UNIVERSIDAD SAN FRANCISCO DE QUITO

Implementación de Seguridad en la Mensajería Celular

Juan Pablo Albuja

Tesis de Grado presentada como requisito para la obtención del título de Ingeniero en Sistemas

Quito, Mayo 2009

UNIVERSIDAD SAN FRANCISCO DE QUITO
Colegio Politécnico

HOJA DE APROBACIÓN DE TESIS

Implementación de Seguridad en la Mensajería Celular

Juan Pablo Albuja

Enrique Vinicio Carrera, DSC ..... (firma)
Director de Tesis

Iván Bernal, PhD ..... (firma)
Miembro del Comité de Tesis

René Játiva, DEA ..... (firma)
Miembro del Comité de Tesis

Fausto Pazmay, MBA ..... (firma)
Director del Departamento de Sistemas

Fernando Romo, MSc ..... (firma)
Decano del Colegio Politécnico

Quito, Mayo 2009

Deseo dedicar esta tesis a mi hijo Juan Diego Albuja, a mi abuelita Sara Palacios, y a mis padres César Albuja y Yomar Riofrío.

Agradecimientos

Agradezco a todos mis profesores y en especial a mi Director de Tesis Enrique Carrera por su incondicional apoyo durante todo el dessarrollo de esta tesis.

Resumen

Uno de los servicios más utilizados que brinda la telefonía celular es el servicio de envío y recepción de mensajes de texto por medio del protocolo SMS, cuyas siglas significan *Short Message Service*. En los últimos años, el número de usuarios y la complejidad de las aplicaciones que hacen uso de este servicio han ido incrementando. Como las aplicaciones son cada vez más complejas y manejan información sensible, ellas requieren medidas de seguridad para la transferencia de información. Lastimosamente este servicio no goza de un estándar de seguridad que brinde los servicios de la recomendación X.800 como son integridad, confidencialidad, autenticación, no negación y control de acceso. Por ello, esta tesis presenta una biblioteca de funciones desarrollada sobre la plataforma J2ME que implementa la protección en la transmisión de los datos sobre SMS basándose en esquemas de seguridad PGP y S/MIME.

Abstract

One of the commonly used services offered by mobile telephony is the sending and receiving text messages via the SMS protocol, whose initials mean *Short Message Service*. In recent years, the number of users and complexity of applications that make use of this service have been increasing. As applications become increasingly complex and handle sensitive information, they require security measures for data transfer. Unfortunately, this service does not have a safety standard that provides services such as X.800 recommendation like integrity, confidentiality, authentication, no denial and access control. Therefore, this thesis presents a library of functions developed on the J2ME platform that implements the protection in the transmission of data on SMS based on data security schemes PGP and S/MIME.

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

La telefonía celular desde su invención en 1947 por la empresa Norteamericana AT&T, ha estado en una constante evolución mejorando el servicio celular añadiendo otros servicios alternativos como complemento, tal es el caso del servicio SMS (*Short Message Service*). SMS es el servicio de envío de mensajes cortos de máximo 160 caractéres entre celulares (9).

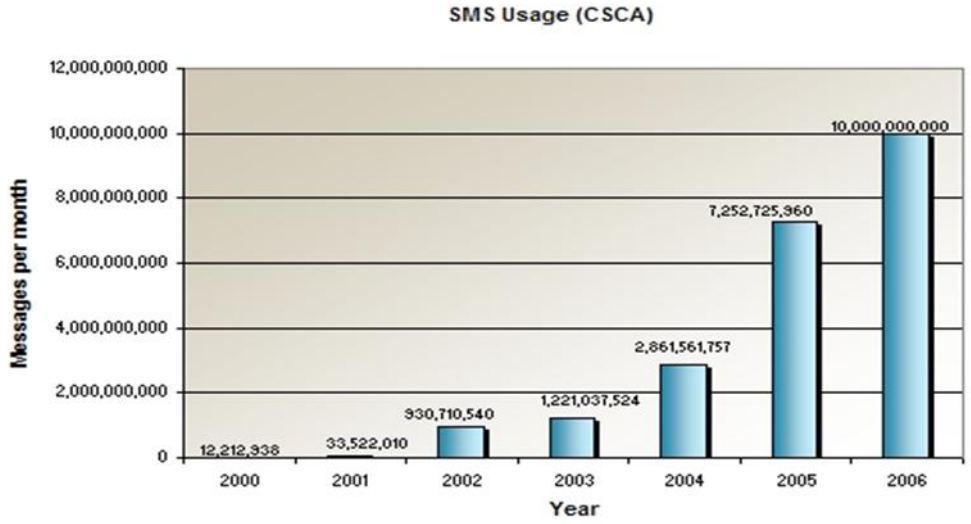

Figura 1.1: Estadísticas de Envío SMS (24)

Este servicio ha tenido gran acogida a nivel mundial porque de acuerdo a las estadísticas estudiadas por el portal Tech Crunchies (21), se enviaron en el 2008 2.3 trillones de mensajes de texto sobre Internet y estimaron que el número de usuarios

de banca móvil en los Estados Unidos en 2012 serán de 40 millones. Potio Research predijo que en el 2013 los ingresos al mercado mundial SMS serán de 224 mil millones de dólares siendo en el 2008 118 mil millones de dólares (12). Además, la empresa consultora Norteamericana Upsolve (24) presentó datos estadísticos del crecimiento de SMS en su país tal como se puede apreciar en la Figura 1.1.

Este impresionante crecimiento del número de clientes SMS también se ha experimentado en nuestro país. Según un artículo de la revista Online Frecuencia, Movistar, segunda operadora celular más grande del Ecuador con 3 millones de usuarios en el 2006, contabiliza sorprendentemente un promedio de 220 millones de mensajes de texto por mes (7), dándo un promedio de 270 SMS por mes y por usuario.

Dentro de lo que comprende el servicio SMS, existen aplicaciones complejas y no complejas. Entre las aplicaciones no complejas existen sorteos televisivos, servicios de horóscopos, servicios de entretenimiento, etc. En estas aplicaciones las interacciones son sencillas entre el dispositivo celular y una aplicación que tiene la capacidad de enviar y recibir mensajes de texto. Entre aplicaciones complejas tenemos banca SMS y M-commerce. Tal es el caso del banco Australiano NetBank (15) que implementa códigos de seguridad para obtener acceso a transacciones bancarias, y la compra de tickets de bus vía SMS en Australia (36). Tenemos que considerar que cada vez las aplicaciones ejecutadas en dispositivos móviles son cada vez más complejas y críticas debido a la importancia de su uso.

Existen riesgos causados por no tomar las suficientes medidas de seguridad en aplicaciones críticas como realizar banca SMS. Uno de los problemas más graves es que el atacante pueda obtener información confidencial como claves de acceso y realizar una suplantación de identidad realizando transferencia de dinero a otras cuentas, o simplemente que el atacante desea obtener información confidencial del cliente como los estados de cuenta.

A pesar del crecimiento significativo de la mensajería celular, no existe un estándar

globalmente aceptado que permita garantizar confidencialidad en la comunicación, integridad de los mensajes, no negabilidad de los participantes y autenticación las partes involucradas. Por ello, la propuesta de esta tesis es crear una biblioteca que implemente seguridad en SMS para cubrir la no existencia de estándares a pesar de haber eminentes riesgos. Esta seguridad se la implementará a nivel de aplicación por medio de la construcción de una biblioteca código abierto modular y eficiente en consumo de energía y en tiempo de operaciones, y que trabaje sobre la plataforma J2ME. Se ha elegido trabajar sobre la plataforma J2ME debido a su madurez, es libre, es soportada por los fabricantes más populares como Sony Ericsson, Motorola, Nokia ; y por su facilidad de integración con diferentes bibliotecas como por ejemplo la biblioteca criptográfica Bouncy Casttle (22).

Cabe recalcar ques esta biblioteca propuesta utilizará un esquema de seguridad similar al utilizado por el protocolo de seguridad de mail PGP (*Pretty Good Privacy*) y S/MIME (*Secure Multipurpose Internet Mail Extensions*).

Capítulo 2

Fundamentos

2.1 Introducción

Las principales bases teóricas que sustentan a esta tesis son la Seguridad Informática y la tecnología SMS. La combinación de estos temas sirven como base para la construcción de un prototipo que aplique la arquitectura de seguridad OSI X.800 al protocolo de mensajería celular SMS.

2.2 Seguridad Informática

Se entiende como seguridad informática el garantizar que los recursos del sistema de información sean utilizados de la manera como se ha decidido desde su creación, y que el acceso y modificación de la información sea concedida sólo a las personas acreditadas de derechos y dentro de los límites de su autorización (30).

2.2.1 Arquitectura de Seguridad

La arquitectura de seguridad OSI está dada por la recomendación X.800. Esta recomendación forma parte de la Unión Internacional de Telecomunicaciones aprobada en Ginebra en 1991 (35). Esta consta de una descripción de los servicios de seguridad que

debe tener un sistema para que sea considerado seguro. Los servicios de seguridad son los servicios que amplían la seguridad en los sistemas de procesamiento y transferencia de datos. Los elementos básicos que debe tener un sistema para ser considerado seguro son (18) :

- Confidencialidad : La información debe ser legible sólo para los usuarios autorizados.
- Integridad : La información recibida debe ser exactamente igual que la información enviada por el emisor. No alteración de los datos durante la transmisión.
- No negación : No puede el usuario negar sus operaciones en el sistema.
- Autenticación : Confirmar la identidad del usuario para permitirle acceso a los recursos del sistema.
- Control de acceso : Es la prevención no autorizada del uso de recursos del sistema. Este servicio controla quién puede tener acceso a los distintos recursos y bajo qué circunstancias se permite el acceso.

2.2.1.1 Confidencialidad

Entendemos como confidencialidad a la no exposición de contenidos o información a los usuarios no autorizados. La criptografía de los datos sirve para evitar ataques pasivos como el monitoreo y análisis de tráfico. Existen dos formas de criptografía conocidas como criptografía simétrica y asimétrica.

Criptografía Simétrica : Se conoce como criptografía simétrica si tanto el emisor como el receptor del mensaje utilizan una misma llave compartida para realizar las operaciones de encriptación y desencriptación. El proceso de desencriptación es exactamente el proceso inverso de encriptación tal como se puede apreciar en la Figura 2.1. Los algortimos más conocidos son AES (*Advanced Encryption Standard*) y 3DES (*Triple Data Encryption Standard*).

Criptografía Asimétrica : Se conoce como criptografía asimétrica cuando el emi-

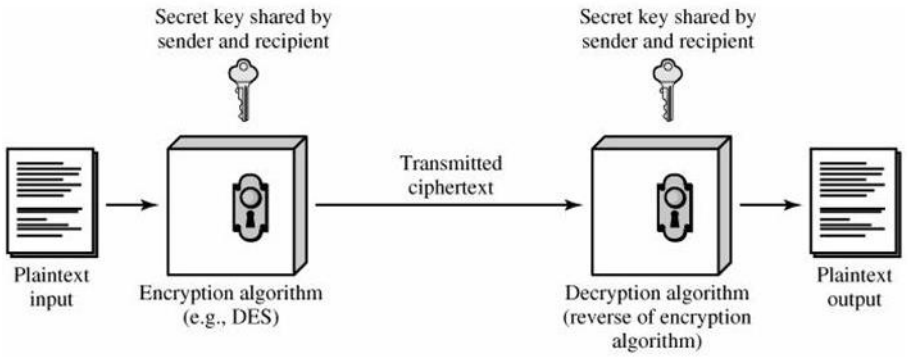

Figura 2.1: Criptografía Simétrica (18)

sor y el receptor utilizan diferentes llaves, dónde una llave es utilizada para la encriptación de los datos y otra diferente es utilizada para desencriptar los datos. Uno de los algoritmos más típicos es el conocido RSA (*Rivest Shamir Adleman*) y se caracteriza por tener la capacidad de encriptar y desencriptar datos utilizando ya sea la llave pública o privada. La Figura 2.2 muestra cómo es el típico proceso de criptografía asimétrica dónde el emisor encripta al mensaje utilizando la llave pública del Receptor (PU_b) y el receptor desencripta el mensaje utilizando su llave privada PR_b.

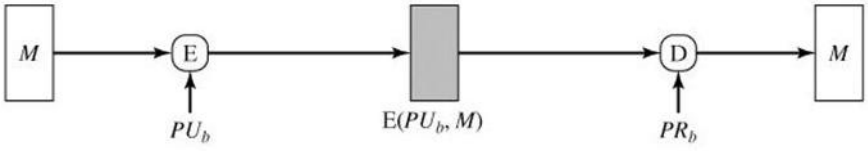

Figura 2.2: Criptografía Asimétrica (18)

2.2.1.2 Integridad

Garantizar la integridad de los datos es sumamente importante ya que si los datos son alterados, el receptor de la información recibirá información errónea o distorsionada que puede afectar gravemente al comportamiento del sistema. Existen técnicas computacionales que pueden realizar la verificación de la integridad de los datos y

que consisten en verificar si realmente la información ha sido alterada o no durante la transmisión. Una de las técnicas más conocidas para garantizar la integridad de los mensajes es el uso de funciones hash. Esta técnica consiste en generar una cadena de bytes obtenida a partir del contenido del mensaje por medio del uso de una función hash conocida y no reversible. Las funciones hash más conocidas son MD5 (*Message Digest 5*) y SHA1 (*Secure Hash Algorithm 1*) debido a su seguridad y desempeño.

Se conoce como firma digital a los bytes obtenidos despúes de aplicar una función hash y encriptados con la llave privada (PR_a) del emisor. La Figura 2.3 muestra el uso de funciones hash en combinación con criptografía de llave pública.

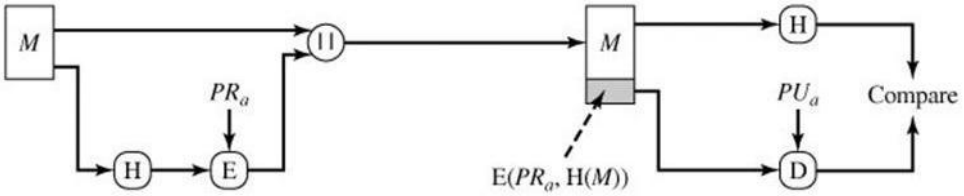

Figura 2.3: Funcionamiento Firma Digital (18)

Los pasos realizados para la obtención de las firmas digitales son :

1. El emisor calcula el código hash H aplicando una función hash conocida.
2. El emisor utiliza su llave privada PRa para enciptar el código hash.
3. El emisor envía el mensaje más el código hash encriptado.
4. El receptor obtiene el hash del mensaje utilizando la misma función hash utilizada por el emisor.
5. El receptor desencripta el código hash recibido, utilizando la llave pública del emisor (PUa), y compara si corresponde al mismo código hash calculado.
6. Si los códigos hash son iguales, el mensaje no ha sido alterado en la transmisión.

2.2.1.3 No Negación

Se refiere a la protección contra la negación de la participación de las partes involucradas en la comunicación. Este servicio prueba que el mensaje haya sido enviado por una específica parte y prueba que el mensaje haya sido recibido por el destino deseado.

Los métodos más conocidos para la no negación en la participación de la comunicación es la encriptación de llave pública y el uso de firmas digitales. El uso de encriptación garantiza no repudiación o no negación de destino debido a que si el mensaje es encriptado con la llave pública de una entidad en el envío, se sabe que sólo el destino tiene la llave privada correspondiente para poder desencriptar el mensaje. Entonces, el emisor tiene la certeza que sólo el propietario de la llave privada será capáz de decifrar el mensaje encriptado. Otro de los métodos más utilizados para la no negación es el uso de firmas digitales debido a que el emisor cada vez que envía un mensaje, lo firma con su llave privada. Entonces si el receptor tiene la llave pública del emisor, puede verificar que el mensaje ha sido firmado por el propietario de la llave privada y comprueba su identidad.

2.2.1.4 Autenticación

La autenticación se refiere a identificar que la entidad que actúa en la comunicación es quien dice ser. Es decir, se asegura que la comunicación es auténtica. Los métodos más conocidos para la autenticación de las partes son criptografía asimétrica, el uso de firmas digitales y el uso de certificados digitales. El uso de firmas digitales y criptografía asimétrica autentica debido a que cada entidad es propietaria de una llave privada y ésta no puede ser vista por otras entidades. Las operaciones realizadas con las llaves privadas sólo pueden ser realizadas por sus propietarios como es el caso de la operación de firma de mensajes y la desencriptación.

Un certificado digital es un documento digital emitido por una Autoridad de Certificación o entidad confiable que garantiza la identidad de las partes que desean comu-

nicarse. Con estos certificados, las entidades pueden reconocer la identidad de la otra, la fecha de creación del certificado, la firma digital que valida el documento y puede ser verificado por una tercera parte en caso de disputas. Existen dos formas de certificación Directa y Arbitrada.

- Directa : La certificación directa es aquella que no requiere de una tercera entidad que certifique o valide la identificación de las partes que participan en la comunicación. Cada una de las entidades tiene su llave privada con la cuál firma digitalmente sus mensajes y su llave pública que es repartida a las otras entidades para que éstas puedan desencriptar los mensajes y verificar la firma de la entidad emisora. Un ejemplo de este esquema es la Certificación RSA.
- Arbitrada : La Autoridad de Certificación (CA) es una entidad que participa activamente en la comunicación de las partes. Ésta valida la veracidad de los certificados de las partes que desean comunicarse y se encarga de la generación y repartición de los certificados digitales que incluye la firma y llave pública de la entidad a la que está certificando.

2.2.1.5 Control de Accceso

Este servicio limita y controla el acceso al sistema y aplicaciones que son accesadas vía enlaces de comunicación. Se permite un acceso controlado a los recursos del sistema según las circunstancias y características del usuario ; es decir, que los usuarios que están permitidos a realizar operaciones en el sistema lo podrán hacer según sus permisos asignados. Una de las técnicas más utilizadas es el uso de login. Login es una técnica dónde los usuarios para poder identificarse ante el sistema insertan su nombre de usuario y una contraseña previamente creada en el sistema. En el caso de SMS, el control de acceso es realizado por la red celular. Por ejemplo, en una red GSM cada teléfono tiene un chip SIM el cuál tiene información que identifica al usuario y permite el acceso del celular a la red (23).

2.3 SMS

SMS es un servicio bidireccional que sirve para enviar mensajes cortos desde teléfonos celulares y así mismo poder recibir mensajes cortos en teléfonos celulares sobre sistemas de comunicación inalámbricos. La palabra "Corto" significa que se pueden enviar mensajes de hasta 160 caracteres cuando se usa alfabetos latinos, y 70 caracteres cuando se usa alfabetos no latinos como el alfabeto chino. Para aclarar este punto, SMS maneja 160 caracteres de largo con caracteres de 7 bits, 140 caracteres de 8 bits y 70 caracteres con caracteres de 16 bits. Este servicio fue creado a finales de los 80´s para que trabaje con la tecnología GSM (*Global System for Mobile Communications*). Este servicio fue creado por ingenieros del grupo GSM W3P que querían que exista un sistema de mensajes que continúe trabajando cuando los usuarios que reciban mensajes de texto tengan apagado su celular o estén fuera del rango de servicio.

2.3.1 Arquitectura de la Red SMS

Para entender de una mejor manera a la red SMS, analicemos la Figura 2.4 que muestra los principales componentes de la red SMS y como se enlazan entre ellos.

Los principales componentes son los siguientes :

- Dispositivo Móvil : Es una terminal inalámbrica que es capaz de recibir y originar mensajes de texto cortos. Generalmente, estos dispositivos son teléfonos celulares o PDAs. El dispositivo móvil se comunica con la estación base que se encuentra en la misma célula del teléfono vía la AI (*Air Interface*) Interface de Aire.
- BSS (*Base Station Subsystem*) : El BSS está constituido por dos elementos : BSC (*Base Station Controller*) y BTS (*Base Transceiver Station*). La BTS define cada célula y ésta incluye una radio antena, un radio transmisor y un enlace al BSC. La principal responsabilidad del BSC es la transmisión de tráfico de voz y datos entre los dispositivos móviles y el MSC (*Mobile Switchig Center*) por medio de las BTSs.

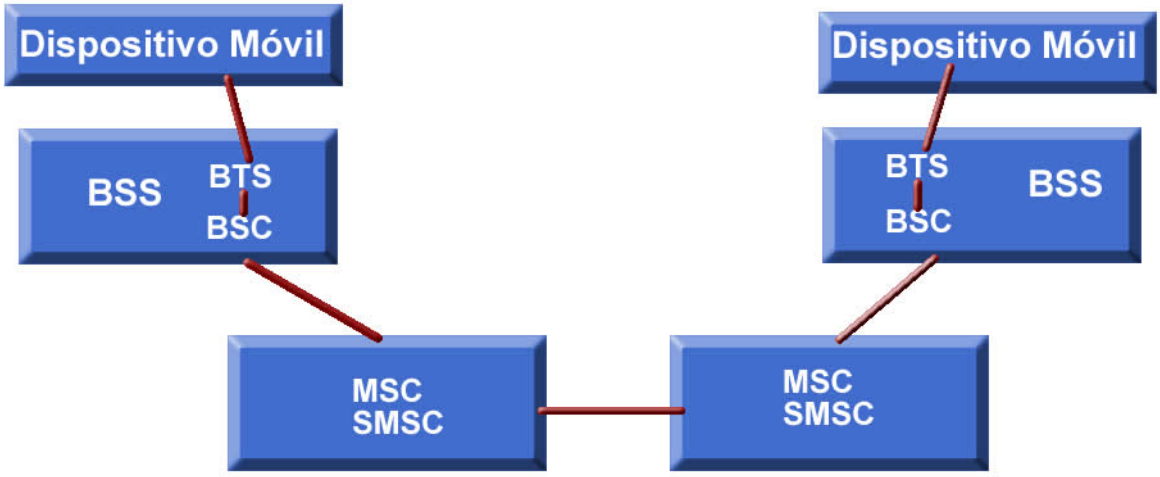

Figura 2.4: Red SMS

– MSC (*Mobile Switching Center*) : El MSC realiza las funciones de conmutación del sistema y controla llamadas a o desde otros teléfonos y sistemas de datos. Este componente también se encarga de crear enlaces entre la red celular y la red pública telefónica para la comunicación, controlar *hand-offs*, y establecer y rutear comunicaciones entre dispositivos móviles. MSC para cumplir sus operaciones se ayuda en dos bases de datos llamadas HLR (*Home Location Register*) y VLR (*Visitor Location Register*). El HLR almacena información permanente y es usado constantemente para almacenar y controlar los perfiles de los subscriptores, de esta base de datos se puede obtener la identidad y datos de los subscriptores necesarios para establecer una comunicación con los dispositivos móviles de aquellos subscritores. El VLR almacena información temporal de los subscritores que físicamente se encuentran cubiertos por el MSC. También almacena la ruta actual en la que se encuentra el subscriptor y si éste se encuentra activo o no.
– SMSC (*Short Message Service Center*) : Se define como una combinación de software y hardware que se encarga del almacenamiento y reenvío de los mensajes de texto cortos entre dispositivos móviles. Cada operadora celular cuenta con un

SMSC para el envío y recepción de mensajes de texto. Los mensajes de texto son almacenados temporalmente aquí hasta que el subscriptor esté activo y pueda recibir el mensaje de texto.

Después de haber listado los componentes analizamos como estos interactúan entre sí (31) :

1. El celular envía un mensaje de texto con destino a otro celular, éste se conecta con su BSS la misma que reenvía esta información al MSC.
2. El MSC consulta al VLR para tarifar y reenvía el mensaje al SMSC de origen.
3. El SMSC de origen envía el mensaje de texto al SMSC destino.
4. El SMSC destino almacena el mensaje en su base de datos y consulta al HLR la información de ruteo del usuario destino.
5. Si el usuario destino está disponible, el SMSC envía al MSC el mensaje con la información de en qué BSS se encuentra el destino, caso contrario se almacena el mensaje en el SMSC por un período de vigencia.
6. El MSC envía el mensaje al teléfono destino.
7. Finalmente, el MSC notifica al SMSC que el mensaje ha sido entregado y que puede ser borrado de su base de datos.

2.4 SMS y Sus Posibles Ataques Informáticos

Se entiende como ataque al evento exitoso o no exitoso que atenta al buen funcionamiento del sistema, es un asalto a la seguridad que se genera al tratar de explotar una vulnerabilidad del sistema.

SMS, como otros protocolos de comunicación sufre, de eminentes riesgos de ataques que atentan contra la confidencialidad de la información, la integridad de la información, autenticación de las partes y la repudiación. Este protocolo de comunicación no goza de protocolos que ayuden a la transmisión de datos seguros tal como WTLS para WAP,

WPA y WEP para redes WiFi (WLAN). Por ello, es necesario crear un esquema de seguridad que puede ser aplicado a SMS para la protección en la transmisión de datos.

2.4.1 Ataques SMS Contra la Confidencialidad

Un ejemplo de un modelo que atenta a la confidencialidad es el uso de un celular captor que pueda capturar mensajes entrantes de un celular en particular llamado interceptor (8). El celular interceptor es un celular modificado que cada vez que reciba o envía un mensaje de texto, envía una copia encubierta al celular captor con fecha y hora. Además, acorde con el artículo presentado por la Universidad de Pretoria Sud África de seguridad SMS (5), se menciona que la integridad y la confidencialidad pueden ser afectadas en la transmisión de mensajes SMS debido a que la transmisión de los mensajes de texto entre el dispositivo móvil y la estación base (BSS) puede o no ser encriptada. Esto depende si la estación base tiene configurado encriptación o no, e incluso existen países que la encriptación no es permitida (5). Además, los algoritmos de encriptación simétricos más conocidos que proveen confidencialidad en la transmisión de datos a las estaciones bases son los conocidos A5 con llaves compartidas de 16 y 64 bits. Shamir Biryukov (5) demostró en la convención "Fast Software Encryption Workshop", que es posible quebrantar estos algoritmos en minutos usando un analizador de tráfico. Para entender de una manera ejemplar este fenómeno, fijarse en la Figura 2.5 que muestra como el oponente lee los contenidos de los mensajes transmitidos entre dos entidades.

2.4.2 Ataques SMS Contra la Integridad

El artículo de seguridad de la Universidad de Petroria menciona que si el atacante tiene acceso al SMSC de la operadora celular, no sólo pudiera leer los datos en tránsito, sino también pudiera manipularlos. Para entender este fenómeno fijémonos en la Figura 2.6, donde el oponente captura el mensaje y modifica su contenido según sus necesidades

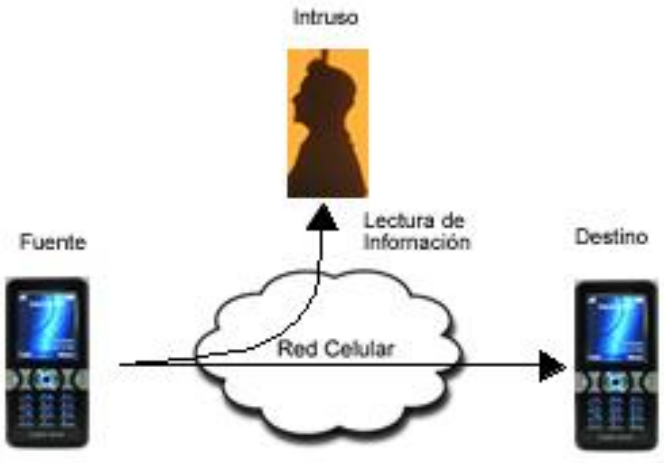

Figura 2.5: Ataque SMS a la Confidencialidad

y lo reenvía a la entidad destino.

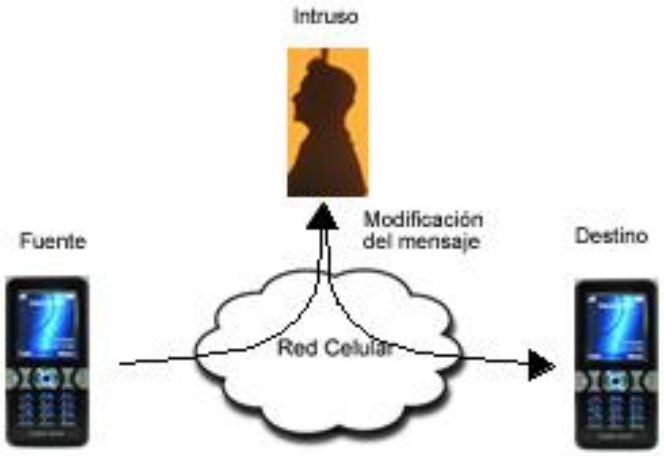

Figura 2.6: Ataque SMS a la Integridad

2.4.3 Ataques SMS de Suplantación de Identidad y Repudio

La suplantación de identidad es cuando una tercera entidad pretende ser otra entidad y la repudiación es cuando un determinado usuario niega sus operaciones en el sistema. Un ejemplo es que el oponente después de haber obtenido información de las

credenciales y dirección destino de la entidad fuente, éste puede hacer uso de esta información y suplantar la identidad para realizar, por ejemplo, transacciones bancarias sin autorización. En la Figura 2.7 se muestra cómo es este ataque, dónde una entidad externa logra obtener las credenciales de la entidad fuente y luego ésta hace uso de las credenciales para suplantar la identidad.

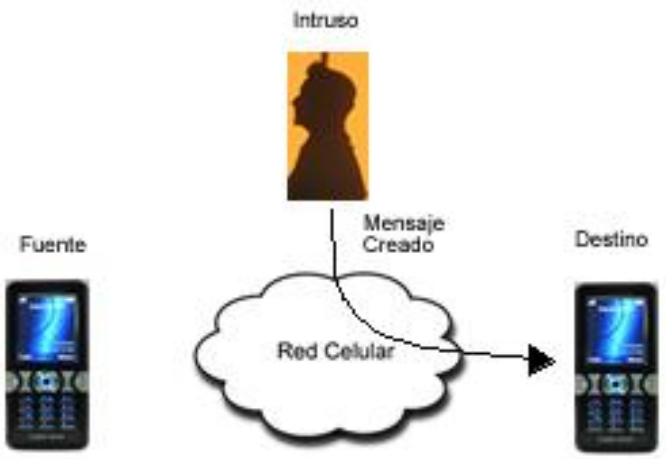

Figura 2.7: Ataque SMS a la Autenticación

2.5 SMS y sus Mecanismos Actuales Contra Ataques Informáticos

Como mecanismo de protección existe el uso de comandos SMS para realizar transacciones seguras y la implementación de la seguridad a nivel de aplicación. Por ejemplo, el Banco Pichincha (3) ofrece un servicio para realizar transacciones bancarias vía SMS. Para tener acceso a este servicio, los usuarios en la página Web del Banco deben inscribir su número telefónico y registrar que cuentas fuentes y destino van a ser utilizadas para las transacciones. Por ejemplo, para hacer una transferencia se debe enviar un mensaje de texto al 299 con el comando "TRF fuente valor destino". Otro ejemplo es la compra de tickets de bus en Australia, la forma de obtener este servicio es registrar

en la página Web de la empresa de buses un número de tarjeta de crédito y un número telefónico que va a ser utilizado para realizar compras de tickets.

Como segundo mecanismo de seguridad mencionado es la implementación de seguridad a nivel de la aplicación. Éste se refiere a la implementación de la seguridad en la aplicaciones que se ejecutan en dispositivos móviles para implementar seguridad. Existen celulares que soportan aplicaciones Java, con este lenguaje se pueden crear aplicaciones que envíen y reciban mensajes de texto encriptados. La ventaja de esta forma de seguridad es que se puede implementar varios mecanismos de seguridad para proteger la transferencia de los datos. Por ejemplo existen los paquetes Fortress SMS (6) y Spider SMS (4) que trabajan sobre Symbian e implementan un mecanismo de generación de firmas propias sobre los datos. También existen formas de implementar seguridad en SMS por medio de personalizaciones a nivel de hardware. *Trusted Short Message Service* (T - SMS) es un producto para teléfonos GSM enfocado en servicios de banca y de pagos (17) ; éste mecanismo requiere la instalación de un chip en el teléfono celular para generar mensajes seguros.

2.6 Tecnologías para la Implementación de SMS

Existen varias tecnologías para la implementación de aplicaciones que usen mensajes SMS para la comunicación entre dispositivos móviles. A continuación se presenta una descripción de las tecnologías existentes para SMS.

Dentro del lenguaje de programación JAVA, una de las tecnologías más conocidas es el API llamado *Wireless Message API* (10). Este API trabaja sobre la plataforma J2ME que se basa en el uso de Midlets, que son programas ejecutables empaquetados en un archivo de extensión *.jar* que usan el perfil MIDP (*Mobile Information Device Profile*) (conjunto de APIS que implementa funcionalidades en dispositivos móviles) y la configuración CLDC (*Connected Limited Device Configuration*) (plataforma JAVA para dispositivos móviles con recursos limitados). Su uso trata en abrir conexiones en

modo cliente o servidor según si se desea sólo enviar mensajes SMS o si también se desea recibir SMSs. Este API puede enviar y recibir mensajes binarios o mensajes de texto por medio del uso de la interface *javax.wireless.messaging*. Además se encarga de la segmentación y reensamblaje de mensajes binarios y de texto. Los límites de segmentación de los mensajes están dados por la Tabla 2.1 :

Codificación	Largo con puerto		Largo sin puerto	
	Largo	Mensajes SMS	Largo	Mensajes SMS
7 bit Alphabet	0-160 caracteres	1	0-152 caracteres	1
	161-304 caracteres	2	153-290 caracteres	2
	305-456 caracteres	3	291-435 caracteres	3
8 bit bynary data	0-140 bytes	1	0-133 bytes	1
	141-266 bytes	2	134-254 bytes	2
	267-399 bytes	3	255-381 bytes	3

Tabla 2.1: Segmentación de WMA

Existe soporte SMS en la plataforma de Android (1). Android es un software para dispositivos móviles que implementa un sistema operativo con sus propias bibliotecas y está siendo desarrollo por Google. Esta plataforma consta del paquete *android.telephony.gsm* que implementa las funcionalidades de enviar mensajes SMS binarios y de texto, segmentación y reensamblaje, pero sólo funciona para redes GSM.

Otra plataforma que soporta aplicaciones para enviar mensajes SMS es Symbian OS (20) ; este sistema operativo tiene una biblioteca desarrollada en C++ que implementa las funcionalidades de enviar y recibir mensajes de texto. Esta biblioteca es de bajo nivel debido a que presenta funciones que permiten el acceso al PDU del SMS.

Python (11) sobre teléfonos Nokia de la serie 60 también ofrece la funcionalidad del envío y recepción de mensajes SMS. Esta biblioteca ofrece los mismos servicios que la biblioteca WMA de Java pero su desventaja es que es soportado sólo por celulares de la serie 60 de Nokia.

En el caso de iPhone existe un API que maneja las comunicaciones llamado CF-Network (2). Este API sólo porporciona funcionalidades de red sobre Inernet para administrar conexiones HTTP y FTP. Lastimosamente, la parte de comunicaciones SMS en IPhone es propietaria de Apple y por ende no existe un API con documentación abierta, tal como existe para otras plataformas (13).

Capítulo 3

Biblioteca de Seguridad SMS

3.1 Introducción

Esta tesis propone la creación de una biblioteca J2ME, que implementa seguridad a nivel de la aplicación para la trasmisión de mensajes de texto SMS. Esta biblioteca implementa básicamente los mismos niveles de seguridad que los conocidos protocolos de seguridad de e-mail PGP y S/MIME y son sólo autenticación, sólo confidencialidad y la combinación de las dos. Cabe recalcar que esta biblioteca denominada como SMS^2 (SMS Seguro) es un proyecto de código abierto y se apoya principalmente en otros dos proyectos de código abierto y son :

- Bouncy Castle (22) : Este proyecto implementa la funcionalidad de algoritmos criptográficos utilizando el lenguaje de programación JAVA. Existen varias versiones de este proyecto compatibles para las máquinas virtuales JDK1.0, JDK1.1, JDK1.2, JDK1.3, JDK1.4, JDK1.5 y para la máquina virtual de J2ME. Cabe recalcar que este proyecto es el resultado de la contribución de más de 100 personas e importantes organizaciones.
- WMA *Wireless Messaging API* (10) : API de la Sun que provee acceso a recursos de comunicación inalámbrica como es el servicio SMS. Este API permite el envío y recepción de mensajes binarios o de texto usando el protocolo SMS.

Esta biblioteca fue creada por medio del uso de la metodología Proceso Unificado de Rational (14). El ciclo de vida de esta metodología se basa en 4 fases que son inicio, elaboración, construcción y transmisión. Estas fases han sido aplicadas de la siguiente manera :

1. Inicio : Se determinó como visión del proyecto la creación de una biblioteca J2ME que implemente seguridad en la mensajería celular.
2. Elaboración : La arquitectura de la biblioteca se basa en un conjunto de clases que provee seguridad en la transmisión de mensajes de texto de forma similar a la que implementa PGP y S/MIME. Como resultado de esta etapa, se obtuvo el diagrama de casos de uso (ver Anexo 1) que describe el comportamiento de la biblioteca.
3. Construcción : En esta etapa se realizó la implementación de la arquitectura propuesta. Por cada caso de uso, se realizó un detenido análisis, diseño, implementación y pruebas para así de cada uno hacer un entregable. Como resultado de esta etapa se obtuvo el diagrama de secuencia (ver Anexo 2) y de clases (ver Anexo digital 3) los cuales fueron la base para el desarrollo y organización del código fuente de la biblioteca SMS^2.
4. Transmisión : En esta etapa se realizó la liberación del proyecto, es decir se PUblicó el código fuente en la biblioteca de la Universidad San Francisco de Quito y se realizó el manual de usuario (ver Anexo 4) y la descripción técnica de la biblioteca en formato Javadoc (ver Anexo Digital 5).

3.2 Descripción Funcional

La biblioteca SMS^2 propone implementar seguridad en el envío y recepción de mensajes SMS basándose en la forma cómo lo hace PGP y S/MIME. Se ha dividido a la funcionalidad de la biblioteca en 5 categorías y son : seguridad SMS con sólo autenti-

cación, seguridad SMS con sólo confidencialidad, seguridad SMS con autenticación y confidencialidad, funcionalidades extras y otros servicios.

Esta biblioteca se basa en un conjunto de servicios que proveen diferentes funcionalidades tanto de seguridad como funcionalidades que sirven para mejorar la calidad del software. Se entiende como servicio a mecanismos que permiten el acceso a una o más funcionalidades. El principal servicio de seguridad de la biblioteca, que a su vez utiliza servicios de más bajo nivel, es el llamado "Servicio de Seguridad SMS". Este servicio está implementado en la clase *com.usfq.services.SmsSecurityService* y se basa en métodos emisores y métodos receptores. Los métodos emisores son aquellos que dados los parámetros necesarios, aplican un esquema de seguridad (autenticación, confidencialidad, ambos) y envían mensajes SMS de forma binaria. Los métodos receptores son aquellos que reciben al SMS binario y aplican el esquema inverso aplicado por el emisor para obtener el mensaje original. Para mayor detalle de los métodos emisores y receptores ver el Anexo digital 5, clase *com.usfq.services.SmsSecurityService*. A continuación se listan las capacidades de la biblioteca SMS[2]. Cabe recalcar que para las Figuras 3.1, 3.2 y 3.3 se aplica la siguiente nomenclatura :

- K_s : Llave de sesión para encriptación simétrica
- PR_a : Llave privada RSA del usuario A
- PR_b : Llave privada RSA del usuario B
- PU_a : Llave pública RSA del usuario A
- PU_b : Llave pública RSA del usuario B
- EP : Encriptación asimétrica
- DP : Desencriptación asimétrica
- EC : Encriptación simétrica
- DC : Desencriptación simétrica
- H : Función Hash
- || : Concatenación

3.2.1 Seguridad SMS con sólo Autenticación

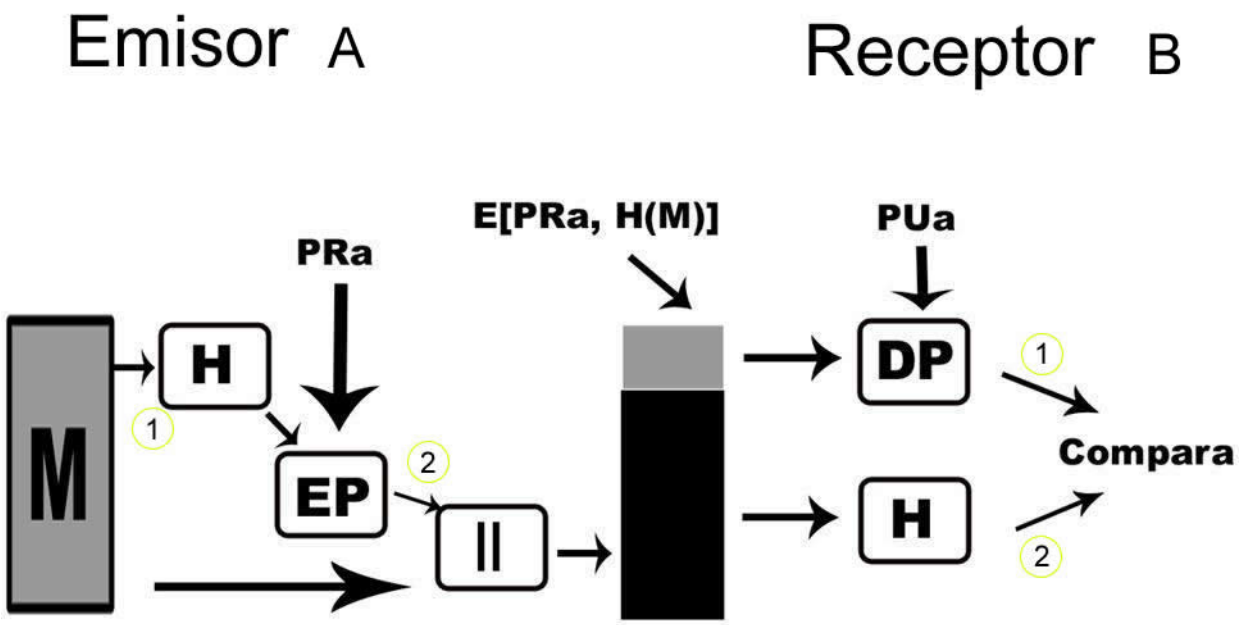

Figura 3.1: Biblioteca SMS^2 con sólo Autenticación

Esta funcionalidad es aquella que sirve para el envío y recepción de mensajes SMS seguros dónde se confirma las identidades de las partes involucradas. La Figura 3.1 presenta la implementación de esta funcionalidad :

Observando el gráfico, los pasos realizados son los siguientes :

Emisor :

1. Obtiene el hash del mensaje usando algún algoritmo hash soportado por esta biblioteca.
2. El código hash es encriptado usando la llave RSA privada PRa del emisor, y el resultado es concatenado al mensaje en claro.

Receptor :

1. Usa la llave pública RSA del emisor PU_a para desencriptar y obtener el código original hash.
2. Genera un nuevo hash del mensaje y lo compara con el hash desencriptado. Si los dos coinciden, el mensaje es aceptado.

Nota : Lo que en el gráfico se denota como E(PRa,H(M)), se conoce como la firma del mensaje. Este esquema de seguridad provee autenticación y no negabilidad, debido a que el receptor debe utilizar la llave pública del emisor para verificar si la firma es válida, y en caso de que sea válida, el emisor no puede negar que él emitió el mensaje. Adicionalmente, se está comprobando la integridad del mensaje, debido a que si el mensaje ha sido modificado durante la transmisión, la verificación de la firma falla.

3.2.2 Seguridad SMS con sólo Confidencialidad

Esta funcionalidad es aquella que sirve para el envío y recepción de mensajes SMS seguros dónde los datos transmitidos son encriptados. Esto lo podemos observar en la Figura 3.2.

Los pasos son :

Emisor :

1. El emisor genera una llave de sesión aleatoria Ks.
2. El mensaje es encriptado con la llave de sesión utilizando uno de los algoritmos permitidos por esta biblioteca.
3. La llave de sesión es encriptada con la llave pública RSA del receptor PU_b.
4. Envía un SMS que tenga la concatenación de la llave de sesión encriptada y el mensaje encriptado.

Receptor :

1. El receptor desencripta la llave de sesión usando su llave privada RSA PRb.
2. El receptor desencripta el mensaje usando la llave de sesión Ks.

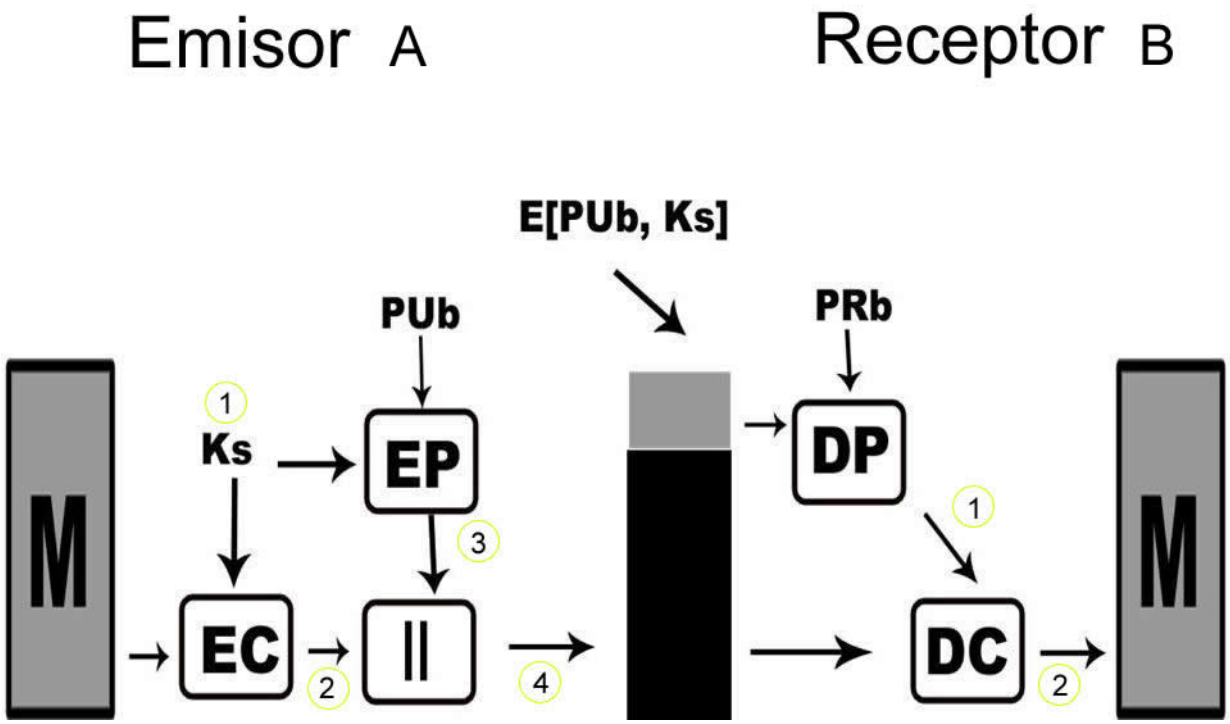

Figura 3.2: Biblioteca SMS^2 con sólo Confidencialidad

Cabe recalcar que esta biblioteca para la generación de la llave de sesión, utiliza la clase *com.usfq.services.RandomBytesKeyGenerator*. El programador debe generar la llave del tamaño adecuado para el algoritmo simétrico que se desea usar. Para ver documentación acerca de los distintos algoritmos simétricos y sus tamaños de llaves permitidos referirse al Anexo 5, clase *com.usfq.service.EncryptServices*. Nótese que este esquema provee confidencialidad debido a que los datos que viajan son encriptados con la llave compartida o de sesión.

3.2.3 Seguridad SMS con Confidencialidad y Autenticación

Este esquema es la unión de los dos esquemas anteriores, por ende es el que implementa mayor seguridad en la transmisión de datos SMS. Como resultado se obtienen mensajes SMS que ofrecen autenticación, integridad, confidencialidad y no negabilidad, dándo como resultado una transmisión de datos segura y confiable. La Figura 3.3

muestra lo que acontece con este esquema.

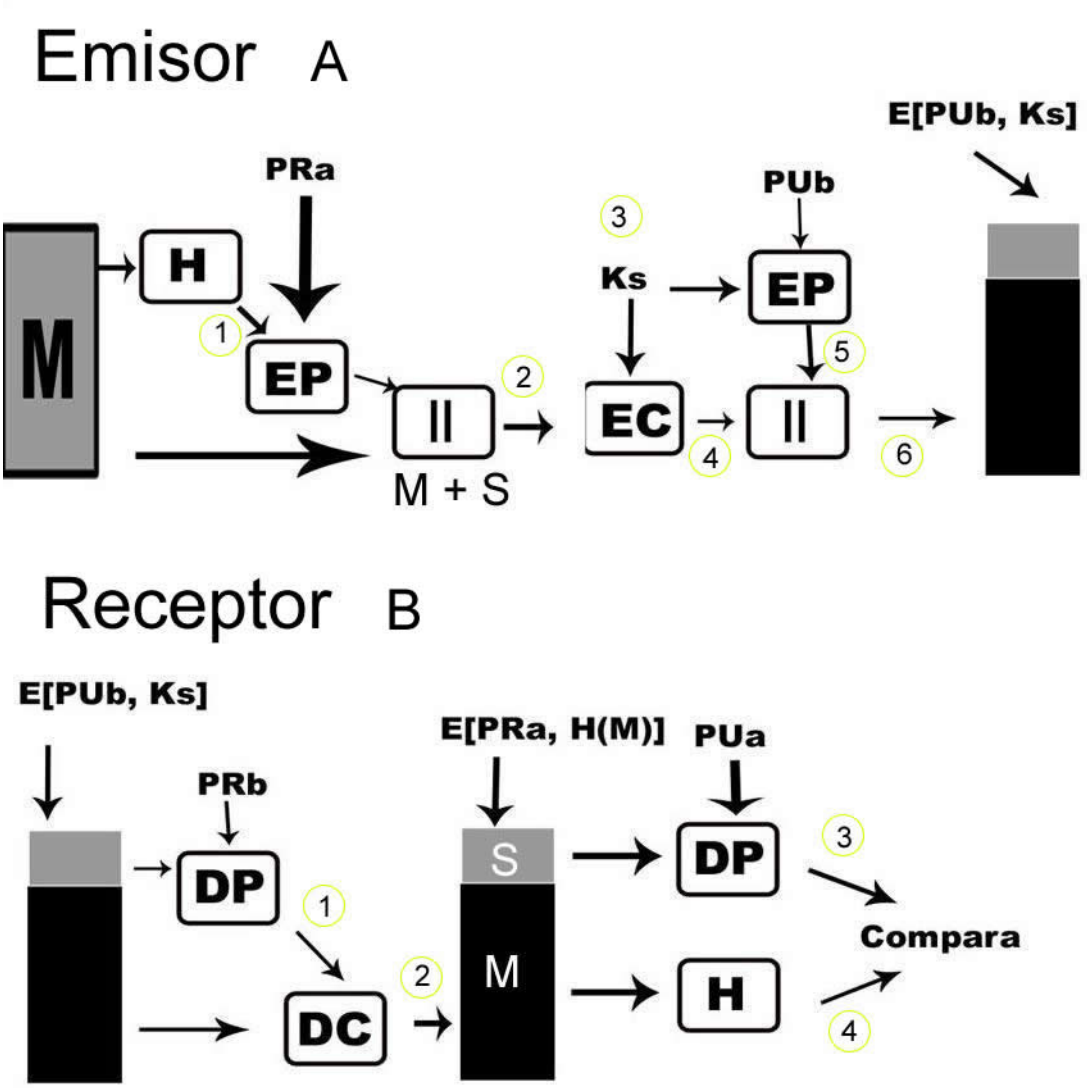

Figura 3.3: Biblioteca SMS^2 con Autenticación y Confidencialidad

Emisor :

1. Obtiene el hash del mensaje usando algún algoritmo hash soportado por esta biblioteca.
2. El código hash es encriptado usando la llave RSA privada del emisor PRa, y el resultado es concatenado al mensaje original y se obtiene M + S, dónde S equivale a E[PRa, H(M)], firma del mensaje.

3. El emisor genera un llave de sesión.
4. M + S es encriptado con la llave de sesión utilizando uno de los algoritmos de ésta biblioteca.
5. La llave de sesión es encriptada con la llave pública RSA del receptor PU_b.
6. Envía un SMS que tenga la concatenación de la llave de sesión encriptada y S + M encriptado.

Receptor :

1. Desencripta la llave compartida usando su llave privada RSA PRb.
2. Desencripta el mensaje usando la llave compartida y obtiene M + S.
3. El receptor usa la llave pública RSA del emisor PU_a para desencriptar y obtener el código original del hash.
4. Genera un nuevo hash del mensaje y lo compara con el hash desencriptado, si los dos coinciden, el mensaje es aceptado.

3.2.4 Funcionalidades Extras de la Biblioteca

Adicionalmente, esta biblioteca ofrece métodos emisores y receptores para enviar mensajes SMS binarios con encriptación simétrica y para enviar llaves públicas RSA. La implementación de encriptación simétrica fue realizada con el objectivo de brindar una opción más para implementar un esquema de seguridad simple y con alto desempeño. En cambio, la implementación del servicio para el intercambio de llaves públicas fue creada con el objetivo de dar soporte a los esquemas de seguridad complejos mencionados anteriormente.

3.2.5 Otros Servicios

Existen otros servicios que ofrece la biblioteca que son complementarios o parte de otros servicios, o que simplemente son utilizados para ayudar a desarrollar software

de mejor calidad con funcionalidades interesantes (Si requiere mayor detalle de los servicios, su implementación y formas de uso, referirse a los Anexos 4 y 5). Los servicios son :

- Servicio de Envío de Mensajes SMS : Este servicio sirve para enviar mensajes SMS binarios o de texto. No es necesario que se especifique el tipo de mensaje debido a que este servicio detecta el tipo de mensaje que se enviará.
- Servicio de Encolamiento de Mensajes SMS : Su función es encolar los mensajes SMS que van llegando al celular en un Vector y así evitar que mensajes se queden sin procesar. Un hilo se encarga de escuchar mensajes SMS entrantes al celular e insertar los mensajes en un Vector, y otro hilo se encarga de tomar los mensajes SMS, eliminarlos de la cola y llamar al método *notifyMessageArrive(Object messageObject, String messageType)* de la Interfaz *NotifierInterface* para notificar que un mensaje ha llegado.
- Servicio Administración de Aplicación : Este servicio sirve para crear una capa adicional entre el Midlet y la lógica del programa, es decir, separa la capa de presentación, de la lógica. Como parámetro a este servicio se le pasa el nombre completo (*fully qualified name*) de la clase que va a manejar los mensajes binarios y de la clase que va a manejar los mensajes de texto.
- Servicio de Llavero RSA : Este servicio se encarga de almacenar llaves RSA tanto públicas como privadas. El almacenamiento es en la memoria no volátil del celular y el usuario no tiene que lidiar con la forma de cómo se almacena la información debido a que este servicio es el encargado de implementar aquella lógica.
- Servicio Analizador de Encabezados : Este servicio sirve para que al recibir el mensaje binario, no nos preocupemos de cuál receptor de la clase *SmsSecurityService* debemos llamar. Este servicio analiza los encabezados y llama automáticamente al receptor correspondiente, caso contrario lanza una excepción *HeaderReceiveAnalyzerServiceException()*.

Capítulo 4

Evaluación

4.1 Introducción

Este capítulo trata de la evaluación de la seguridad, desempeño y consumo de energía de la biblioteca SMS^2. Se han desarrollado dos aplicaciones prototipos para poder demostrar que la biblioteca de seguridad puede ser utilizada por varios tipos de aplicaciones que utilicen el protocolo SMS. Además, éstas utilizan conexión bluetooth para conectarse con una computadora para intercambiar información e interactuar. La primera aplicación es una aplicación segura que implementa el sorteo de un premio vía SMS. Esta aplicación no utiliza bases de datos debido a que sólo registra los eventos del celular servidor en la computadora vía bluetooth. En cambio, la segunda aplicación utiliza algoritmos más seguros que la primera debido a que implementa voto electrónico y usa una base de datos para registrar los votos insertados por los usuarios.

4.2 Bluetooth

Como las aplicaciones utilizan conexión bluetooth entre el celular y una aplicación J2SE, vamos a realizar una pequeña descripción de los APIs utilizados para lograr esta comunicación y son :

- API JSR 82 javax.bluetooth : API que oculta la complejidad del stack del protocolo bluetooth y trabaja para el entorno CLDC/MIDP.
- API JSR 82 BlueCove : API que oculta la complejidad del stack del protocolo Bluetooth y trabaja en la plataforma J2SE.

JSR 82 *Java Specification Request* : Es una especificación estándar aprobada por la Comunidad de Desarrollo de la Tecnología Java que describe los servicios que deben tener los APIs JSR 82 para implementar bluetoooth. Los dos APIs anteriormente mencionados implementan los servicios de : Registro de servicios, descubrimiento de servicios y dispositivos, establecimiento de conexiones RFCOMM, L2CAP y OBEX, envío y recepción de datos, control de la comunicación y seguridad.

La conexión bluetooth utilizada en las dos aplicaciones es usando el servicio RFCOMM. El servicio RFCOMM es la conexión serial entre los dos dispositivos para el envío y recepción de flujos de bytes. Para finalizar, la última característica de estas dos aplicaciones es que usan seguridad en la conexión bluetooth. Esto se configura en el servidor al momento de asignar su URL, ejemplo :

String serverUrl = “btspp ://localhost :” + RFCOMM_UUID + “ ;name=rfcommtest ; authenticate=true ;authorize=true ;encrypt=true” ;

Dónde RFCOMM_UID es el código hexadecimal del servicio RFCOMM y es el Ox003.

La seguridad que implementa el API bluetooth ofrece lo siguiente :

- Autenticación : Consiste en verificar la identidad del dispositivo remoto. La autenticación implica un reto que se lanza entre los dispositivos, que requiere una clave compartida de 128 bits derivada de un PIN compartido por ambos dispositivos. Si el PIN en ambos dispositivos falla, falla el proceso de autenticación.
- Encriptación : Todos los datos transmitidos en ambas direcciones se encriptan con una llave de 128 bits.
- Autorización : Procedimiento por el cual un usuario de un dispositivo servidor

garantiza el acceso a un servicio específico a un cliente específico.

4.3 Aplicación Básica de Seguridad

Esta aplicación implementa el típico sorteo donde se envía mensajes de texto a un número específico para registrarse y participar por un premio. La diferencia es que esta aplicación va más allá ya que implementa seguridad debido a que se desea identificar correctamente a cada participante. Además, muestra las funcionalidades que tiene la biblioteca debido a que implementa las tres formas de seguridad que son sólo autenticación, sólo confidencialidad y autenticación más confidencialidad en el envío y recepción de mensajes SMS. Cabe recalcar que la estructura de esta aplicación es sólo para fines demostrativos de cómo se puede integrar las funcionalidades de la biblioteca propuesta con un computador. La Figura 4.1 muestra como es la estructura de la aplicación.

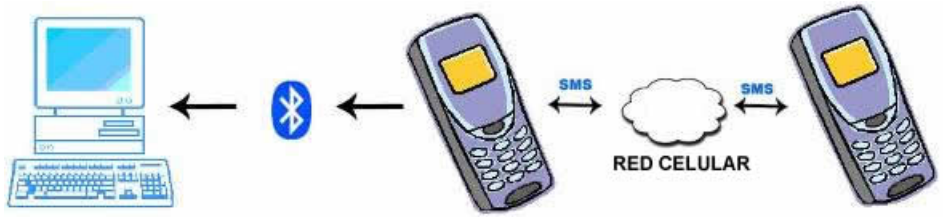

Figura 4.1: Sorteo SMS

Vemos en la Figura 4.1 que existe un computador que se conecta con un celular por medio de bluetooth. Por fines demostrativos, en este caso la lógica del servidor de sorteo se encuentra implementado en el celular ; por ende, el celular es el encargado de procesar el login, registrar usuarios en el sorteo (registra los usuarios en memoria), y realizar el sorteo. Se utiliza el computador sólo para dejar constancia de que se puede implementar la lógica en un computador y no cargar de procesamiento a un celular, esto se lo realiza en el caso de la aplicación compleja voto electrónico que se revisará más adelante.

4.3.1 Demostración de la Aplicación

1. Después de haber establecido la conexión bluetooth entre el computador y el celular, el cliente celular SMS ya puede interactuar con la aplicación. La Figura 4.2 muestra la pantalla de login del celular cliente. En esta pantalla se debe ingresar el nombre, contraseña y número de teléfono al cuál se desea comunicar y elegir si se desea comunicar con el servidor sin seguridades, con sólo confidencialidad, con sólo autenticación o con autenticación y confidencialidad.

Figura 4.2: Sorteo Login

2. Cuando el mensaje llega al celular servidor SMS, éste se comunica con el computador vía bluetooth y le reenvía el mensaje al computador para que éste imprima en consola el mensaje llegado ; ejemplo :

   LLEGO

   >>SMS ://+59393681970 :5432>> Texto en claro : requestadmin ?admin

   Luego, el servidor SMS se encarga de procesar el login y responde "aceptado" o "rechazado" al celular cliente.

3. Si el servidor responde "aceptado" a la aplicación cliente, el cliente muestra una pantalla dónde se muestra una pregunta que debe responder el cliente correctamente tal como muestra la Figura 4.3.

Figura 4.3: Respuesta Sorteo

4. Cuando la respuesta llega al servidor, éste verifica si es correcta, y si es correcta registra el número de teléfono en memoria y envía un mensaje de "Contestaste Correctamente" o "Mala respuesta no contestaste correctamente". La Figura 4.4 muestra la pantalla que le aparece al cliente SMS, en caso de que su respuesta haya sido incorrecta.

4.3.2 Análisis Sorteo sin Seguridad

Como se mencionó anteriormente, la aplicación permite la interacción sin seguridad. En este caso no se implementa seguridad en el envío y recepción de mensajes, la única seguridad es el login que debe realizar el celular cliente para poder registrarse en el sorteo. El login sólo está proporcionando control de acceso pero tenemos que recalcar

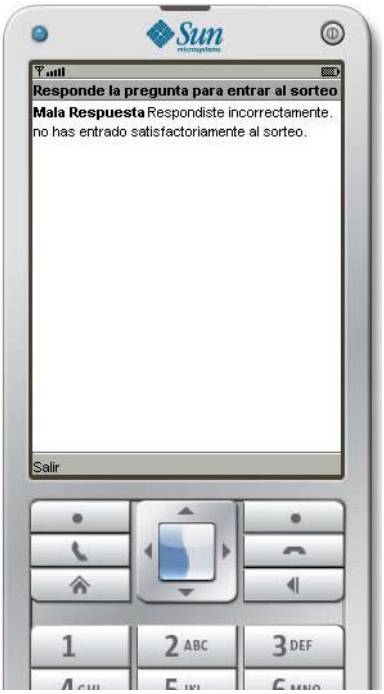

Figura 4.4: Resultado de regreso al Sorteo

que existe riesgo de ataques porque los datos viajan en claro, no existe autenticación y los mensajes pueden ser modificados sin que ninguna de las dos partes se enteren. Por ello, a continuación se analizan los tres niveles de seguridad que la biblioteca ofrece para la seguridad en la transmisión de datos.

4.3.3 Análisis Sorteo con sólo Confidencialidad

Este nivel de seguridad brinda confidencialidad en la transmisión de los datos. A diferencia del anterior escenario, éste requiere que tanto cliente como servidor generen un par de llaves público / privado RSA e intercambien las llaves públicas. El intercambio de llaves es por medio de esta biblioteca debido a que ésta tiene generadores de llaves público privado y además tiene un módulo que permite el envío y recepción de llaves RSA. En este caso, se utilizan llaves RSA de tamaño de 256 bits que son almacenadas en la memoria no volátil tanto del celular conectado al computador como por el celular participante del sorteo.

El envío de mensajes confidenciales consiste en :

1. El emisor genera una llave compartida DES de 64 bits.
2. El emisor encripta al mensaje con la llave DES.
3. El emisor encripta la llave compartida con la llave pública del receptor.
4. El emisor envía un SMS encriptado más la llave compartida encriptada.
5. El receptor desencripta la llave compartida utilizando su llave privada.
6. El receptor desencripta al SMS usando la llave compartida.

Con este esquema, el receptor garantiza que el mensaje ha sido enviado por una entidad que tiene su llave pública y el emisor tiene las garantías de que sólo el dueño de la llave privada podrá desencriptar la llave compartida para obtener el mensaje SMS. Cabe recalcar que este método no sólo es encriptación de los mensajes con las llaves privadas, es un esquema dónde cada vez que se envía un mensaje, el emisor genera una llave aleatoria simétrica DES de 64 bits. Con esto conseguimos mejorar tanto la seguridad como el desempeño debido a que el proceso de encriptación y desencriptación de los datos con llaves compartidas es más rápido que RSA y además mejora la seguridad porque no se cae en el problema de como distribuir la llave compartida. Este esquema tiene un costo respecto al tamaño del mensaje, por ejemplo para enviar el mensaje de login "requestadmin ?admin" de 18 bytes, este módulo transforma este mensaje a 62 bytes.

4.3.4 Análisis Sorteo con sólo Autenticación

Este esquema de seguridad también requiere un previo intercambio de llaves públicas. A continuación se describe el proceso de transmisión SMS :

1. El emisor genera un hash del mensaje utilizando el algoritmo MD2.
2. El emisor encripta al hash utilizando su llave de 256 bits y así logra formar su firma.

3. El emisor envía un SMS con datos más la firma.
4. El receptor verifica la firma con la llave pública del emisor y acepta o rechaza al mensaje.

Cabe recalcar que este esquema brinda autenticación del usuario y protege la integridad de los datos. Brinda autenticación de usuarios porque si el receptor logra verificar efectivamente la firma del mensaje con la llave pública del emisor, sabe que el mensaje ha sido enviado por dicho emisor. Además, brinda protección contra la integridad de los datos debido a que la firma es la encriptación con la llave privada del hash del mensaje y éste depende del contenido de los datos. En caso de que el mensaje haya sido alterado en la transmisión, el receptor, al momento de verificar la firma, puede detectar que el mensaje ha sido alterado y rechazarlo. El costo del tamaño del mensaje con este esquema de seguridad es superior al anterior, el mensaje de login "requestadmin ?admin" de 18 bytes es transformado a un SMS de 88 bytes.

4.3.5 Análisis Sorteo con Autenticación y Confidencialidad

Este esquema, al igual que los dos anteriores, requiere el previo intercambio de llaves públicas entre las dos partes que se van a comunicar. Se implementa tanto confidencialidad como autenticación, es decir es la combinación de los dos esquemas anteriores de seguridad. Tenemos que recalcar que conseguimos un alto nivel de seguridad en la transmisión de mensajes SMS debido a que existe confidencialidad, integridad del mensaje, autenticación, control de acceso y no negación. Para entender de una manera más fácil a continuación se describe el proceso :

1. El emisor firma el mensaje con su llave privada.
2. El emisor genera una llave compartida DES de 64 bits y encripta al mensaje más su firma.
3. El emisor encripta la llave DES de 64 bits con la llave pública del receptor.
4. Envía el mensaje SMS firmado y encriptado más la llave compartida encriptada.

5. El receptor desencripta la llave compartida con su llave privada.
6. Desencripta al mensajes SMS encriptado y firmado usando la llave compartida.
7. Y verifica la firma del emisor usando la llave pública del emisor.

Analizando un poco más allá, se puede recalcar que la confidencialidad existe debido a que los datos viajan encriptados. Integridad del mensaje existe debido a que la verificación de la firma depende de la firma y de los datos. Autenticación y no negación existe debido a que el receptor puede verificar la firma del emisor usando su llave pública. El control de acceso se da debido a que la aplicación implementa login para controlar el acceso de los usuarios al sorteo. Así como la seguridad de este esquema es superior a los anteriores, el costo en términos de longitud del mensaje también lo es debido a que 18 bytes son transformados a 129 bytes.

4.4 Aplicación con Altos Requisitos de Seguridad

Esta aplicación es más compleja y crítica que la anterior. En este caso se intenta explotar el máximo las seguridades ofrecidas por la biblioteca. En esta ocasión se va a utilizar la biblioteca SMS^2 para implementar voto electrónico sobre SMS. El voto electrónico es una de las aplicaciones más desafiantes porque requiere altos requisitos de seguridad y lo que se quiere demostrar con este prototipo es que se puede conseguir alta seguridad.

La Figura 4.5 muestra la estructura de la aplicación de voto electrónico. En este caso, el computador interactúa más con el celular debido a que la lógica del voto electrónico se encuentra implementada en el computador y el celular es utilizado sólo como interface de encriptación, desencriptación de mensajes y como llavero público. En el computador corre una aplicación J2SE que se comunica vía bluetooth con el midlet del celular y esta aplicación administra la base de datos que almacena a los candidatos con su número de votos y a los votantes con un indicador de si ya voto o no.

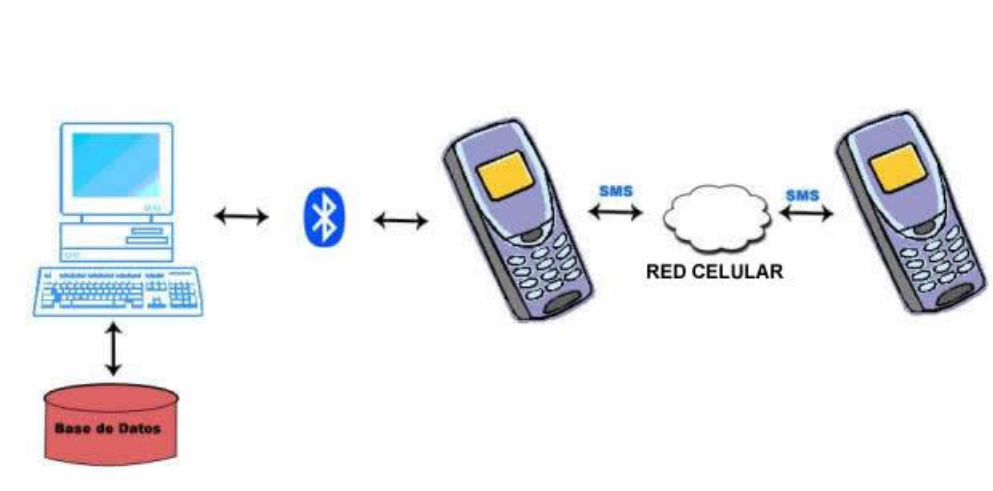

Figura 4.5: Voto SMS

4.4.1 Demostración de la Aplicación

Después de un intercambio de llaves públicas entre el celular votante y el celular conectado al servidor, empieza el proceso de votación, la Figura 4.6 muestra la pantalla de login de la aplicación votante donde el usuario debe insertar su cédula y clave preconfigurada en la base de datos del servidor de votaciones.

Si el login ha sido aceptado, el siguiente paso es escoger el candidato favorito y proceder al voto tal como muestra la Figura 4.7. El último paso es mostrado en la Figura 4.8 dónde el votante recibe una confirmación de que su voto ha sido almacenado satisfactoriamente.

En cambio del lado del servidor, existe una aplicación J2SE que administra el proceso de votación. Esta aplicación tiene un botón de salir, de depuración que muestra la llegada de los mensajes, de inicio de escucha bluetooth y de ver resultados. La Figura 4.9 muestra la llegada de un mensaje que ha sido firmado con SHA1 y encriptado usando el algoritmo Rijndael.

La Figura 4.10 muestra los resultados obtenidos en el proceso de votación.

Figura 4.6: Login Voto SMS

Figura 4.7: Votación Voto SMS

Figura 4.8: Confirmación Voto SMS

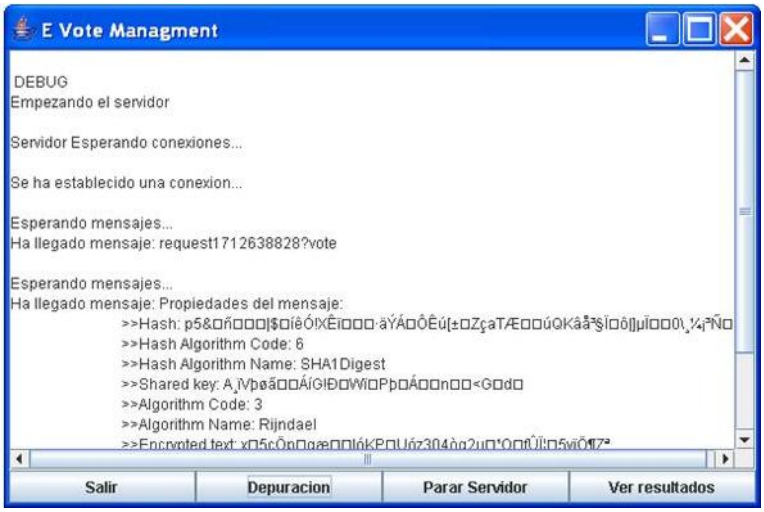

Figura 4.9: Aplicación J2SE

Figura 4.10: Aplicación J2SE Resultados

4.4.2 Análisis de Seguridad del Voto Electrónico

Esta aplicación utiliza el módulo de máxima seguridad ofrecida por la biblioteca SMS[2], usa el módulo de envío de mensajes de texto con confidencialidad y autenticación. Tenemos que considerar que con este esquema el control de acceso a la aplicación está dado por el login, la integridad del mensaje está dada por la firma del mensaje, la autenticación de las partes también se consigue por medio del uso de las firmas digitales, la confidencialidad se consigue por medio de la encriptación del mensaje, y la no negación se garantiza por el uso de la criptografía y el uso de los mensajes firmados.

En este caso se utiliza algoritmos más fuertes para mejorar la seguridad, pero cabe recalcar que el desempeño de la aplicación baja, esto último será analizado en la sección 5.3. Además, por cada mensaje de texto seguro que se envía, se llega a utilizar hasta dos mensajes de texto, por ende el API WMA entra en acción para realizar la segmentación y reensamblaje de los mensajes SMS.

Se utiliza llaves RSA de 512 bits para firmar a los mensajes usando el algoritmo

SHA1, para la encriptación se utiliza el algoritmo Rijndael de 256 bits. Cabe recalcar que el aumento al tamaño del mensaje es crítico en esta aplicación debido a que se usa más de dos mensajes de texto para enviar mensajes de texto cortos. Por ejemplo, para enviar el mensaje de login "request1712638822 ?vote" que son 22 bytes, se transforman a 169 bytes.

4.5 Desempeño

El desempeño de la biblioteca SMS[2] se ha medido por medio del uso de bechmarks JAVA que miden el tiempo de ejecución de las operaciones en milisegundos. Después de 10 corridas se han obtenido promedios de las principales operaciones que pueden ser ejecutadas con esta biblioteca. Las pruebas se han realizado usando dos celulares de distintas marcas y de distintas características. Se usó un Nokia 3500 con un procesador ARM9 de 104MHz, con una batería de 860 mAh y un Sony Ericsson K550i que tiene un procesador ARM9 de 220 MHz y con una batería de 950 mAh.

La tabla 4.1 muestra la comparación de desempeño de los celulares para la generación de llaves RSA y encriptación de llave pública, cuando se encriptan 32 bytes de datos equivalente a 256 bits. En este caso se observa que el orden del desempeño en los dos celulares son prácticamente iguales. La tabla 4.2 muestra que el desempeño de los

	Nokia 3500			Sony Ericsson K550i		
Tamaño de Llave	Generación RSA	Encriptar	Desencriptar	Generación RSA	Encriptar	Desencriptar
128	0,2 seg	2 ms	4 ms	0,3 seg	2,1 ms	5 ms
256	1,0 seg	5 ms	20 ms	1,2 seg	6,2 ms	37,2 ms
512	8,2 seg	13 ms	117 ms	9,89 seg	18,1 ms	247,5 ms
1024	79,2 seg	44 ms	873 ms	102,442 seg	99,3 ms	1000 ms

Tabla 4.1: Desempeño RSA

dos teléfonos para la generación de llaves compartidas, encriptación y desencriptación de 64 bytes de datos es muy similar. Nótese que el orden de las operaciones es el mismo y no varían mucho con diferentes algoritmos. Lo importante que hay que recalcar es la

diferencia del desempeño entre la encriptación y desencriptación de llave pública privada y de llave compartida. La encriptación de llave compartida tiene mejor desempeño tanto en la generación de llaves como en el proceso de encriptación y desencriptación.

	Nokia 3500			Sony Ericsson K550i		
Tamaño de Llave	Generación Llave	Encriptar	Desencriptar	Generación Llave	Encriptar	Desencriptar
DES 64 bits	1,8 ms	2,2 ms	2,8 ms	1,4 ms	1 ms	1 ms
3DES 128 bits	2,1 ms	3,2 ms	3,4 ms	1,5 ms	2 ms	1,8 ms
3DES 192 bits	2,6 ms	3,6 ms	4,0 ms	1,4 ms	2,3 ms	2,1 ms
Rijndael 128 bits	2,1 ms	4,2 ms	5,8 ms	1,5 ms	2,4 ms	2,9 ms
Rijndael 256 bits	2,9 ms	5,4 ms	7,0 ms	1,4 ms	3,8 ms	4,3 ms

Tabla 4.2: Desempeño Llaves Compartidas

La tabla 4.3 muestra los tiempos de procesamiento de los principales algoritmos hash y son MD5 y SHA1. Nótese que el celular Nokia tiene mejor desempeño en esta operación, pero lo importante es recalcar que el algoritmo SHA1 tiene mejor desempeño que MD5, por esa razón y por su mejorada seguridad es mejor utilizar SHA1.

	Nokia 3500	Sony Ericsson K550i
Algoritmo	Tiempo de Ejecución	Tiempo de Ejecución
MD5	7,6	8,3 ms
SHA1	5,6 ms	6,1 ms

Tabla 4.3: Desempeño Algoritmos Hash

Para finalizar esta sección, la tabla 4.4 muestra el desempeño medido de los dos celulares cuando se realizan operaciones de envío de mensajes de texto con sólo confidencialidad, con sólo autenticación y combinación de los dos. Cabe recalcar que estos tiempos se refieren al procesamiento o preparación que toma la biblioteca para el envío

y recepción de SMS seguros, descartándose la latencia de la red. Estos datos son tomados utilizando RSA de 512 bits, SHA1 como algoritmo de hash y encriptación de llave compartida Rijndael de 256 bits. Cabe recalcar que los tiempos de ejecución son suficientemente pequeños y por ende se puede utilizar estos tipos de seguridades en cada mensaje de texto.

	Nokia 3500		Sony Ericsson K550i	
Tipo Servicio	Procesamiento de Envío	Procesamiento de Recepción	Procesamiento de Envío	Procesamiento de Recepción
SMS Firmado	163 ms	27 ms	404,2 ms	170,8 ms
SMS Encriptado	40 ms	138 ms	224,4 ms	365,8 ms
SMS Encriptado y Firmado	181 ms	182 ms	423,8 ms	443,8 ms

Tabla 4.4: Desempeño Servicios SMS

4.6 Consumo de Energía

Una métrica muy importante que se debe considerar es el consumo de energía en dispositivos móviles al momento de usar esta biblioteca. La idea de esta sección es calcular cuantos mensajes seguros utilizando SMS[2] una batería completamente cargada soporta. Antes de mostrar el consumo de energía que gastan los celulares al usar esta biblioteca, primero vamos a presentar datos generales de consumo de potencia de los dos celulares estudiados Nokia 3500 y Sony Ericsson K550i. La tabla 4.5 muestra la potencia de ambos celulares en diferentes estados, la potencia está dada en Watts y es la transferencia de energía por unidad de tiempo.

Cabe recalcar que el celular Nokia 3500 tiene una batería con capacidad de 860 mAh, eso es equivalente a 11,45 KJ de energía, en cambio el celular Sony Ericsson K550i tiene una batería de 950 mAh que equivale a 12,7 KJ de energía. La tabla 4.6 muestra el gasto de energía que realizan ambos celulares al usar la biblioteca propuesta. La energía está dada en Joules y es el producto de la potencia por el tiempo.

	Nokia 3500	Sony Ericsson K550i
Condición	Poder	Poder
Stand By	11mW	25,9 mW
Activo + Salva pantalla	48 mW	151,7 mW
Activo	227 mW	466,2 mW
Activo + Llamada	796 mW	836,2 mW
Activo + Cámara	851 mW	1258 mW

Tabla 4.5: Valores Consumo de Potencia

	Nokia 3500	Sony Ericsson K550i
Operación	Energía	Energía
Encriptación RSA 512 bits	7,66 mJ	12,05 mJ
Encriptación Rijndael de 256 bits	3,18 mJ	2,53 mJ
Función Hash SHA1	3,30 mJ	4,06 mJ
Envio SMS Firmado	96,007 mJ	269,20 mJ
Envio SMS Encriptado	23,56 mJ	149,45 mJ
Envio SMS Firmado y Encriptado	106,609 mJ	282,25 mJ

Tabla 4.6: Valores Consumo de Energía

Asumiento que la descarga de la batería es lineal se llega la conclusión estimada que con el teléfono Nokia con su batería cargada puede enviar hasta 107 mil SMS seguros con confidencialidad y autenticación, en cambio con el teléfono Sony Ericsson se puede enviar con una batería totalmente cargada hasta aproximadamente 46 mil SMS seguros.

Capítulo 5

Conclusiones y Trabajos Futuros

5.1 Conclusiones

La biblioteca SMS construida para esa tesis permite a programadores y a usuarios intercambiar mensajes de texto SMS de forma confidencial, íntegra, no repudiable y autenticada. Además, esta biblioteca es muy flexible debido a que puede ser usada para varios tipos de aplicaciones y usando diferentes niveles de seguridad, tal como se ha mostrado en la aplicación de sorteo SMS y de voto electrónico.

En la aplicación de sorteo se han usado algoritmos menos seguros pero con mayor desempeño. Se usó llaves RSA de 256 bits, DES 64 y hash MD2 para enviar mensajes SMS autenticados y confidenciales. De acuerdo a los cálculos realizados en el teléfono Sony Ericsson, el tiempo total de envío de cada mensaje de longitud de 64 bytes con estas seguridades es de 369 milisegundos y de recepción es de 313 milisegundos. En cambio, la aplicación de voto electrónico utiliza llaves RSA de 512 bits, firmas SHA1 y algoritmo confidencial Rijndael de 256 bits, es decir utiliza distintos algoritmos pero más seguros. Por ello, el tiempo de envió promedio de un SMS con el teléfono Sony Ericsson es de 423,8 ms, y el tiempo de recepción es de 443,8 milisegundos. En cambio, con el teléfono Nokia marca los tiempos de 181 ms para el envío y 182 para la recepción.

Cabe recalcar que la flexibilidad de esta biblioteca permite escoger entre varias opciones dependiendo de las necesidades. Por ejemplo, si es muy crítico el desempeño

pero necesitamos un nivel de seguridad no tan fuerte podemos escoger los algoritmos utilizados para el sorteo, pero si la seguridad es preferente ante el desempeño podemos utilizar los algoritmos utilizados en voto electrónico.

Otro aspecto que hay que considerar es el número de mensajes utilizados. Por ejemplo, con la opción 1 (seguridad en aplicación sorteo) se utiliza 1 mensaje SMS para enviar de 0 a 23 bytes, 2 SMS para enviar de 24 a 143 bytes, y 3 SMS para enviar de 144 bytes a 271 bytes. En cambio con la opción 2 (seguridad en aplicación voto electrónico) se utilizan 2 SMS para enviar de 0 a 103 bytes y 3 SMS para enviar de 104 bytes a 231 bytes. Algo muy importante de notar es que no se aplicó la opción de enviar mensajes SMS seguros con llave RSA de 1024, Rijndael de 256 bytes y SHA1 debido al bajo desempeño, el envío demora 1654 ms y la recepción 1808,2 ms. Además, el número de mensajes es crítico debido a que se utiliza 3 SMS para enviar de 0 a 103 bytes. Esto es crítico debido a que los usuarios deberán gastar más dinero por cada mensaje seguro que desean enviar y también el API WMA en su documentación explica que la correcta segmentación y reensamblaje de mensajes SMS es garantizada hasta 3 SMS.

El aspecto de consumo de energía no es tan crítico debido a que la biblioteca SMS^2 es muy eficiente. Asumiendo que la descarga de la batería del celular es lineal, se ha llegado a la conclusión estimada de que se puede enviar 107 mil mensajes de texto seguros aplicando la seguridad utilizada para voto electrónico si la batería del celular Nokia 3500 está completamente cargada, y en el caso del teléfono Sony Ericsson se puede enviar hasta 46 mil SMS seguros con autenticación y confidencialidad. Con estos datos se ha demostrado que el manejo de energía de la biblioteca SMS^2 es muy eficiente al igual que el desempeño debido a sus cortos tiempos en las operaciones y al poco consumo de energía de la batería.

Para culminar, SMS^2 es una biblioteca bastante modular que presenta varios niveles de seguridad y algoritmos según las necesidades de las aplicaciones. Por ello antes de

escoger alguna opción es muy importante analizar si es más importante el desempeño que la seguridad o viceversa y también es necesario considerar el número de mensajes SMS necesarios para enviar mensajes seguros.

5.2 Trabajos Futuros

Después de haber desarrollado la biblioteca SMS^2 para la plataforma J2ME, sería interesante implementar estas funcionalidades y expandirlas en otras plataformas como Symbian OS y Android que son plataformas nuevas que están creciendo en el mercado de teléfonos celulares. Como nuevas funcionalidades a esta biblioteca se incorporará nuevos algoritmos de criptografía asimétrica como ECC (*Elliptic curve cryptography*) para así tener más opciones de este tipo de criptografía como es el caso de criptografía simétrica.

Otros trabajos que se realizará es la creación de más aplicaciones seguras SMS con diferentes estructuras y comportamientos para poder demostrar de forma más clara la modularidad y el acoplamiento de la biblioteca a distintos tipos de aplicaciones con diferentes objetivos.

Y por último se está considerando incluir en la evaluación de la biblioteca la latencia, que significa considerar el tiempo de envío de mensajes seguros considerando también el tiempo de transmisión de los mensajes en la red celular. Con estos tiempos se pudiera presentar tablas no sólo considerando los tiempos de ejecución de las operaciones de la biblioteca sino el tiempo total del envío de mensajes seguros.

Anexos

Anexo 1

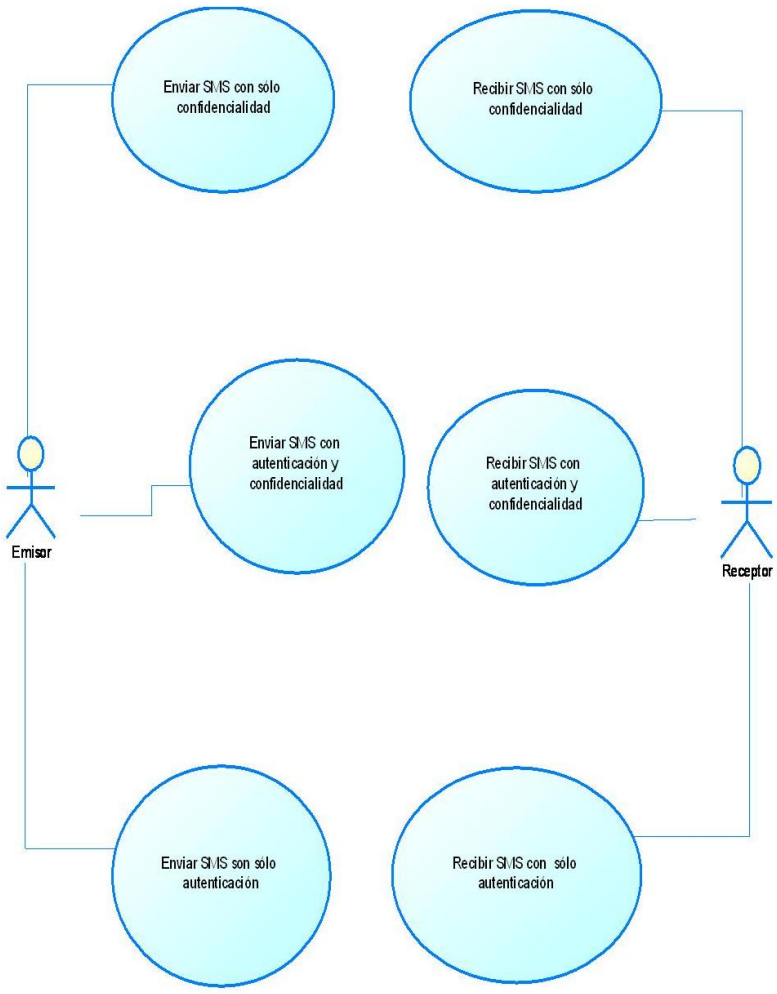

Figura 5.1: Diagrama de Casos de Uso

Anexo 2

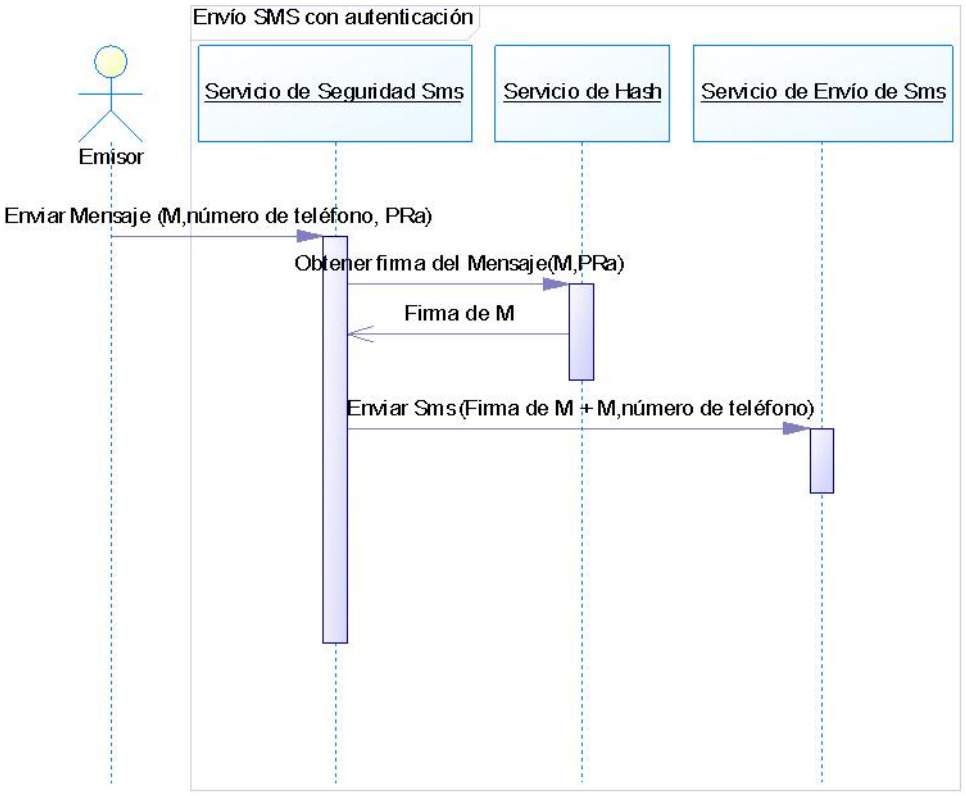

Figura 5.2: Anexo 2 sólo Autenticación Envío

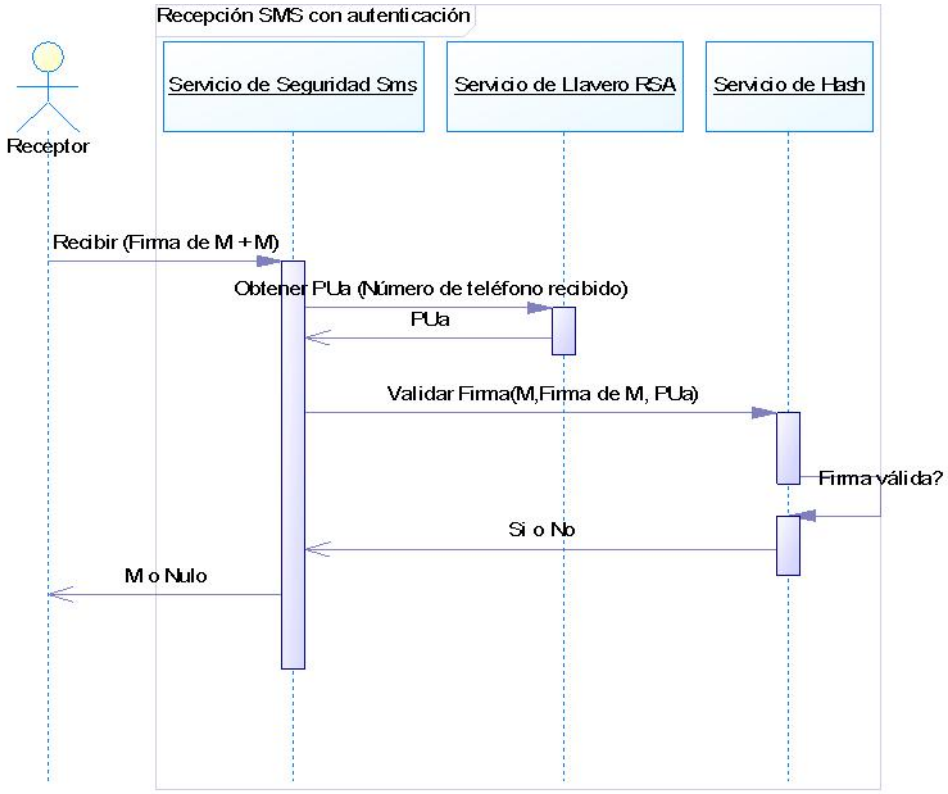

Figura 5.3: Anexo 2 sólo Autenticación Recepción

.

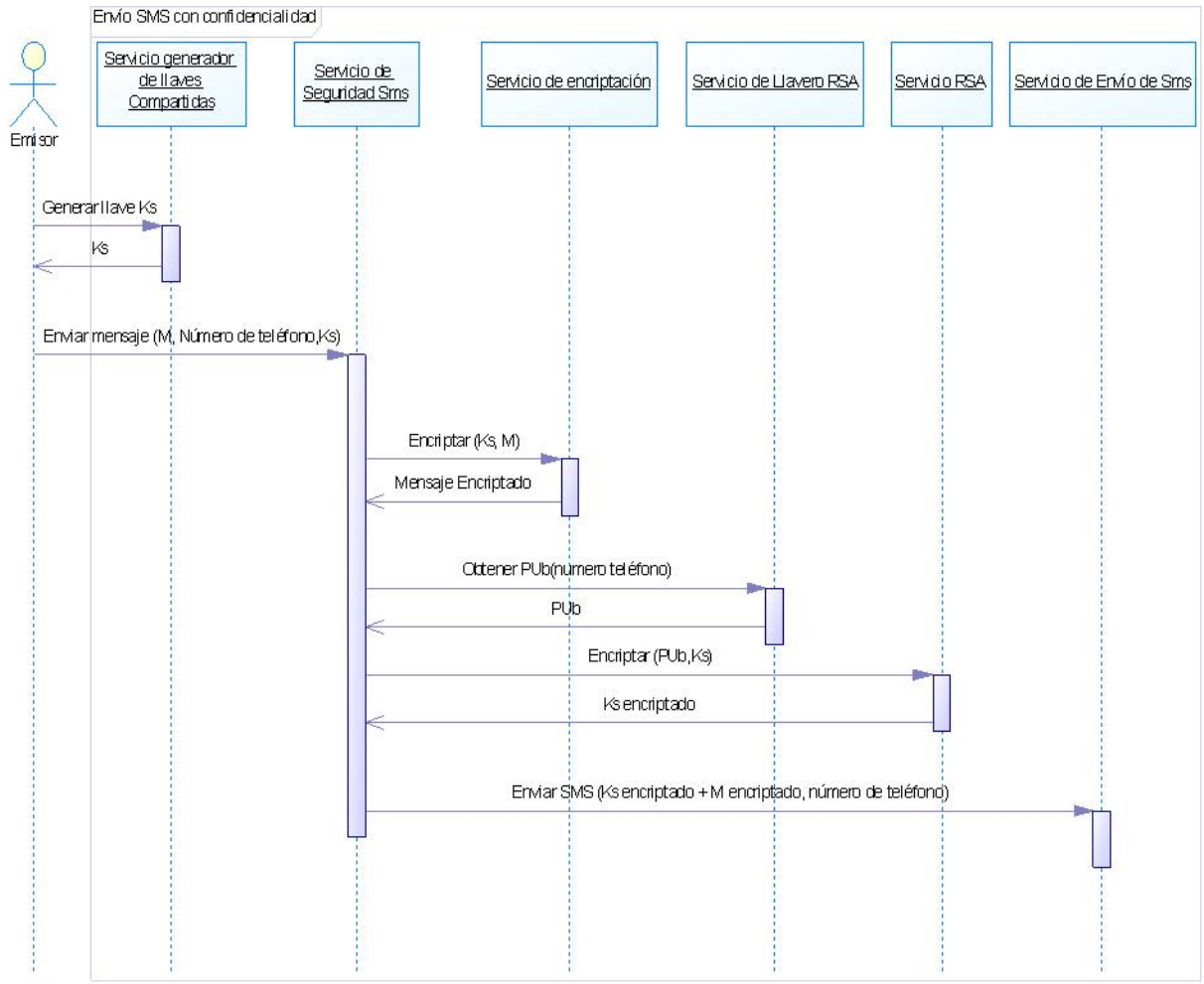

Figura 5.4: Anexo 2 sólo Confidencialidad Envío

.

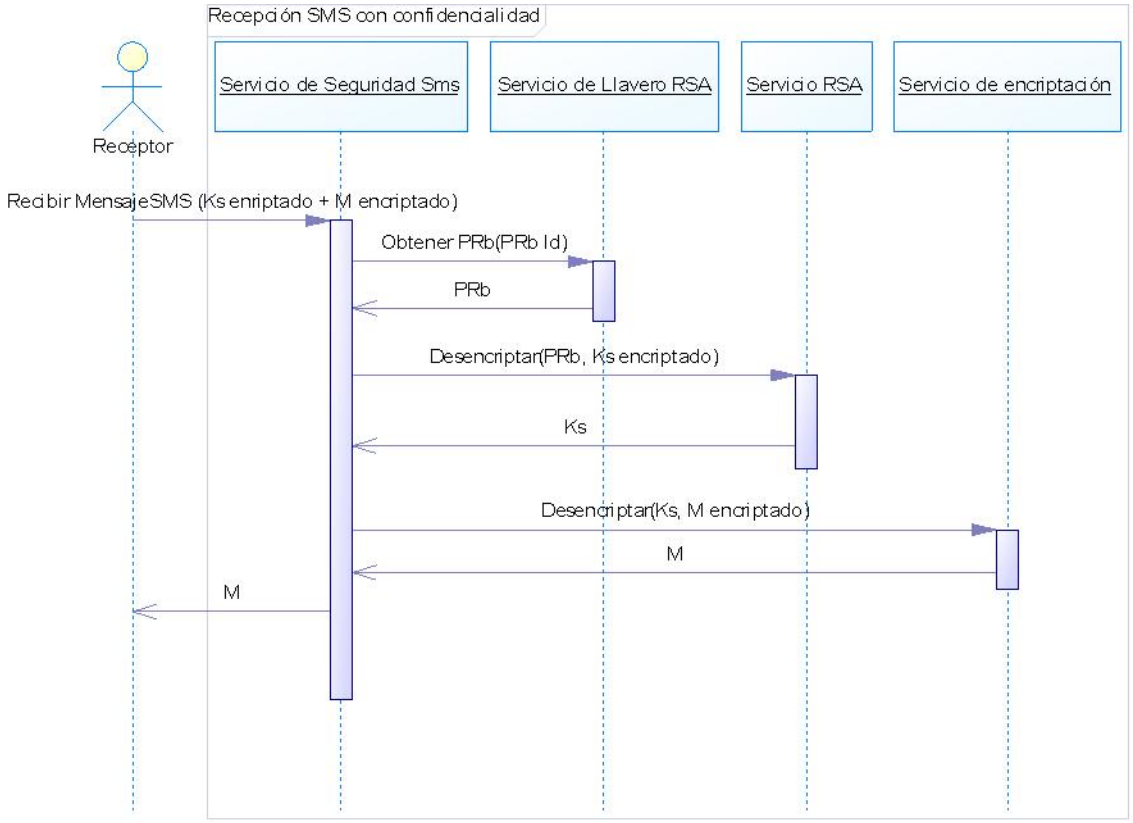

Figura 5.5: Anexo 2 sólo Confidencialidad Recepción

.

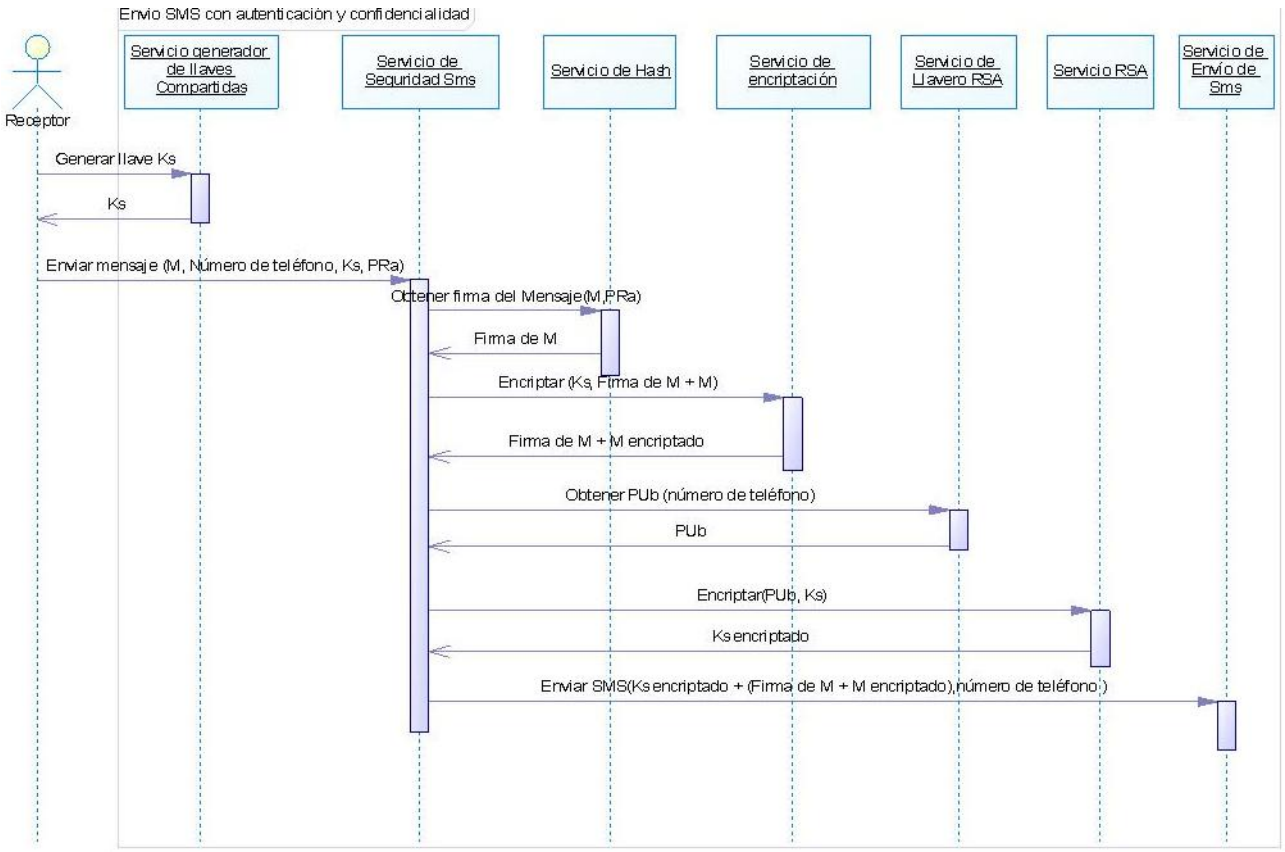

Figura 5.6: Anexo 2 Autenticación y Confidencialidad Envío

.

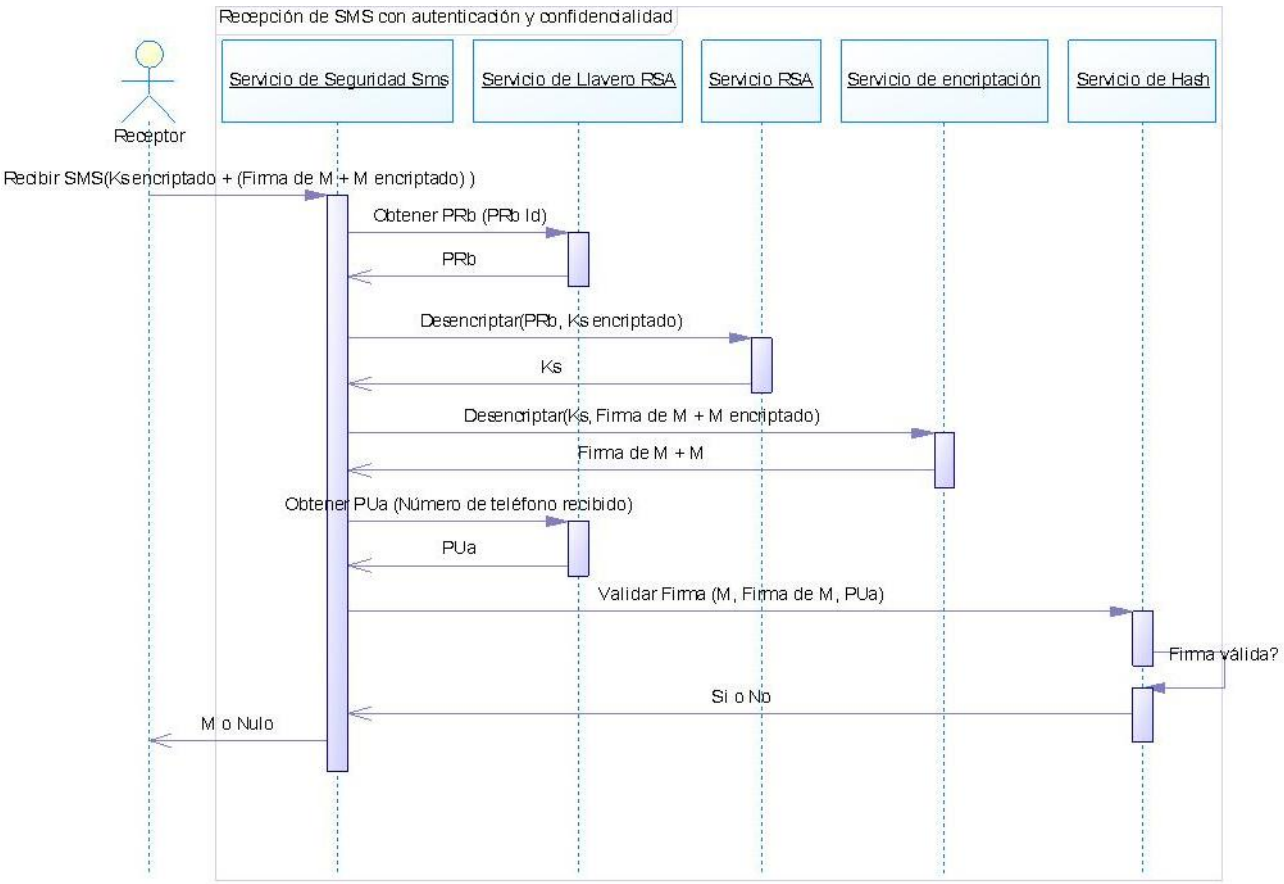

Figura 5.7: Anexo 2 Autenticación y Confidencialidad Recepción

.

Anexo 3

Diagrama de Clases

Nota : Anexo digital en formato CD que viene junto con esta tesis.

Anexo 4

Manual de Usuario de la Biblioteca SMS^2

Este documento es una guía para programadores que desean implementar seguridad en el envío de mensajes SMS usando la biblioteca SMS^2. Este manual empieza con una pequeña descripción de la biblioteca SMS^2, como se debe instalar el ambiente de desarrollo y una descripción de los servicios que ofrece. Si se desea mayor información acerca de esta biblioteca, por favor referirse al anexo Manual Técnico (Anexo 5) de la implementación.

Descripción

La biblioteca SMS^2 brinda seguridad en el envío de mensajes SMS implementando básicamente 3 modos de seguridad tal como lo implementa PGP y S/MIME y son, sólo autenticación, sólo confidencialidad y la combinación de ambos. Además soporta el envió y recepción de mensajes encriptados por una llave compartida. Para el soporte de estos modos de seguridad, la biblioteca brinda funcionalidades extras como el manejo y encolamiento de mensajes, generación de llaves compartidas, generación de llaves públicas y privadas RSA, etc. Esta biblioteca está hecha puramente en Java y funciona para la plataforma J2ME, por lo tanto es soportada por dispositivos móviles que brindan soporte para dicha plataforma que son la mayoría. La biblioteca consta con más de 30 clases que implementan su funcionalidad y se distribuyen en 5 paquetes. Además, funciona gracias al soporte de dos librerías que son indispensables para su funcionamiento :

- BouncyCastle Light API : biblioteca que implementa las funciones criptográficas y es orientada a J2ME.
- WMA *Wireless Messaging API* : API de la Sun que provee acceso a recursos de comunicación SMS en la plataforma J2ME

Instalación del Ambiente de Desarrollo

Para utilizar esta biblioteca se requiere que el programador tenga un ambiente de desarrollo orientado a aplicaciones móviles J2ME con características de optimización para el *Mobile Information Device Profile* (MIDP) y para el *Connected Limited Device Configuration* (CLDC). Una excelente herramienta para el desarrollo de aplicaciones móviles es Netbeans (16) debido a que es muy amigable para el usuario, es modular y tiene el gran soporte para el componente *Mobility Pack*. Este componente sirve para el desarrollo, depuración y publicación de aplicaciones móviles Java Micro Edition (J2ME) y soporta dos tipos de configuraciones :

- CLDC *Connected Limited Device* : Dispositivos con poca memoria y capacidad de procesamiento como por ejemplo teléfonos celulares.
- CDC *Connected Device Configuration* : Dispositivos con mayor memoria y capacidad de procesamiento.

Los pasos para la preparación del ambiente de desarrollo para aplicaciones SMS seguras son :

1. Descargar el IDE Netbeans 6.0 o superior de la página http ://www.netbeans.org. Es importante bajar la versión que tiene soporte para Java ME, debido a que esta ya tiene incluido el Mobility Pack de Netbeans y las librerías necesarias para el desarrollo de aplicaciones móviles. Adicionalmente ya están incluida la biblioteca WMA que da soporte a la mensajería celular por medio del protocolo SMS.
2. El siguiente paso es crear un nuevo proyecto móvil por medio del menú File → New Project → Mobility → MIDP Application.

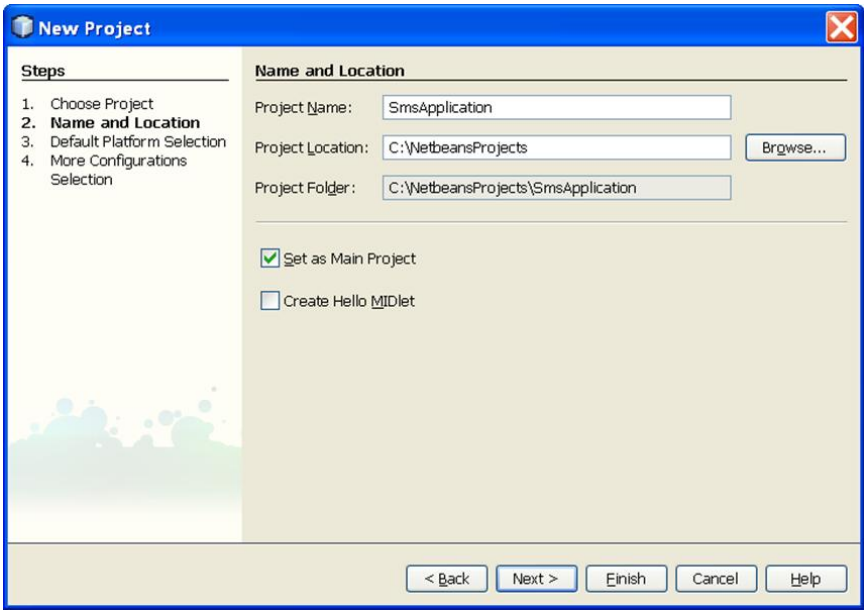

Figura 5.8: Anexo 4 Segundo Paso de Instalación de Ambiente

3. Después de presionar Next en la pantalla anterior, poner el nombre del proyecto, locación del proyecto y presionar Next.

4. En la siguiente pantalla se escoge para que versión de CLDC y de MIDP se desea desarrollar. La versión que se debe escoger depende de la configuración del teléfono en dónde se va a correr la aplicación y presionar "Finish". Este paso es importante porque el JAR que se crea al compilar la aplicación es diferente y es soportado para la configuración establecida. Esta configuración puede ser posteriormente modificada dando click derecho al Proyecto → Propiedades → Plataforma.

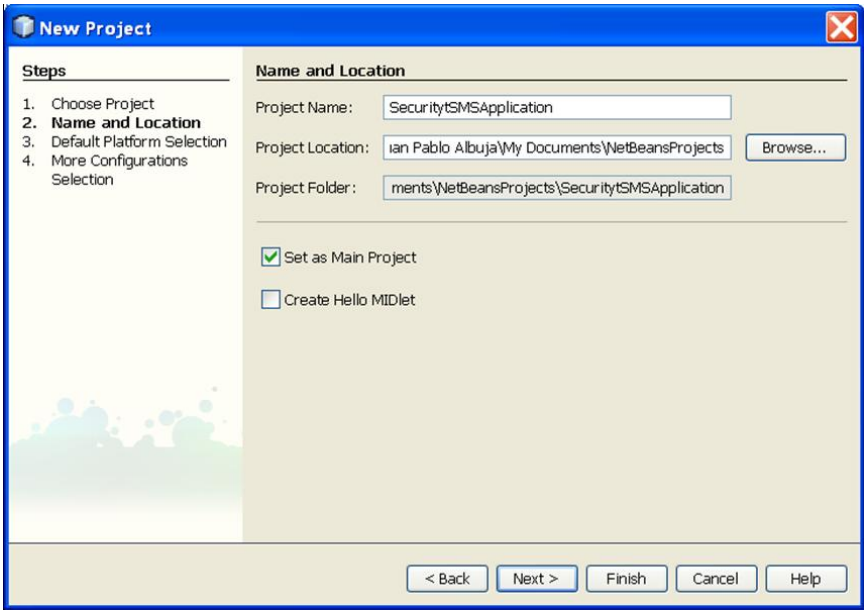

Figura 5.9: Anexo 4 Tercer Paso de Instalación de Ambiente

5. Con el proyecto ya creado se debe añadir el jar SMS2.jar. Este jar tiene incluido al Bouncy Castle 1.38 y clases de soporte para el envío de mensajes SMS seguros. Para cumplir con este paso, dar click derecho sobre los recursos del proyecto recién creado, Add Jar/Zip, y escoger el archivo SMS2.jar. En caso de que si se desea actualizar la versión de Bouncy Castle, se lo debe volver a compilar junto con las clases propias de la biblioteca SMS[2].

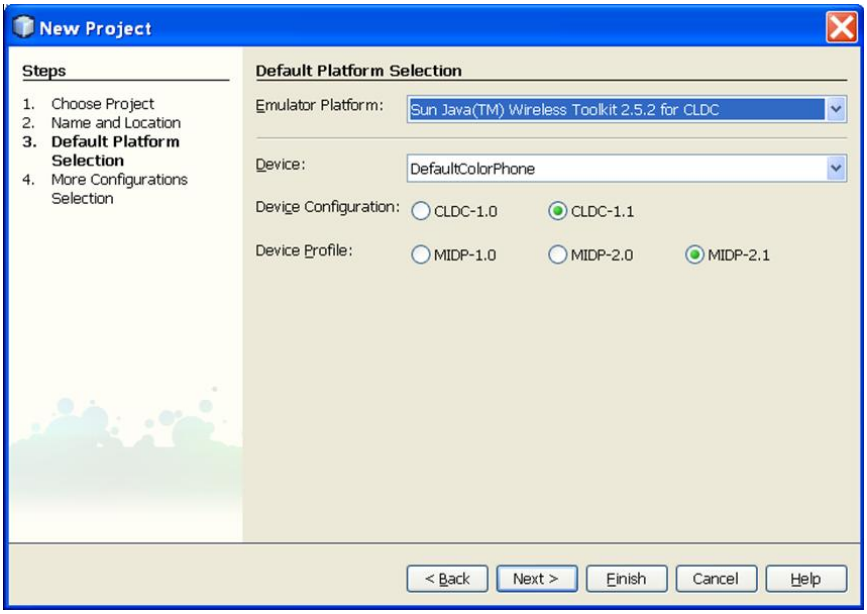

Figura 5.10: Anexo 4 Cuarto Paso de Instalación de Ambiente

6. El último paso es configurar al proyecto con el nivel máximo de ofuscación para el funcionamiento del Bouncy Castle, este paso se debe a que Bouncy Castle sobrescribe ciertas clases del J2SE para su funcionamiento, y el compilador Java no permite que las clases sean nombradas al igual que una clase original de J2SE. Para ofuscar el código click derecho sobre el nombre del Proyecto → Propiedades → Ofuscación y mover la barra a alto y aceptar. Con este último paso estamos listos para empezar a desarrollar.

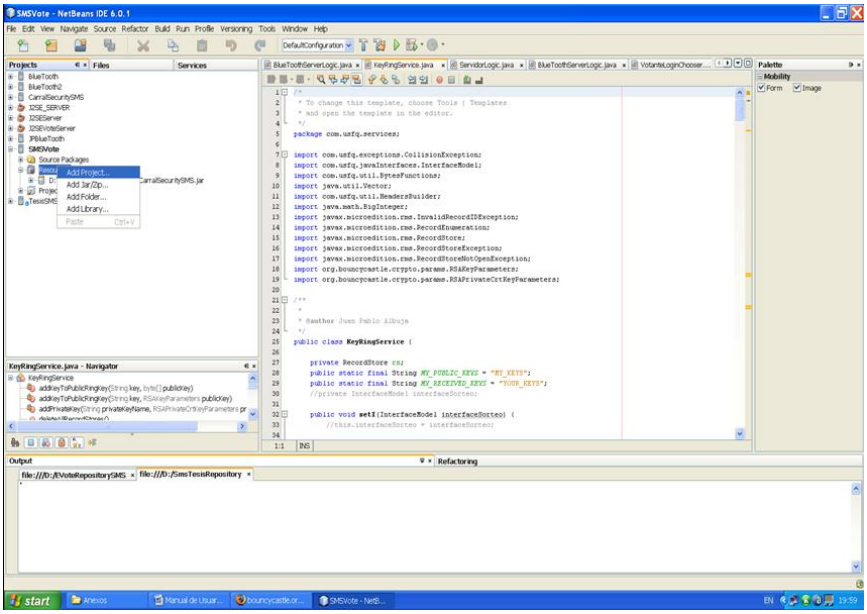

Figura 5.11: Anexo 4 Quinto Paso de Instalación de Ambiente

Descripción de los Servicios Ofrecidos por la Biblioteca

Esta biblioteca contiene 5 paquetes JAVA y son :

- com.usfq.exceptions : Paquete que contiene un conjunto de clases que extienden la clase java.lang.Exception para implementar excepciones personalizadas propias de esta biblioteca.
- com.usfq.javaInterfaces : Paquete que contiene un conjunto de interfaces Java que deben ser implementadas para utilizar ciertos servicios de la biblioteca.
- com.usfq.services : Paquete que contiene las funcionalidades de la biblioteca, las clases que están en este paquete son las que deben ser utilizadas por el programador. Los servicios se los define como mecanismos que permiten tener acceso a una o más funcionalidades o capacidades. Existen servicios que utilizan otros servicios para así convertirse en servicios de alto nivel.
- com.usfq.services.entities : Este paquete contiene tres clases usadas por el servicio de encolamiento de mensajes que será explicado más adelante. Este servicio utiliza una entidad que representa la cola de recepción de mensajes SMS y dos hilos que

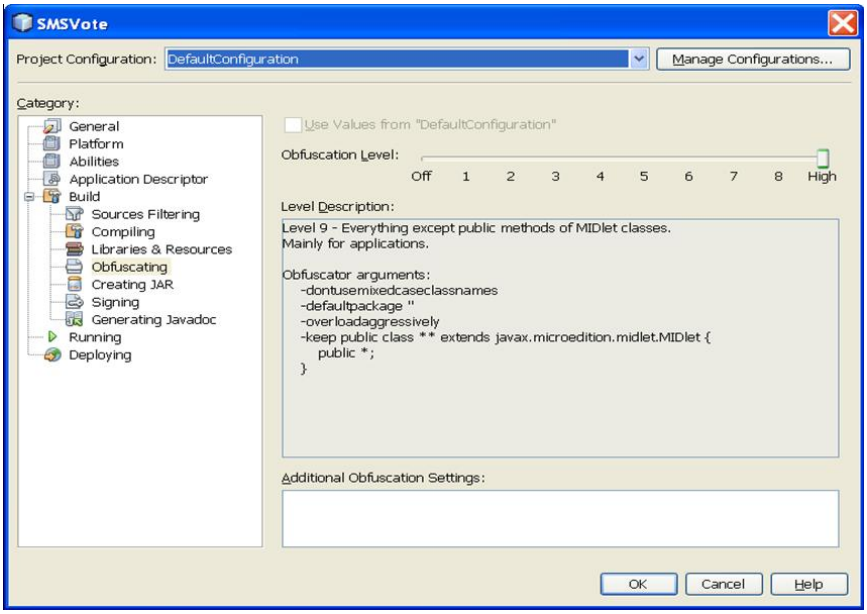

Figura 5.12: Anexo 4 Sexto Paso de Instalación de Ambiente

se encargan de insertar y sacar elementos de la cola.

– com.usfq.util : Paquete que contiene clases que brindan utilidades para el funcionamiento de los servicios como clases que manejan relleno de los mensajes, manipulación de bytes, etc. Estas clases también pueden ser utilizadas directamente por el programador.

En este manual vamos a enfocar nuestra atención a los servicios ofrece la biblioteca y mostrar cómo deben ser usados desde el punto de vista del programador.

1. **Nombre del Servicio :** Servicio de Envío de Mensajes SMS.

   Clase Principal : com.usfq.services. SendSmsService

   Descripción : Este servicio sirve para enviar mensajes SMS binarios o de texto. No es necesario que se especifique el tipo de mensaje debido a que este servicio detecta el tipo de mensaje que se enviará. Puede mandar mensajes SMS del tipo caractéres o bytes, en caso de que el objeto a enviar es de otro tipo se lanzará una excepción.

   Uso : El uso consiste en usar únicamente el método *sendMessage()* de la siguiente manera :

```
1 //Envío de mensaje de texto
2 SendSmsService sendService = new SendSmsService();
3 String mensajeAEnviar ="hola";
4 sendService.sendMessage("SMS://+5550000","5432", mensajeAEnviar);
5 //Envío de binario
6 sendService.sendMessage("SMS://+5550000","5432", mensajeAEnviar.
     getBytes());
7 /*Comprobación si se generó algún error. Si es nulo, no existió
     ningún error.*/
8 sendService.getErrorMessage();
```

2. **Nombre del Servicio :** Servicio de Encolamiento de Mensajes SMS.

   Clase Principal : com.usfq.services.SmsQueueService

   Clases Secundarias : com.usfq.javaInterfaces.NotifierInterface, com.usfq.entities.PopBufferEntity, com.usfq.entities.PushBufferEntity, com.usfq.services.BufferEntity, com.usfq.util. Constants

   Descripción : Su función es encolar los mensajes SMS que van llegando al celular en un Vector y así evitar que mensajes se queden sin procesar. El hilo *PushBufferEntity* se encarga escuchar mensajes SMS entrantes al celular usando el API WMA e insertar los mensajes en un Vector, y otro hilo llamado *PopBufferEntity* se encarga de tomar los mensajes SMS, eliminarlos de la cola y llamar al método *notifyMessageArrive(Object messageObject, String messageType)* de la Interface *NotifierInterface*, dónde messageObject representa el mensaje SMS de texto *javax.wireless.messaging.TextMessage* o binario *javax.wireless.messaging.BinaryMessage* y el messageType describe el tipo del mensaje que puede ser *Constants.TEXT_MESSAGE_TYPE* o *Constants.BINARY_MESSAGE_TYPE*. Además este servicio llama al método *notifyException(Exception e)* de la interface *NotifierInterface* en caso de que exista una excepción. Para concluir, el programador puede suspender temporalmente o

definitivamente la inserción o lectura de mensajes SMS a la cola.

Uso : Su uso es muy simple, suponiendo que la clase que usa este servicio implementa la interface NotifierInterface el código sería el siguiente :

```
1  SmsQueueService  smsQueueService = new SmsQueueService(this, "SMS
       ://:5432" );
2  /*Inicia a los hilos receptor de mensajes SMS y lector del Vector
       */
3  queueService.startReceivingPickingMessages();
4  /*Éste método de la interface es llamado cada vez que un mensaje
       ha llegado al celular */
5  public void notifyMessageArrive(Object messageObject, String
       messageType){
6          if (messageType.equals(Constants.TEXT_MESSAGE_TYPE)) {
7          /*Función del programador que va a manejar el mensaje de
               texto SMS*/
8                  processTextMessage(((TextMessage) messageObject))
                       ;
9          } else{
10         /*Función del programador que va a manejar al mensaje SMS
                binario*/
11         processBinaryMessage(((BinaryMessage) messageObject);
12       }
13 }
```

3. **Nombre del Servicio :** Servicio de Administración de Aplicación

   Clase Principal : com.usfq.services.ApplicationManager

   Clases Secundarias : com.usfq.javaInterfaces.Executer, com.usfq.util.Constants

   Descripción : Este servicio sirve para crear una capa adicional entre el midlet y la lógica del programa, es decir separa la capa de presentación de la lógica. Como parámetro a este servicio se le pasa el nombre completo *fully qualified name* de la clase que va a manejar los mensajes binarios y de la clase que

va a manejar los mensajes de texto. Estas clases deben implementar la interface Executer, debido a que éste servicio llama al método void *execute(Object message, String messageType)* de esta interface dónde message es un objeto *javax.wireless.messaging.TextMessage* o *javax.wireless.messaging.BinaryMessage*, y messsageType tiene el valor de la constante *Constants. TEXT_MESSAGE_TYPE* o *Constants. BINARY_MESSAGE_TYPE*. Además, este servicio llama al método void *setParameters(Hashtable parameters)* de la interface Executer para pasar parámetros a las clases que van a manejar la lógica.

Nota : Como la carga de la clase es en tiempo de ejecución y no en tiempo de compilación, se debe aumentar en las propiedades de ofuscación la opción -keep public class <fully qualified name de la clase>.

Uso : El siguiente ejemplo asume que se está utilizando el servicio de encolamiento SMS y se va a utilizar el administrador de aplicaciones en el método *notifyMessageArrive* del anterior ejemplo :

```
1  //Creación de la instancia ApplicationManager
2  ApplicationManager  appManager = new ApplicationManager("com.usfq
       .applicationLogic.ManejadorTexto", "com.usfq.applicationLogic.
       ManejadorBinario");
3  //Creación de parámetros
4  Hashtable parametersForTextManager = new Hashtable();
5  Hashtable parametersForBinaryManager= new Hashtable();
6  //Método de la interface NotifierInterface
7  public void notifyMessageArrive(Object messageObject, String
       messageType) {
8          try {
9                  if (messageType.equals(Constants.
                       TEXT_MESSAGE_TYPE)) {
10                         appManager.setParameters(
                               parametersForTextManager);
11                         appManager.executeApplication(
```

```
                                messageObject);
12                      } else {
13                              appManager.setParameters(
                                    parametersForBinaryManager);
14                              appManager.executeApplication(
                                    messageObject);
15                      }
16              }
17              catch(Exception e){}
18      }
```

4. **Nombre del Servicio :** Servicio de Encriptación

   Clase Principal : com.usfq.services.EncryptServices

   Descripción : Este servicio implementa encriptación de llave compartida, es decir que con la misma llave se encripta la cadena de caracteres o arreglo de bytes y también se la desencripta. Este servicio implementa 5 diferentes algoritmos, que son “DES”, “DESede”, “IDEA”, “Rijndael”, “Twofish”.

   Uso : Para usar este servicio sólo se debe crear una instancia de la clase *EncryptServices* y usar los métodos *getEncryptedBytes()* y *getEncryptedString()*. Ejemplo :

```
1  //Llave compartida de 256 bytes
2  byte [] sharedKey = "01234567890123456789123456123456".getBytes()
       ;
3  String sharedAlgorithm = EncryptServices.RIJNDAEL;
4  EncryptServices encryptService = new EncryptServices(sharedKey,
       sharedAlgorithm);
5  // Encriptación del texto}
6  byte [] data = encrypt.getEncryptedBytes("hola");
7  // Desencriptación de la cadena de bytes. Retornará "hola".
8  String texto = encryptService.getDencryptBytes(data);
```

5. **Nombre del Servicio :** Servicio RSA

   Clase Principal : com.usfq.services.RSAService.

   Descripción : Este servicio implementa encriptación de llave asimétrica RSA, es decir que con la llave pública se encripta los datos y con la llave privada se desencripta los datos. Este servicio consta de métodos para la generación de llaves pública privada, encriptación y desencriptación de cadena de bytes.

   Uso : Su uso involucra directamente la utilización de la clase del Bouncy Castle *java.math.BigInteger* para representar el exponente público RSA y así proceder a la generación de las llaves pública y privada. Ejemplo :

```
1  // Entidad A instancia la clase RSAService
2  RSAService rsaA = new RSAService();
3  //Exponente BigInteger 65537
4  BigInteger thePublicExponent = new BigInteger("10001", 16);
5  /*Generación de par público privado de 1024 bytes con certeza de
       80*/
6  rsaA.generatePrivateAndPublicKeys(1024, 80, thePublicExponent);
7  /*Una segunda entidad B encripta los datos con la llave pública
       de A para enviar datos seguros a A.*/
8  RSAService rsaB = new RSAService();
9  rsaB.setPublicKey(rsaA.getPublicKey());
10 byte [] encriptado = rsaB.encrypt("hola".getBytes());
11 //A desencripta usando su llave privada.
12 byte [] desencriptado = rsaA.decrypt(encriptado);
```

6. **Nombre del Servicio :** Servicio de Hash

   Clase Principal : com.usfq.service.EncryptHashService

   Descripción : Este servicio se encarga de calcular el hash del mensaje utilizando una gama de algoritmos hash como MD5, SHA1 y firmarlo con la llave privada. Además provee el mecanismo de comprobar la firma del mensaje usando la llave pública de la entidad que envía el mensaje.

Uso : Su uso se basa en instanciar la clase EncryptHashService de la siguiente manera.

```
1  //Instancia el hash con el algoritmo SHA1Digest
2  EncryptHashService hashService = new EncryptHashService(
       EncryptHashService.SHA1DIGEST);
3  //Creación del par RSA
4  RSAService rsaA = new RSAService();
5  BigInteger thePublicExponent = new BigInteger("10001", 16);
6  rsaA.generatePrivateAndPublicKeys(256, 80, thePublicExponent);
7  //Generación de la firma
8  byte [] signedHash = hashService.RSASign("hola".getBytes(), rsaA.
       getPrivateKey());
9  //Comprobación de la firma. Retorna true si es válida la firma.
10 boolean valid = hashService.RSAVerify("hola".getBytes(),
       signedHash, rsaA.getPublicKey());
```

7. **Nombre del Servicio :** Servicio de Llavero RSA

   Clase Principal : com.usfq.services.KeyRingService

   Descripción : Este servicio se encarga de almacenar llaves RSA en la memoria no volátil del celular. El usuario no tiene que lidiar con la forma de cómo se almacena, sólo tiene que usar este servicio como si fuera un Hashtable, es decir almacena las llaves con un key o clave usando el método *put()* y toma las llaves con el método *get()*. La única restricción es que el key o identificador del almacenamiento debe tener máximo 127 caracteres o se producirá una excepción. En caso de que se guarde una llave con un nombre existente se sobrescribirá con la nueva llave. Se aconseja almacenar a las llaves públicas recibidas con nombre igual al número de teléfono de la entidad emisora. Cabe recalcar que al momento de almacenar una llave pública, se debe indicar si es llave pública generada por la entidad local, o es llave pública recibida. Ejemplo :

   Uso :

```
1  //Entidad A genera llave pública privada
2  RSAService rsaA = new RSAService();
3  BigInteger thePublicExponent = new BigInteger("10001", 16);
4  rsaA.generatePrivateAndPublicKeys(256, 80, thePublicExponent);
5  KeyRingService keyRingService = new KeyRingService();
6  //Almacenamiento de llaves generadas
7  keyRingService.putPrivateKey("miLlavePrivada", rsaA.getPrivateKey
       ());
8  keyRingService.putPublicKeyObject("miLlavePublica", rsaA.
       getPublicKey(), KeyRingService.MY_PUBLIC_KEYS);
9  /*Recuperación de llaves almacenadas en memoria no volátil*/
10 RSAPrivateCrtKeyParameters privateKey = keyRingService.
       getPrivateKeyByName("miLlavePrivada");
11 RSAKeyParameters publicKey = keyRingService.getPublicKeyObject(
       KeyRingService.MY_PUBLIC_KEYS, "miLlavePublica");
12 /*Entidad B con número de teléfono +593973675 78 genera llave
       pública privada y envía a A.*/
13 RSAService rsaB = new RSAService();
14 BigInteger thePublicExponentB = new BigInteger("10001", 16);
15 rsaA.generatePrivateAndPublicKeys(256, 80, thePublicExponentB);
16 /*Entidad A recibe llave pública y almacena la llave recibida de
       B en la memoria no volátil del teléfono*/
17 keyRingService.putPublicKeyObject("+59397367578", rsaB.
       getPublicKey(), KeyRingService.MY_RECEIVED_KEYS);
18 //Entidad A lee de la memoria no volátil la llave almacenada del
       teléfono +593973675 78
19 RSAKeyParameters publicKeyOfB = keyRingService.getPublicKeyObject
       (KeyRingService.MY_RECEIVED_KEYS, "+59397367578");
```

8. **Nombre del Servicio :** Servicio Generador de Llaves Compartidas

   Clase Principal : com.usfq.services.RandomBytesKeyGenerator

   Descripción : Este servicio genera una cadena randómica de bytes dada una

longitud en bits. Los tamaños permitidos son : 64, 128, 160, 192, 224, 256, 512 bits. Si se trata de genera una cadena de algún tamaño que no se encuentre en esta lista se producirá una excepción *RandomBytesKeyGeneratorException*.

Uso :

```
1 //Instancia de la clase
2 RandomBytesKeyGenerator keyGenerator = new
     RandomBytesKeyGenerator();
3 //Generación de una llave de 256 bits. //
4 byte[] sharedKey = keyGenerator.generateKey(
     RandomBytesKeyGenerator.KEY256_BITS_LENGTH);
```

9. **Nombre del Servicio :** Servicio de Seguridad SMS.

   Clase Principal : com.usfq.services.SmsSecurityService

   Clases Secundarias : com.usfq.util.BytesFunctions, com.usfq.util.HeadersBuilder

   Servicios Usados : Servicio de envío de mensajes SMS, servicio de encriptación, servicio RSA, servicio de Llavero RSA y servicio de Hash

   Descripción : Este es el servicio más útil de toda la biblioteca ya que integra todos los servicios en una sola clase y tiene métodos para enviar SMS seguros y recibir estos mismos. Este servicio realiza el envío de mensajes SMS utilizando tres niveles de seguridad como PGP, autenticación, confidencialidad y la combinación de los dos. Además esta clase provee mecanismos para envió y recepción de llaves RSA públicas y el envió y recepción de mensajes criptografados por llave compartida.

   Uso Envío de Mensajes Seguros : Para el envío de los mensajes seguros y llaves RSA el servicio utiliza internamente la clase HeadersBuiler. Lo que hace esta clase es crear un encabezado de acuerdo al tipo de mensaje que se va a enviar. Por ejemplo el siguiente código ilustra cómo se envía un mensaje de seguro que tenga confidencialidad y autenticación.

```
1 // Instancia de la clase SmsSecurityService
```

```
2 SmsSecurityService securitySercvice=new SmsSecurityService();
3 //Generación de la llave compartida
4 byte[] sharedKey = keyGenerator.generateKey(
     RandomBytesKeyGenerator.KEY64_BITS_LENGTH);
5 /* Envio del mensaje seguro. Este método asume que existe una
     llave pública recibida y guardada con el número de teléfono
     +59397367578 como clave */
6 securityService.sendAuthenticatedConfidencialMessage("
     stringToSend", keyRingService.getPrivateKeyByName("
     myPrivateKey"), sharedKey, EncryptServices.DES,
     EncryptHashService.MD2DIGEST, "+59397367578", "5432");
```

El método sendAuthenticatedConfidentialMessage hace lo siguiente :

(a) Busca en la memoria no volátil usando la clase KeyRingService una llave pública recibida con la clave "+59397367578".

(b) Firma al mensaje "stringToSend" con la llave privada pasada como parámetro, en este caso usa la llave llamada "myPrivateKey" para generar S.

(c) Concatena S + M, dónde M es el mensaje stringToSend.

(d) Encripta S + M con la llave de sesión *sharedKey* usando el algoritmo DES.

(e) Encripta con la llave pública de la otra entidad a la llave de sesión.

(f) Envía un mensaje SMS binario que tiene S + M encriptado concatenado la llave de sesión encriptada.

Otra manera de enviar el mismo mensaje y pasando como parámetro la llave pública con la cual se desea encriptar la llave compartida se usa el siguiente método.

```
1 public byte[] sendAuthenticatedConfidencialMessage(String
     smsMessageText, RSAPrivateCrtKeyParameters privateKey, byte[]
     sharedKey, String sharedAlgorithm, String hashAlgorithm,
     RSAKeyParameters otherEntityPublicKey, String address, String
     port)
```

dónde RSAKeyParameters es la llave pública de la entidad a la cual se le está enviando el mensaje seguro. Así mismo existen métodos de esta clase para enviar mensajes con sólo autenticación y sólo confidencialidad y son :

Sólo Autenticación :

```
1 /*Asume que la llave pública de la otra entidad está almacenada
     con la clave address usando el servicio KeyRingService. Este
     método busca la llve pública de la otra entidad usando el
     parámetro addres de éste método.*/
2 public void sendAuthenticatedSms(String smsMessageText,
     RSAPrivateCrtKeyParameters privateKey, RSAKeyParameters
     otherEntityPublicKey, String hashAlgorithm, String address,
     String port)
3 /*O también */
4 /* Manera explícita de enviar el mensaje. Se pasa como parámetro
     la llave pública de la otra entidad*/
5 public void sendAuthenticatedSms(String smsMessageText,
     RSAPrivateCrtKeyParameters privateKey, String hashAlgorithm,
     String address, String port)
```

Note : Que éste modo de seguridad también requiere la llave pública de la otra entidad. Es para adjuntar en el header el módulo de la llave pública y así la otra entidad (receptor), en caso de que tenga varias llaves privadas, sepa cuál llave privada debe usar para firmar el mensaje de respuesta.

Sólo Confidencialidad :

```
1 /*Asume que la llave pública de la otra entidad está almacenada
     con el parámetro address del método que está a continuación.
     Forma implícita de enviar el mensaje*/
2 public void sendConfidencialMessage(String smsMessageText, byte[]
      sharedKey, String sharedAlgorithm, String address, String
     port)
3 /*O también */
```

```
4 //Forma explícita de enviar el mensaje
5 public void sendConfidencialMessage(String smsMessageText, byte[]
      sharedKey, RSAKeyParameters publicKey, String sharedAlgorithm
     , String address, String port)
```

Envío de Llave RSA :

```
1 public void sendRSAPublicKey(RSAKeyParameters publicKeyToSend,
     String address, String port)
```

Envío SMS Protegido por Llave Compartida :

```
1 sendConfidencialSharedMessage(String smsMessageText, byte[]
     sharedKey, String sharedAlgorithm, String address, String port
     )
```

Uso Recepción de Mensajes Seguros

La recepción de mensajes seguros y de llaves públicas RSA es más sencilla aún, debido a que el emisor ya envía en los bytes de preámbulo del mensaje, información de cuál es el algoritmo de encriptación simétrica, cuál algoritmo de hash es, etc.

Con el siguiente método se ilustra cómo se debe recibir un mensaje enviado con autenticación y confidencialidad.

```
1 String message = receiveAuthenticatedConfidencialMessage(
     BinaryMessage binaryMessage)
```

El anterior método es una forma de recibir el mensaje seguro de manera implícita. Es implícita debido a que asume que la llave pública de la entidad que envía el mensaje se encuentra almacenada en la memoria no volátil por medio del uso del método *keyRingService.putPublicKeyObject("+59397367578", publicKeyToStore KeyRingService.MY_RECEIVED_KEYS)* de la clase KeyRingService, cuya clave es el número de teléfono emisor. La manera explícita es cuando se pasa como parámetro

la llave pública con la cual se desea recibir el mensaje. El proceso que realiza el método anterior es el siguiente :

(a) Obtiene la llave pública de la otra entidad almacenada en la memoria no volátil, por medio del servicio KeyRingService.

(b) Obtiene la llave privada que debe usar. La forma de realizar esta tarea es leer de los bytes de preámbulo del mensaje recibido para obtener el identificador de cuál llave privada debe obtener de su llavero para desencriptar.

(c) Desencripta la llave de sesión con su llave privada RSA.

(d) Utiliza la llave de sesión para desencriptar y obtener S + M.

(e) Con la llave pública de la otra entidad verifica la firma, en caso de que sea correcto el método retorna M o caso contrario nulo.

O también se puede recibir de una forma más explícita usando el método

```
1 public String receiveAuthenticatedConfidencialMessage(byte[]
    binaryReceivedMessage, RSAPrivateCrtKeyParameters privateKey,
    RSAKeyParameters otherEntityPublicKey)
```

Así mismo para la recepción de las otras formas de seguridad existe :

Recepción de Mensaje Seguro sólo Autenticación :

```
1 /*Asume que la llave pública de la otra entidad está almacenada
    en la memoria no volátil. Como se mencionó anteriormente, es
    una manera implícita de recibir el mensaje debido a que este
    método busca qué llave pública está almacenada para el emisor
    del mensaje por medio del uso del servicio KeyRingService.*/
2 receiveAuthenticatedMessage(BinaryMessage receivedBinaryMessage)
3 /*O también */
4 /*Forma explícita de recepción del mensaje. Se pasa como
    parámetro la llave pública con la cual se desea recibir el
    mensaje*/
5 String receiveAuthenticatedMessage(byte[] binaryReceivedMessage,
    RSAKeyParameters publicKey)
```

Recepción de Mensaje Seguro sólo Confidencialidad :

```
1 // Forma implícita de recepción
2 receiveConfidencialMessage(BinaryMessage binaryMessage)
3 /*O también */
4 // Forma explícita de recepción
5 public String receiveConfidencialMessage(byte[]
    binaryReceivedMessage, RSAPrivateCrtKeyParameters privateKey)
```

Recepción de Llave RSA :

```
1 public RSAKeyParameters reveiveRSAPublicKey(byte[] publicKeyBytes
    )
```

Recepción de Mensaje Protegido por Llave Compartida :

```
1 public String receiveConfidencialSharedMessage(byte[]
    binaryReceivedMessage, byte[] sharedKey)
```

Existe también un servicio que es complementario a éste, se llama Servicio Analizador de Encabezados y está representado por la clase *com.usfq.services.HeaderReceiveAnalizerService*. Este servicio sirve para que al recibir el mensaje binario, no nos preocupemos de cuál receptor de la clase SmsSecurityService debemos llamar. Este servicio analiza los encabezados y llama automáticamente al receptor correspondiente, caso contrario lanza una excepción *HeaderReceiveAnalyzerServiceException()*. La forma de cómo se lo usa es de la siguiente manera :

```
1 HeaderReceiveAnalizerService analyzer = new
    HeaderReceiveAnalizerService()
2 String receivedMessage = analyzer.getAnalizedMessageReceive((
    BinaryMessage) messageObject);
```

Este servicio analizador de encabezados tiene extractores muy importantes que nos dan información sobre la última operación realizada como :

– El tipo de mensaje recibido :

```
1 /*Valores posibles son: Constants.CONFIDENTIAL_MODE, Constants.
     AUTHENTICATION_MODE, Constants.AUTHCONF_MODE, Constants.
     SHAREDKEY_MODE, Constants.RSA_KEY*/
2 analyzer.getRecognizedMode()
```

– El algoritmo criptográfico de llave compartida y la llave compartida : Esta información está a disposición cuando se recibe mensajes seguros de sólo confidencialidad, autenticación y confidencialidad, y seguridad con llave compartida :

```
1 public String getCryptoAlgorithmName()
2 public byte[] getSharedKey()
```

– El algoritmo hash utilizado : Esta información está a disposición cuando se recibe mensajes seguros con sólo autenticación y autenticación más confidencialidad.

```
1 public String getHashAlgorithmName()
```

– El módulo de la llave privada a buscar : Este valor está disponible para sólo autenticación, sólo confidencialidad y autenticación más confidencialidad.

```
1 public String getModuledModule()
```

Anexo 5

Manual Técnico

Nota : Anexo digital en formato CD que viene junto con esta tesis.

Anexo 6

Glosario de Términos

- 3DES (*Triple Data Encryption Standard*) : Algoritmo criptográfico simétrico creado por la IBM en 1978 (33).
- AES (*Advanced Encryption Standard*) : Algoritmo de criptografía simétrica adoptado como estándar de cifrado por el Gobierno de los Estados Unidos (34).
- AI (*Air Interface*) : Enlace de comunicación de radio entre el celular y la estación base (31).
- BSC (*Base Station Controller*) : Componente de la red celular que se encarga de la transmisión de tráfico de voz y datos entre los dispositivos móviles y el MSC (31).
- BSS (*Base Station Subsystem*) : Componente de la red celular compuesta por el BTS y BSC (31).
- BTS (*Base Transceiver Station*) : Componente de la red celular que define cada célula y esta incluye una radio antena, un radio transmisor y un enlace al BSC (31).
- CLDC (*Connected Limited Device Configuration*) : Plataforma JAVA que provee un conjunto de funcionalidades para correr aplicaciones JAVA en dispositivos móviles con recursos limitados (19).
- HLR (*Home Location Register*) : Base de datos utilizada en la red celular dónde se almacena información permanente y es usado constantemente para controlar los perfiles de los subscriptores (31).
- J2ME (*Java 2 Micro Edition*) : Plataforma robusta que provee un ambiente flexible para correr aplicaciones en dispositivos móviles (19).
- MD5 (*Message Digest 5*) : Algoritmo de hash desarrollado por el Instituto Tecnológico de Massachusetts en 1991 (26).

- MIDP (*Mobile Information Device Profile*) : Conjunto de APIS que implementan funcionalidades en dispositivos móviles (19).
- MSC (*Mobile Switchig Center*) : Componente de la red celular que se encarga de realizar las funciones de conmutación del sistema, y controlar llamadas a o desde otros teléfonos y sistemas de datos (31).
- PGP (*Pretty Good Privacy*) : Protocolo de seguridad de e-mail desarrollado en 1991 por el estadounidense Phil Zimmermann (27).
- SHA1 (*Secure Hash Algorithm 1*) : Algoritmo de hash dessarrollado en 1995 por Instituto Nacional de Estándares y Tecnología (28).
- SMS (*Short Message Service*) : Servicio bidireccional soportado por la red celular que sirve para el envío y recempción de mensajes cortos desde dispositivos móviles como celulares (31).
- SMSC (*Short Message Service Center*) : Componente de la red celular que es la combinación de software y hardware que se encarga del almacenamiento y reenvío de mensajes de texto cortos entre dispositivos móviles (31).
- S/MIME (*Secure Multipurpose Internet Mail Extensions*) : Protocolo de Seguridad de e-mail desarrollado originalmente por la compañía RSA Data Security en 1999 (25).
- RSA (*Rivest Shamir Adleman*) : Algoritmo de criptograía asimétrica dessarrollado en 1983 por el Instituto Tecnológico de Massachusetts (29).
- Symbian OS : Sistema operativo que fue producto de la alianza de varias empresas de telefonía móvil, entre las que se encuentran Nokia y Sony Ericsson (32).
- VLR (*Visitor Location Register*) : Base de deatos utilizada en la red celular dónde se almacena información temporal de los subscritores que físicamente se encuentran cubiertos por el MSC (31).

Anexo 7

Código Fuente del Proyecto

Nota : Anexo digital en formato CD que viene junto con esta tesis.

Referencias

[1] Android - An Open Hadset Alliace Project. What is Android. http://code.google.com/android/what-is-android.html, Enero 2009. 17

[2] Apple Inc. iPhone Dev Center. http://developer.apple.com/iphone/index.html, Enero 2009. 18

[3] BANCO PICHINCHA. Portal Banco Pichincha. http://wwwp2.pichincha.com/web/index.php, Enero 2006. 15

[4] Barbi, L. Spider SMS : Sending and reception of encrypted SMS. http://www.lucabarbi.it/lec/spidersms.htm, Enero 2009. 16

[5] Jhony Li Judith Bishop. Smssec : an end-to-end protocol for secure sms. *Universidad de Petroria*, 1(1) :5–7, 2005. 13

[6] Fortress. Fortress SMS. http://www.fortressmail.net/fortresssms.htm, Enero 2009. 16

[7] Frecuencia Online. Movistar Ecuador : un caso de éxito mundial en sms. http://www.espanol.frecuenciaonline.com, Enero 2006. 2

[8] GSM-ESPIA. Espía Monitoreo Sin Límites. http://www.gsmespia.com/celular_sms_interceptor.htm, Febrero 2008. 13

[9] Scoot B. Guttery and Mary J Cronin. *Mobile Application Development with SMS and SIM toolkit.* MC Graw-Hill, New York., 2002. 1

[10] Java Community Process. Wireless Messaging Api . `http://java.sun.com/products/wma/index.jsp`, Marzo 2003. 16, 19

[11] Jukka Laurila. Python for S60. `http://opensource.nokia.com/projects/pythonfors60/`, Enero 2006. 17

[12] Portio Research Ltd. Mobile messaging futures 2009-2013. *Portio Research Ltd*, 2009. 2

[13] MacNN Media. How to code to send and receive SMS using iPhone SDK. `http://forums.macnn.com/79/developer-center/375045/how-code-send-receive-sms-using/`, Enero 2009. 18

[14] María A. Mendoza Sanchez. Metodologías De Desarrollo De Software. `http://www.informatizate.net/articulos/pdfs/metodologias_de_desarrollo_de_software_07062004.pdf`, Junio 2004. 20

[15] Netbank. Security. `http://www.commbank.com.au/help/faq/netbank/security.aspx`, Enero 2009. 2

[16] Netbeans. Netbeans IDE. `www.netbeans.com`, Enero 2009. 57

[17] PrivyLink. Mobile payment. `http://www.privylink.com.sg/products`, Enero 2009. 16

[18] William Stallings. *Cryptography and Network Security Principles and Practices*. Prentice Hall, New Jersey., 2005. x, 5, 6, 7

[19] Sun Microsystems Inc. Java Technology. `http://java.sun.com`, Junio 2008. 78, 79

[20] Symbian Software Ltd. Sybiam Os. `http://www.symbian.com/index.asp`, Enero 2008. 17

[21] Tech Crunchies. How Many SMSes Sent WorldWide. `http://techcrunchies.com`, Enero 2009. 1

[22] The Legion of Bouncy Castle. The Legion of Bouncy Castle. `http://www.bouncycastle.org/java.html`, Octubre 2008. 3, 19

[23] TIA. Short message services for spread spectrum systems. TIA/EIA-637-B, Telecommunications Industry Association, Enero 2002. 9

[24] Upsolve. The Power of SMS. `http://www.upsolv.com`, Junio 2006. x, 1, 2

[25] Wikipedia Enciclopedia. S MIME. `http://es.wikipedia.org/wiki/S/MIME`, Diciembre 2008. 79

[26] Wikipedia Enciclopedia. MD5. `http://es.wikipedia.org/wiki/Algoritmo_MD5`, Marzo 2009. 78

[27] Wikipedia Enciclopedia. MD5. `http://es.wikipedia.org/wiki/PGP`, Marzo 2009. 79

[28] Wikipedia Enciclopedia. MD5. `http://es.wikipedia.org/wiki/SHA`, Marzo 2009. 79

[29] Wikipedia Enciclopedia. RSA. `http://es.wikipedia.org/wiki/Claves_RSA`, Abril 2009. 79

[30] Wikipedia Enciclopedia. Seguridad Informática. `http://es.wikipedia.org/wiki/Seguridad_informatica`, Enero 2009. 4

[31] Wikipedia Enciclopedia. Servicio de Mensajes Cortos. `http://es.wikipedia.org/wiki/Servicio_de_mensajes_cortos`, Marzo 2009. 12, 78, 79

[32] Wikipedia Enciclopedia. Symbian OS. `http://es.wikipedia.org/wiki/Symbian`, Abril 2009. 79

[33] Wikipedia Enciclopedia. Triple DES. `http://es.wikipedia.org/wiki/3DES`, Abril 2009. 78

[34] Wikipedia Enciclopedia. Triple DES. `http://es.wikipedia.org/wiki/Advanced_Encryption_Standard`, Mayo 2009. 78

[35] Wikipedia Enciclopedia. X 800. `http://es.wikipedia.org/wiki/X.800`, Enero 2009. 4

[36] Lan Willians. Bus Passengers buy tickets via SMS. `http://www.computing.co.uk/vnunet/news/2206717/bus-passengers-mobile-tickets`, Enero 2008. 2