceso
denegado. Usted no tiene permiso para acceder al sistema", Toast.LENGTH_LONG).show();
                                                Intent intent = new Intent(ValidacionUsuario.this,
ElegirAccion.class);
                                                intent.putExtra("DIR_IP", IP);
                                                startActivity(intent);
                                                ValidacionUsuario.this.finish();
                                        }
                                }
                        }
                    });
                } catch (KeyManagementException e) {
                        e.printStackTrace();
                } catch (UnrecoverableKeyException e) {
                        e.printStackTrace();
```

```
                } catch (InvalidKeyException e) {
                        e.printStackTrace();
                } catch (KeyStoreException e) {
                        e.printStackTrace();
                } catch (NoSuchAlgorithmException e) {
                        e.printStackTrace();
                } catch (CertificateException e) {
                        e.printStackTrace();
                } catch (NotFoundException e) {
                        e.printStackTrace();
                } catch (NoSuchProviderException e) {
                        e.printStackTrace();
                } catch (SignatureException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                        runOnUiThread(new Runnable() {
                        public void run() {
                                mensaje =(TextView)findViewById(R.id.msg);
                                mensaje.setText("");
                                Toast.makeText(getApplicationContext(), "Acceso denegado. Usted no
tiene permiso para acceder al sistema", Toast.LENGTH_LONG).show();
                                Intent intent = new Intent(ValidacionUsuario.this,
ElegirAccion.class);
                                intent.putExtra("DIR_IP", IP);
                                startActivity(intent);
                                ValidacionUsuario.this.finish();
                                }
                            });
                } catch (ClassNotFoundException e) {
                        e.printStackTrace();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
        }).start();
}
/**Metodo que permite el establecimiento de una conexion segura con el servidor
* @throws SignatureException
* @throws InvalidKeyException
* @throws InterruptedException
* @see #comprobarAcceso()*/
@TargetApi(Build.VERSION_CODES.GINGERBREAD)
public String EstablecerConexion() throws KeyStoreException, NoSuchAlgorithmException,
CertificateException, NotFoundException, IOException, KeyManagementException,
UnrecoverableKeyException, NoSuchProviderException, ClassNotFoundException, InvalidKeyException,
SignatureException, InterruptedException{
StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitNetwork().build();
StrictMode.setThreadPolicy(policy);
        String acceder=null;
        try{
        //Instanciación de la clase keystore
                KeyStore ts = KeyStore.getInstance("BKS");
                //cargar la trustore del cliente
                ts.load(getResources().openRawResource(R.raw.tsclienteaut),ksPass);
                TrustManagerFactory tmf =
TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
                tmf.init(ts);
                //Creación del contexto que usa el TrustManager
                SSLContext sslcontext = SSLContext.getInstance("TLS");
                sslcontext.init(null, tmf.getTrustManagers(), null);
                //Creación del socket
                socket = sslcontext.getSocketFactory().createSocket(IP, Puerto);
                GestionDatos gd = new GestionDatos();
                // Si el socket está conectado se llama a una serie de métodos
                if(socket.isConnected())
                {
                        gd.enviarCertificadoAut(socket, alias, certificado);
                        Thread.sleep(1500);
                        acceder=gd.recibirEntradaAut(socket);
                        progreso.dismiss();
                        socket.close();
                }
        } catch(UnknownHostException e) {
```

```
        // Si se produce un error se vuelve a inicio
        Toast.makeText(getApplicationContext(), "Se ha producido un error de conexión. La
aplicación volverá a Inicio", Toast.LENGTH_LONG).show();
                Intent intent = new Intent(this, ElegirAccion.class);
                intent.putExtra("DIR_IP", IP);
                startActivity(intent);
                this.finish();
                acceder=null;
       }
        return acceder;
}
/**Metodo que permite volver a la primera actividad si el usuario pulsa sobre el boton Aceptar
* @param view permite ver el boton para volver al inicio*/
public void VolverInicio(View view)
{
        Intent intent = new Intent(this, ElegirAccion.class);
        intent.putExtra("DIR_IP", IP);
        startActivity(intent);
        this.finish();
}
/**Metodo que permite salir de la aplicacion si el usuario pulsa el boton de atras del movil.
* @param keyCode codigo del boton de atras del movil que es transparente al usuario
* @param event detecta la accion del usuario*/
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
  if (keyCode == KeyEvent.KEYCODE_BACK) {
        Intent intent = new Intent(this, ElegirCertificado.class);
        intent.putExtra("DIR_IP", IP);
        startActivity(intent);
        this.finish();
        // Si el listener devuelve true, significa que el evento esta procesado, y nadie debe hacer
nada mas
        return true;
    }
//para las demas cosas, se reenvia el evento al listener habitual
  return super.onKeyDown(keyCode, event);
}
/**Metodo que muestra al usuario un mensaje para que elija si desea salir o no*/
@SuppressLint("NewApi")
public void mensajeSalir()
{
         new AlertDialog.Builder(this)
      .setIcon(android.R.drawable.ic_dialog_alert)
      .setIconAttribute(android.R.attr.alertDialogIcon)
      .setTitle("Salir")
      .setMessage("¿Está seguro?")
      .setNegativeButton("Cancelar", null)//sin listener
      .setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {//un listener que al
pulsar, cierre la aplicacion
        @Override
        public void onClick(DialogInterface dialog, int which){
          //Salir
        ValidacionUsuario.this.finish();
        }
      })
      .show();
}
/**Realiza una accion u otra dependiendo del item elegido por el usuario*/
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_volver_inicio5:
                Intent intent = new Intent(this, ElegirAccion.class);
                intent.putExtra("DIR_IP", IP);
                startActivity(intent);
                this.finish();
            return true;
        case R.id.menu_salir6:
                mensajeSalir();
            return true;
        default:
            return super.onOptionsItemSelected(item);
        }
```

```
}
/**Muestra una serie de opciones en el menu al usuario*/
@Override
public boolean onCreateOptionsMenu(Menu menu) {
// Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.activity_validacion_usuario, menu);
        return super.onCreateOptionsMenu(menu);
}
}
```

EliminarCertificado.java

```
package com.id_digital_pfc;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.cert.CertificateException;
import java.util.StringTokenizer;
import android.os.Bundle;
import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;
import certCSR.EliminarKeystore;

/**
 * Actividad que permite eliminar un certificado.
 * @author Eva M. Blanco Delgado
 * @version 18/01/2014_v0.0
*/
public class EliminarCertificado extends Activity {
String[] listado;
String Alias=null;
String IP;
/**String que almacena el path en el que se van a guardar los certificados
 * @see #EliCert(String)*/
static final String PATH="/data/data/com.id_digital_pfc/files/Certificados/";
/**
* Metodo que permite el arranque inicial de la actividad.
* Recupera la <code>IP</code> introducida por el usuario desde la actividad anterior.
* Tambien llama a los metodos mostrarDialogo2 y listaCertificados
* @param savedInstanceState Mantiene el estado de la actividad.
* @see #mostrarDialogo2()
* @see #listaCertificados()
* @see #comprobarCertificados()
*/
@Override
protected void onCreate(Bundle savedInstanceState) {
        int existe;
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_eliminar_certificado);
        Bundle bundle=getIntent().getExtras();
        IP = bundle.getString("DIR_IP");
        existe = comprobarCertificados();
        if(existe==1)
        {
                mostrarDialogo();
```

```
                listaCertificados();
        }
}
/**Metodo que comprueba si hay certificados almacenados. Si no los hay vuelve a ElegirAccion*/
private int comprobarCertificados(){
         int n;
         File f = new File(PATH);
         if(!f.exists())
         {
                f.mkdirs();
         }
         final String[] listacerts=GestionDatos.devolverListadoArchivos(PATH);
         if(listacerts.length==0)
         {
                 Toast.makeText(getApplicationContext(), "Todavía no tiene archivos almacenados.
Tiene que solicitar un certificado", Toast.LENGTH_LONG).show();
                 Intent intent = new Intent(this, ElegirAccion.class);
                 intent.putExtra("DIR_IP", IP);
                 startActivity(intent);
                 this.finish();
                 n=0;
         }
         else
                 n=1;

 return n;
}
/**Metodo para avisar al usuario de que debe elegir un certificado, el cual se va a eliminar*/
public void mostrarDialogo(){
        // String que se mostrará al usuario
        String mensajeDialogo="Seleccione el certificado que desea eliminar";
        //Creación del diálogo
        AlertDialog.Builder dialogo = new AlertDialog.Builder(this);
        dialogo.setMessage(mensajeDialogo);
        dialogo.setCancelable(false);
        dialogo.setNeutralButton("Aceptar", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialogo, int id) {
                        dialogo.dismiss();
                }
        });
        dialogo.show();
}
/**Metodo que elimina el certificado seleccionado por el usuario y vuelve a la segunda actividad.
* @param nombreFichero String con el nombre del fichero que se desea eliminar
* @see #listaCertificados()*/
private void EliCert(String nombreFichero){
        File f = new File(PATH+nombreFichero);
        if(f.delete())
           Toast.makeText(getApplicationContext(), "El certificado \""+nombreFichero+"\" ha sido
borrado correctamente", Toast.LENGTH_SHORT).show();
        else
            Toast.makeText(getApplicationContext(), "El certificado "+nombreFichero+" no se puede
eliminar", Toast.LENGTH_SHORT).show();

        Intent intent = new Intent(this, ElegirAccion.class);
        intent.putExtra("DIR_IP", IP);
        startActivity(intent);
        this.finish();
}
/**Metodo que permite obtener el alias de un certificado a partir del
* nombre del archivo en el que esta el certificado. (Los certificados se guardan en el keystore
* con el nombre que el usuario le da al certificado como alias)
* @param nombre String con el nombre del archivo en el que se guarda el certificado.
* @see #listaCertificados()*/
public String NombreAlias(String nombre)
{
        String alias=null;
        String s2=null;
        int numTokens=0;
        StringTokenizer st = new StringTokenizer(nombre);
        while(st.hasMoreTokens())
        {
                s2=st.nextToken(".");
```

```
                if(numTokens==0)
                {
                        alias=s2;
                }
                numTokens++;
        }
        return alias;
}
/**Metodo para mostrar al usuario una lista de los certificados que tiene almacenados,
* entre los que el usuario deberá elegir el que desea eliminar
* @see GestionDatos#devolverListadoArchivos(String)
* @see EliminarKeystore#EliminarEntrada(String)
* @see #NombreAlias(String)
* @see #EliCert(String)*/
public void listaCertificados(){
    ListView list;
    final String[] listacerts=GestionDatos.devolverListadoArchivos(PATH);
    list = (ListView)findViewById(R.id.listview);
    ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, listacerts);
    list.setAdapter(adaptador);
    list.setOnItemClickListener(new OnItemClickListener(){
        @Override
        public void onItemClick(AdapterView<?> arg0, View arg1, int position, long id) {
            Alias=NombreAlias(listacerts[position]);
            EliminarKeystore ek = new EliminarKeystore();
            try {
                    ek.EliminarEntrada(Alias);
                } catch (KeyStoreException e) {
                    e.printStackTrace();
                } catch (NoSuchProviderException e) {
                    e.printStackTrace();
                } catch (NoSuchAlgorithmException e) {
                    e.printStackTrace();
                } catch (CertificateException e) {
                    e.printStackTrace();
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            EliCert(listacerts[position]);
        }
    });
}
/**Metodo que permite salir de la aplicacion si el usuario pulsa el boton de atras del movil.
* @param keyCode codigo del boton de atras del movil que es transparente al usuario
* @param event detecta la accion del usuario*/
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
  if (keyCode == KeyEvent.KEYCODE_BACK) {
          Intent intent = new Intent(this, ElegirAccion.class);
          intent.putExtra("DIR_IP", IP);
          startActivity(intent);
          this.finish();
// Si el listener devuelve true, significa que el evento esta procesado, y nadie debe hacer nada
mas
    return true;
   }
//para las demas cosas, se reenvia el evento al listener habitual
  return super.onKeyDown(keyCode, event);
}
/**Metodo que muestra al usuario un mensaje para que elija si desea salir o no*/
@SuppressLint("NewApi")
public void mensajeSalir()
{
         new AlertDialog.Builder(this)
      .setIcon(android.R.drawable.ic_dialog_alert)
      .setIconAttribute(android.R.attr.alertDialogIcon)
      .setTitle("Salir")
      .setMessage("¿Está seguro?")
      .setNegativeButton("Cancelar", null)//sin listener
      .setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {//un listener que al
pulsar, cierre la aplicacion
```

```
        @Override
        public void onClick(DialogInterface dialog, int which){
          //Salir
                EliminarCertificado.this.finish();
            }
        })
        .show();
}
/**Realiza una accion u otra dependiendo del item elegido por el usuario*/
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_volver_inicio6:
                Intent intent = new Intent(this, ElegirAccion.class);
                intent.putExtra("DIR_IP", IP);
                startActivity(intent);
                this.finish();
             return true;
        case R.id.menu_salir7:
                mensajeSalir();
             return true;
        default:
             return super.onOptionsItemSelected(item);
        }
}
/**Muestra una serie de opciones en el menu al usuario*/
@Override
public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; Añade las opciones al menú.
        getMenuInflater().inflate(R.menu.activity_eliminar_certificado, menu);
        return super.onCreateOptionsMenu(menu);
}
}
```

generarCSR.java

```
package certCSR;

import java.io.IOException;
import java.io.StringWriter;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.SecureRandom;
import java.security.Security;
import org.bouncycastle.asn1.DERPrintableString;
import org.bouncycastle.asn1.pkcs.PKCSObjectIdentifiers;
import org.bouncycastle.asn1.x500.X500Name;
import org.bouncycastle.asn1.x509.SubjectPublicKeyInfo;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.openssl.PEMWriter;
import org.bouncycastle.operator.ContentSigner;
import org.bouncycastle.operator.OperatorCreationException;
import org.bouncycastle.operator.jcajce.JcaContentSignerBuilder;
import org.bouncycastle.pkcs.PKCS10CertificationRequest;
import org.bouncycastle.pkcs.PKCS10CertificationRequestBuilder;

/**
 * Clase que permite generar un par de claves y un CSR.
 * @author Eva M. Blanco Delgado
 * @version 18/01/2014_v0.0
*/
public class generarCSR {
/**String que contiene el certificado con el CSR que se va a enviar al usuario
* @see #certUsuario(String, String, String, String, String, String, String)*/
static String certificado;
/**Variable que contiene un par de claves
 * @see #generarClaves(int)
 * @see #CertificateSigningRequest(String, String, String, String, String, String, String)*/
```

```
static KeyPair kp;
/**Añade el proveedor BouncyCastleProvider*/
        static {
       // Security.addProvider(new org.spongycastle.jce.provider.BouncyCastleProvider());
         Security.addProvider(new BouncyCastleProvider());
         }

/**Metodo que genera un par de claves: una publica y una privada
* @param size tamaño en bits que se le da a las claves
* @return <code>kp:</code> variable que contiene el par de claves
* @see #CertificateSigningRequest(String, String, String, String, String, String, String) */
private static KeyPair generarClaves(int size) throws NoSuchAlgorithmException,
NoSuchProviderException
{
        //Generación de claves con un tamaño que depende de la variable size
        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA", "BC");
        keyPairGenerator.initialize(size, new SecureRandom());
        KeyPair kp= keyPairGenerator.generateKeyPair();
        return kp;
}
/**Metodo que genera un Certificate Signing Request (CSR)
* @param password password que se usa para la generacion del CSR (Se usa como password el nº de
telefono)
* @param Nombre nombre del usuario
* @param Apellido1 primer apellido del usuario
* @param Apellido2 segundo apellido del usuario
* @param Ciudad ciudad en la que reside el usuario
* @param Pais país en el que reside el usuario
* @param Email email del usuario
* @return <code>csr</code>: un objeto de tipo PKCS10CertificationRequest con el CSR del usuario
* @see #certUsuario(String, String, String, String, String, String, String)
* @see #generarClaves(int)*/
public static PKCS10CertificationRequest certificateSigningRequest (String password, String Nombre,
String Apellido1, String Apellido2, String Pais, String Ciudad, String Email) throws
OperatorCreationException, NoSuchAlgorithmException, NoSuchProviderException
{
        kp = generarClaves(2048);
        //Requested Certificate Name and things
        X500Name name = new X500Name("CN="+Nombre+" "+Apellido1+" "+Apellido2+", C="+Pais+",
L="+Ciudad+", EmailAddress="+Email);
        SubjectPublicKeyInfo publicKeyInfo =
SubjectPublicKeyInfo.getInstance(kp.getPublic().getEncoded());
        PKCS10CertificationRequestBuilder csrBuilder = new PKCS10CertificationRequestBuilder(name,
publicKeyInfo);

        JcaContentSignerBuilder signerBuilder= new JcaContentSignerBuilder("SHA1withRSA");
        csrBuilder.addAttribute(PKCSObjectIdentifiers.pkcs_9_at_challengePassword, new
DERPrintableString(new String(password.toCharArray())));
        ContentSigner signer = signerBuilder.build(kp.getPrivate());
        PKCS10CertificationRequest csr = csrBuilder.build(signer);
        return csr;
}
/**Metodo que convierte el csr creado en un String que se devuelve a la actividad que ha solicitado
la creacion del CSR
* @param password password que se usa para la generacion del CSR (Se usa como password el nº de
telefono)
* @param Nombre nombre del usuario
* @param Apellido1 primer apellido del usuario
* @param Apellido2 segundo apellido del usuario
* @param Ciudad ciudad en la que reside el usuario
* @param Pais país en el que reside el usuario
* @param Email email del usuario
* @return <code>certificado</code>: Un string que recoge el CSR en formato PEM
* @see #CertificateSigningRequest(String, String, String, String, String, String, String)*/
public static String certUsuario (String password, String Nombre, String Apellido1, String
Apellido2, String Pais, String Ciudad, String Email) throws NoSuchAlgorithmException,
NoSuchProviderException, OperatorCreationException, IOException
{
        PKCS10CertificationRequest CSR = certificateSigningRequest(password, Nombre, Apellido1,
Apellido2, Pais, Ciudad, Email);
     /* String que habría que enviar al servidor
     */
        StringWriter sw =new StringWriter();
        PEMWriter pem = new PEMWriter(sw);
```

```
        pem.writeObject(CSR);
        pem.close();
        //Convierte el archivo PEM en un String que se le enviará al servidor
        certificado = sw.toString();
        return certificado;
        }
}
```

CrearKeystore.java

```
package certCSR;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.security.KeyPair;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.PrivateKey;
import java.security.Security;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.security.spec.InvalidKeySpecException;
import org.bouncycastle.jce.provider.BouncyCastleProvider;

/**
 * Clase que permite la creacion de una Keystore.
 * @author Eva M. Blanco Delgado
 * @version 18/01/2014_v0.0
*/
public class CrearKeystore {
//Inicialización de variables
/**String que almacena el path o ruta en el que se almacena el keystore del usuario dentro del
dispositivo.
* @see #GenerarKeystore(KeyPair, String, String)
* @see #GuardarKeystore(String, String)*/
final static String PATH="data/data/com.id_digital_pfc/files/keystores/";
/**String que almacena el path o ruta en el que se almacenan los certificados dentro del
dispositivo.
* @see #obtenerCertificado(String)*/
final static String PATH2="data/data/com.id_digital_pfc/files/Certificados/";
/**Metodo que almacena el archivo del certificado dentro de una variable que sea instanciable en
* una keystore
* @return <code>cert:</code> un objeto de tipo x509Certificate que contiene el certificado del
usuario
* @see #GenerarKeystore(KeyPair, String, String)*/
public static X509Certificate obtenerCertificado(String NomCert) throws CertificateException,
IOException, NoSuchProviderException
{
        Security.addProvider(new BouncyCastleProvider());
        InputStream inStream = null;
        try{
                inStream = new FileInputStream(PATH2+NomCert+".pem");
                CertificateFactory cf = CertificateFactory.getInstance("X.509", "BC");
                X509Certificate cert = (X509Certificate)cf.generateCertificate(inStream);
                return cert;
        }finally{
                if(inStream!=null)
                {
                        inStream.close();
                }
        }
}
/**Metodo que crea un archivo para la keystore si no existe y almacena en el la clave privada
* y el certificado tanto si existe el archivo como si no
* @see #obtenerCertificado(String)
```

```
* @see #GuardarKeystore(String, String)*/
public static void generarKeystore(KeyPair kp, String CerUsr, String NomCert) throws
KeyStoreException, NoSuchAlgorithmException, CertificateException, IOException,
NoSuchProviderException
{
        //Se almacena la clave privada en la variable priv
        PrivateKey priv = kp.getPrivate();
        //Se obtiene el certificado firmado a partir del archivo en el que se almacena
        //llamando al método obtenerCertificado
        X509Certificate certificado = obtenerCertificado(NomCert);
        //Se instancia el keystore de tipo BKS
        KeyStore ks = KeyStore.getInstance("BKS", "BC");
        String password = "109638";
        char[] ksPass = password.toCharArray();
        File f = new File(PATH+"ksUsuario.bks");
        if(!f.exists())
        {
                //Carga un keystore nuevo si no existe uno previamente
                ks.load(null, ksPass);
        }
        else
                //Si existe previamente una keystore, se carga
                ks.load(new FileInputStream(PATH+"ksUsuario.bks"), ksPass);
                Certificate[] certChain = new Certificate[1];
                certChain[0]=certificado;
                //Se crea una entrada en el keystore
                ks.setKeyEntry(NomCert, priv, ksPass, certChain); //alias, clave privada,
contraseña y certificado
                //El alias que le he puesto a la entrada es el nombre del certificado
                OutputStream writeStream = new FileOutputStream(PATH+"ksUsuario.bks");
                ks.store(writeStream, ksPass);
                writeStream.close();
}
/**Metodo que comprueba si existe un directorio para las keystores, si no existe lo crea y llama
* al metodo que se encarga de guardar la entrada en el keystore.
* @see #GenerarKeystore(KeyPair, String, String)*/
public static void guardarKeystore(String certificado, String nombre) throws KeyStoreException,
NoSuchAlgorithmException, CertificateException, IOException, InvalidKeySpecException,
NoSuchProviderException{
        File f = new File(PATH);
        if(!f.exists())
        {
                f.mkdir();
        }
        generarKeystore(generarCSR.kp, certificado, nombre);
}
}
```

EliminarKeystore.java

```
package certCSR;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.cert.CertificateException;
/**
 * Clase que permite eliminar una entrada dentro del Keystore Keystore.
 * @author Eva M. Blanco Delgado
 * @version 18/01/2014_v0.0
*/
public class EliminarKeystore {
//Inicialización de variables
/**String en el que se almacena el path o ruta en el que se guardan los keystore dentro del movil.
         * @see #EliminarEntrada(String)*/
        final static String PATH="data/data/com.id_digital_pfc/files/keystores/";
/**Metodo para eliminar una entrada de una keystore a partir del alias con el que esta guardado*/
```

```
public void EliminarEntrada(String alias) throws KeyStoreException, NoSuchProviderException,
NoSuchAlgorithmException, CertificateException, FileNotFoundException, IOException
{
        KeyStore ks = KeyStore.getInstance("BKS", "BC");
        String password = "109638";
        char[] ksPass = password.toCharArray();
        ks.load(new FileInputStream(PATH+"ksUsuario.bks"), ksPass);
        ks.deleteEntry(alias);
        OutputStream writeStream = new FileOutputStream(PATH+"ksUsuario.bks");
        ks.store(writeStream, ksPass);
        writeStream.close();
}
}
```

Anexo G

Presupuesto

Anexo G

PRESUPUESTO

1. **Ejecución Material**
   - Compra de ordenadores personales (software incluido).....2000 €.
   - Dispositivo móvil LG Optimus L5.....130 €.
   - Material de oficina.....150 €.
   - Total de ejecución material.....2280 €.

2. **Gastos generales**
   - 21 % sobre Ejecución Material.....478,80 €.

3. **Beneficio Industrial**
   - 6 % sobre Ejecución Material.....136,80 €.

4. **Honorarios Proyecto**
   - 1500 horas a 15 € / horas.....22500 €.

5. **Material Fungible**
   - Gastos de impresión.....200 €.
   - Encuadernación.....50 €.

6. **Subtotal del presupuesto**
   - Subtotal Presupuesto.....25645,60 €.

7. **I.V.A. aplicable**
   - 21 % Subtotal Presupuesto.....5385,60 €.

8. **Total presupuesto**
   - Total Presupuesto.....**31031,20 €.**

Madrid, Mayo de 2014

El Ingeniero Jefe de Proyecto

Fdo.: Eva Milagros Blanco Delgado

Ingeniero de Telecomunicación

Anexo H

Pliego de condiciones

Anexo H

PLIEGO DE CONDICIONES

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de un *diseño y desarrollo de una aplicación Android para el uso de identidades digitales, autenticación y firmas digitales*. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4% del presupuesto y la provisional del 2%.

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma, por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no

sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad "Presupuesto de Ejecución de Contrata" y anteriormente llamado "Presupuesto de Ejecución Material" que hoy designa otro concepto.

Condiciones particulares

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.