Universidad de las Ciencias Informáticas

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Título:
API para el trabajo con llamadas de datos en el Sistema Operativo Android.

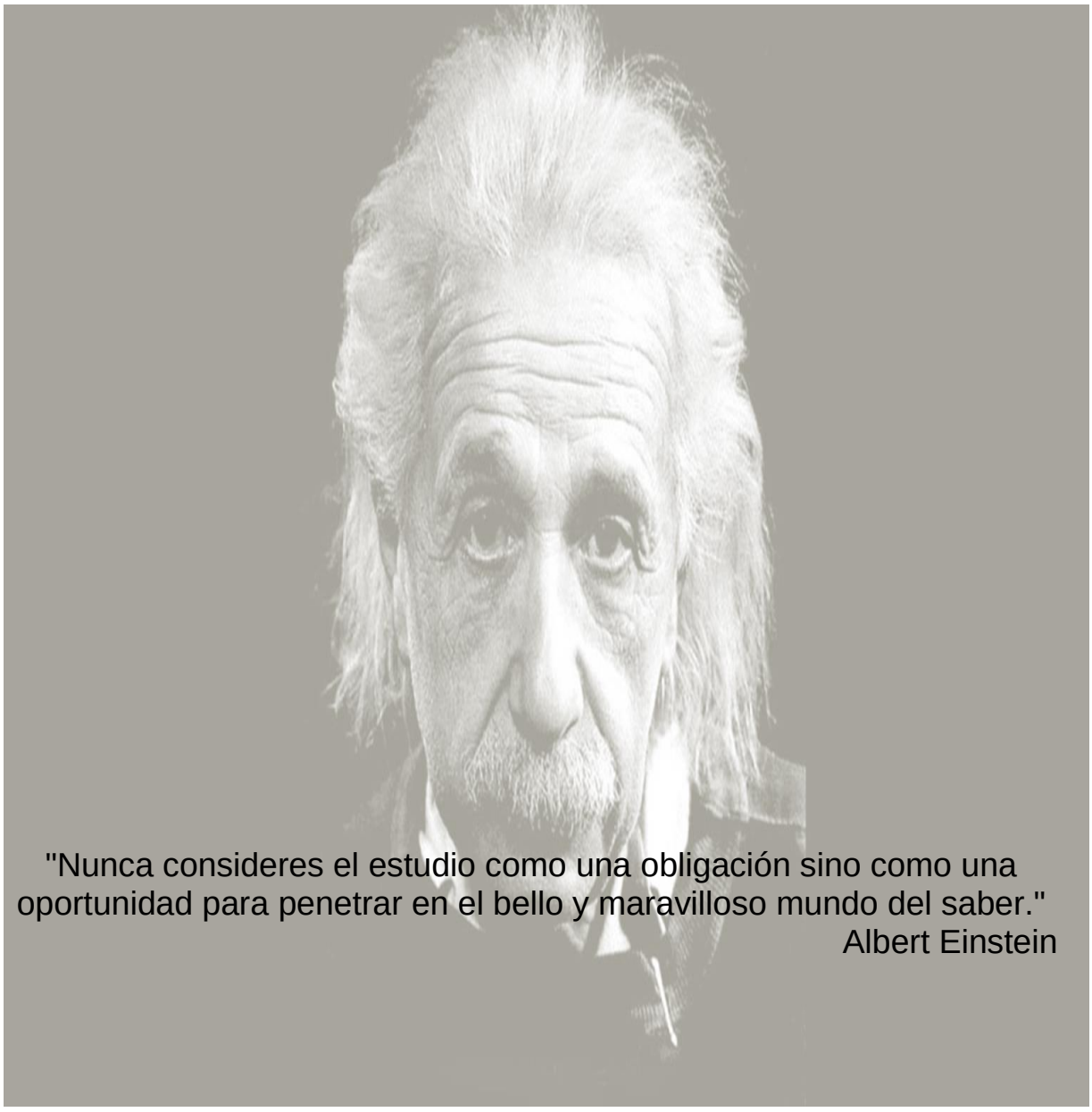
Autores: Roberto Gutiérrez Peña

Ariel Álvarez García

Tutores: MSc. Allan Pierra Fuentes

Ing. Yaiselis Ramírez Mastrapa

La Habana, 2015



"Nunca consideres el estudio como una obligación sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber."

Albert Einstein

Declaración de Autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

Roberto Gutiérrez Peña	Ariel Álvarez García
_____	_____
Firma del Autor	Firma del Autor
MSc. Allan Pierra Fuentes	Ing. Yaiselis Ramírez Mastrapa
_____	_____
Firma del Tutor	Firma del Tutor

Datos de Contacto

Tutor:

MSc. Allan Pierra Fuentes, graduado en Ingeniería en Ciencias Informáticas, MSc. en Informática Aplicada, Jefe del Centro de Soporte, con más de 10 años de experiencia en Software Libre y Código Abierto, Profesor de Programación y cursos de Android en Postgrado.

Tutora:

Ing. Yaiselis Ramírez Mastrapa, graduada en Ingeniería en Ciencias Informáticas.

Agradecimientos

Roberto:

Agradezco a todas las personas que hicieron posible la realización de este trabajo. A toda mi familia, a mi madre Graciela, por todo su apoyo, a mis hermanos Yober y Alberto, a mis primos Yadir, Adita, Yosmani, Ailín, Gaito, a mi tía Helen. Agradezco a Biky, a María y a Isidro por todo su apoyo en estos 5 años.

Además, agradezco a mis amigos Yoel, Arianna, Eblis, Lester, Daldis, que durante los 5 años estuvieron a mi lado.

Doy gracias a mis tutores Yaiselis, por su ayuda incondicional durante la realización de este trabajo, y Allan, por su apoyo.

Ariel:

Agradezco a mi familia, a mi madre Betty, mi tía Ile, mi prima Anaely y principalmente a mi hermano Armando, que en los momentos más difíciles siempre supo apoyarme a pesar de que siempre me criticaba como todo un padre, y quiero que sepa que para mí lo es.

También quisiera agradecer a mi tutora Yaiselis, por su dedicación y Allan por su apoyo. A Diwel, Jorge, Ernesto, Dayrelis, Alexis, Carlos, Osvaldo que me apoyaron mucho en esta última etapa. Y a todos aquellos que de una forma u otra se preocuparon por apoyarme aunque fuera regañándome. También quisiera agradecerle a mi novia Sheyla que en los momentos finales supo darme su apoyo.

Dedicatoria

Roberto:

A mi madre, por haberme dedicado su vida, por darme siempre su apoyo en todos los momentos que lo necesité y por confiar siempre en mí.

A mi padre, que aunque hoy no está a mi lado, hubiese estado orgulloso de ver mi sueño hecho realidad.

Ariel:

Dedico este trabajo al esfuerzo de mi familia, por lograr que yo me superara y tuviera siempre las condiciones necesarias y posibles para realizar mis estudios. A mi hermano Armando. Y a esa persona que nunca pude llegar a conocer, mi padre.

Me hubiera gustado haber hecho mucho más.

Resumen

Las vulnerabilidades que poseen las redes de telecomunicaciones inciden en la confidencialidad que se pueda tener en el momento de establecer una llamada telefónica siendo posible la intercepción no autorizada de la misma. Por lo que fue necesario realizar un estudio de los componentes relacionados con la llamada telefónica, con el objetivo fundamental de elaborar una solución informática capaz de mejorar la seguridad de la información en el proceso de realización de llamadas telefónicas en el Sistema Operativo Android. Para lograr el correcto funcionamiento de la solución fue necesario modificar algunos componentes del Sistema Operativo y adicionar nuevos módulos. Estos módulos se encargan de realizar las llamadas haciendo uso de la transmisión de Datos por Conmutación de Circuitos y de cifrar el flujo de audio durante dicha llamada. Lográndose establecer una llamada cifrada con alto nivel de seguridad. El sistema está implementado sobre la plataforma Android, en el lenguaje Java y C, a través de la herramienta Eclipse. Además, se tuvieron en cuenta todos los pasos y métodos que ofrece la metodología OpenUP. El sistema desarrollado fue evaluado mediante pruebas de Caja Blanca y Caja Negra, arrojando resultados satisfactorios.

Palabras claves: redes de telecomunicaciones, confidencialidad, seguridad, llamadas telefónicas, Sistema Operativo Android.

Abstract

Telecommunication networks have vulnerabilities that affect negatively the confidentiality at the moment of establishing a phone call since an unauthorized interception is possible. So it was necessary to conduct a study of the components related to the phone call, with the primary objective of developing a software solution capable of improving the security of information in the process of making phone calls in the Android Operating System. In order to achieve the correct operation of this solution it was necessary to modify some of the components of the Operating System and add new modules. These modules are responsible for placing calls using Circuit Switched Data transmission and encrypt the audio stream during the call, improving the security of the call through encryption. The system is implemented on the Android platform, using Java and C with the Eclipse tool. In addition, all steps and methods provided by the OpenUP methodology where taken into consideration. The developed system was tested with White Box and Black Box, obtaining satisfactory results.

Keywords: telecommunications networks, confidentiality, security, telephone calls, Android Operating System.

Índice

Introducción

En el mundo actual, contar con información de forma rápida y precisa es vital para la toma de decisiones, especialmente en altos niveles de gestión. En este aspecto, las tecnologías inalámbricas marcan un desarrollo en los últimos años. La telefonía celular es una de las tecnologías inalámbricas que mayor auge ha tenido. Desde sus inicios, a finales de los 70 ha revolucionado enormemente las actividades que se realizan diariamente. Los teléfonos celulares se han convertido en una herramienta primordial para las personas comunes y de negocios, les proporciona seguridad e incrementa su productividad. El creciente uso de los teléfonos móviles y su aplicación en casi todas las facetas de la comunicación personal y corporativa, los convierten en una herramienta necesaria para el intercambio de información confidencial (Martínez, 2001).

En la telefonía móvil la confidencialidad no es uno de los puntos más fuertes, debido a que los mecanismos y niveles de seguridad dependen directamente de la red sobre la cual se opera. Gran parte de las comunicaciones se realizan sobre las redes del Sistema Global de Comunicaciones Móviles (GSM, *Global System for Mobile Communications*), siendo el estándar más extendido en el mundo, alcanzando cifras de 1.6 mil millones de los teléfonos móviles que utilizan estas redes a nivel mundial (Gsma, 2015). Redes tales como GSM carecen de mecanismos que les posibiliten a los usuarios de teléfonos móviles, realizar llamadas seguras sin que corran el peligro de sufrir espionaje o intervenciones que permitan acceder al contenido de la información y comprometan la seguridad. Esto se debe a que solo existe cifrado de la información entre el teléfono móvil y la Estación Base[1], pero entre las mismas estaciones no existe ningún mecanismo de protección implementado, estos quedan en manos de los proveedores. La mayoría de los países no implementan estos servicios de seguridad, incluyendo Cuba (Ronquillo, 2006).

Debido a las vulnerabilidades que presentan las redes GSM, terceras personas pueden suplantar la Estación Base y escuchar la información que por ella circula, dando paso a los espionajes telefónicos (Cubasi, 2014). Generalmente estos espionajes son empleados como herramientas de ataque político y empresarial al infiltrarse en conversaciones entre personajes públicos o autoridades y divulgarlas. Un ejemplo concreto es el caso de la presidenta de Brasil, Dilma

[1] Estación Base: estación de transmisión y recepción situada en un lugar fijo, compuesta de una o más antenas de recepción/transmisión, y utilizada para manejar el tráfico telefónico.

Rousseff, la cual fue objeto de espionaje por parte de la Agencia de Seguridad Nacional (NSA, *National Security Agency*) de Estados Unidos. Fueron escudriñados los contenidos de llamadas telefónicas, correos electrónicos y mensajes de texto (Arias, 2013). Otra evidencia es el espionaje al candidato presidencial Peña Nieto, en plena campaña por la presidencia de México, con el objetivo de revisar sus datos personales (Eleconomista, 2014).

La evolución de las tecnologías y las nuevas ventajas de los *smartphones*[2], como sus capacidades de procesamiento, brindan un camino para solucionar estas vulnerabilidades en las redes GSM.

Entre los Sistemas Operativos para *smartphones* se encuentra Android. El cual fue creado por Google, y se ha convertido en el Sistema Operativo (SO) móvil más utilizado a nivel mundial. La cifra de dispositivos activados diariamente, que usan este SO, asciende a más 1.5 millones. Una de las razones que potencian su popularidad son su naturaleza de código abierto (*open source*) y que posee una curvatura de aprendizaje relativamente baja (Warren, 2014). Actualmente Android carece de mecanismos fijos, que le posibiliten superponerse a las limitaciones de seguridad de las redes sobre las que operan los terminales.

Algunas soluciones alternativas para realizar llamadas seguras, incluyen la creación e instalación de aplicaciones en el SO. Tal es el caso desarrollado por la empresa *Whisper Systems*[3] que pretende brindar una solución mediante un software con la aplicación RedPhone, la cual tiene como objetivo establecer una comunicación VoIP[4] cifrada. Pero presentando la limitante, de no funcionar en redes GSM que no estén conectadas a Internet, por lo que un grupo de estas redes están imposibilitados a emplear esta aplicación (Eddy, 2014).

Debido a la situación expuesta anteriormente se plantea el siguiente **problema a resolver**: ¿Cómo mejorar la seguridad de la información a transmitir en el proceso de realización de llamadas telefónicas entre *smartphones* con SO Android?

[2] *Smartphones*: término que se utiliza para denominar teléfonos inteligentes. Se trata de un teléfono celular que ofrece prestaciones similares a las que brinda una computadora, y presentando como característica el uso de un SO.

[3] *Whisper Systems*: empresa fundada por el investigador de seguridad Moxie Moulinsart y el investigador de robótica Stuart Anderson en el año 2010 y posteriormente adquirida por Twitter.

[4] VoIP: Voz sobre Protocolo de Internet del inglés *Voice over IP*.

Luego, se identifica como **objeto de estudio**: la seguridad en las llamadas telefónicas sobre las redes GSM y como **campo de acción** el proceso de realización de llamadas telefónicas entre *smartphones* con SO Android.

Para darle solución al problema planteado anteriormente se propone como **objetivo general** de la investigación desarrollar una solución informática que sea capaz de mejorar la seguridad de la información a transmitir en el proceso de realización de llamadas telefónicas entre *smartphones* con SO Android.

Para dar cumplimiento al objetivo planteado se han definido los siguientes **objetivos específicos**:

- Sistematizar los fundamentos teóricos del proceso de llamadas telefónicas entre *smartphones* con SO Android.
- Analizar el mecanismo de realización de llamadas telefónicas entre *smartphones* con SO Android.
- Diseñar e implementar un sistema para el cifrado de audio en el proceso de realización de llamadas telefónicas entre *smartphones* con SO Android.
- Implementar y realizar pruebas funcionales al sistema elaborado para comprobar su correcto funcionamiento.

Para darle cumplimiento a los objetivos específicos planteados se realizan las siguientes **tareas de investigación**:

- Revisión bibliográfica para generar el marco teórico conceptual en lo referente a las comunicaciones sobre las redes GSM.
- Estudio de soluciones similares para conocer aspectos referentes a la seguridad en las llamadas telefónicas.
- Estudio de la arquitectura del SO Android y los mecanismos que utiliza en la comunicación a través de las redes telefónicas para comprender su funcionamiento.
- Estudio del protocolo de transmisión de Datos por Conmutación de Circuitos para su posterior uso.
- Caracterización de la metodología y herramientas para guiar el desarrollo del sistema.
- Estudio del algoritmo de cifrado AES para su integración en el proceso de llamada telefónica.

Se plantea como **idea a defender:** si se desarrolla una solución informática que sea capaz de mejorar la seguridad de la información a transmitir en el proceso de realización de llamadas telefónicas entre *smartphones* con SO Android se reduce la posibilidad de escuchar la información de las llamadas interceptadas.

Diseño metodológico de la investigación:

Métodos Teóricos:

- **Histórico-Lógico**: se utiliza para indagar sobre el devenir histórico de las llamadas seguras y el proceso de realización de llamadas telefónicas en el SO Android, los principales conceptos que giran alrededor de la misma y los sistemas que ya existen que implementan algún mecanismo de seguridad.
- **Analítico-Sintético**: permite realizar el estudio teórico de la investigación facilitando el análisis de documentos y la extracción de los elementos más importantes acerca de la seguridad en las llamadas telefónicas y del proceso de realización de las llamadas telefónicas en el SO Android.
- **Modelación:** permite reflejar la estructura, relaciones y características de la solución a través de diagramas, facilitando también el diseño y la comprensión de las clases necesarias para la implementación del sistema.

Métodos Empíricos:

- **Observación**: permite la obtención de conocimiento e información acerca del comportamiento de la seguridad presente en las llamadas telefónicas utilizando los *smartphones* con SO Android.

El presente trabajo de diploma se ha estructurado de la siguiente manera:

Capítulo 1: Fundamentación Teórica: en este capítulo se exponen los principales conceptos que contribuyen al mejor entendimiento del problema en cuestión. Se especifican detalladamente todos los argumentos que esclarecen el objeto de estudio, a partir del análisis de soluciones similares, enfocadas en aumentar la seguridad en las llamadas telefónicas utilizando el SO Android. Además, se realiza la caracterización de la metodología, herramientas y tecnologías que serán utilizadas en el desarrollo de la solución.

Capítulo 2: Análisis y Diseño: contiene un análisis profundo del proceso que lleva a cabo el SO Android para realizar llamadas telefónicas, detallándose como se realizará el diseño del sistema a desarrollar. Se elaboraran los diagramas pertinentes para un mejor entendimiento del sistema.

Capítulo 3: Implementación y Prueba: contiene las principales características de la implementación, representando el diagrama de componentes. Se documentan y ejecutan las pruebas de software que servirán para comprobar el correcto funcionamiento de la solución.

Capítulo 1. Fundamentación Teórica

1.1 Introducción

En este capítulo se analizan soluciones similares vinculadas al campo de acción. Se caracterizan las redes GSM y la arquitectura del SO Android. Se hace un análisis para definir las tecnologías, metodologías y herramientas a utilizar para la construcción de la solución propuesta haciendo un estudio comparativo de las mismas y se fundamenta la metodología de desarrollo de software utilizada para guiar el proceso de construcción de la solución.

1.2 Red de Comunicaciones

La comunicación es un fenómeno inherente a la relación que los seres vivos mantienen cuando se encuentran en grupo. A través de la comunicación, las personas obtienen información respecto a su entorno y pueden compartirla con el resto. El proceso comunicativo implica la transmisión y recepción de señales (sonidos, gestos, señas) con la intención de dar a conocer un mensaje (Santo, 2012).

El término “red” se utiliza para definir una estructura enlazada que cuenta con un patrón característico. Es decir un conjunto de elementos conectados entre sí que comparten características similares (López, 2014).

Entonces una red de comunicaciones es un conjunto de elementos conectados entre sí, capaz de establecer una comunicación a través de la transmisión y recepción de señales. Las redes de comunicación permite que un conjunto de equipos enlazados entre si puedan compartir recursos y brindar servicios. La implantación de una red mundial de comunicaciones es uno de los grandes avances de las tecnologías en la actualidad.

1.3 Tecnología móvil

La tecnología móvil no es más que una de las ramas existentes dentro de los sistemas de comunicación. Esta se basa exclusivamente en sus sistemas de conexión inalámbrica. Posibilita la conversión a señales electromagnéticas de los sonidos que viajan a través del aire hasta ser

captadas. Posteriormente son transformadas por antenas repetidoras o vía satélite en mensajes hasta llegar a su receptor (Meyers, 2011).

1.3.1 Telefonía móvil

La telefonía móvil o telefonía celular, está formada por dos grandes partes: una red de comunicaciones y los teléfonos celulares que permiten el acceso a dicha red. La conexión entre ambas partes se realiza a través de ondas o frecuencias. Gran parte de las comunicaciones de la telefonía celular en el mundo se realizan sobre las redes GSM (Definiciónabc, 2015).

La telefonía móvil se ha convertido en un instrumento muy útil, debido a la fácil comunicación entre personas y al auge de los teléfonos celulares. Esta permite a la persona moverse tranquilamente por cualquier lugar sin depender de cables o aparatos estáticos que deban ser mantenidos en un espacio específico.

1.3.2 Redes GSM

En los comienzos de los años ochenta los sistemas celulares analógicos experimentaron un gran crecimiento en Europa. Cada país desarrolló su propio sistema, incompatible con el resto de los países, lo que impedía la interoperabilidad más allá de sus fronteras. Esta situación no era deseable debido a que la movilidad se limitaba a cada país y lo que es muy importante, los mercados eran muy limitados. En 1982, la Conferencia Europea de Correos y Telégrafos (CEPT, *Conference of European Post and Telegraphs*) formó un grupo de estudio denominado Grupo Especial para Móviles, para desarrollar un sistema para la telefonía móvil. El grupo propuso desarrollar un nuevo sistema inalámbrico móvil con las siguientes premisas: itinerancia[5] (*roaming*) internacional y soporte para la introducción de nuevos servicios (Cruz, 2006).

La evolución de GSM ha estado marcada por tres fases, la Fase 1, en la que se produjeron sus especificaciones; la Fase 2, en la que se propuso la inclusión de servicios de datos y de fax; y finalmente, la Fase 2+, en la que se realizan mejoras sobre la codificación de voz y se implementan servicios de transmisión de datos avanzados (Cruz, 2006).

[5] La itinerancia es la capacidad de enviar y recibir llamadas en redes móviles fuera del área de servicio local de la propia compañía, es decir, dentro de la zona de servicio de otra empresa del país, o bien durante una estancia en otro país diferente, con la red de una empresa extranjera.

GSM es un sistema de conmutación de circuitos que estandariza la telefonía móvil, diseñado originalmente para voz, al que posteriormente se le adicionaron algunos servicios de datos: servicio de mensajes de textos y un servicio de datos GSM, que permite el enviar y recibir mensajes por correo electrónico, faxes y navegar por Internet.

1.3.3 Arquitectura de la Red GSM

El sistema GSM necesita una serie de equipos y funciones para proporcionar un servicio de telefonía móvil, esencialmente se conforma por los teléfonos celulares de los suscriptores, llamado Estación Móvil (MS, *Mobile Station*), el Subsistema de Estación Base (BSS, *Base Station System*), el Subsistema de Conmutación de Red (NSS, *Network Switching Subsystem*) y el Subsistema de Soporte a la Operación (OSS, *Operation Support Subsystem*). En la figura 1 se muestran los componentes principales de la arquitectura GSM:

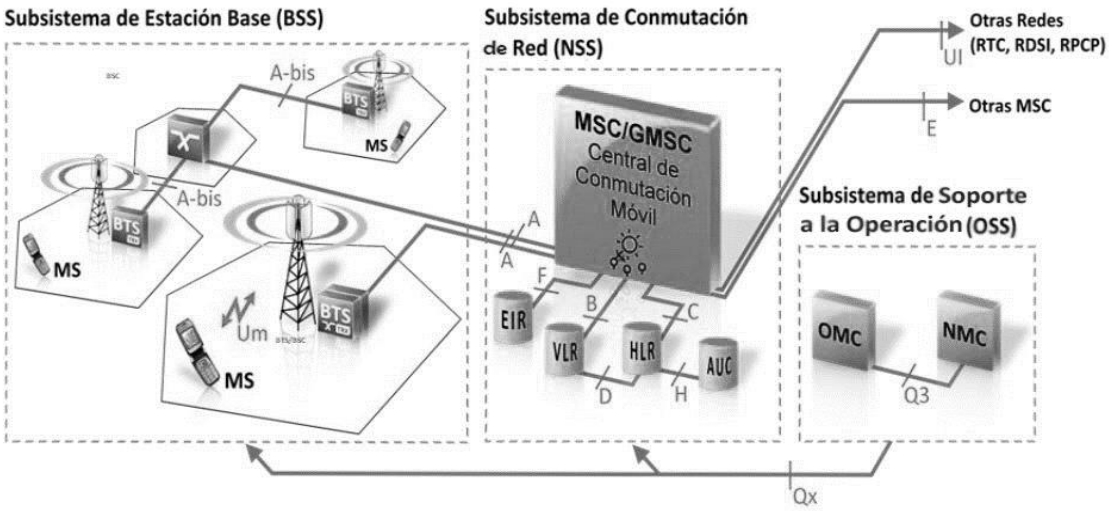


Figura 1. Componentes principales de la arquitectura GSM.

Fuente: Figura seleccionada de (Halonen, y otros, 2004).

Subsistema de Estación Base

Los teléfonos móviles se comunican con la Estación Base a través de ondas de radio. Varias estaciones base están asignadas a un Controlador de la Estación Base (BSC, *Base Station Controller*), cuya principal tarea es la de asignación y liberación de canales radio para la

comunicación con los terminales móviles y garantizar que los procesos funcionen correctamente. La operación normal del BSS requiere que cada terminal móvil proporcione información de su ubicación. El terminal móvil monitoriza también las señales procedentes de las estaciones bases más cercanas y selecciona aquella cuya señal llega con mayor potencia y establece la conexión a través de dicha estación (Halonen, y otros, 2004).

Subsistema de Conmutación de Red

Es el componente que realiza las funciones de portar y administrar las comunicaciones entre los teléfonos móviles y la Red Conmutada de Telefonía[6] para una red GSM. Dicho subsistema permite gestionar las comunicaciones entre los usuarios GSM y los usuarios de otras redes de telecomunicaciones. Dentro del NSS existen un conjunto de funcionalidades que se encargan de controlar el teléfono móvil, estas funciones son (Sánchez, 2005):

- Registro de Ubicación Base (HLR, *Home Location Register*), es una base de datos que contiene información sobre los usuarios conectados a un determinado MSC. Entre la información que almacena el HLR se tiene fundamentalmente la localización del usuario y los servicios a los que tiene acceso.
- Registro de Suscriptores Visitantes (VLR, *Visitor Location Register*), almacena temporalmente la información más reciente sobre la situación de un terminal móvil en el rango de su MSC. El VLR solicita y obtiene datos del HLR y si el terminal móvil abandona la zona visitada, sus datos se eliminan del VLR.
- Central de Autenticación (AUC, *Authentication Center*), proporciona los parámetros necesarios para la autentificación de usuarios dentro de la red; también se encarga de soportar funciones de encriptación.
- Equipo de Registro de Identidad (EIR, *Equipment Identity Register*), almacena información acerca de los terminales móviles (por ejemplo, la lista de equipos autorizados o equipos robados).

En este Subsistema se encuentra también la Central de Conmutación Móvil (MSC, *Mobile Switching Central*) se encarga de iniciar, terminar y canalizar las llamadas a través del BSC y la

[6] Red Conmutada de Telefonía*:* conjunto de elementos constituido por todos los medios de transmisión y conmutación necesarios para enlazar a voluntad dos equipos terminales mediante un circuito físico que se establece específicamente para la comunicación y que desaparece una vez que se ha completado la misma.

BSS correspondientes al abonado[7] llamado (Halonen, y otros, 2004).

Subsistema de Soporte a la Operación

El OSS es responsable de la operación de BSS y NSS. Contiene principalmente un bloque de supervisión, que se encarga de las tareas administrativas (por ejemplo, informe de facturación), un bloque de gestión global del flujo de información, y un bloque de operación y mantenimiento, que se encarga del mantenimiento y explotación de la red (Halonen, y otros, 2004).

1.3.4 Datos por Conmutación de Circuito

La conmutación de circuitos se define como un tipo de conexión que realizan los diferentes nodos de una red para lograr un camino apropiado para conectar dos usuarios de una red de telecomunicaciones. En la conmutación de circuitos se establece un canal de comunicaciones dedicado entre dos estaciones, se reservan recursos de transmisión y de conmutación de la red para su uso exclusivo en el circuito durante la conexión (Textoscientíficos, 2005).

La trasmisión de Datos por Conmutación de Circuito (CSD) es la forma original de transmisión de datos desarrollada para los sistemas de telefonía móvil basados en el Acceso Múltiple por División de Tiempo (TDMA). CSD usa un intervalo de tiempo (*time slot*) de radio individual para enviar 9,6 Kb/s de datos al NSS de una red GSM, donde podría conectarse por medio del equivalente a un módem a la Red Conmutada de Telefonía permitiendo llamadas directas a cualquier servicio de marcación (Pérez, 2003).

Antes del CSD, la transmisión de datos sobre teléfonos móviles se hacía usando un módem, integrado en el terminal o conectado a él. Tales sistemas estaban limitados por la calidad de la señal de audio a 2,4 Kb/s o menos. Con la introducción de la transmisión digital en los sistemas basados en TDMA, como el GSM, el CSD proporcionó acceso casi directo a la señal digital subyacente, permitiendo mayores velocidades. Al mismo tiempo, la compresión de audio orientada a la voz que se usa en GSM realmente significaba que las velocidades de datos usando un módem tradicional conectado al teléfono habrían sido incluso inferiores a las de los antiguos sistemas analógicos (Pérez, 2003).

[7] Abonado: persona que mantiene un contrato con un proveedor de servicios de comunicaciones para recibir un determinado servicio.

Una llamada CSD funciona de una forma muy similar a una llamada de voz en una red GSM. Se asigna en exclusiva un único intervalo de tiempo de radio entre el teléfono y la Estación Base.

Ventajas de realizar una llamada haciendo uso de CSD:

- El ancho de banda es definido y se mantiene constante durante la comunicación.
- La llamada no presentaría retardo ya que el circuito es fijo y no se pierde tiempo en el almacenamiento de la información.
- La transmisión se realiza en tiempo real.
- Permite el envío de datos no estructurados por lo que brinda la posibilidad de la transmisión de cualquier tipo de datos.

Para realizar la comunicación de datos por CSD es necesario que el operador de telefonía móvil autorice y active el servicio de datos CSD. El operador asignará un número de teléfono para la comunicación de datos o en ocasiones es posible que el operador asigne un número único para todos los servicios de voz y datos.

1.4 Sistema Operativo Android

Android es un SO basado en el *kernel* de Linux[8]. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como *smartphones*, *tablets*, relojes inteligentes, televisores y automóviles. Android se ha convertido en la plataforma más popular de los teléfonos inteligentes, al ser de código abierto. Este ha sido la elección de muchas empresas que fabrican teléfonos (Michelone, 2013).

1.4.1 Reseña Histórica

En noviembre del 2007 Google anunciaba Android como una plataforma móvil, a raíz de la adquisición de una pequeña empresa llamada Android Inc. que comenzaba el desarrollo de este SO para móviles. Casi un año después del anuncio de este SO, salió al mercado el primer equipo con SO Android, el HTC G1, precursor de este SO y responsable de la posterior incorporación a

[8] *Kernel* de Linux: es un software desarrollado por Linus Torvalds, que constituye una parte fundamental del SO.

este mercado de otras marcas como Samsung[9], Motorola[10] y Sony Ericsson[11] (Michelone, 2013).

Integrantes de Android

Google es su principal impulsor, es quien lo desarrolla. Para hacerlo más abierto y que no sea un SO manejado por una sola empresa, se creó una organización sin fines de lucro denominada la Alianza Abierta de Dispositivos (OHA, *Open Handset Alliance*). Esta incluye a más de 65 empresas del sector de fabricación de dispositivos móviles, empresas proveedoras de servicios móviles, y fabricantes de microprocesadores y placas de video. Ellas son responsables de mantener a Android como SO (Openhandsetalliance, 2007).

1.4.2 Características de Android

Las características más importantes del SO Android son (Vílchez, 2009):

- **Navegador integrado:** basado en los motores de renderizado de *open source Webkit*[12].
- **SQLite:** base de datos para almacenamiento estructurado que se integra directamente con las aplicaciones.
- **Multimedia:** soporte para medios con formatos comunes de audio, video e imágenes planas (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF).
- **Máquina virtual Dalvik:** basado en el concepto de máquina virtual utilizado en Java. Facilita la optimización de recursos como, la ejecución de ficheros Dalvik y ahorro de memoria.
- **Entorno de desarrollo:** Incluye un conjunto de herramientas para facilitar el trabajo a los desarrolladores como un emulador de dispositivos, herramientas para depuración[13] de memoria y análisis del rendimiento del software.

[9] Samsung: www.samsung.com

[10] Motorola: www.motorola.com

[11] Sony Ericsson: www.sonymobile.com

[12] *WebKit***:** es una plataforma para aplicaciones que funciona como base para navegadores web, que cumple estrictamente los estándares web.

[13] Depuración: proceso que lleva a cabo un programa para probar y eliminar los errores del programa objetivo.

1.4.3 Arquitectura Android

A continuación, se detallarán las diferentes capas de la arquitectura de componentes del SO Android (Gironés, 2012):

- **Aplicaciones:** incluyen todas las aplicaciones del dispositivo, tanto las que tienen interfaz de usuario como las que no.
- **Marco de aplicación:** formada por todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones. La mayoría de los componentes de esta capa son librerías Java que acceden a los recursos de las capas anteriores a través de la máquina virtual Dalvik.
- **Librerías:** Android incluye un conjunto de librerías desarrolladas en C o C++ y compiladas para la arquitectura de hardware específica del dispositivo. Estas normalmente están hechas por el fabricante del dispositivo. El objetivo de las librerías es proporcionar funcionalidad a las aplicaciones para tareas que se repiten con frecuencia.
- **Entorno de ejecución:** cada aplicación Android ejecuta su propio proceso, con su propia instancia de la máquina virtual Dalvik que ejecuta archivos en el formato Ejecutables Dalvik (.dex), optimizado para utilizar memoria mínima.
- ***Kernel* de Linux:** Android depende en la última de sus versiones, del núcleo 3.0.31 de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red, y modelo de *drivers*. El núcleo también actúa como una capa de abstracción entre el hardware y las aplicaciones que acceden a él.

En la figura 2 se muestra la arquitectura de los componentes de Android descrita anteriormente:

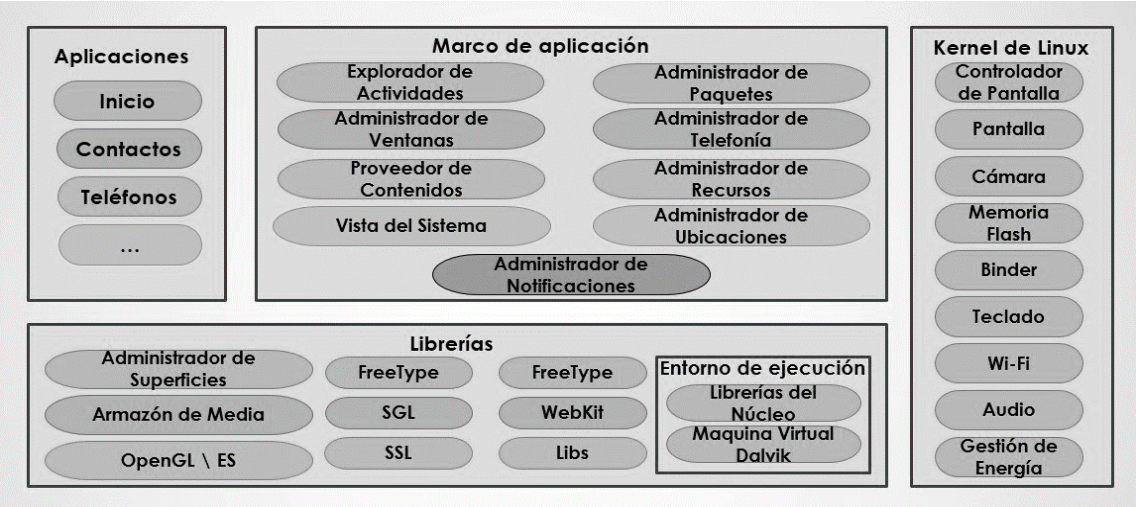


Figura 2. Arquitectura de componentes de Android.

Fuente: Elaboración propia con apoyo de (Gironés, 2012).

1.5 Análisis de soluciones similares

Para garantizar la seguridad en las redes telefónicas existen varias soluciones, entre ellas se destacan dos aplicaciones para el SO Android, que protegen la privacidad de los usuarios que utilizan estas herramientas. A continuación se detalla un análisis de dichas aplicaciones, donde se exponen sus características, ventajas y limitaciones.

1.5.1 RedPhone

RedPhone es una aplicación para dispositivos móviles con SO Android, creada por la empresa *Whisper Systems*, para realizar llamadas telefónicas cifradas, asegurando las conversaciones de modo que nadie pueda escuchar. RedPhone tiene como objetivo establecer una comunicación VoIP cifrada de extremo a extremo. Utiliza el protocolo ZRTP[14] para cifrar la conversación y Diffie-Hellman para negociar las claves privadas, que se intercambian entre los dos interlocutores para establecer el canal seguro (Eddy, 2014).

[14] ZRTP es un protocolo desarrollado por Phil Zimmermann, y es utilizado para negociar las claves para el cifrado entre dos puntos a través de una llamada VoIP.

Una vez instalado RedPhone en el *smartphone*, la aplicación se conectará a un servidor central que hace como pasarela de datos, y pone en contacto a los usuarios (usando como identificador su número de teléfono). En la figura 3 se muestra la Arquitectura de RedPhone:

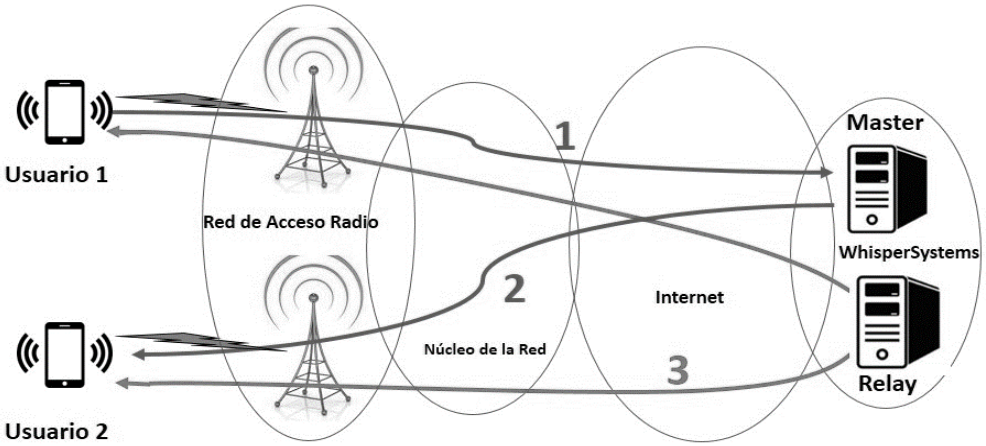


Figura 3. Arquitectura de RedPhone.

Fuente: Elaboración propia con apoyo de (Eddy, 2014).

1. El Usuario 1, que es el teléfono con el cliente RedPhone, contacta con el Servidor *Master*[15] y señaliza que quiere establecer una llamada con el Usuario 2. El Servidor *Master* será el encargado del proceso de señalización y cifrado de la llamada.
2. El Usuario 2 recibe una llamada cifrada y conecta con el Servidor *Master* que indica que ha recibido la señal de llamada correctamente.
3. Si el Usuario 2 elige contestar la llamada, se reenvía la comunicación al Servidor *Relay*[16] más cercano.

Ventajas de utilizar RedPhone:

- Establecer una comunicación cifrada de extremo a extremo para las llamadas, asegurando las conversaciones de modo que nadie pueda escuchar.

[15] Servidor *Master*: servidor diseñado y ubicado en la empresa *Whisper Systems*.

[16] Servidor *Relay*: servidor encargado de la comunicación entre dos usuarios.

Limitaciones de utilizar RedPhone:

- Se requiere de un servidor de confianza que manejará las claves de cifrado. Esto se convierte en un gran problema si en dicho servidor no se confía.
- RedPhone es una aplicación de espacio de usuario. Esto significa que es incapaz de proporcionar garantías de seguridad sobre el SO.
- Se limita a establecer una comunicación VoIP, y no está basada en la red de telefonía móvil que se utilice.

1.5.2 Cellcrypt

La aplicación Cellcrypt para el SO Android, establece una llamada cifrada de óptima calidad entre *smartphones*. Cellcrypt utiliza el Protocolo de Cifrado de Contenido Móvil (EMCP, *Encrypted Mobile Content Protocol*) y un conjunto de protocolos basados en estándares tecnológicos, con el fin de optimizar el intercambio de contenido entre *smartphones* en tiempo real, a través de redes inalámbricas con bajo ancho de banda (Cellcrypt, 2014). En la figura 4 se muestra la Arquitectura de Cellcrypt:

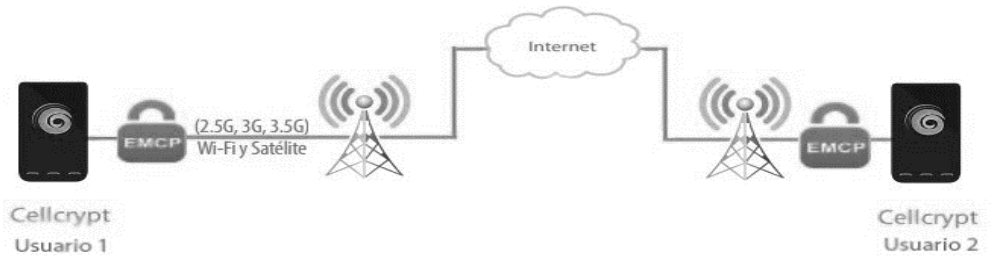


Figura 4. Arquitectura de Cellcrypt.

Fuente: Figura seleccionada de (Cellcrypt, 2014).

Cellcrypt utiliza tecnologías de cifrado estándar, incluyendo:

- AES para criptografía simétrica.
- Algoritmo de Firma Digital de Curva Elíptica (ECDSA) para firmas digitales.
- Curva Elíptica Diffie-Hellman (ECDH) para acuerdo de clave.

- Algoritmo de *Hash* Seguro (SHA) para resumen de mensaje.

Para realizar una llamada segura con Cellcrypt, el Usuario 1 marcara el número telefónico del Usuario 2, Cellcrypt se encargará de cifrar la llamada primeramente utilizando el RC4-256 bit y después es cifrada de nuevo utilizando el AES-256 bit. Posteriormente se utilizará RSA y ECDSA para la autenticación, es decir para el intercambio de claves. Y por último se utiliza SHA para incrementar el aseguramiento de la integridad del cifrado (Cellcrypt, 2014).

Ventajas de utilizar Cellcrypt:

- Sólido cifrado de extremo a extremo.
- Certificado por el gobierno de los EE.UU. bajo la norma FIPS 140-2 del NIST (Cert # 1310).
- Alta calidad de llamadas con baja latencia.
- Interopera a través de redes celulares.

Limitaciones de utilizar Cellcrypt:

- Se limita a establecer una comunicación VoIP, y no está basada en la red de telefonía móvil que se utilice.

Después de realizar un análisis detallado de las soluciones existentes se llega a la conclusión de que es necesario realizar una nueva solución pues las existentes no resuelven las necesidades actuales. Puesto que RedPhone y Cellcrypt, no realizan comunicaciones basadas en las redes telefónicas, como las que se utilizan en Cuba, sino en redes que estén conectadas a Internet. El análisis de estas herramientas sirve de guía para la elaboración de la nueva solución, además de propiciar un conjunto de funcionalidades que se implementarán en el nuevo sistema.

1.6 Algoritmos de cifrado

El cifrado de datos es el proceso por el que una información legible se transforma mediante un algoritmo en información ilegible, llamada criptograma o secreto. Esta información ilegible se puede enviar a un destinatario con muchos menos riesgos de ser leída por terceras partes. El destinatario puede volver a hacer legible la información (descifrarla), introduciendo la clave del cifrado (Microsoftdevelopernetwork, 2005).

Para el cifrado digital se utiliza la criptografía simétrica y la asimétrica. La criptografía simétrica, también llamada criptografía de clave secreta, es un método criptográfico en el cual se usa una misma clave para cifrar y descifrar mensajes, a diferencia de la criptografía asimétrica que utiliza una clave privada y una clave pública para el cifrado. Dentro de los algoritmos simétricos más usados se encuentra AES.

1.6.1 AES

El Estándar de Encriptación Avanzada (AES, *Advanced Encryption Standard*), es uno de los algoritmos simétricos más utilizados hoy en día, al estar disponible para el uso público. Este algoritmo está clasificado por la NSA, de los Estados Unidos para la seguridad más alta de información secreta que esta maneja. Es un algoritmo de cifrado por bloques, donde la longitud del bloque y de la clave es variable. Los bloques tienen un tamaño de 128 bits y las llaves de 128, 192 o 256 bits (Boxcryptor, 2015).

Ventajas:

- AES es rápido tanto en software como en hardware.
- Es relativamente fácil de implementar, y requiere poca memoria.
- Es de dominio público, disponible para todo el mundo.

En la figura 5 se representa el esquema para el cifrado de datos utilizando el algoritmo de cifrado AES:

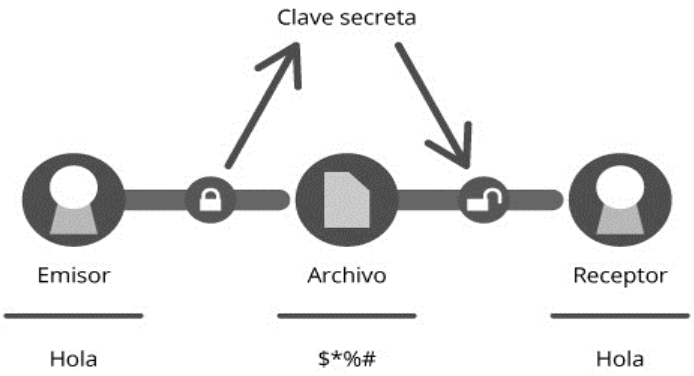


Figura 5. Esquema para el cifrado de datos utilizando AES.

Fuente: Figura seleccionada de (Gutiérrez, 2013).

AES fue seleccionado para el cifrado de datos en el sistema a construir debido a que su velocidad para el cifrado/descifrado es alta. Este algoritmo también brindará al sistema un alto grado de seguridad en el envío de los datos a través de las redes móviles.

1.7 Metodología de Desarrollo de Software

Una metodología de desarrollo de software se puede definir como un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda en la construcción de un software (Pressman, 2001).

En la actualidad existen varias metodologías divididas en dos grandes grupos, las tradicionales y las ágiles. Las primeras se centran en elaborar una documentación exhaustiva de todo el desarrollo del software y en cumplir con la planificación realizada en la fase inicial del proyecto, sin dar margen a posibles cambios, puesto que al ocurrir uno, se ven afectados varios componentes del proceso de desarrollo del software. Otro elemento que las caracteriza es la gran cantidad de participantes con los que cuenta, pues requieren de un equipo de trabajo capaz de administrar un proceso complejo en varias etapas. Alternativamente, surgen las metodologías ágiles, caracterizadas por ser más orientadas al desarrollo de software, con bajos niveles de formalización en la documentación requerida y por ser, a diferencia de las tradicionales, más adaptables a los cambios, requerir de pequeños grupos de trabajo y por ser apropiadas para entornos volátiles (Lara Almarales, y otros, 2013).

1.7.1 Proceso Unificado Abierto

OpenUP (*Open Unified Process*) es un proceso unificado que aplica enfoques iterativos e incrementales dentro de un ciclo de vida estructurado. Utiliza una filosofía ágil que se enfoca en la naturaleza de colaboración en el desarrollo de software. Es una herramienta que se utiliza para hacer frente a una gran variedad de proyectos. OpenUP está basada en RUP (Proceso Unificado de Desarrollo), por lo tanto, comparte las mismas prácticas en el flujo de trabajo y los roles. Las prácticas son las siguientes: gestionar requisitos, utilizar una arquitectura basada en componentes, modelar software de forma visual, desarrollar software de manera iterativa, controlar los cambios aplicados al software y verificar la calidad del software (Eclipse, 2010).

Los 4 principios básicos por los que se caracteriza OpenUP son:

- Colaboración para unificar intereses y compartir conocimientos.
- Balancear las prioridades para maximizar las necesidades de los *stakeholders*[17].
- Centrado en la arquitectura.
- Desarrollo Iterativo.

Beneficios en el uso del OpenUP (Monte, 2013):

- Es apropiado para proyectos pequeños y de bajos recursos. En ellos permite disminuir las probabilidades de fracaso en ellos e incrementando las probabilidades de éxito.
- Permite detectar errores tempranos a través de un ciclo iterativo.
- Evita la elaboración de documentación, diagramas e iteraciones innecesarios requeridos en la metodología RUP.
- Por ser una metodología ágil tiene un enfoque centrado al cliente y con iteraciones cortas.

La metodología de desarrollo de software OpenUP permite la generación de los artefactos necesarios en cada fase del ciclo de desarrollo, tiene como principal objetivo evolucionar para estar siempre en una mejora continua.

1.8 Lenguajes de Desarrollo de Software

1.8.1 Lenguaje Unificado de Modelado (UML)

UML es un lenguaje, éste posee más características visuales que programáticas, las mismas facilitan a integrantes de un equipo multidisciplinario participar e intercomunicarse fácilmente, estos integrantes son los analistas, diseñadores, especialistas de área y desde luego los programadores (Mastermagazine, 2015).

Se utilizará como notación UML, para lograr un mayor entendimiento ya que permite modelar y describir secuencialmente por pasos todos los procesos que se llevan a cabo según la problemática planteada. UML es un lenguaje estándar para construir planos de software, que se

[17] *Stakeholder*: usuario, grupo de personas, organización u otra entidad que posee un interés directo o indirecto en un sistema, que puede afectar o afectarse por las acciones u objetivos del mismo.

utiliza para visualizar, especificar, construir y documentar los artefactos de un sistema.

1.8.2 Lenguaje de Programación

Los lenguajes de programación son un conjunto de reglas, herramientas y condiciones que permiten crear programas o aplicaciones dentro de una computadora. Estos programas son los que permitirán ordenar distintas acciones a la computadora en un idioma comprensible por ella (Lanzillotta, 2015).

Lenguaje de Programación Java

La principal característica de Java es la de ser un lenguaje compilado e interpretado, a la vez de ser de código abierto. Todo programa en Java ha de compilarse y el código generado es interpretado por una máquina virtual. De este modo se consigue la independencia de la máquina, el código compilado se ejecuta en máquinas virtuales que si son dependientes de la plataforma. Java es un lenguaje orientado a objetos de propósito general. Aunque Java comenzará a ser conocido como un lenguaje de programación de *applets*[18] que se ejecutan en el entorno de un navegador web, se puede utilizar para construir cualquier tipo de proyecto (Frozzt, 2015).

Se selecciona este lenguaje de programación porque permite optimizar el tiempo y el ciclo de desarrollo (compilación y ejecución). Además, mediante su uso permite la creación de aplicaciones para el SO Android. Cientos de aplicaciones desarrolladas en Java corriendo satisfactoriamente en la plataforma Android, certifican lo idóneo que resulta este lenguaje de programación.

Lenguaje de Programación C

El lenguaje de programación C está orientado a la implementación de Sistemas Operativos, concretamente Unix[19]. El lenguaje C es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones. El SO Android utiliza el lenguaje C para implementar su núcleo y librerías

[18] *Applets*: programas que pueden incrustarse en un documento HTML.

[19] Unix: es un SO portable, multitarea y multiusuario. Desarrollado en el año 1969, por un grupo de empleados de los laboratorios Bell de AT&T.

para la comunicación con el hardware (Ruiz, 2003).

1.9 Herramientas de Desarrollo de Software

1.9.1 Herramienta Case Visual Paradigm para UML

Las herramientas CASE (Ingeniería de Software Asistida por Computadora) son utilizadas para automatizar o apoyar una o más fases del proceso de desarrollo de software, sirven de ayuda a los ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software. Entre las herramientas CASE para el modelado de los artefactos se encuentra Visual Paradigm, de la cual a continuación se describen algunas de sus características más importantes (Mastermagazine, 2015).

Visual Paradigm 8.0

Visual Paradigm para UML es una herramienta de modelado que soporta el modelado mediante UML y proporciona asistencia tanto a los analistas, ingenieros de software como a los desarrolladores durante todo el ciclo de vida del proyecto (Visualparadigm, 2015).

Características:

- **Dibujo**: facilita el modelado de UML, ya que proporciona herramientas específicas para ello. Esto también permite la estandarización de la documentación, ya que la misma se ajusta al estándar soportado por la herramienta.
- **Coherencia entre diagramas**: al disponer de un repositorio común, es posible visualizar el mismo elemento en varios diagramas, enviando duplicados.
- **Reutilización**: facilita la reutilización, ya que dispone de una herramienta centralizada donde se encuentran los modelos utilizados para otros proyectos.
- **Generación de código**: permite generar código de forma automática, reduciendo los tiempos de desarrollo y evitando errores con la codificación del software.
- **Generación de informes**: permite generar diversos informes a partir de la información inducida en la herramienta.

La herramienta de modelado Visual Paradigm para UML en su versión 8.0 soporta el lenguaje de

modelado UML, facilita el trabajo del equipo de desarrollo durante el ciclo de vida del software y permite la estandarización de la documentación que se va generando.

1.9.2 Entorno de desarrollo integrado

El Entorno de Desarrollo Integrado (IDE) es un programa informático compuesto por herramientas de programación que pueden ser partes de aplicaciones existentes. Estos sistemas pueden manejar uno o varios lenguajes de programación ejemplo Java, C# y C++. Han sido empaquetados como programas de aplicaciones, editores de código, depuradores, compiladores o constructores de interfaces gráficas (Docsetools, 2015).

Eclipse 3.8

Eclipse es un entorno de desarrollo integrado, de código abierto y multiplataforma. Es un potente y completo entorno de desarrollo que posibilita la creación y compilación de elementos variados como sitios web, programas en C++ o aplicaciones Java (Eclipse, 2015).

Ventajas (PicandoJava, 2015)**:**

- El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (*plugin*) para proporcionar toda su funcionalidad al frente de la plataforma utilizada, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no.
- Este mecanismo de módulos es una plataforma ligera para componentes de software.
- La arquitectura *plugin* permite escribir cualquier extensión deseada en el ambiente. Se provee soporte para Java y el Sistema de Versiones Concurrentes (SVC) en el SDK de Eclipse. No tiene por qué ser usado únicamente para soportar otros lenguajes de programación, sino que puede familiarizarse con otros entornos y plataformas.

La elección del IDE Eclipse fue basada en las ventajas que brinda y su compatibilidad con los SO, siendo un IDE multiplataforma. Eclipse también permite el desarrollo de aplicaciones para el SO Android utilizando herramientas como el SDK de Android y el ADT (*Android Development Tools*) como *plugin*, diseñada para facilitar un ambiente potente e integrado en la construcción de aplicaciones para Android.

1.9.3 Sistema Gestor de Base de Datos SQLite

SQLite es un sistema de gestión de bases de datos que se enlaza con el sistema pasando a ser parte integral del mismo. El sistema utiliza la funcionalidad de SQLite a través de llamadas simples a funciones. Esto reduce la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más rápidas y seguras que la comunicación entre procesos (Álvarez, 2007).

SQLite es una de las mejores alternativas para la persistencia de los datos sobre texto plano o ficheros en XML ya que es un sistema gestor muy ligero que posibilita la portabilidad local de los datos y no requiere de un gran procesamiento en memoria para su gestión. Android posee herramientas muy útiles que facilitan las tareas de manipulación y consulta sobre este tipo de base de datos.

1.9.4 Android SDK

Android SDK es un conjunto de recursos fundamentales de los que dispone la plataforma Android para el desarrollo de aplicaciones. Se debe configurar el IDE Eclipse para que localice el SDK de Android y disponga de sus recursos en el entorno de desarrollo. Entre los recursos de los que dispone el SDK de Android se encuentra un emulador de dispositivos, para probar los desarrollos sin necesidad de realizar instalaciones sobre un dispositivo físico (Developerandroid, 2015).

El paquete de Desarrollo de Software de Android (SDK, *Software Development Kit*), no es más que un conjunto de herramientas de desarrollo que le permite a un programador crear aplicaciones para Android. Estas proporcionan bibliotecas de interfaz de programación de aplicaciones creadas con el fin de permitir un mejor uso de las técnicas a desarrollar en este sistema.

El SDK continuamente contiene códigos de ejemplo y notas técnicas de soporte para ayudar a clarificar ciertos puntos del material de referencia. Este marco de trabajo es tan sencillo como las APIs, creadas con el objetivo de permitir el uso de cierto lenguaje de programación.

1.9.5 ADT *Plugin* para Eclipse 21.1

Plugin que extiende las funcionalidades de Eclipse para proporcionar un entono de desarrollo

orientado a la plataforma Android, dispone de toda una serie de utilidades como: componentes basados en el UI de Android, herramientas de depuración del SDK de Android, generación de aplicación en formato .apk (Developerandroid, 2015).

1.10 Conclusiones parciales

En este capítulo fueron analizadas distintas herramientas que brindan mecanismos para aumentar la seguridad en las llamadas telefónicas en el SO Android indicando que estas herramientas no resuelven las necesidades actuales. Para dar cumplimiento al problema de la investigación se hará uso de una llamada CSD, por las ventajas que presenta, y utilizando el algoritmo AES para el cifrado de los datos a transmitir en dicha llamada. Se seleccionaron JAVA como lenguaje de programación, Android como plataforma de desarrollo y Eclipse como entorno de desarrollo, al ser las herramientas de desarrollo más usadas para este tipo de solución. La metodología para el desarrollo seleccionada fue OpenUP, al ser ágil y centrada en la naturaleza colaborativa de desarrollo de software.

Capítulo 2. Análisis y diseño

2.1 Introducción

En este capítulo se analiza el proceso que lleva a cabo el SO Android para realizar llamadas telefónicas, basado en la versión 4.0 del SO Android (*Ice Cream Sandwich*). Este estudio se realiza con el objetivo de elevar el nivel de seguridad en las llamadas telefónicas al incorporar un mecanismo de cifrado en dicho proceso. Se detalla cómo se realiza el diseño del sistema a desarrollar, siguiendo la metodología OpenUP, y desarrollándose los diagramas pertinentes para mejor entendimiento del sistema.

2.2 Proceso de realización de llamadas telefónicas en el SO Android

A continuación se describen un conjunto de conceptos que permiten comprender mejor el proceso de llamadas telefónicas:

- **Dialer**: aplicación con la que el usuario interactúa, en la cual se puede introducir información como el número telefónico con el que desea comunicar. Esta será la encargada de iniciar el proceso de realización de una llamada telefónica.
- **Phone**: provee un conjunto de APIs para el seguimiento de la información básica del *smartphone*, como el tipo de red y el estado de la conexión, más los servicios públicos para la manipulación de secuencias de números telefónicos.
- **RIL**: provee una capa de abstracción entre los servicios telefónicos del SO Android (android.telephony) y el hardware. De esta forma el SO se independiza de los aspectos técnicos de cada modelo de *smartphone*, ya que esta capa es la que implementa las características específicas de cada *smartphone*.

La RIL está integrada por dos componentes:

- **RIL Daemon** (RILD): inicializa el Vendor RIL, procesa todas las comunicaciones provenientes del android.telephony y realiza las llamadas al Vendor RIL como comandos solicitados[20].

[20] Comandos solicitados: son comandos originados por la RIL como marcar y colgar.

- **Vendor RIL**: procesa todas las comunicaciones con el hardware de radio y realiza las llamadas al RILD a través de comandos no solicitados[21]. Contiene las librerías que brinda el fabricante de *smartphones* para la comunicación del SO y el hardware de comunicación.

2.2.1 Servicios de telefonía:

Para el uso de los servicios de telefonía móvil el SO Android cuenta con una serie de elementos que interactúan entre sí, estos se analizarán como una estructura de capas, en las que se aprecia la capa de aplicación, la capa del marco de aplicación, la capa de librerías, la Capa de Interfaz de Radio (RIL, *Radio Interface Layer*), la capa del *kernel* de Linux y la capa de radio (*baseband*) (Shell, y otros, 2004).

En la capa de aplicación se encuentran las aplicaciones que se usan comúnmente para enviar y recibir mensajes de texto, y para la realización y recibo de llamadas telefónicas. Dichas aplicaciones generalmente usan un conjunto de APIs que facilitan funcionalidades y una estructura común, para organizar y optimizar el trabajo con los componentes responsables de las llamadas telefónicas a través de las redes telefónicas. Estos mecanismos se ubican en la capa de marco de aplicación y en la capa de librerías. Estas dos capas son las que interactúan con la RIL, siendo esta la encargada de comunicar los servicios de telefonía de Android y el hardware de radio. En esta capa también se encuentran librerías específicas para el hardware de cada fabricante de dispositivos. Por último se encuentran la capa del *kernel* de Linux y la capa de radio. El *kernel* de Linux brinda un conjunto de *drivers* para la comunicación de la RIL con la capa de radio, la cual se encargará de comunicarse con las redes externas (Shell, y otros, 2004).

[21] Comandos no solicitados: son comandos que se originan en el *baseband* como la notificación de una llamada o un mensaje nuevo.

Para mejor comprensión, en la figura 6, se muestra la estructura antes descrita:

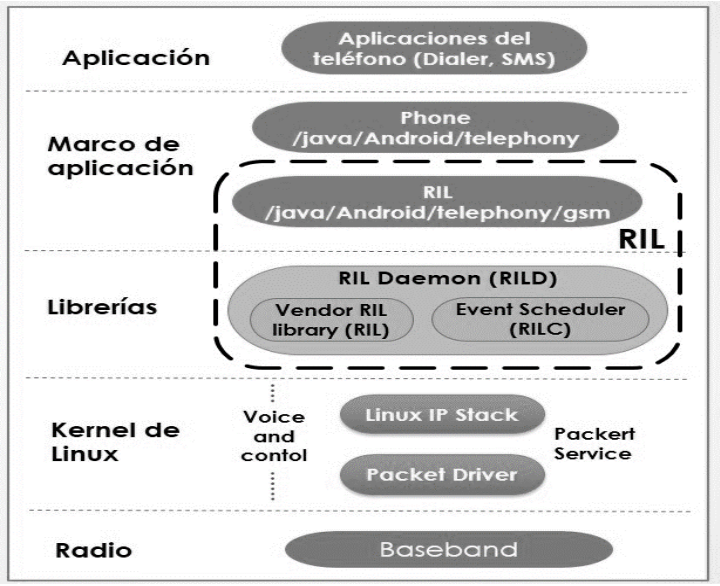


Figura 6. Estructura de los servicios de telefonía móvil de Android.

Fuente: Elaboración propia con apoyo de (Shell, y otros, 2004).

2.2.2 Componentes de la comunicación en las llamadas telefónicas

Los componentes de los medios de comunicación de la capa del marco de aplicaciones son responsables del soporte multimedia de Android. El audio que parte de esta capa es de interés para las llamadas telefónicas. La unidad básica del audio del SO Android es el *stream*. Un *stream* puede ser la entrada o salida de cualquier dispositivo de audio o aplicación. Los componentes que hacen posible el flujo de audio son:

- ***Audio System*:** clase que proporciona constantes genéricas para el dispositivos, como el flujo de *stream* en una llamada. También brinda métodos significativos que controlan los medios de comunicación en el SO.
- ***Audio Flinger*:** maneja el flujo de *stream* del micrófono y altavoces disponibles. También establece

modos de enrutamiento para todo el sistema, como MODE_IN_CALL[22] o MODE_RINGTONE[23].

- **Lib Hardware:** librerías que proporciona el fabricante de *smartphones* para la comunicación con el hardware de sus dispositivos.

2.2.3 Llamadas telefónicas

Las llamadas de telefónicas en el SO Android son el resultado final de una colaboración entre los servicios de telefonía y los medios de comunicación. Una llamada telefónica comienza cuando el usuario interactúa con el Dialer, introduciendo el número a marcar y seleccionando la opción llamar, el Dialer realizará una solicitud de una llamada telefónica con el número especificado. Esta solicitud se propagará por las diferentes capas hasta llegar al Vendor RIL. El Vendor RIL le enviará la solicitud directamente al *baseband* (Burns, y otros, 2015).

El *baseband* hará la solicitud a la red telefónica apropiada. En caso de éxito, se comenzará a transmitir a través del altavoz del teléfono, la señal recibida y se enviará la voz capturada del micrófono.

Toda la interacción descrita anteriormente se muestra en figura 7:

Figura 7. Estructura de una llamada telefónica en el SO Android.

Fuente: Figura seleccionada de (Burns, y otros, 2015).

[22] MODE_IN_CALL: modo que se establece cuando se está ejecutando una llamada telefónica.

[23] MODE_IN_RINGTONE: modo que se establece cuando se recibe o se envía una solicitud de llamada.

1. Se realiza una solicitud de llamada.
2. Se establece la llamada.
3. Los servicios de telefonía ponen a los componentes de los medios de comunicación del teléfono en MODE_IN_CALL.
4. Se retorna al Dialer, mostrando la llamada en curso.

2.3 Propuesta de solución basada en los componentes del SO Android

Para la implementación de una llamada CSD cifrada es necesario realizar un conjunto de cambios en el SO Android. Estos cambios implican la creación de nuevos módulos que realizaran las funciones necesarias para llevar a cabo dicha llamada.

Primeramente se adicionarán al SO dos módulos con la función de controlar el flujo de audio y el cifrado del mismo. El Módulo de Cifrado del Audio (MCA) y el Módulo de Control del Cifrado de Audio (MCCA). Estos módulos crean una pila de soporte para el cifrado del audio de una llamada. El MCA tendrá la función de cifrar el flujo de audio que proviene del micrófono y descifrarlo antes de enviarlo a los altavoces disponibles, durante una llamada telefónica. Este módulo se añadirá en la capa de librerías en conjunto con las librerías que proporciona el fabricante de dispositivos para la comunicación con el hardware de audio. El MCCA proporcionará los ajustes necesarios para la interacción entre los componentes de los medios de comunicación y los componentes del servicio de telefonía. Estos ajustes permitirán el cifrado del *stream* de audio durante una llamada telefónica.

Posteriormente se adicionarán los módulos encargados de realizar la llamada CSD, estos son:

- **Dialer**: interfaz que brindará al usuario las funcionalidades de realizar y responder llamadas CSD cifradas. Esta interfaz también gestiona los contactos y las claves de cifrado que usará el usuario en una llamada cifrada.
- **Módulo CSDCall**: módulo que proporciona las funcionalidades necesarias para la comunicación del Dialer con el RIL.
- **Librería CSD Vendor RIL**: librería encargada de gestionar las comunicaciones de una llamada CSD entre el RIL y el *baseband*.

En la figura 8 se muestra una visión general de las interacciones de los módulos durante el proceso de ejecución de una llamada CSD:

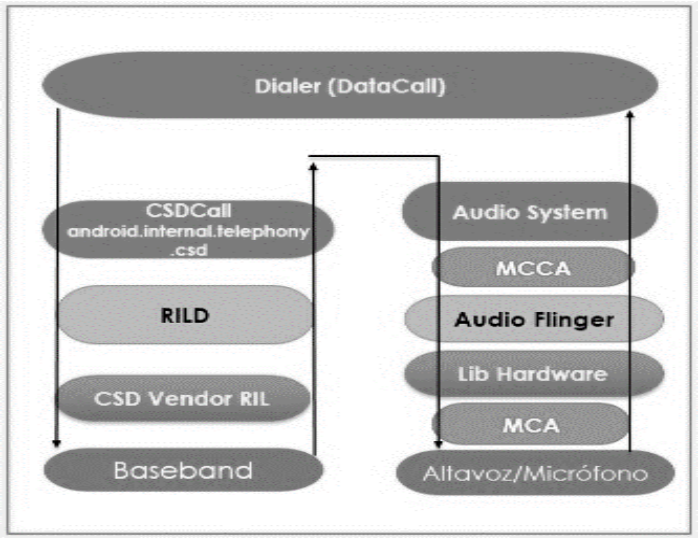


Figura 8. Interacciones de los módulos durante el proceso de una llamada CSD.

Fuente: Elaboración propia con apoyo de (Burns, y otros, 2015).

2.4 Propuesta de solución general

Para darle solución al problema de investigación identificado se propone desarrollar un sistema para el trabajo con llamadas de datos que permita realizar llamadas telefónicas seguras, extremo a extremo, entre *smartphones* con SO Android, usando la red GSM de cualquier operador. Las llamadas establecerán una conexión de datos entre ambos dispositivos, y para que progrese correctamente es necesario que ambos teléfonos seguros estén previamente autorizados por los organismos o entidades correspondientes.

Dicho sistema utiliza el número de teléfono que trae por defecto la línea para hacer y recibir llamadas o un número dado por el operador para realizar llamadas de datos por lo que no necesita otro identificador. A partir de las modificaciones del SO Android que se proponen, el usuario puede realizar una llamada cifrada, teniendo en cuenta que el receptor de la llamada también este autorizado y tenga el acceso a dicha llamada cifrada.

La aplicación usará un cifrado simétrico, en este caso AES, para emplear menor cantidad de recursos (espacio y memoria). Se aplicará a partir de un secreto establecido por el organismo o entidad responsable del acceso a este tipo de comunicación, en función de las políticas de seguridad que en particular para este caso se establezcan.

Se usará como códec de audio[24] para la compresión de voz el que trae por defecto el SO Android, que incluye un conjunto de algoritmos que permiten codificar y descodificar el audio, con el fin de comprimir las señales de voz para que ocupen el menor espacio posible, y así lograr que la información se ajuste al ancho de banda 9,6 Kb/s, de los canales de transmisión que poseen los protocolos CSD.

Las llamadas se establecerán mediante los protocolos y mecanismos que usan las redes GSM. Para realizar pruebas de las funcionalidades se propone realizar un almacenamiento temporal de la información de la llamada, cifrada y codificada, permitiendo validar la seguridad, obteniéndose una interfaz genérica que se ocupará de hacer la implementación de la transmisión posteriormente. Esta información puede ser recibida por el otro extremo previo descifrado y descodificado. De existir las facilidades tecnológicas, dígase terminal móvil autorizado, redes GSM y SIM, que tengan servicio de conmutación de circuito, se deben realizar las modificaciones de esta implementación que se dejan plasmadas como guía en el documento de la investigación.

2.5 Modelo de Dominio

Un modelo conceptual es una representación de conceptos en un dominio del problema cuya cualidad esencial es representar elementos del mundo real y no componentes del software. Dicho modelo es el encargado de comunicar a los interesados (ejemplo: desarrolladores) cuáles son los procesos del negocio y como se relacionan entre sí. La elaboración del modelo de dominio es una de las primeras actividades dentro del ciclo de desarrollo (Craig, 1999). En la figura 9 se representa el Modelo de Dominio del sistema:

[24] Códec de audio: conjunto de algoritmos que permite comprimir señales o ficheros de audio con un flujo de datos (*stream*) con el objetivo de que ocupan el menor espacio posible.

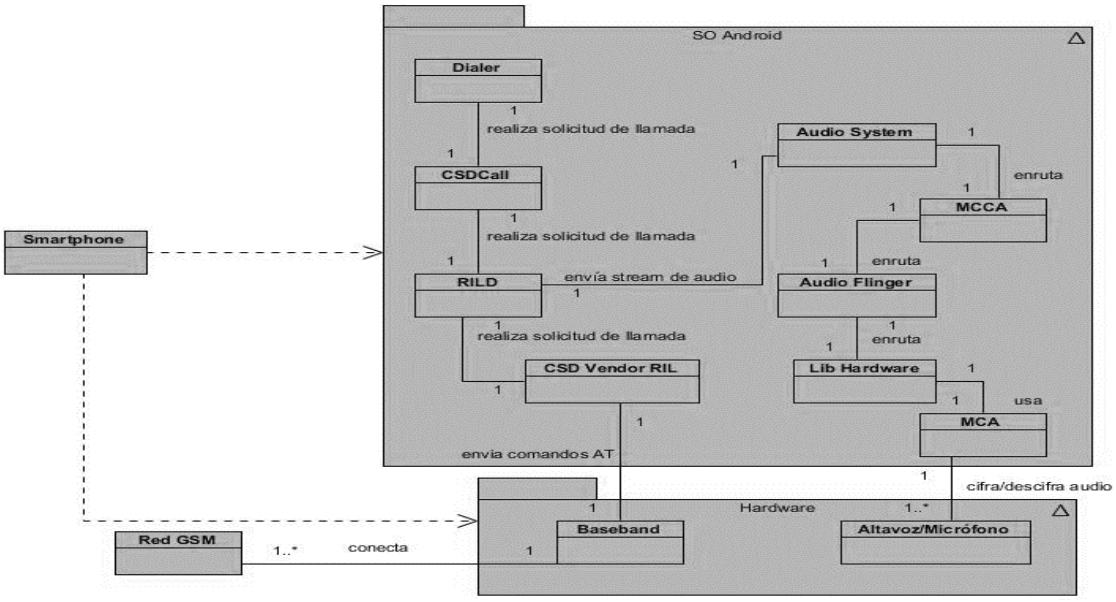


Figura 9. Modelo de Dominio.

Fuente: Elaboración propia.

2.5.1 Descripción de las clases del Modelo de Dominio

Smartphone: dispositivo con SO Android donde se ejecutará el sistema.

Dialer: interfaz que brindará al usuario las funcionalidad de realizar y responder llamadas CSD cifrada.

CSDCall: módulo que proporciona los servicios necesarios para la comunicación del Dialer con el RILD.

RILD: entidad que procesa todas las comunicaciones.

CSD RIL Vendor: librería encargada de gestionar las comunicaciones de una llamada CSD.

Baseband: hardware encargado de la comunicación con las redes externas.

Red GSM: red de comunicaciones.

Audio System: controla los medios de comunicación.

MCCA: módulo encargado de proporcionar los ajustes necesarios para la interacción entre los componentes de los medios de comunicación y los componentes del servicio de telefonía.

Audio Flinger: maneja el flujo de *stream* del altavoz y micrófono.

MCA: módulo encargado de cifrar el flujo de audio que proviene del micrófono y descifrarlo antes de enviarlo a los altavoces.

Altavoz/Micrófono: hardware encargado de reproducir y capturar el audio.

2.6 Requisitos del sistema

Los requisitos de un sistema establecen con detalle las funciones, servicios y restricciones operativas de un sistema. Deben ser precisos y deben definir qué es lo que exactamente se desea implementar. También son conocidos como las entradas en la etapa de diseño del ciclo de desarrollo de un software. En resumen, son una cualidad o característica que el sistema debe poseer (Sommerville, 2005).

2.6.1 Requisitos funcionales

Los requisitos funcionales son declaraciones para definir el funcionamiento interno de un sistema, de la manera que este debe reaccionar a entradas particulares y de cómo debe comportarse en determinadas situaciones. Para su obtención se utilizó la técnica lluvia de ideas (Sommerville, 2005).

RF1. Realizar llamada cifrada.
Descripción: permite que el usuario realice una llamada cifrada.

RF2. Responder llamada cifrada.
Descripción: permite que el usuario responda una llamada cifrada.

RF3. Visualizar agenda de contactos.
Descripción: permite visualizar la agenda de contactos del usuario.

RF4. Gestionar contacto.
Descripción: permite gestionar los contactos de la agenda del usuario.

2.6.2 Requisitos no funcionales

Los requisitos no funcionales son aquellos que están dirigidos a las propiedades emergentes del sistema como son la fiabilidad y seguridad, para tan solo citar algunos ejemplos. Estos requisitos no solo se refieren al software que se desea desarrollar, también existen algunos que restringen el proceso que se debe utilizar para el desarrollo de dicho sistema (Sommerville, 2005).

Requisito de Usabilidad

RNF1. Facilidades de uso al usuario del sistema

- El sistema debe presentar una vista sencilla, descriptiva y fácil de usar, con el objetivo de propiciarles a los usuarios finales un mejor entendimiento con la aplicación.

RNF2. Requerimientos de software

- Para la utilización del sistema se requiere el SO Android en su versión 4.0 (*Ice Cream Sandwich*).

RNF3. Requerimientos de hardware

- Capacidad disponible de 6 Mb en la memoria del *smartphone* para la instalación y ejecución de la aplicación.
- Memoria RAM mínima: 256 Mb.
- *Smartphone* con soporte de redes GSM.
- Tarjetas SIM autorizada por el operador de servicios telefónicos.
- El *baseband* debe soportar trasmisión de datos usando CSD.

RNF4. Servicio de datos activado

- Canal de datos *standard* de 9,6 Kb/s.
- Número del servicio de datos requerido o no (dependiendo del operador).

RNF5. Finalidad

- El objetivo que persigue el sistema es que se puedan realizar una llamada CSD segura entre *smartphones* con SO Android.

Requisito de Confiabilidad

RNF6. Cifrado de la base de datos

- El sistema cifrara la base de datos que almacenará los contactos, propiciando mayor confiabilidad de los datos.

RNF7. Confiabilidad de la información

- La información trasmitida debe coincidir con la información recibida por el otro sistema.

Requisito de Eficiencia

RNF8. Eficiencia

- El tiempo para que el sistema realice el cifrado/descifrado de la información debe ser menos de 0.3 segundos.
- La tasa de transmisión de la información debe ser menor que 9,6 kb/s.

Requisito de Interfaz

RNF9. Interfaz

- La solución debe tener una interfaz sencilla, similar a las interfaces para realizar llamadas telefónicas, comúnmente usadas.

Requisito de Soporte

RNF10. Soporte

- La aplicación será diseñada teniendo en cuenta que pueda integrar nuevos algoritmos de cifrados de forma sencilla sin que interfiera en el resto de los módulos.

Requisito de Documentación

RNF11. Documentación

- Debe existir la documentación del sistema para que sirva de guía para la puesta en práctica de la solución en un *smartphone* autorizado.

2.7 Diagrama de Casos de Uso (CU) del sistema

Los diagramas de Casos de Uso explican gráficamente un conjunto de casos de uso de un sistema, los actores y las relaciones que existen entre ambos componentes. Los casos de uso son las herramientas que permiten mejorar la comprensión de los requisitos, es decir, son las descripciones narrativas de los procesos del dominio (Craig, 1999).

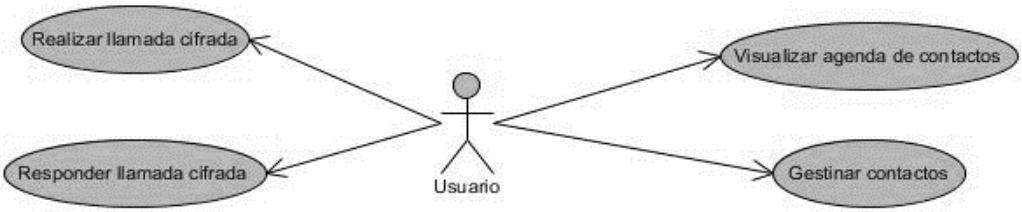


Figura 10. Diagrama de Casos de Uso del Sistema.

Fuente: Elaboración propia.

Actor	Objetivo
Usuario	Realizar llamada cifrada, responder llamada cifrada, visualizar agenda de contactos, gestionar contactos.

Tabla 1. Descripción del actor del sistema.

Fuente: Elaboración propia.

2.7.1 Descripción de Casos de Uso significativo del sistema

En la tabla 2 se muestra la descripción del CU Realizar llamada cifrada y las demás descripciones se encuentran en el Anexo No.1.

Objetivo	Realizar llamada cifrada.
Actores	Usuario.
Resumen	El caso de uso inicia al seleccionar la opción “Llamar”, esta ejecuta una llamada CSD cifrada entre dos *smartphones*, permitiendo mejorar la seguridad de la información que se transmite en dicha llamada. El caso de uso finaliza cuando el usuario selecciona la opción finalizar llamada.
Complejidad	Alta
Prioridad	Alta
Precondiciones	Tarjeta SIM autorizada y servicio CSD activado.
Postcondición	Se realiza una llamada cifrada.

Flujo de eventos		
Flujo básico <Realizar llamada cifrada>		
	Actor	**Sistema**
1.	Introduce el número telefónico que desea llamar.	
2.	Selecciona la opción “Llamar”.	
3.		Verifica si el número telefónico se encuentra en la agenda de contactos del usuario, si no se encuentra el número telefónico se ejecuta la sección 1 del CU Gestionar contactos.
4.		Ejecuta una llamada CSD al número telefónico que introdujo el usuario y muestra una vista de la llamada en curso que permite la siguiente funcionalidad: • Finalizar la llamada en curso, ver Sección 1: Finalizar llamada.
Flujo Alterno al paso 4		
Nº Evento <No se realizó la llamada CSD>		
	Actor	**Sistema**
1.		4.1 Muestra un mensaje de notificación al Usuario informando que no se puede realizar la llamada CSD.
Sección 1: “Finalizar llamada”		
Flujo básico <Finalizar llamada>		
	Actor	**Sistema**
1.	Selecciona la opción “Finalizar llamada”.	
2.		Termina la llamada CSD y muestra un mensaje de notificación al Usuario informando que se finalizó la llamada y se

		muestra la vista de realizar llamada.
Relaciones	**CU Incluidos**	No aplica.
	CU Extendidos	No aplica.
Requisitos no funcionales	RNF1, RNF2, RNF3, RNF4, RNF5, RNF6, RNF7, RNF8, RNF9	
Asuntos pendientes	No aplica.	
Prototipo de Interfaz		

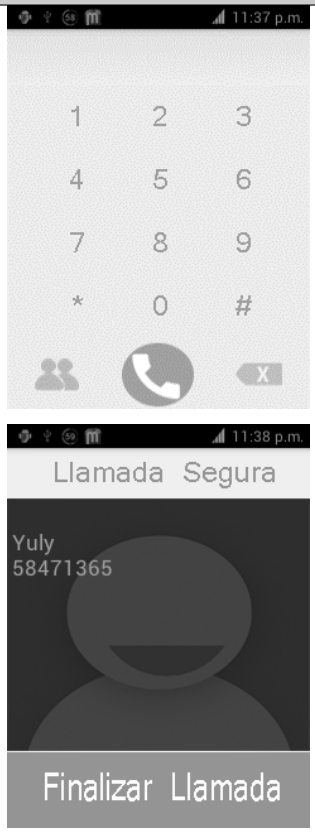


Tabla 2. Descripción del CU realizar llamada cifrada.

Fuente: Elaboración propia.

2.8 Modelo de Diseño

Un Modelo de Diseño describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema. El modelo de diseño es considerado la entrada fundamental de las actividades de implementación (Jacobson, y otros, 2000).

2.8.1 Diagrama de Paquetes

Los diagramas de paquetes son utilizados para reflejar la organización de paquetes y sus componentes. Cuando se usan para representaciones, los diagramas de paquete se emplean para proveer una visualización de estructura física de la herramienta en desarrollo. Los usos más comunes para los diagramas de paquete son para organizar diagramas de casos de uso y diagramas de clase (Sparxsystems, 2015). En la figura 11 se muestra el diagrama de paquetes correspondiente al sistema:

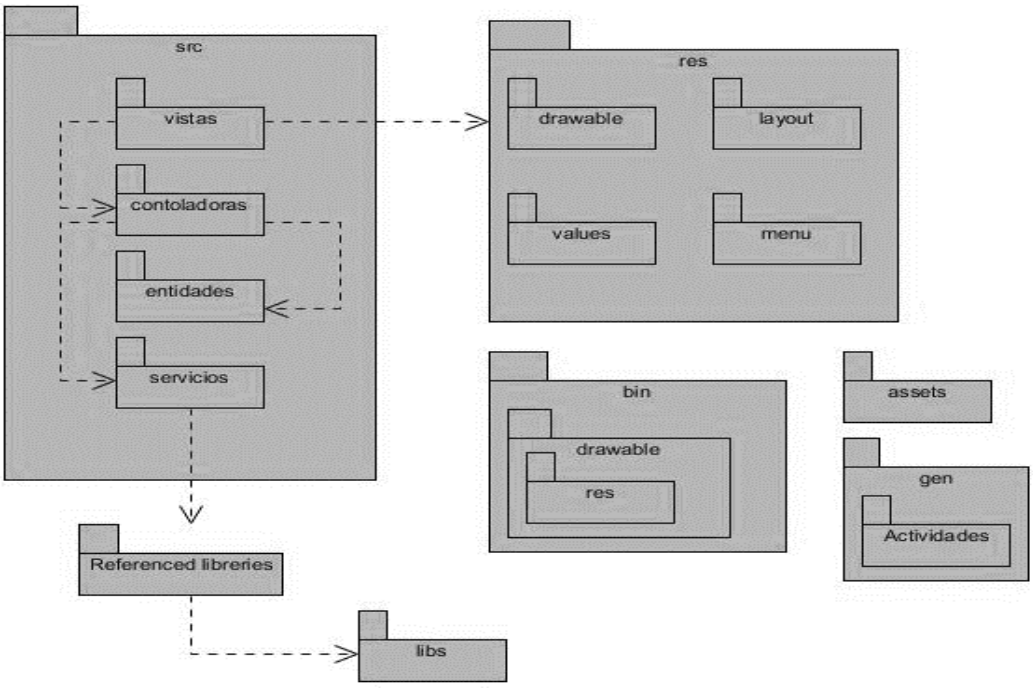


Figura 11. Diagrama de Paquetes.

Fuente: Elaboración propia.

Los proyectos Android eventualmente van construyendo en los archivos .apk todo el código fuente, recursos y elementos necesarios para el funcionamiento del software. Estos archivos .apk son los ejecutables de instalación de las aplicaciones. La estructura de paquetes y archivos de un proyecto de este tipo es la siguiente:

src/: contiene los archivos de clases y actividades. Estos son almacenados en la dirección: src/*your*/*package*/*namespace*/*ActivityName*.Java.

bin/: directorio de salida de la construcción del archivo .apk y otros recursos compilados.

gen/: contiene los archivos de Java generados por el ADT, como el archivo R.java para el control de los identificadores de todos los elementos implicados en la herramienta.

res/: contiene los recursos de la aplicación, como archivos drawable, layouts y los valores string (archivo XML que contiene los textos visualizados en la herramienta).

drawable/: para archivos de imágenes.png, .jpeg o .gif; archivos XML que describen las formas drawable u objetos drawable que contengan múltiples estados.

layout/: archivos XML que son compilados en los layouts de pantalla.

values/: para archivos XML que son compilados en diversos tipos de recursos. A diferencia de otros recursos del directorio res/, los recursos que son escritos a archivos XML en esta carpeta no son referenciados por su nombre, sino que el tipo de elemento XML contenido es controlado a través de la clase R.

libs/: contiene las librerías privadas que son utilizadas internamente en el proyecto.

2.8.2 Diagrama de Secuencia

Los diagramas de secuencia indican los módulos o clases que forman parte de la aplicación y las llamadas que se hacen a cada uno de ellos para realizar una tarea determinada. En dichos diagramas no se deben poner situaciones erróneas, puesto que poner todos los detalles da lugar a un diagrama que no

se entiende o difícil de leer. Para dar solución a ese problema se recomienda acompañar el diagrama con un texto que detalle todas esas situaciones erróneas y particularidades (Vidal, y otros, 2012).

En la figura 12 se muestra el diagrama de secuencia del CU Realizar llamada cifrada, encontrándose los demás diagramas de secuencia en el Anexo No.2:

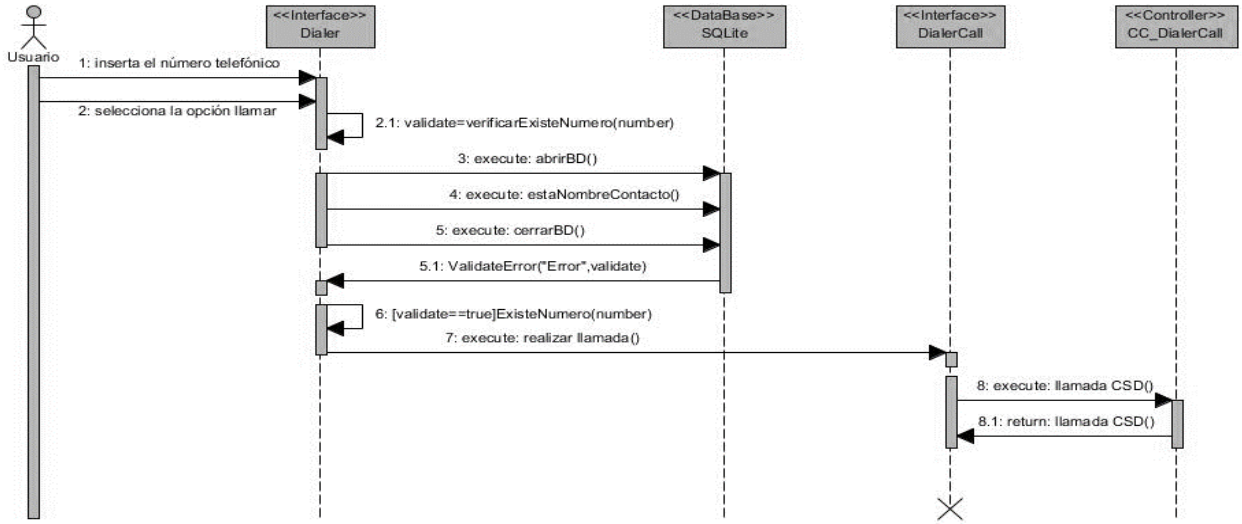


Figura 12. Diagrama de secuencia del CU Realizar llamada cifrada.

Fuente: Elaboración propia.

En el diagrama de secuencia de la figura 12 el actor Usuario inserta el número telefónico que desea llamar y selecciona la opción llamar. Se verifica que el número insertado existe en los contactos del usuario, y si existe se ejecuta el método realizar llamada, el cual mostrará una interfaz con los datos del número marcado por el Usuario si se encuentran en la base de datos. Se solicita a la clase controladora CC_DialerCall una llamada CSD a los servicios del SO Android encargados de este proceso, retornándose la llamada CSD y mostrándose la llamada en curso.

2.9 Patrón arquitectónico

Para lograr la integración del sistema para el trabajo con llamadas de datos al SO Android, se propone el uso de una arquitectura por capas ya que permitirá un mayor acoplamiento a la estructura de los servicios de telefonía del SO y a los componentes de los medios de comunicación. Esta arquitectura permite que

las funciones de una capa utilicen elementos de las capas inferiores de forma transparente.

Algunas de las ventajas de utilizar este patrón son:

- Mejoras en las posibilidades de mantenimiento. Debido a que cada capa es independiente de la otra los cambios o actualizaciones pueden ser realizados sin afectar la aplicación como un todo.
- Flexibilidad, como cada capa puede ser manejada y escalada de forma independiente, la flexibilidad se incrementa.

La descomposición en capas que se propone es:

- **Capa de aplicación**: se integra el Dialer, este brinda las interfaces para la interacción del usuario con el sistema. Esta interfaz tiene las opciones de realizar y responder llamadas CSD cifrada, también gestiona los contactos y las claves de cifrado que usará el usuario en una llamada cifrada.
- **Capa de marco de aplicación**: en esta capa se integran el módulo CSDCall y el MCCA.
- **Capa de librerías**: en esta capa se integra las librería CSD Vendor RIL con la función de gestionar las comunicaciones de una llamada CSD entre RIL y el *baseband*, y el MCA con la función de cifrar el flujo de audio que proviene del micrófono y descifrarlo antes de enviarlo a los altavoces disponibles, durante una llamada telefónica.

2.10 Patrones de diseño

Un patrón de diseño constituye un esquema para refinar subsistemas o componentes. Es una descripción de clases y objetos comunicándose entre sí, adaptada para resolver un problema de diseño general en un contexto particular. Un patrón de diseño identifica: clases, instancias, roles, colaboraciones y la distribución de responsabilidades, además de que ayuda a construir clases y a estructurar sistemas de clases. Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces (Gamma, 2006).

2.11 Patrones GRASP

- **Controlador**: el patrón controlador es utilizado por todas las clases implicadas en la capa de controladora de la lógica del negocio. Poseen la responsabilidad de controlar el flujo de eventos

mediante las actividades correspondientes. Este patrón se representa en clases como CC_DialerCall.java.

- **Creador**: el patrón creador se basa en asignarle a la clase la responsabilidad de crear una instancia de otra clase. Este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos como se muestra en la figura 13:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    setContentView(R.layout.dialer_call);

    Bundle parametros = this.getIntent().getExtras();
    number = parametros.getString("telefono");

    end_call = (Button) findViewById(R.id.btnEnd_Call);
    call_finish = (Button) findViewById(R.id.btnCall_Finish);
    dateContact = (TextView) findViewById(R.id.tvDateContact);

    end_call.setOnClickListener(this);
    timeCall = (Chronometer) findViewById(R.id.chTime);
    timeCall.start();
    try {
        SQLite sqLite = new SQLite(DialerCall.this);
        sqLite.abrir();
        nameContact = sqLite.nombreContacto(number);
        sqLite.cerrar();
        dateContact.setText(" " + nameContact + "\n " + number);
    } catch (Exception e) {
        dateContact.setText(" " + number);
    }
```

Figura 13. Ejemplo del uso del patrón Creador en la clase DialerCall.java.

Fuente: Elaboración propia.

- **Experto**: el patrón experto plantea asignar una responsabilidad al experto en información, es decir, la clase que tiene los datos necesarios para cumplir con la responsabilidad. El problema que resuelve el patrón experto está referido al principio más básico mediante el cual las responsabilidades son asignadas en el diseño orientado a objetos. En la figura 14 se muestra el uso de este patrón con el que se pretende que los objetos realicen las acciones relacionadas con la información que poseen.

```
try {
    SQLite sqLite = new SQLite(ContactBook.this);
    sqLite.abrir();
    String[] columnas = sqLite.obtenerDatosDeLaTablaContactos();
    sqLite.cerrar();

    list = new ArrayList<Contacts>();
    for (int i = 0; i < columnas.length; i += 3) {
        Contacts contacts = new Contacts(columnas[i], columnas[i + 1],
                columnas[i + 2]);

        list.add(contacts);
    }
    ContactsInfo contactsInfo = new ContactsInfo(this, list);
    listContacts.setAdapter(contactsInfo);

} catch (Exception e) {
    // TODO: handle exception
}
```

Figura 14. Ejemplo del uso del patrón Experto en la clase ContactBook.java.

Fuente: Elaboración propia.

- **Bajo Acoplamiento**: el patrón bajo acoplamiento consiste en asignar responsabilidades de manera que el acoplamiento se mantenga bajo. El patrón propone el diseño de clases más independientes, lo que reduce el impacto del cambio y facilita la reutilización en otros sistemas. En la aplicación se evidencia este patrón en la clase Contacts.java ya que no tiene dependencias de ninguna otra clase.
- **Alta cohesión**: la cohesión es una medida de la fuerza con la que se relacionan las responsabilidades de un elemento, un elemento con responsabilidades altamente relacionadas y que no hace una gran cantidad de trabajo tiene alta cohesión. El patrón Alta Cohesión se ve reflejado en la relación existente entre las clases controladoras y los servicios, ya que estas se auxilian entre sí para ejecutar una determinada funcionalidad, disminuyendo las responsabilidades que tendrían los elementos de estas clases si no existiera esta dependencia.

2.12 Patrones GOF

- **Adaptador**: se emplea para generar los elementos de los componentes visuales de la lista de contactos, que requieren de un adaptador para crear los grupos, que se muestran al usuario. Este patrón se utiliza en la clase ContactBook.java.

2.13 Modelo de Datos

El modelo de datos responde a una serie de preguntas específicas e importantes para cualquier aplicación del procesamiento de datos. Como por ejemplo: ¿Cuáles son los objetos de datos primarios que va a procesar el sistema?, ¿Cuál es la composición de cada objeto de datos y que atributos describe el objeto?, ¿Dónde residen actualmente los objetos?, y por último ¿Cuál es la relación entre los objetos y los procesos que los transforman? Para dar respuesta a esas preguntas, el modelo de datos se apoya en el diagrama Entidad-Relación, el cual define todos los datos que se introducen, se almacenan, se transforman y se producen dentro de una aplicación. En la figura 15 se representa el modelo de datos del sistema para el trabajo con llamadas de datos:

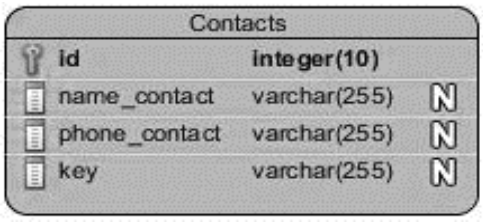


Figura 15. Modelo de Datos del sistema para el trabajo con llamadas de datos.

Fuente: Elaboración propia.

2.13.1 Descripción de la tabla *Contacts* del Modelo de Datos

Nombre: *Contacts*		
Descripción: almacena los datos de los contactos del usuario.		
Atributo	**Tipo**	**Descripción**
id	integer(10)	Identificador auto-incremental.
name_contact	varchar(250)	Nombre del contacto.
phone_contact	varchar(250)	Teléfono del contacto.
key	varchar(250)	Llave para el cifrado.

Tabla 3. Descripción de la tabla Contacts del Modelo de Datos.

Fuente: Elaboración propia.

2.14 Diagrama de Despliegue

El diagrama de despliegue modela la topología del hardware sobre la que se ejecuta un sistema, mostrándose la configuración de los nodos que participan en la ejecución de los componentes que residen en ellos. Además, representa el despliegue físico de un componente. El sistema será desplegado en un *smartphone* con SO Android, comunicándose con la red GSM a través del protocolo CSD para la realización de llamadas CSD cifradas. En la figura 16 se muestra el diagrama de despliegue para el sistema:

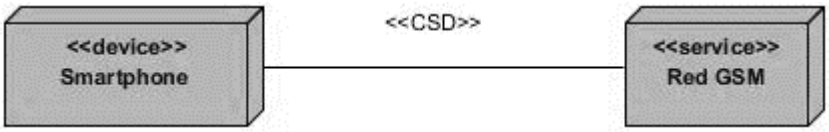


Figura 16. Diagrama de Despliegue.

Fuente: Elaboración propia.

2.15 Conclusiones parciales

En este capítulo se completó el diseño del sistema para aumentar de seguridad en las llamadas telefónicas entre los *smartphones* con SO Android. Se definieron los requisitos funcionales y no funcionales, permitiéndose identificar las funcionalidades con las que contará el sistema, las cuales darán respuesta a las necesidades del problema. Se elaboró el patrón arquitectónico basado en capas que permite un mejor acoplamiento del sistema diseñado con la arquitectura que presenta el SO Android. Se seleccionaron los patrones GRASP y GOF como una buena práctica documentada, o solución, que se ha aplicado con éxito en ambientes múltiples con el fin de tener una guía a la hora de la implementación de la aplicación.

Capítulo 3. Implementación y Prueba

3.1 Introducción

En el presente capítulo se muestra el modelo de implementación del sistema para elevar la seguridad en el proceso de realización de llamadas telefónicas entre *smartphones* con SO Android basándose en el modelo de diseño realizado con anterioridad. Se definen los estándares de codificación utilizados para el desarrollo del sistema y se ejecutan las pruebas de software para verificar el correcto funcionamiento de la aplicación.

3.2 Modelo de Implementación

El Modelo de Implementación es un conjunto de componentes y subsistemas que constituyen la estructura física de la implementación del sistema. El modelo de implementación describe como se organizan los componentes de acuerdo al o los lenguajes de programación utilizados. Tiene como objetivo describir las relaciones existentes entre los diferentes componentes, basándose en las especificaciones de diseño (Jacobson, y otros, 2000).

3.2.1 Diagrama de componentes

Un diagrama de componentes no es más que la representación del sistema en componentes físicos y las dependencias entre ellos, los cuales están compuestos por interfaces, componentes y relaciones. Un componente de software representa una parte de un sistema modular, desplegable y reemplazable que se encuentra en la computadora. Su principal punto es el potencial que tienen para volver a ser utilizados (Jacobson, y otros, 2000).

En la figura 17 se mostrará el diagrama de componentes del sistema para una mayor comprensión del mismo:

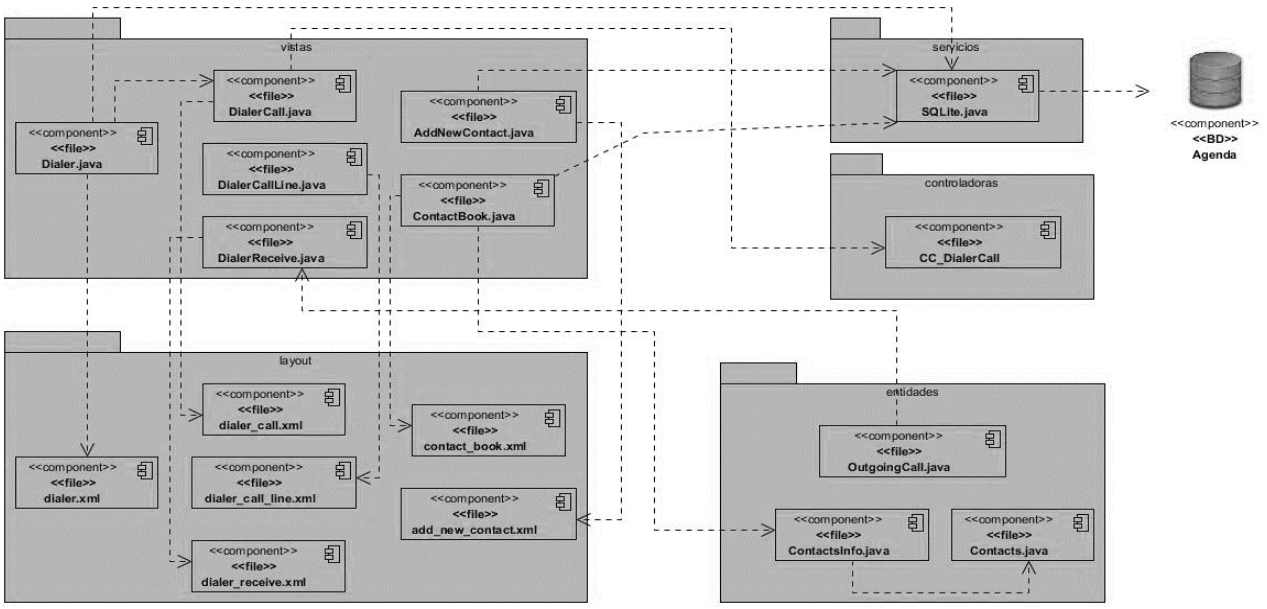


Figura 17. Diagrama de componentes del Sistema.

Fuente: Elaboración propia.

3.3 Código Fuente

El código fuente de un software es un conjunto de líneas de texto que son las instrucciones que debe seguir la máquina para ejecutar dicho software. Dicho código está escrito por un programador en un lenguaje de programación específico, pero para que la máquina lo entienda primero tiene que ser traducido a lenguaje de máquina mediante un compilador.

3.3.1 Estándares de codificación

Un estándar de codificación comprende todos los aspectos de generación de código dentro de un proyecto. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. La confección de estos estándares debe ser definida al comienzo de la implementación para garantizar que todos los programadores trabajen de manera coordinada. Estos estándares deben cumplir con dos principios: legibilidad y mantenibilidad, lo cual repercute en lo bien que el programador entienda el código y la facilidad

con que el sistema puede modificarse, depurar errores o mejorar el rendimiento (Microsoftdevelopernetwork, 2003).

Para el desarrollo del sistema se definieron los siguientes estándares de codificación:

- **Número de declaraciones por línea**: se debe declarar cada variable en una línea distinta, de esta forma cada variable se puede comentar por separado.

  Ejemplo:

```
private Button call_finish;
private Button end_call;
private TextView dateContact;
private String nameContact;
private String number;
private MediaPlayer media;
private int pos = 0;
```

- **Espacio en blanco**: se debe usar una línea en blanco entre: métodos, variables locales de un método y la primera sentencia, entre diferentes secciones lógicas dentro de un fichero (más legibilidad).

  Ejemplo:

```
@Override
public View getView(int posicion, View convertView, ViewGroup parent) {
    Contacts contacts = listContacts.get(posicion);
    View v = convertView;

    if (v == null) {
        LayoutInflater inflater = LayoutInflater.from(mContext);
        v = inflater.inflate(R.layout.contact_list_item, null);
    }

    TextView nombreContact = (TextView) v.findViewById(R.id.tvNombreContacto);
    TextView numeroContact = (TextView) v.findViewById(R.id.tvNumeroContacto);

    nombreContact.setText(contacts.getContactsName());
    numeroContact.setText(contacts.getContactsNumber());

    return v;
}
```

- **Asignación de nombres**: cada tipo de elemento debe nombrarse con una serie de reglas determinadas.

  Clases e interfaces: la inicial en mayúscula ya sea simple o compuesta su nombre.

Ejemplo:

```
public class DialerCall extends Activity implements OnClickListener,
        OnCompletionListener {
```

Métodos: la primera letra de la primera palabra en minúsculas, el resto de las palabras empiezan por mayúsculas.

Ejemplo:

```
public String nameContact(String numberContact) {
```

Variables: deben comenzar por minúscula. No se utilizará en ningún caso el carácter "_".

Ejemplo:

```
String nameContact;
String[] columnas = new String[] { ID_FILA, NAME, PHONE };
```

3.4 Pruebas del software

Las pruebas tienen como principal objetivo detectar los errores del sistema para obtener un software de mayor calidad. Son aplicadas durante todo el ciclo de desarrollo y en diferentes niveles: pruebas de desarrollador, independientes, de unidad, integración, de sistema y aceptación. Durante el desarrollo del proceso de pruebas del sistema se aplicarán las pruebas unitarias y pruebas de aceptación (Juristo, y otros, 2004).

3.4.1 Pruebas unitarias

Las pruebas unitarias son las encargadas de detectar errores en los datos, y en la lógica. Estas aseguran que un único componente de la aplicación produzca una salida correcta para una determinada entrada.

Las pruebas unitarias se realizan porque (Torío, 2004):

- Aseguran la calidad del código entregado, pero no aseguran detectar todos los errores, por tanto las pruebas de integración y aceptación siguen siendo necesarias.
- Permiten encontrar errores en los inicios del desarrollo.

- Facilitan que el programador cambie el código para mejorar su estructura (refactorización), puesto que permiten hacer pruebas sobre los cambios y asegurarse de que no han introducido errores.
- Las pruebas funcionales se hacen más sencillas pues la mayoría de los aspectos individuales de cada unidad ya están probados. De este modo, las pruebas funcionales se centran solo en verificar la correcta cooperación de las distintas unidades y los funcionamientos generales del programa.

3.4.2 Resultados de las pruebas unitarias

Las pruebas unitarias se desarrollan utilizando la extensión de Android JUnit, que es una parte integral del SDK de Android. Permitiendo probar los componentes específicos mediante clases de casos de pruebas. Estas clases proporcionan métodos auxiliares para comprobar el correcto funcionamiento del sistema. En las figuras 18, 19, 20 y 21 se mostrarán fragmentos de códigos y los resultados de las pruebas aplicadas:

```
import junit.framework.TestCase;

public class TestAccesoADatos extends TestCase {
    public void testinsertarContactos() {
        CC_AccessDataBase accessDataBase = new CC_AccessDataBase();
        accessDataBase.insertarContactos("Tito", "53632331", "sdufys");
    }

    public void testobtenerContactos() {
        CC_AccessDataBase accessDataBase = new CC_AccessDataBase();
        accessDataBase.obtenerContactos();
    }

    public void testobtenerNombreContacto() {
        CC_AccessDataBase accessDataBase = new CC_AccessDataBase();
        accessDataBase.obtenerNombreContacto("53632331");
    }

    public void testobtenerLlaveContacto() {
        CC_AccessDataBase accessDataBase = new CC_AccessDataBase();
        accessDataBase.obtenerLlaveContacto("53632331");
    }

}
```

Figura 18. Fragmento de código de prueba de la clase CC_AccessDataBase.

Fuente: Elaboración propia.

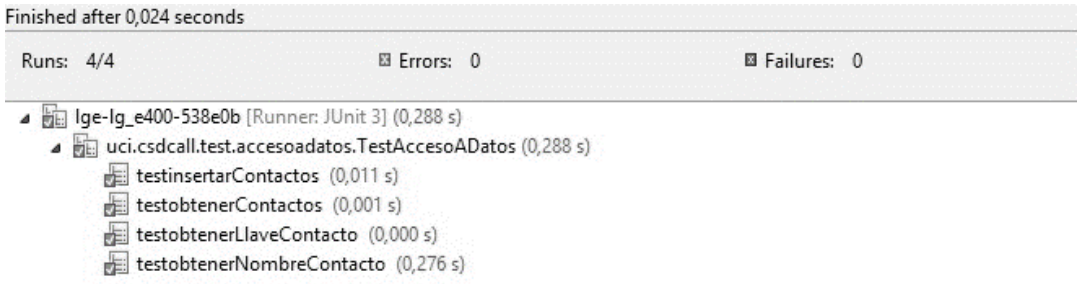


Figura 19. Resultado de la prueba realizada a la clase CC_AccessDataBase.

Fuente: Elaboración propia.

```
public class TestCifradoDeAudio extends TestCase {

    public void testcifrar() {
        Cifrador cifrador = new Cifrador();
        cifrador.cifrar("asgadghaasass");
    }

    public void testdescifrar() {
        Cifrador cifrador = new Cifrador();
        cifrador.descifrar("asgadghaasass");
    }

}
```

Figura 20. Fragmento de código de prueba de la clase Cifrador.

Fuente: Elaboración propia.

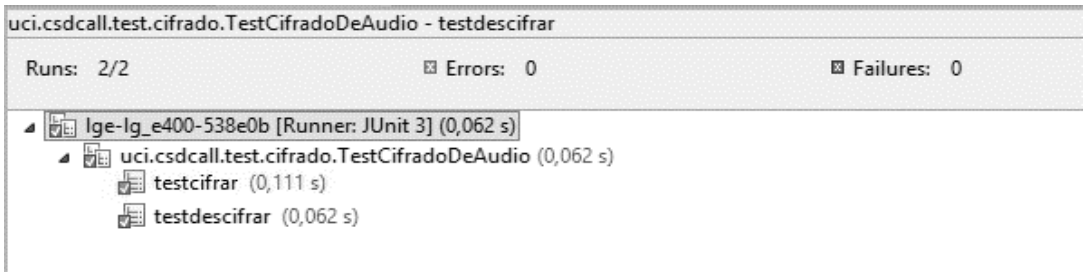


Figura 21.Fragmento de código de prueba de la clase Cifrador.

Fuente: Elaboración propia.

3.4.3 Pruebas de aceptación

Las pruebas de aceptación se elaboran a lo largo de la iteración, en paralelo con el desarrollo del sistema, y adaptándose a los cambios que el sistema sufre. La prueba de software, es un elemento crítico para la garantía de la calidad y representa una revisión final de las

especificaciones del diseño y de la codificación. La prueba está enfocada principalmente en la evaluación y determinación de la calidad del producto. La técnica seleccionada es la prueba de caja negra, la cual se centra en los requisitos funcionales de la aplicación, sin entrar en detalles del funcionamiento interno de la misma. Estas pruebas se desarrollan sobre la interfaz visual del software y se utiliza para detectar errores en la interfaz, funciones incorrectas, errores de salida y problemas con el acceso a datos (Van Wyk, y otros, 2013).

Para confeccionar los casos de prueba de caja negra se utilizó la técnica de Partición de Equivalencia, esta divide el campo de entrada de un programa en variables de equivalencia, las cuales representan un conjunto de datos válidos e inválidos. Las variables válidas son las entradas correctas que acepta el sistema y las inválidas son las entradas erróneas; además de que existen valores no relevantes a los que no es necesario asignarle un valor real de dato. El propósito principal de esta técnica es descubrir y corregir la mayor cantidad de errores antes de que el software llegue a manos del cliente (Pressman, 2001).

3.4.4 Diseño de Casos de Prueba

En la tabla 4 y 5 se muestra la sección a probar y los casos de prueba del CU Realizar llamada, las demás descripciones se encuentran en el Anexo No.3.

Sección a probar CU Realizar llamada cifrada

Escenario de la sección	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Insertar número de teléfono.	Se inserta un número telefónico seleccionando los números que lo componen.	El sistema mostrará en un campo los números que se seleccionen.	1. En la interfaz principal se insertan los números. 2. El sistema muestra los números insertados en pantalla.
EC 1.2 Realizar llamada.	Se selecciona la opción llamar.	El sistema realiza una llamada CSD al número telefónico introducido por el usuario y muestra una	1. Se selecciona la opción “Llamar”. 2. El sistema mostrará una vista de la llamada

		vista de la llamada en curso.	en curso.

Tabla 4. Diseño de casos de prueba del CU Realizar llamada cifrada.

Fuente: Elaboración propia.

Casos de prueba del CU Realizar llamada cifrada

Escenario	Clases válidas	Clases inválidas	Resultado esperado	Resultado de la prueba
EC 1.1	El usuario inserta los números.		Se muestran los números insertados por el usuario.	Se muestran los números que el usuario ha insertado.
EC 1.2	El usuario selecciona la opción llamar.		Se muestra una vista con la llamada en curso.	Se muestra una vista con la llamada en curso, además se muestra el nombre y el número telefónico del usuario llamado.

Tabla 5. Casos de prueba del CU Realizar llamada cifrada.

Fuente: Elaboración propia.

3.5 Resultado de pruebas

Se realizaron tres iteraciones de pruebas, detectándose de forma general 14 no conformidades, en la figura 22 se muestra el resultado:

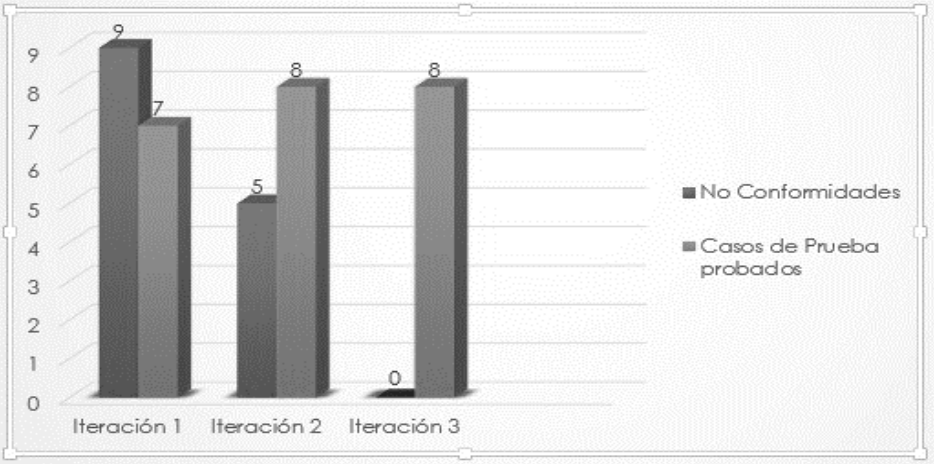


Figura 22. No conformidades detectadas en cada iteración de prueba.

Fuente: Elaboración propia.

Iteración 1. Se probaron 7 casos de prueba y se detectaron 9 No Conformidades.
Iteración 2. Se probaron 8 casos de prueba y se detectaron 5 No Conformidades.
Iteración 3. Se probaron 8 casos de prueba y se detectaron 0 No Conformidades.

3.6 Conclusiones parciales

Durante el desarrollo del capítulo se realizó el modelo de implementación del sistema con el objetivo de estructurar dicho modelo en términos de subsistemas de implementación y mostrar las relaciones entre sus elementos mediante la elaboración del diagrama de componentes. Se reflejó el uso de los estándares de codificación para mantener una concordancia y uniformidad en el código durante la implementación del sistema. Las pruebas unitarias y de aceptación se realizaron para validar el correcto funcionamiento del sistema, haciendo uso del método de caja negra basado en la técnica de partición de equivalencia, y obteniendo resultados satisfactorios de dichas pruebas.

Conclusiones

Después de culminada la investigación se puede afirmar que se le dio cumplimiento al objetivo general planteado, llegando a las siguientes conclusiones:

- El estudio de las redes GSM y del proceso para realizar llamadas telefónicas en el SO Android permitió identificar las vulnerabilidades existentes, posibilitando realizar un mecanismo dentro del SO para reducir dichas vulnerabilidades.
- El uso de CSD permitió el intercambio de información en tiempo real, lográndose reducir la perdida de información en las redes GSM.
- Con el desarrollo del sistema para el trabajo con llamadas de datos se elevó la seguridad en el proceso de realización de llamadas telefónicas entre *smartphones*, reduciendo la posibilidad de escuchar la información de las llamadas interceptadas.
- Las pruebas realizadas arrojaron resultados satisfactorios, garantizando así el cumplimiento de los requisitos planteados.

Recomendaciones

- Se recomienda el despliegue del sistema en un dispositivo real que cumpla los requerimientos necesarios.
- Se recomienda evaluar la integración de un protocolo para el intercambio de la llave simétrica entre el emisor y el receptor basado el cifrado asimétrico.

Bibliográficas Consultadas

Álvarez, Sara. 2007. Características de un sistema gestor de base de datos relacional. [En línea] 2007. [Citado el: 28 de Marzo de 2015.] http://www.desarrolloweb.com/articulos/12-reglas-sgbdr.html.

Androideity. 2013. Arquitectura de Android. [En línea] 2013. [Citado el: 15 de Febrero de 2015.]

Androidpit. 2012. Android para Principiantes - ¿Qué quiere decir SDK de Android? [En línea] 2012. [Citado el: 10 de Febrero de 2015.]

Arias, Juan. 2013. Brasil convoca al embajador de EE UU tras nuevas acusaciones de espionaje. [En línea] 2013. [Citado el: 19 de Enero de 2015.] http://internacional.elpais.com/internacional/2013/09/02/actualidad/1378133902_556483.html.

Aroche, Stephanie Falla. 2015. La historia de Google. [En línea] 2015. [Citado el: 10 de Febrero de 2015.] http://www.maestrosdelweb.com/googlehis/.

Boxcryptor. 2015. Cifrado AES y RSA para la nube. [En línea] 2015. [Citado el: 12 de Febrero de 2015.] https://www.boxcryptor.com/es/cifrado.

Burns, I., Gabert, K. y Zheng, J. 2015. *End-to-End Encrypting Android Phone Calls.* 2015.

Cellcrypt. 2014. Cellcrypt Mobile para Android. [En línea] 2014. [Citado el: 11 de Diciembre de 2014.] www.cellcrypt.com.

Craig, Larman. 1999. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* s.l. : Pretince Hall, Hispanoamérica México, 1999.

Cruz, Álvaro Pachón de la. 2006. *Evolución de los sistemas móviles celulares GSM.* s.l. : Sistemas y Telemática, 2006.

Cubasi. 2014. EEUU usa aviones para espionaje telefónico. [En línea] 2014. [Citado el: 20 de Enero de 2015.] http://cubasi.cu/cubasi-noticias-cuba-mundo-ultima-hora/item/33618-eeuu-usa-aviones-para-espionaje-telefonico#addcomments.

Definiciónabc. 2015. Definición de Telefonía celular » Concepto en Definición ABC. [En línea] 2015. [Citado el: 12 de Febrero de 2015.] http://www.definicionabc.com/tecnologia/telefonia-celular.php.

Developerandroid. 2015. ADT Plugin Release Notes. [En línea] 2015. [Citado el: 6 de Febrero de 2015.] http://developer.android.com/tools/sdk/eclipse-adt.html.

—. 2015. Android SDK. [En línea] 2015. [Citado el: 22 de Enero de 2015.] http://developer.android.com/sdk/index.html.

Docsetools. 2015. Entorno de Desarrollo Integrado. [En línea] 2015. [Citado el: 14 de Febrero de 2015.] http://campodocs.com/articulos-para-saber-mas/article_56561.html.

Eclipse. 2015. Eclipse desktop & web IDEs. [En línea] 2015. [Citado el: 6 de Febrero de 2015.] https://eclipse.org/ide/.

—. 2010. OpenUP. [En línea] 2010. [Citado el: 12 de Febrero de 2015.] http://epf.eclipse.org/wikis/openup/.

Eddy, Max. 2014. RedPhone (for Android) Review & Rating. [En línea] 2014. [Citado el: 19 de Enero de 2015.] http://www.pcmag.com/article2/0,2817,2415410,00.asp.

Eleconomista. 2014. Destapan espionaje de EU a Rousseff y Peña Nieto. [En línea] 2014. [Citado el: 19 de Enero de 2015.] http://eleconomista.com.mx/comment/reply/461744#comment-form.

Fraga, Miguel Angel. 2015. El sistema de claves públicas RSA - DigiSign Data Security. [En línea] 2015. [Citado el: 4 de Febrero de 2015.] http://digisign.50megs.com/info040.html.

Frozzt. 2015. Definicion de Java. [En línea] 2015. [Citado el: 5 de Febrero de 2015.] http://www.buenastareas.com/ensayos/Definicion-y-Significado-De-Java/1395551.html.

Gamma, Erich. 2006. *Patrones de diseño: elementos de software orientado a objetos reutilizables, ed. P.* 2006. Educación.

Gironés, Jesús Tomás. 2012. *El gran libro de Android.* Marcombo : s.n., 2012. págs. 26-30.

Gsma. 2015. Brief History of GSM & the GSMA. [En línea] 2015. [Citado el: 5 de Mayo de 2015.] http://www.gsma.com/aboutus/history.

Gutiérrez, Pedro. 2013. Tipos de criptografía: simétrica, asimétrica e hibrida. [En línea] 2013. [Citado el: 12 de Febrero de 2015.] http://www.genbetadev.com/seguridad-informatica/tipos-de-criptografia-simetrica-asimetrica-e-hibrida.

Halonen, Timo, Romero, Javier y Melero, Juan. 2004. *GSM, GPRS and EDGE Performance: Evolution Towards 3G/UMTS.* Segunda Edición. 2004. págs. 45-52. 0-470-86694-2.

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2000. *El proceso unificado de desarrollo de software.* s.l. : Addison Wesley Reading, 2000. págs. 255-259. Vol. 7.

Juristo, N., Moreno Ana, M. y Vegas, Sira. 2004. *Técnicas de evaluación de Software.* 2004.

Lanzillotta, Analía. 2015. Definición de Lenguaje de programación. [En línea] 2015. [Citado el: 5 de Febrero de 2015.]

Lara Almarales, Bárbara y Terrero Sencial, Robin. 2013. *Desarrollo de un portal web para la gestión de servicios en el Consejo Nacional de Patrimonio Cultural.* Universidad de las Ciencias Informáticas. La Habana : s.n., 2013. Tesis.

López, Mireya Esperanza. 2014. Definición de red. [En línea] 2014. [Citado el: 21 de Febrero de 2015.] http://www.buenastareas.com/ensayos/Definici%C3%B3n-De-Redes/62298184.html.

Martínez, Evelio. 2001. *La evolución de la telefonía móvil.* 2001.

Mastermagazine. 2015. Definición de CASE - Significado y definición de CASE. [En línea] 2015. [Citado el: 27 de Enero de 2015.] http://www.mastermagazine.info/termino/4182.php.

—. 2015. Definición de UML - Significado y definición de UML. [En línea] 2015. [Citado el: 12 de Marzo de 2015.] http://www.mastermagazine.info/termino/7006.php.

Meyers, Justin. 2011. A Visual History of The Mobile Phone. [En línea] 2011. [Citado el: 22 de Enero de 2015.] http://www.businessinsider.com/complete-visual-history-of-cell-phones-2011-5?op=1.

Michelone, Manuel López. 2013. La historia de Android. [En línea] 2013. [Citado el: 22 de Enero de 2015.] http://www.unocero.com/2013/09/23/la-historia-de-android/.

Microsoftdevelopernetwork. 2005. Cifrado de datos. [En línea] 2005. [Citado el: 12 de Enero de 2015.] http://msdn.microsoft.com/es-es/library/cc785633(v=ws.10).aspx.

—. 2003. Revisiones de código y estándares de codificación. [En línea] 2003. [Citado el: 14 de Marzo de 2015.] https://msdn.microsoft.com/es-es/library/aa291591%20(v=vs.71).aspx.

Monte, Yusleydi Fernández del. 2013. *Metodología para desarrollar la distribución cubana de GNU/Linux Nova.* Universidad de las Ciencias Informáticas. La Habana : s.n., 2013.

Openhandsetalliance. 2007. Open Handset Alliance. [En línea] 2007. [Citado el: 22 de Enero de 2015.] http://www.openhandsetalliance.com/oha_faq.html.

Pérez, Enrique Herrera. 2003. *Tecnologías y redes de transmisión de datos.* 2003. págs. 197-200.

PicandoJava. 2015. IDE Eclipse. [En línea] 2015. [Citado el: 12 de Febrero de 2015.] https://sites.google.com/site/picandojavacom/mundo-java/ide-eclipse.

Pressman, Roger. 2001. *Ingeniería del Software: una tecnología estratificada.* Quinta Edición. 2001. pág. 201.

—. 2001. *Ingeniería del Software: una tecnología estratificada.* Quinta Edición. 2001. págs. 433-434.

Preukschat, Alex. 2014. Criptografía asimétrica: Sistemas de Cifra con Clave Pública - Bitcoin. [En línea] 2014. [Citado el: 20 de Febrero de 2015.] http://www.oroyfinanzas.com/2014/01/criptografia-asimetrica-sistemas-cifra-clave-publica-bitcoin/.

Ronquillo, Tony José Lamilla. 2006. *Migración de GSM a UMTS.* Universidad de San Carlos de Guatemala. Guatemala : s.n., 2006. Tesis.

Ruiz, Marcelo Hernán. 2003. *Programación C.* MP Ediciones S.A. Buenos Aires : s.n., 2003. págs. 16-17.

Sánchez, Juan Andrés Wevar. 2005. *Análisis y Estudio de Redes GPRS.* Universidad Austral de Chile. Chile : s.n., 2005. pág. 75, Tesis.

Santo, Karito. 2012. Definición de comunicación. [En línea] 2012. [Citado el: 21 de Febrero de 2015.] http://www.buenastareas.com/ensayos/La-Comunicacion-Definicion/4463837.html.

Shell, Scott R., Sherman, Roman y Shen, Alan W. 2004. *Radio interface layer in a cell phone with a set of APIs having a hardware-independent proxy layer and a hardware-specific driver layer.* s.l. : Google Patents, 2004.

Sommerville, Ian. 2005. *Ingeniería del Software.* Séptima edición. Madrid : s.n., 2005.

Sparxsystems. 2015. Diagrama de Paquete UML 2. [En línea] 2015. [Citado el: 1 de Abril de 2015.] http://www.sparxsystems.com.ar/resources/tutorial/uml2_packagediagram.html.

Stallings, William. 2004. Fundamentos de seguridad en redes: aplicaciones y estándares. [En línea] 2004. [Citado el: 4 de Febrero de 2015.] https://books.google.com.cu/books?id=cjsHVSwbHwoC&pg=PA237&lpg=PA237&dq=algoritmo+Diffie-Hellman&source=bl&ots=ZnxKZ-LaNF&sig=sq13Lq4M8U9bakvbzBhpyr1U2i8&hl=es&sa=X&ei=Yw3TVMzbAoSiyASvhILQAQ&ved=0CCkQ6AEwAjgU#v=onepage&q=algoritmo%20Diffie-Hellman&f=false.

Stubing, H., Pfalzgraf, M. y Huss, S.A. 2011. *IEEE Xplore Abstract - A Diffie-Hellman based privacy protocol for Car-to-X communication.* 2011. págs. 1147-1154.

Textoscientíficos. 2005. Conmutación de circuitos. [En línea] 2005. [Citado el: 12 de Marzo de 2015.] http://www.textoscientificos.com/redes/conmutacion/circuitos.

Torío, Julio Mellado. 2004. *Estrategias de pruebas de líneas de producto de sistemas de tiempo real especificados con diagramas de estados jerárquicos.* Telecomunicacion. 2004. Tesis.

Unam. 2012. Fundamentos de Criptografía. [En línea] Facultad de Ingeniería, 2012. [Citado el: 4 de Febrero de 2015.] http://redyseguridad.fi-p.unam.mx/proyectos/criptografia/criptografia/index.php/4-criptografia-simetrica-o-de-clave-secreta/41-introduccion-a-la-criptografia-simetrica/413-principales-algoritmos-simetricos?showall=&start=1.

Van Wyk, Ken, Radosevich, Will y C.C., Michael. 2013. Black Box Security Testing Tools. [En línea] 2013. [Citado el: 26 de Abril de 2015.] https://buildsecurityin.us-cert.gov/articles/tools/black-box-testing/black-box-security-testing-tools.

Vidal, Cristian L., y otros. 2012. *Extensión del Diagrama de Secuencias UML (Lenguaje de Modelado Unificado) para el Modelado Orientado a Aspectos.* 2012. págs. 51-62, Información tecnológica.

Video2brain. 2015. Aprende Android con video2brain. [En línea] 2015. [Citado el: 22 de Enero de 2015.] https://www.video2brain.com/mx/android-os.

Vílchez, Ángel. 2009. Que es Android: Características y Aplicaciones. [En línea] 2009. [Citado el: 22 de Enero de 2015.] http://www.configurarequipos.com/doc1107.html.

Visualparadigm. 2015. Software Design Tools for Agile Teams, with UML, BPMN and More. [En línea] 2015. [Citado el: 11 de Marzo de 2015.] http://www.visual-paradigm.com/.

Warren, Tom. 2014. Google touts 1 billion active Android users per month. [En línea] 2014. [Citado el: 16 de Marzo de 2015.] http://www.theverge.com/2014/6/25/5841924/google-android-users-1-billion-stats.

Anexos

Anexo No.1: Especificación de los casos de uso

CU Responder llamada cifrada

Objetivo	Responder llamada cifrada.	
Actores	Usuario.	
Resumen	El caso de uso inicia cuando el sistema muestra una vista indicando que se recibe una llamada. El Usuario tendrá las opciones de responder la llamada o rechazarla. El caso de uso finaliza cuando el Usuario selecciona la opción rechazar llamada o finalizar llamada.	
Complejidad	Alta	
Prioridad	Alta	
Precondiciones	Tarjeta SIM autorizada y servicio CSD activado.	
Postcondición	Se responde la llamada cifrada.	
Flujo de eventos		
Flujo básico <Responder llamada cifrada>		
	Actor	**Sistema**
1.		Muestra una vista indicando que se recibe una llamada, con las opciones siguientes: • Responder llamada, ver Sección 1: Responder llamada. • Rechazar llamada ver Sección 2: Rechazar llamada.
2.	Selecciona una de las opciones disponibles.	
Sección 1: "Responder llamada"		
Flujo básico <Responder llamada>		
	Actor	**Sistema**
1.	Selecciona la opción "Responder llamada"	
2.		Establece la llamada con el Usuario que la solicita y se muestra una vista de la

		llamada en curso.
3.	Selecciona la opción “Finalizar llamada”.	
4.		Termina la llamada y muestra un mensaje de notificación al Usuario informando que finalizó la llamada.
Sección 2: “Rechazar llamada”		
Flujo básico <Rechazar llamada>		
	Actor	**Sistema**
1.	Selecciona la opción “Rechazar llamada”.	
2.		Rechaza la llamada.

Relaciones	**CU Incluidos**	No aplica.
	CU Extendidos	No aplica.
Requisitos no funcionales	RNF1, RNF2, RNF3, RNF4, RNF5, RNF6, RNF7, RNF8, RNF9	
Asuntos pendientes	No aplica.	

Prototipo de Interfaz

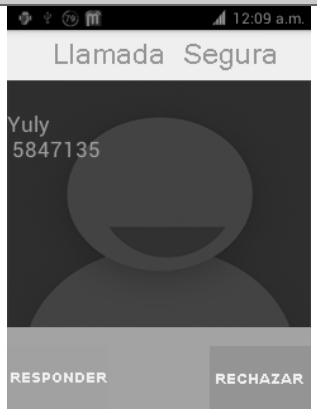


Tabla 6. Descripción del CU responder llamada cifrada.

Fuente: Elaboración propia.

Anexo No.2: Diagramas de Secuencia

CU Responder llamada

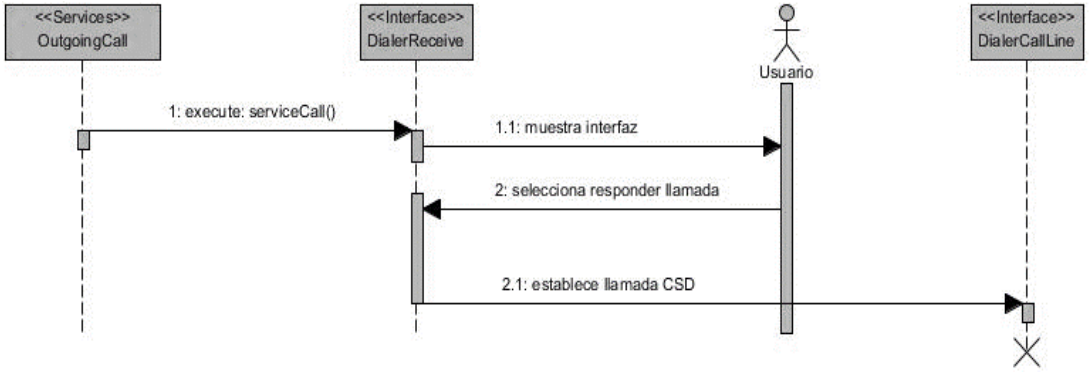


Figura 23. Diagrama de secuencia del CU Responder llamada cifrada.

Fuente: Elaboración propia.

Anexo No.3: Casos de prueba

Sección a probar CU Responder llamada cifrada

Escenario de la sección	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Responder llamada.	Se responderá la llamada entrante.	Se mostrará una vista con la llamada en curso.	1. Se selecciona la opción "Responder Llamada". 2. El sistema muestra una vista con la llamada en curso.
EC 1.2 Rechazar llamada.	Se rechazará la llamada entrante.	El sistema cancelará la llamada entrante.	1. Se selecciona la opción "Rechazar Llamada". 2. El sistema cancela la llamada entrante.

Tabla 7. Diseño de casos de prueba del CU Responder llamada cifrada.

Fuente: Elaboración propia.

Casos de prueba del CU Responder llamada cifrada

Escenario	Clases válidas	Clases inválidas	Resultado esperado	Resultado de la prueba
EC 1.1	El usuario selecciona la opción "Responder Llamada".		Se muestra una vista con la llamada en curso.	Se muestra una vista con la llamada en curso, además se muestra el nombre y el número telefónico del usuario que está llamando.
EC 1.2	El usuario selecciona la opción "Rechazar Llamada".		Se cancela la llamada entrante.	El sistema cancela la llamada entrante.

Tabla 8. Casos de prueba del CU Responder llamada cifrada.

Fuente: Elaboración propia.