Sistemas Informáticos

Curso 2007-2008

Investigación y Desarrollo de Aplicaciones Criptográficas en Symbian

Bernardo Chenlo García

Marta Garbayo Sesma

Elisa Sanz Troyano

Dirigido por:

Prof. Luis Javier García Villalba

Facultad de Informática

Universidad Complutense de Madrid

ÍNDICE

1. PRESENTACIÓN

Este documento pretende mostrar la portación, diseño y construcción de una herramienta de comunicación sobre terminales móviles, segura, confiable y de fácil manejo, orientada a las necesidades de autenticidad y no repudio de las sociedades de información actuales.

¿Qué es lo que hace, entonces, CryptoSMS? Esta primera versión permite a los usuarios de la aplicación mantener un medio de comunicación privado y fiable mediante mensajes de texto y mensajes multimedia y restringirlo a un grupo conocido de personas. Siguiendo la filosofía de su antecesor, CryptoSMS utiliza algoritmos de encriptación de tipo simétricos para el envío de mensajes y proporciona mecanismos de intercambio de claves entre contactos que se almacenan en una agenda privada en cada terminal.

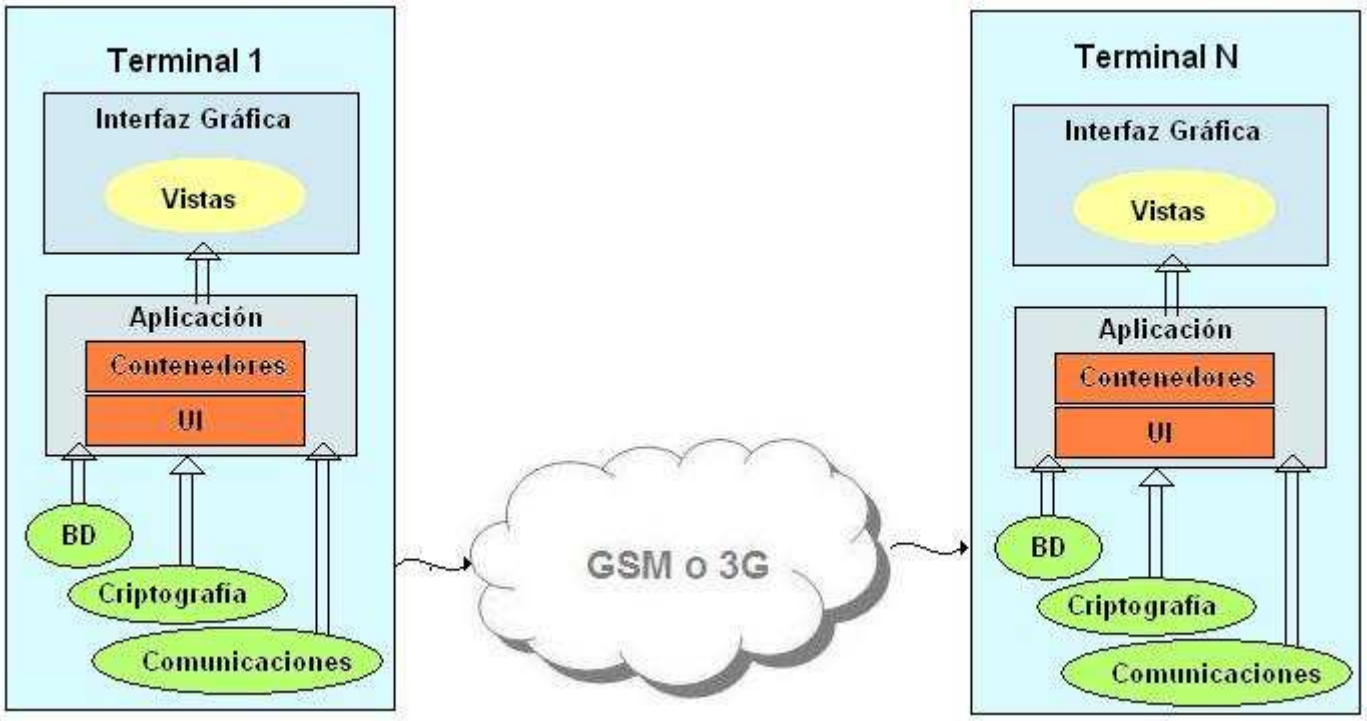
Nuestros objetivos dentro del proyecto han estado dedicados a la parte de criptografía:

Crear una API en lenguaje Symbian intuitiva, segura y robusta para permitir la encriptación y desencriptación de mensajes de texto.

Actualmente, las API´s existentes están destinadas a la criptografía en general, no están diseñadas específicamente para el módulo de SMS y MMS. Sería una de las primeras API´s diseñadas para este propósito.

2. ARQUITECTURA

En una primera aproximación a la aplicación se muestra el siguiente dibujo:



Existen tres módulos fundamentales diferenciados con tareas específicas, el módulo de base de datos, el módulo de Criptografía, y el módulo de comunicaciones. Todos ellos se usan e interactúan a través de la parte del control de la aplicación, que se encarga por otra parte del control de vistas de la interfaz gráfica.

Para la arquitectura de la interfaz gráfica se ha tenido en cuenta que la aplicación tiene mucha interacción con el usuario y constantemente hace uso de la entrada - salida para mostrar o pedir información externa y avanzar con la ejecución.

Symbian propone varias alternativas prediseñadas para la creación de aplicaciones.

- Arquitectura tradicional.
- Basadas en Dialogos.
- Arquitectura Avkon.

La arquitectura tradicional tiene la desventaja de no proporcionar manejo de vistas, así que es necesario implementarlo aparte. La arquitectura basada en Diálogos proporciona una gran independencia entre aplicaciones que comparten funcionalidad, hace que cada aplicación sea

más eficiente y reduce la complejidad entre aplicaciones simples; pero no puede ser usada en todos los casos y es incompatible con el uso de vistas.

En la arquitectura de Vistas Avkon cada vista maneja sus propios eventos y se proporciona una serie de métodos para el control y cambio entre vistas, de tal manera que el entorno que rodea a cada vista puede ser distinto. Se ha elegido la Arquitectura Avkon por la flexibilidad y modularidad que ofrece junto con Diálogos para mejorar el rendimiento de la aplicación.

Esta arquitectura sigue el modelo tradicional y añade el control de vistas.

2.1 MODULO DE CRIPTOGRAFÍA

Se ha estructurado el trabajo orientándolo a la creación de una interfaz que pueda ser usado fácilmente por los grupos encargados de realizar la GUI y el almacenamiento.

Nuestro objetivo es conseguir la implementación de una API destinada a la encriptación de mensajes de texto y mensajes multimedia.

En dicha API, se incluirán los métodos estrictamente necesarios para llevar a cabo la encriptación.

Se intenta en todo momento que la API sea lo suficientemente segura y robusta como requiere una aplicación de este tipo.

Dado que la aplicación responde a la necesidad de transferir información delicada, la API criptográfica no debe permitir el acceso a partes sensibles de los algoritmos de encriptación.

Para lograr lo anterior, se intenta crear una interfaz clara, sencilla, intuitiva y rigurosamente documentada.

Comenzamos estudiando en profundidad los algoritmos de encriptación empleados en la aplicación:

Para implementar los algoritmos de cifrado, la aplicación hace uso de la librería criptográfica Bouncy Castle.

Bouncy Castle es un proyecto de software libre que pretende desarrollar una serie de librerías criptográficas libres y, entre otros, ofrece un *provider* para el JCE de java.

En la página web del proyecto: http://www.bouncycastle.org/ es posible descargar la versión actual del provider Bouncy Castle para distintas versiones de la máquina virtual de Java.

Ofrece un API (*application programming interface*) que permite:

- Generación de claves (claves secretas y pares de claves pública y privada).
- Cifrado simétrico (DES, 3DES, IDEA, etc).
- Cifrado con curva elíptica (ECIES).
- Cifrado simétrico (RSA, DSA, Diffie-Hellman, ElGamal...).
- Funciones de resumen (MD5 y SHA1) y algoritmos MAC (Message Authentication Code).
- Acuerdo de claves.

Dentro de dicha librería, se usan las siguientes clases:

org.bouncycastle.crypto.AsymmetricCipherKeyPair: Una clase de mantenimiento para los pares de parámetros públicos/privados.

org.bouncycastle.crypto.BasicAgreement: Interfaz con los métodos init y calculateAgreement del método Diffie-Hellman. Éste método permite el intercambio secreto de claves entre dos partes que no han tenido contacto previo, utilizando un canal inseguro, y de manera anónima (no autenticada).

org.bouncycastle.crypto.BlockCipher: Cifra bloques de datos.

org.bouncycastle.crypto.BufferedBlockCipher: Clase para manejar los bloques encriptados mediante un buffer.

org.bouncycastle.crypto.CipherParameters: Parámetros de la encriptación.

org.bouncycastle.crypto.DataLengthException: Esta expeción se lanza si un buffer que tiene que almacenar datos es pequeño, o le hemos dado datos insuficientes para almacenar. Esta excepción se lanza antes que la expcepción ArrayOutOfBounds

org.bouncycastle.crypto.DerivationFunction: Interfaz general para las clases de derivación. Se usa para la generación de bytes

org.bouncycastle.crypto.DerivationParameters: Interface de la clase básica de los parámetros usados para la derivación

org.bouncycastle.crypto.Digest: Interfaz genérica para objetos que generan bytes aleatorios

org.bouncycastle.crypto.ExtendedDigest: Clase de apoyo a digest.

org.bouncycastle.crypto.InvalidCipherTextException: Excepción que se lanza cuando encontramos algo anormal en un mensaje

org.bouncycastle.crypto.Mac: Interfaz para la implementación de los códigos de autenticación de los mensajes (MACs).

org.bouncycastle.crypto.agreement.ECDHBasicAgreement: Método que obtiene un valor secreto a partir de una clave privada y otra publica.

org.bouncycastle.crypto.digests.SHA256Digest: Implementación del algoritmo SHA-256 mediante FIPS 180-2.

org.bouncycastle.crypto.digests.GeneralDigest: Implementación básica del resúmen de la familia MD4, tal y como se explica en el libro "Handbook of Applied Cryptography", páginas 344 – 347

org.bouncycastle.crypto.digests.SHA1Digest: Implementación del algoritmo SHA-1 como se explica en "Handbook of Applied Cryptography", páginas 346 – 349. A parte del IV extra, la otra diferencia que tiene con MD5 es que la palabra que se procesa no tiene final.

org.bouncycastle.crypto.engines.AESFastEngine: Implementación del Motor AES(Rijndael), a partir de FIPS-197.

org.bouncycastle.crypto.engines.IESEngine: Clase de apoyo para construir encriptadores para el intercambio básico de mensajes en la aplicación. En resumen, procesa (encripta, desencripta) bloques de bits de los mensajes

org.bouncycastle.crypto.generators.KDF2BytesGenerator: Construye un generador de KDF2 bytes.

org.bouncycastle.crypto.macs.HMac: Message authentication code algorithm

org.bouncycastle.crypto.params.ECDomainParameters: Clase para la creación de parámetros de las claves de curva elíptica.

org.bouncycastle.crypto.params.ECPublicKeyParameters: Parámetros de la clave pública en curva elíptica.

org.bouncycastle.crypto.params.ECPrivateKeyParameters: Parámetros de la clave privada en cruva elíptica.

org.bouncycastle.crypto.params.KDFParameters: Parámetros de la Key Derivation Functions (KDF). Estas funciones sacan una o más keys a partir de un valor secreto.

org.bouncycastle.crypto.params.KeyParameter: Parámetros de la key

org.bouncycastle.crypto.params.IESParameters: Parámetro para usar cifras en modo streaming. Es la clave de agreement (Diffie-Hellman) usada como base de la encriptación

org.bouncycastle.crypto.params.IESWithCipherParameters: Clase base para las claves de curva elíptica

org.bouncycastle.crypto.prng.RandomGenerator: Interfaz genérica para objetos que generan bytes aleatorios.

org.bouncycastle.crypto.prng.DigestRandomGenerator: Generación aleatoria basada en el “digest” con contador. Llamar a “addSeeMaterial” incrementará siempre la entropía del hash. El acceso interno al “digest” está sincronizado de manera que uno de estos puede ser compartido.

org.bouncycastle.crypto.prng.DigestRandomGenerator: Generación aleatoria basada en el “digest” con contador.

org.bouncycastle.crypto.prng.RandomGenerator: Interfaz genérica para objetos que generan bytes aleatorios.

org.bouncycastle.math.BigInteger: Clase que implementa los enteros grandes

org.bouncycastle.math.ec.ECPoint: Clase base para puntos y curvas elipticas.

org.bouncycastle.math.ec.ECConstants: Clase de apoyo para los parámetros de las claves de curva elíptica.

org.bouncycastle.math.ec.ECCurve: Clase base para una curva elíptica.

org.bouncycastle.math.ec.ECFieldElement: Realiza operaciones matemáticas.

org.bouncycastle.security.SecureRandom: Implementación del SecureRandom especificada para una API con “poco peso”, JDK 1.0 y J2ME. La generación aleatoria está basada en SHA1 con contador. Llama a setSEED, lo que incrementará la entropía de la tabla hash.

A partir de esta información, comenzamos a estudiar que herramientas provee el sistema operativo para conseguir un prototipo de la aplicación en lenguaje Symbian.

Una vez analizada la librería, pasamos a portarla a Symbian.

2.1.1 Opciones para su desarrollo

Como encargados de todo a lo referente a criptografía, pensamos como estructurar nuestro trabajo orientándolo a la creación de una interfaz que pudiera ser usado fácilmente por el grupo encargado de realizar la GUI.

Los objetivos específicos del módulo de criptografía son conseguir una API que proporcione los métodos necesarios para encriptar y desencriptar de forma segura, así como la generación de claves.

Otro objetivo a destacar es una vez enviada la clave de un usuario al resto, como almacenar dicha clave en un fichero, del cual se extraerá

la información para ser almacenada en la agenda de cada usuario. De esta manera conseguimos que un usuario siempre mande la misma clave.

2.1.1.1 Portación directa:

En un primer momento, comenzamos con el estudio profundo de los métodos y clases que se usan de la librería BouncyCastle. Pensamos que podríamos intentar realizar una portación directa a Symbian de dichas clases, bien mediante un programa o bien programando directamente la conversión.

Nos encontramos con un primer problema. La librería de criptografía que usa la aplicación está escritas en los lenguajes Java y C#.

Buscamos aplicaciones capaces de realizar la conversión de lenguajes directa.

Encontramos varios programas:

Para portar de c# a c++:

Encontramos el programa Instant CPlus Plus CSharp Edition. (http://tangiblesoftwaresolutions.com/Product_Details/Instant_CPlusPlus_CSharp_Edition.htm). La versión de prueba limita la conversión a un número limitado de líneas de código, con lo cual la portación se convertía en algo tedioso. La conversión no era en absoluto perfecta, y requería más tiempo revisando los resultados que el tiempo que tardaríamos en portarlo directamente. Además, el lenguaje c++ generado es mucho menos restrictivo que el empleado por Carbide. Por estas razones descartamos esta opción.

Portar de Java a C puro:

Investigamos el programa Toba para portar bouncycastle de Java a C.(http://www.cs.arizona.edu/projects/sumatra/toba/).Este programa tampoco dio buenos resultados.

Portar directamente el código fuente Java a código Symbian:

Para establecer un primer prototipo, consideramos que esta opción iba a ser demasiado costosa. Empezamos a encontrar problemas con las clases que realizan las operaciones matemáticas. Por ejemplo, en Carbide no existe ninguna librería que implemente los enteros grandes. Por otro lado, necesitamos estar seguros de que la encriptación sea totalmente segura. Los testeos y depuración iban a costar muchísimo tiempo. No desestimamos esta opción en un futuro, pero no a corto plazo.

2.1.1.2 OpenC y OpenSSL:

Investigando acerca de cómo conseguir librerías básicas que contuvieran operaciones matemáticas, descubrimos este plug-in para Carbide.

Permite escribir en código C y además incluye una librería para encriptación (OpenSSL).

En un primer momento nos pareció sumamente interesante. Implementaba un gran número de algoritmos, así como las operaciones matemáticas que necesitábamos.

Tras varias reuniones con los directores del proyecto, se decidió intentar implementar un prototipo sin usar ningún plug-in adicional. No obstante, esta opción quedó abierta por si no consiguiéramos portar directamente a Symbian.

Llegados a este punto, comenzamos a buscar información acerca de librerías o interfaces que pudieran servirnos como base o referencia, y que estuvieran implementadas en Symbian. De este modo, podríamos estar seguros de que no existían bugs.

2.1.1.3 IAIK Krypto Group AES Loungue:

Encontramos una API que provee las funciones necesarias para implementar el algoritmo AES. (http://www.newlc.com/AES-Encryption.html). La API original estaba escrita en C, y se tradujo manualmente a Symbian, como se puede comprobar en el enlace web.

Creamos un primer prototipo de encriptación usando este algoritmo. Conseguimos que cifrara correctamente, pero no descifraba. A parte, no nos ofrecía toda la seguridad que necesitábamos, ya que la API no estaba testeada o comprobada por Nokia. Decidimos seguir buscando más soluciones.

2.1.2 Opción definitiva

Tras varias deliberaciones acerca de cuál de las alternativas estudiadas emplear para implementar nuestra API de encriptación, decidimos que la mejor solución era emplear la Symbian Cryptography Library.

Cómo ya hemos visto, la portación directa de código Java a Código C hubiera sido muy tediosa e insegura.

Podríamos haber usado OpenC y la librería OpenSSL, ya que esta última librería provee ciertos módulos de criptografía, pero pensamos que preferíamos emplear lenguaje Symbian en vez de lenguaje C, ya

que hay muchas menos implementaciones de encriptación escritas en Symbian, y nos pareció una opción mucho más innovadora.

Por todo esto, nos pareció la mejor solución. Empezamos a desarrollar nuestra interfaz criptográfica a partir de esta librería.

Las referencias a Symbian Cryptography Libraries las encontramos en el foro de Nokia.

Fue lanzada por Nokia el 9 de Agosto del 2007.

Se hicieron disponibles las cabeceras de las funciones para programadores Symbian, tanto del SDK S60 3rd Edition, como del UIQ 3.

La librería provee API´s para encriptación y desencriptación simétrica y asimétrica, encriptación basada en claves, firma digital, implementación de enteros largos con generadores de números primos etc.

A primera vista, nos pareció la mejor solución de todas las que habíamos encontrado. Estaba creada directamente por Nokia con lo cual no tendríamos que buscar bugs. Implementaba la clase de los enteros largos e implementaba muchos métodos para crear las claves así como algoritmos de encriptación.

2.2 PROTOTIPOS IMPLEMENTADOS

Necesitamos dos prototipos, uno con un algoritmo de cifrado simétrico para guardar los mensajes encriptados en el teléfono, y otro con un algoritmo de cifrado asimétrico para el envío de mensajes.

Cifrado Simétrico

Para el almacenamiento de los mensajes en el teléfono se emplea el algoritmo AES. AES es hoy en día el algoritmo de seguridad de mayor confianza por su grado de fortaleza, seguridad, velocidad y bajo consumo de recursos, y para quienes lo hicieron, este debe ser representar su mayor logro, ya que dicho algoritmo fue elegido para ser el Advanced Encryption Standard (AES) luego de pasar un largo proceso de selección y haber ganado el concurso debido para ser considerado como tal.

A partir de la librería encontrada en IAIK Krypto Group, comenzamos primero por implementar el algoritmo AES, ya que al ser un algoritmo simétrico era más simple. Gracias a esto, empezamos a familiarizarnos con la API.

Conseguimos crear una pequeña aplicación que encriptaba y desencriptaba correctamente mensajes.

Al descubrir la Symbian Cryptography Libraries, decidimos reimplementar la API usando esta librería. Al estar firmada por Nokia, nos parecía mucho mas seguro que la anterior. Creamos la siguiente API:

```
class CriptoCryptoAES{

public:

static CriptoCryptoAES* NewL();

static CriptoCryptoAES* NewLC();

virtual ~CriptoCryptoAES();

private:

CriptoCryptoAES();

void ConstructL();

public:

void setAesKey(TBuf8<16>);

void EncryptAesL(const TBuf<256>&, TBuf<256>&);

void DecryptAesL(const TBuf<256>&, TBuf<256>&);

private:

TBuf8<16> iAesKey;

};
```

Cifrado Asimétrico

Estudiando la librería Symbian Cryptography Libraries, encontramos los siguientes algoritmos de cifrado:

• **DES:** hoy en día, DES se considera inseguro para muchas aplicaciones. Esto se debe principalmente a que el tamaño de clave de 56 bits es corto; las claves de DES se han roto en menos de 24 horas. Existen también resultados analíticos que demuestran debilidades teóricas en su cifrado, aunque son inviables en la práctica. A parte, el algoritmo es simétrico, con lo cual no nos sirve.

• **3-DES:** Cuando se descubrió que una clave de 56 bits no era suficiente para evitar un ataque de fuerza bruta, TDES fue elegido como forma de agrandar el largo de la clave sin necesidad de cambiar de algoritmo de cifrado. El Triple DES está desapareciendo lentamente, siendo reemplazado por el algoritmo AES. También es simétrico, con lo que queda descartado.

• **DH:** El protocolo Diffie-Hellman (debido a Whitfield Diffie y Martin Hellman) permite el intercambio secreto de claves entre dos partes que no han tenido contacto previo, utilizando un canal inseguro, y de manera anónima (no autenticada). Se emplea generalmente como medio para acordar claves simétricas que serán empleadas para el cifrado de una sesión. No es un algoritmo de cifrado en sí, sino un algoritmo para cifrar claves a la hora de ser intercambiadas. No sirve.

• **DSA:** (Digital Signature Algorithm, Algoritmo de Firma digital).Es un estándar del Gobierno Federal de los Estados Unidos de América o FIPS para firmas digitales. Este algoritmo sirve para firmar digitalmente un mensaje, no para encriptarlo. Queda descartado.

• **RSA:** El sistema criptográfico con clave pública RSA es un algoritmo asimétrico cifrador de bloques, que utiliza una clave pública, la cual se distribuye (en forma autenticada preferentemente), y otra privada, la cual es guardada en secreto por su propietario. Además de ser asimétrico es bastante robusto y rápido, con lo cual nos decantamos por él.

Algoritmo RSA

A la hora de realizar un primer prototipo, elegimos este algoritmo porque podíamos reutilizar la librería de criptografía de Symbian con la seguridad de que no tenía ninguna clase de bugs.

Además, este es un algoritmo de cifrado de clave pública, al igual que ECIES, y es ampliamente utilizado en ámbitos de criptografía.

Es lo suficientemente seguro y potente como para asegurarnos crear una aplicación robusta. Para implementar el algoritmo ECIES, necesitábamos crear clases que implementaran toda la criptografía de curva elíptica.

No nos pareció nada seguro para un primer prototipo, ya que temíamos que el proceso de depuración y testeo de estas clases consumiera demasiado tiempo, retrasando así otras partes del proyecto. Por esto nos decantamos por emplear el algoritmo RSA, cuyo funcionamiento describimos a continuación.

Descripción:

El sistema criptográfico con clave pública RSA es un algoritmo asimétrico cifrador de bloques, que utiliza una clave pública, la cual se distribuye (en forma autenticada preferentemente), y otra privada, la cual es guardada en secreto por su propietario.

Cuando se quiere enviar un mensaje, el emisor busca la clave pública de cifrado del receptor, cifra su mensaje con esa clave, y una vez que el mensaje cifrado llega al receptor, éste se ocupa de descifrarlo usando su clave oculta.

Los mensajes enviados usando el algoritmo RSA se representan mediante números y el funcionamiento se basa en el producto de dos números primos grandes (mayores que 10100) elegidos al azar para conformar la clave de descifrado. Emplea expresiones exponenciales en aritmética modular.

Generación de claves:

Emisor y Receptor están comunicándose a través de un medio de transmisión inseguro (abierto), y Emisor quiere enviar un mensaje privado (o seguro) a Receptor.Usando RSA, Emisor tomará los siguientes pasos para la generación de la clave pública y privada:

1. Cada usuario elige dos números primos largos p y q de manera que p y q sean diferentes, y se calcula n = p · q.
2. Los valores de p y q no se hacen públicos.
3. Cada usuario calcula Φ(n) = (p-1) · (q-1).
4. Cada usuario selecciona una clave publica *e* tal que 1 < e < Φ(n) y que cumplan que *e* y φ(n) sean primos entre sí.
5. Cada usuario calcula un *d* tal que d·e=1 (mod Φ(n)). Finalmente despejamos d: $d = \left(\frac{x(p-1)(q-1)+1}{e} \right)$, con x un valor entero positivo.
6. Se hace publico el grupo n y la clave e.
7. Se guarda en secreto la clave d.

La clave pública consiste en:

- n, el módulo.
- e, el exponente público (a veces exponente de cifrado).

La clave privada consiste en:

- n, el módulo, el cual es público y aparece en la clave pública.
- d, el exponente privado (a veces el exponente de descifrado), el cual debe permanecer oculto.

Cifrado de mensajes:

Ejemplo rápido:

Emisor quiere enviar a Receptor un mensaje secreto que solo el pueda leer.

Receptor envía a Emisor un mensaje con la clave pública suya. Emisor recibe el la clave publica, escribe el mensaje, lo encripta con la clave

publica de Receptor y se lo envía. Receptor abre el mensaje con su llave privada.

Supongamos que Emisor desea enviar un mensaje M a Receptor.

Él cambia M en un número m

Emisor ahora tiene m, y conoce n y e, mientras Receptor fue avisada. El entonces calcula el texto cifrado c correspondiente a m:

Esto puede ser rápido usando el método de exponentiation by squaring (llamado también exponenciación binaria). Emisor transmite c a Receptor.

Como hemos dicho antes, estos serian los pasos a dar:

1. Cada usuario elige n = p·q.
2. Cada usuario calcula φ(n) = (p-1)(q-1).
3. Cada usuario elige una clave pública e de forma que 1 < e < φ(n) y que cumpla con la condición: mcd [e, φ(n)] = 1.
4. Cada usuario calcula la clave privada d = inv [e,φ(n)].
5. Se hace público el grupo n y la clave e.
6. Se guarda en secreto la clave d.

Descifrado de mensajes:

Receptor recibe c de Emisor, y conoce su clave privada d. El puede recuperar m de c por el siguiente procedimiento:

$$m \equiv c^d \pmod{n}$$

Dado m, el puede recuperar el mensaje M. El descifrado procede de la siguiente forma:

$$c^d \equiv (m^e)^d \equiv m^{ed} \pmod{n}$$

Ejemplo:

Aquí tenemos un ejemplo de cifrado/descifrado con RSA.

1. Generamos primos p=137 y q=131.
2. n = pq = 137.131 = 17947 φ(n)=(p-1)·(q-1) = 136·130 = 17680
3. Elegimos e = 3 porque 1<3<17680 y mcd(3,17680)=1
4. Computamos d = $e^{\wedge -1}$ mod φ(n) = $3^{\wedge -1}$ mod 17680 = 11787.

La función de cifrado es:

$$encrypt(m) = m^e (\text{mod } n) = m^3 (\text{mod } 17947)$$

Donde m es el texto sin cifrar

La función de descifrado es:

$$decrypt(c) = c^d (\text{mod } n) = c^{11787} (\text{mod } 17947)$$

Donde c es el texto cifrado.

Para cifrar el valor del texto sin cifrar 09, nosotros calculamos:

$$encrypt(123) = 09^3 (\text{mod } 17947) = 729$$

Para descifrar el valor del texto cifrado, nosotros calculamos

$$\text{decrypt}(855) = 72911787 \ (\text{mod } 17947) = 09$$

Implementación algoritmo RSA:

Comenzamos haciendo pruebas con el algoritmo. Para generar las claves pública y privada empleamos clases ya implementadas. A partir de las claves, se puede llamar a la función que encripta. Conseguimos crear una pequeña aplicación que encriptaba los mensajes, pero no desencriptaba.

Si la librería estaba testada por Nokia, el error estaría en nuestra implementación. Investigando más a fondo el algoritmo, descubrimos que al cifrar por bloques, el tamaño del mensaje tendría que ser divisible por el tamaño de bloque.

Encontramos una clase dentro de la API que permitía convertir el tamaño del mensaje a un tamaño múltiplo del tamaño de bloque. Tras probar esta solución conseguimos que la aplicación encriptara y desencriptara sin problemas. La generación de claves había sido aleatoria.

Nos dimos cuenta de que, de alguna manera, la aplicación tiene que crear una clave pública y otra privada al ser arrancada por primera vez por el usuario, y estas claves tienen que ser almacenadas en la memoria del teléfono. Se nos plantea otro problema, como generar siempre las mismas claves. Existen unos parámetros a partir de los cuales se puede generar una clave pública y una clave privada.

Una vez hemos generado las claves, con los parámetros n y e para la clave pública, y n y d para la clave privada, necesitamos guardarlas en un fichero para de esta manera tenerlas almacenadas y que sea siempre la misma clave publica la que se envíe a los contactos que no la tengan. Se han estudiado varias maneras de guardar estos datos. En primer lugar, se

estudia la Unified Key Stores de Symbian, que es una librería diseñada para este propósito.

Lamentablemente, la librería no funciona con el SDK que empleamos. Por el momento, tenemos la siguiente API para el algoritmo RSA:

```
class CriptoCryptoRSA{

public://Constructors and destructors

static CriptoCryptoRSA* NewL();

static CriptoCryptoRSA* NewLC();

virtual ~CriptoCryptoRSA();

private:

CriptoCryptoRSA();

void ConstructL();

public://Functions

void EncryptRsaL(const TBuf<256>&, TBuf<256>&);

void DecryptRsaL(const TBuf<256>&, TBuf<256>&);

void ExtractCertificate();

private:

CRSAKeyPair* iKeyPair;

};
```

Esta API está totalmente orientada a su uso para encriptación de SMS y MMS.

2.3 INTERFAZ GRÁFICA

Para el desarrollo de los prototipos se han investigado numerosas API´s de Symbian v9.1, utilizando finalmente sólo aquellas que se adaptaban mejor a nuestros propósitos.

La mayor parte de las API´s estudiadas están orientadas al manejo de las comunicaciones sobre Symbian v9.1, y más concretamente al manejo (envío, manipulación, recepción) de SMS y MMS. A continuación pasamos a explicar más detalladamente cada una de ellas.

2.3.1 API

Manejo de SMS

Al ser uno de los principales requisitos de la aplicación el envío de SMS (y MMS), necesitábamos obtener acceso a esos servicios que ofrece Symbian. Lógicamente nuestras primeras investigaciones estuvieron encaminadas al envío de SMS, las cuáles nos llevaron al uso de la API Send UI que pasamos a explicar más detalladamente a continuación:

Send UI

Send UI es una API que proporciona a las aplicaciones cliente las capacidades necesarias para ofrecer al usuario servicios de mensajería interactiva. Dicho de otra manera, Send UI proporciona el uso de cualquier método de mensajería soportado por el dispositivo a través de una serie de elementos gráficos (menús desplegables, editores, ...).

Sin embargo en el contexto de nuestro proyecto, la principal ventaja que proporciona SendUI se vuelve totalmente en nuestra contra. Esto se debe a que el hecho de usar Send UI significa perder totalmente el control de los mensajes entre su creación y envío (es totalmente transparente al programador), o lo que es lo mismo, no podemos realizar ningún tipo de procesamiento del cuerpo del mensaje antes de que sea enviado. Por lo tanto no podríamos encriptar el cuerpo del mensaje antes de que sea enviado, lo que supone no cumplir uno de los principales requisitos de nuestro proyecto. Todo esto nos llevo a desestimar el uso de la API SendUI y a tener que buscar otra alternativa para el envío de los SMS.

La solución que se adoptó finalmente fue recurrir a la API Send As, combinada con el uso de un editor de texto independiente, lo cuál nos permitía editar el cuerpo de los SMS de una manera totalmente independiente de la API utilizada para su envío. A continuación pasamos a comentar más en profundidad la API RSendAs:

Send As

Send As se encuentra en un nivel más bajo que Send UI, de manera que constituye la capa inmediatamente superior al Message Server en la jerarquía de capas de mensajería en Symbian.

Hay que comentar que el uso de Send As se corresponde más con la clásica arquitectura cliente / servidor, donde el cliente sería la aplicación de usuario (nuestra aplicación en este caso), las clases de Send As encapsularían la sesión con el servidor y los datos que se intercambian (los mensajes) y el Message Server haría el papel de servidor.

Nótese que en este caso disponemos de métodos para asignar el cuerpo del mensaje una vez que éste se ha creado y antes de que sea enviado

pudiendo realizar cualquier procesamiento entre esos dos pasos, lo cuál se adapta perfectamente al objetivo de nuestra aplicación.

Sin embargo debido a los requisitos de nuestra aplicación no podemos conformarnos sólo con el envío de mensajes sino que tenemos que incluir un proceso de recepción para desencriptar los mensajes y poder mostrarlos al usuario. Como se indica en el apartado de Diseño hemos optado por no crear una nueva bandeja de entrada, sino usar la bandeja de entrada estándar del teléfono. Es decir, dejamos que Symbian realice el proceso de recepción de los mensajes de la misma manera que con un mensaje normal, para posteriormente explorar la bandeja de entrada en busca de los mensajes encriptados (los detalles de este proceso se indican en el apartado Diseño). Por lo tanto esto nos obligaba a buscar la manera de manipular las bandejas de entrada de Symbian de una manera segura y sin que afectara al resto datos almacenados en las bandejas que no tenían nada que ver con nuestra aplicación.

La solución a este problema fue usar "Client / Server Framework"

En Symbian este framework se usa para el acceso a todos aquellos recursos del sistema (como por ejemplo abrir archivos, acceder a bandejas de entrada, acceder a servicios de comunicación,...) que puedan ser accesibles de forma concurrente por varios procesos y que por lo tanto necesitan ser manipulados de una manera cuidadosa. Más concretamente, cada uno de los diferentes servidores de los que se compone Symbian es el encargado de manejar las peticiones de los clientes (aplicaciones de usuario) sobre los recursos que le corresponden.

2.3.2 Diseño

Las decisiones de diseño para el desarrollo del proyecto se han ido tomando conforme a las dificultades y necesidades que han ido apareciendo.

En este apartado nos centraremos en las decisiones del diseño que han tenido que ver con la parte de criptografía.

Una vez funcionó el envío, era necesario capturar ese mensaje, para ello se decidió entre dejar los mensajes en la bandeja de entrada por defecto o crear una bandeja privada en cada terminal y mover los mensajes enviados por la aplicación a esta bandeja. Por transparencia y confianza del usuario se decidió que los mensajes permanecieran en la bandeja de entrada, y explorar cada vez que se quisieran ver. Inicialmente eran texto plano, pero en posteriores iteraciones aparecerían encriptados, así que no violaría la finalidad del proyecto.

Otra necesidad derivada de la recepción de mensajes de texto fue la distinción de mensajes de la aplicación en desarrollo frente a mensajes de texto normales. Según el estudio de la aplicación en Java, este problema estaba resuelto con el uso de puertos virtuales, pero, en symbian el uso de estos puertos virtuales no existe. Se decidió entonces marcar los mensajes

añadiendo al texto del cuerpo del mensaje una serie de caracteres particulares, en nuestro caso estos caracteres son CRYPTO, esto ha provocado una pérdida de 6 caracteres del mensaje de texto. Si bien es cierto que un mensaje de texto permite hasta 160 caracteres, es una pérdida aceptable.

Otra parte importante del diseño fue decidir como implementar la agenda del teléfono. Se tenía las alternativas de utilizar la agenda propia del usuario o bien implementar una agenda propia de la aplicación, para la primera opción había que tener en cuenta que al añadir el módulo de criptografía sería necesario almacenar las claves de alguna manera externa a la agenda propia del teléfono y corresponder cada contacto con su clave particular; para la segunda opción era necesario diseñar un mecanismo de almacenamiento de cada contacto con su información. Se decidió usar la segunda opción y de esta manera separar ambas agendas y hacer privada esta información.

La inserción de la parte de criptografía hacía necesario un mecanismo para intercambiar claves entre terminales, para el caso de la aplicación se utiliza el envío de un mensaje de texto marcado con la palabra KEY y se ha insertado una opción en el menú principal para escanear la bandeja de entrada en busca de claves e incorporarla a la base de datos.

3. ENTORNO DE DESARROLLO

Para empezar a desarrollar aplicaciones con el sistema operativo symbian es necesario también definir que lenguaje se va a usar, el caso del proyecto es el C++. Si se quiere utilizar C++ hace falta instalar los siguientes componentes.

- Active Perl 5.6.1.635: Active perl es requerido por el carbide, su versión actual es la 5.10.0.1002, no obstante, hasta esta versión y desde la 5.6.1 es incompatible con el carbide.
- Carbide C++ v1.3: hay distintas ediciones dentro de cada versión, dependiendo de la que se escoja se requeríra de un registro gratuito o previo pago de un importe. La edición express limita las herramientas que pueden utilizarse pero permite la construcción de aplicaciones.
- JRE 1.4.2: Es requerido por la herramienta NCF que se explicará mas adelante.Symbian C++ S60_SDK_3rd_Edition: paquete de desarrollo de software proporcionado por nokia y que permite utilizar las librerías estandar de symbian, además de suministrar un emulador para provar las aplicaciones antes de pasarlo a un terminal real.
- Nokia Connectivitty Framework: para simular el envío de mensajes entre distintos emuladores.

3.1 HERRAMIENTAS DE DESARROLLO

Estructura de un programa en Symbian

Todo proyecto que se quiera crear con el carbide C++ para un sdk determinado utiliza una serie de archivos y una estructura de directorios básica para la creación del instalable. Si bien es posible alterar el orden que propone Symbian añadiendo o uniendo distintos archivos.

Hay 6 directorios básicos en todo proyecto symbian, data, inc, src, gfx, group y sis.

En la carpeta data se suelen incluir los archivos con extensión .rls, .rss. Los archivos .rls contienen el texto que se suele usar para etiquetas, opciones, etc. Es el texto que se va a visualizar en la pantalla. Las cadenas de texto se parametrizan con una etiqueta, de esta manera es fácil definir distintos rls para los idiomas que se quiera tener en la aplicación. Los archivos .rss son los archivos de recursos. Un archivo de recurso especifica estructuras de interfaz de usuario como CBA (Command button Area), Status Pane (panel de estado), menus, listbox, etc.

En la carpeta inc se encuentran los archivos .h y los archivos .hrh. Los archivos .h son los archivos de cabecera de código y los archivos .hrh contienen identificadores, como de vistas, comandos, etc.

La carpeta src contiene los archivos de código .cpp.

La carpeta gfx almacena los archivos gráficos .svg, .bmp, etc.

La carpeta group es una de las carpetas principales, contiene los archivos .mmp, .mk y .inf. El archivo .inf es el archivo principal que dice como construir el proyecto global, para que plataforma, incluye los .mmp y .mk. los archivos .mk dicen que gráficos se incluyen, sus características y en qué directorio se almacenan para su posterior carga en memoria. El archivo .mmp es el archivo de especificación del proyecto, incluye el identificador de la aplicación, del desarrollador, el tamaño por defecto para la pila de ejecución, los archivos de recursos que se utilizarán, las librerías que se incluyen, permisos o capacidades que requiere la aplicación, lenguajes de la aplicación y el código a compilar.

Y por último la carpeta .pkg que contiene los archivos .sis, .sisx y .pkg. El archivo .pkg especifica como se construye el archivo de instalación .sis, en que ruta dentro del terminal móvil se almacena cada archivo adicional, las plataformas que soporta la aplicación, y características del instalador. El archivo .sis es el archivo de instalación de la aplicación en un terminal. A partir de la tercera edición de la serie 60 este .sis debe ir firmado para poder instalarse en el móvil. El archivo .sisx es un archivo de instalación firmado, no es necesario que tenga extensión .sisx pero es una estratagema que utiliza carbide para diferenciar entre .sis y .sisx.

3.1.1 Plataforma

Symbian

Symbian es un sistema operativo que fue producto de la alianza de varias empresas de telefonía móvil, entre las que se encuentran Nokia, Sony Ericsson, PSION, Samsung, Siemens, Arima, Benq, Fujitsu, Lenovo, LG, Motorola, Mitsubishi Electric, Panasonic, Sharp, etc. Sus origenes provienen de su antepasado EPOC32, utilizado en PDA's y Handhelds de PSION.

Symbian cuenta con cinco interfaces de usuario o plataformas para su sistema operativo, las denominadas Serie 60, Serie 80, Serie 90, UIQ y FOMA. La mayoría de los móviles utilizan la Serie 60, todos los de Sony Ericsson trabajan bajo UIQ, así como también Motorola.

La plataforma para la que se ha desarrollado la apliacción es la serie 60. La plataforma S60 (formalmente, Interfaz de usuario de serie 60) consiste en un conjunto de bibliotecas y aplicaciones informáticas estándar, tales como telefonía, herramientas de gestión de información personal, y reproductores multimedia Helix. Está pensada para potenciar terminales móviles modernos de amplias características, con pantallas a color muy grandes, que son conocidos comúnmente como terminales smartphone. Se han ido proporcionando nuevas librerias hasta alcanzar la tercera edición y se han ido proporcionando funcionalidades en forma de paquetes FP (feature pack), actualmente la versión mas moderla es la serie 60 3ª edición Feature pack 2, aunque no han lanzado ningún terminal que soporte esta edición por el momento.

3.1.2 Herramientas

On-device debug

Los emuladores de los SDKs emulan en entornos de ejecución similares a dispositivos reales, pero no pueden reproducir exactamente el entorno de ejecución de un dispositivo real. Por ello existen errores que solo aparecen cuando se prueba la aplicación en un móvil real, es entones cuando surge la necesidad de un depurador que trabaja directamente en un dispositivo real.

Las versiones de Carbide C++ Developer, Professional, y OEM disponen de la opción de On-device Debugging.

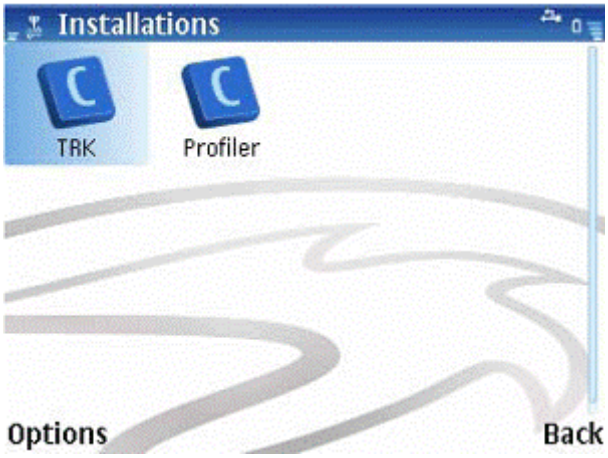
Preparación para On-device Debugging

Antes de poder usar el depurador con Carbide.c++, es necesario instalar TRK en el dispositivo destino, el cual debe estar conectado al PC de desarrollo a través de una conexión serial.

Instalar la aplicación TRK

La aplicación es un sisx que está dentro de la carpeta directorio_de_carbide\plugins\com.nokia.carbide.trk.support_1.3.0.024\trk\s6

Se instala la aplicación con PC suite de Nokia.

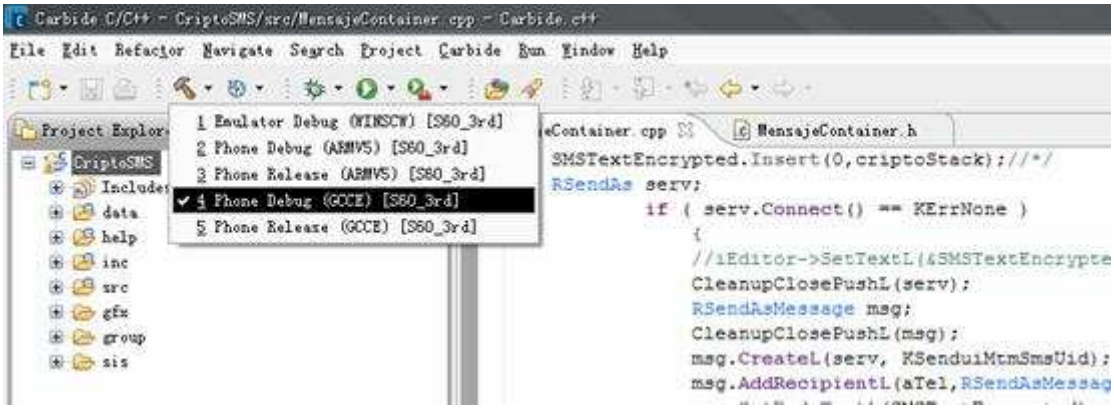


Configuración de la conexión

Existen dos vías para conectar el dispositivo al PC, por cable usb o por Bluetooth, en cualquier de los dos casos hay que tener especialmente en cuenta el número de puerto COM que está siendo utilizado en para después poder configurar correctamente Carbide.c++, dicho puerto se puede comprobar en Device Manager (administrador de dispositivos) de Windows.

Compilación del proyecto

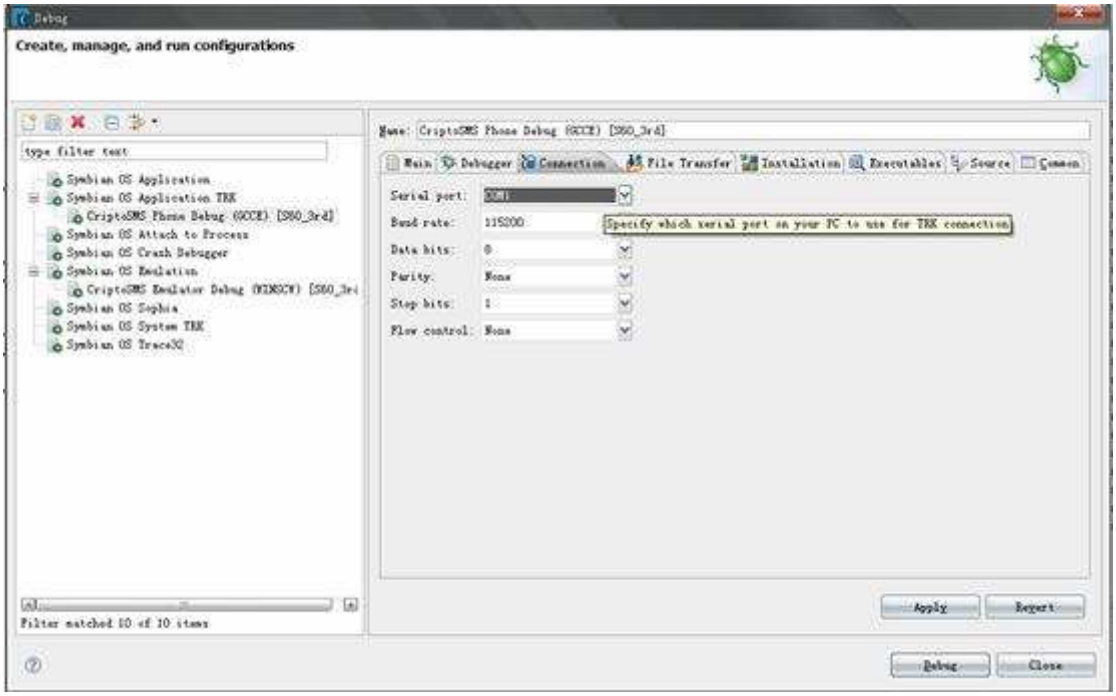
Se selecciona el destino de compilación "Phone Debug". Esta compilación genera los archivos .sis y sisx, para eso hay que tener en cuenta la firma para el paquete de instalación, ciertos permisos necesitan un firmado especial.



Configuración de Carbide.c++

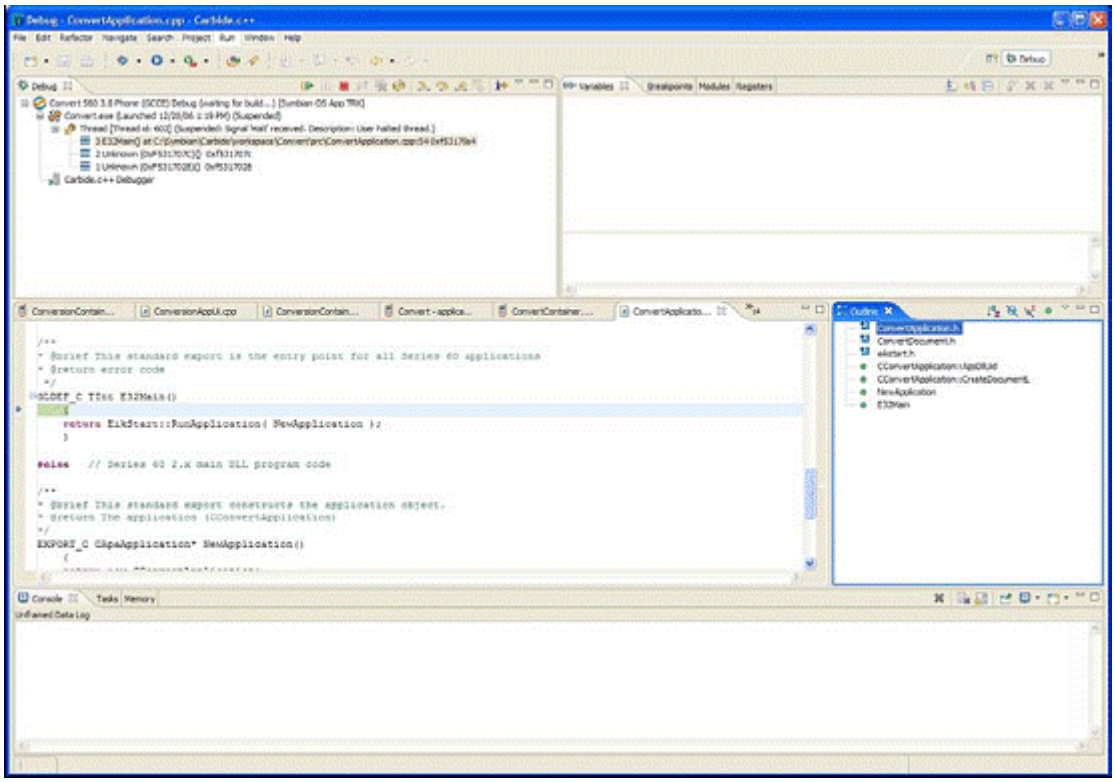
Antes de lanzar el debug, es conveniente verificar la configuración de debug. Para eso se abre el dialogo de debug.

En la parte izquierda de esta ventana selecciona "Symbian OS Application TRK". Aquí se debe prestar especial atención a la pestaña de configuración de conexión, ya que el puerto serial debe coincidir con el puerto de conexión del dispositivo.



Ejecutar On-Device Debug

Después de haber ajustado las configuraciones, al ejecutar el debug en Carbide.c++ se ejecutará automáticamente TRK en el dispositivo conectado al PC. Entonces se puede depurar la aplicación como si fuera ejecutada en



el emulador.

3.1.3 Sistema de Gestión de Base de Datos

Introducción:

La aplicación tiene la necesidad de almacenar de forma permanente en el dispositivo los datos de los contactos: nombre, teléfono y la clave pública asociada al contacto. La manera más cómoda y eficiente para realizar esta tarea es mediante una base de datos. Symbian ofrece el API DBMS (contenido en la librería edbms.dll) para la gestión de las bases de datos.

Para el funcionamiento de la base de datos es necesario crearla, especificando las tablas y los atributos de cada una de ellas.

Symbian DBMS:

Los sistemas de gestión de base de datos (*Database management system*, abreviado DBMS) son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta.

Symbian OS DBMS ofrece mecanismos para la creación y el mantenimiento de bases de datos, implementando un fiable y seguro acceso a estas bases de datos usando sentencias SQL o mediante métodos nativos.

Se trata pues de una potente y ligera base de datos que implementa las funcionalidades de añadir, buscar, acceder, actualizar y borrar además de la especificación para el manejo SQL, DDL y DML.

DBMS: Estructura y elementos:

Symbian OS DBMS es una base de datos relacional. Está representada por una jerarquía de elementos. Las tablas son contenedores de filas, que a su vez éstas están compuestas por campos.

Para su almacenamiento permanente usa la funcionalidad que ofrece Symbian OS en el manejo de ficheros: File Server, Permanent File Stores y Streams.

Creación de una Base de Datos:

Existen dos APIs para la creación de la base de datos:

- RDbStoreDatabase ofrece una interfaz para crear y abrir una base de datos que no va a ser compartida, es decir, solo una aplicación va a acceder a ella. Las operaciones son realizadas directamente a un fichero.
- RDbNamedDatabase ofrece por el contrario una interfaz para crear y abrir una base de datos identificado por su nombre y formato. Esta clase permite tanto el acceso exclusivo desde el cliente como el acceso compartido cliente-servidor.

Para implementar la agenda de la aplicación es más conveniente usar la primera API, ya que solo necesitamos un acceso exclusivo a la base de datos, donde estarán guardados los datos de los contactos: nombre, teléfono y clave pública de dicho contacto.

El fichero que se va a crear va a ser permanente y vamos a concederle permisos de lectura y escritura sobre él. Por último creamos la única tabla que va a tener nuestra base de datos.

```
void CBookDb::CreateBooksTableL()
{
```

```
TDbCol nombreCol(KNombreCol, EDbColText);
TDbCol telCol(KTelCol, EDbColText);
TDbCol claveCol(KClaveCol, EDbColText);
TDbCol tieneClaveCol(KTieneClaveCol, EDbColText);
CDbColSet* agendaColSet = CDbColSet::NewLC();
agendaColSet->AddL(nombreCol);
agendaColSet->AddL(telCol);
agendaColSet->AddL(claveCol);
agendaColSet->AddL(tieneClaveCol);
//Creamos la tabla
User::LeaveIfError(iBookDb.CreateTable(KAgendaTabla,*agendaColSet))
CleanupStack::PopAndDestroy(agendaColSet);
```

Cada tipo TDbCol es una columna. Necesitamos 4: el nombre, el número de teléfono, la clave pública y un flag que nos indique si tiene clave o no. Todas son de tipo texto. Una vez declarados, añadimos el conjunto de columnas a la base de datos usando el método CreateTable().

Apertura de una base de datos:

```
TInt CBookDb::OpenDb(const TFileName& aExistingBookFile){
Close();
if(!BaflUtils::FileExists(iFsSession, aExistingBookFile)){
   return KErrNotFound;
TRAPD(error,  iFileStore  =  CPermanentFileStore::OpenL(iFsSession,
aExistingBookFile, EFileRead|EFileWrite);
iFileStore->SetTypeL(iFileStore->Layout());
iBookDb.OpenL(iFileStore,iFileStore->Root())
);
if(error!=KErrNone)
{
return error;
iOpen = ETrue;
return KErrNone;
}
```

Para poder acceder a una base de datos ya creada es necesario primero abrirla indicando el nombre del fichero que la alberga y declarando qué permisos queremos sobre ella. Intentar abrir una base de datos que ya esté abierta provoca un fallo en la aplicación que provoca su terminación.

Inserción de datos:

```
TInt CBookDb::crearEntrada(const TDesC& aNombre,
```

```
const TDesC& aTel,
const TDesC& clave,
const TDesC& tieneClave)
{
RDbTable table;
TInt err = table.Open(iBookDb, KAgendaTabla, table.EUpdatable);
if(err!=KErrNone)
{
return err;
CDbColSet* booksColSet = table.ColSetL();
CleanupStack::PushL(booksColSet);
table.Reset();
RDbColWriteStream writeStream;
TRAPD(error,
table.InsertL();
table.SetColL(booksColSet->ColNo(KNombreCol), aNombre);
table.SetColL(booksColSet->ColNo(KTelCol), aTel);
table.SetColL(booksColSet->ColNo(KClaveCol),clave);
table.SetColL(booksColSet->ColNo(KTieneClaveCol), tieneClave);
);
if(error!=KErrNone)
{
return error;
writeStream.Close();
TRAP(err, table.PutL());    // Actualiza cambios
if(err!=KErrNone)
{
return err;
CleanupStack::PopAndDestroy(booksColSet);
table.Close();
return KErrNone;
```

Como se ha mencionado antes, existen dos maneras de insertar datos: usando sentencias SQL o usando métodos nativos. Queríamos experimentar las llamadas nativas que ofrece Symbian, por lo que decidimos usar éste último. Indicamos a la tabla que queremos insertar datos usando table. Insert(). Para rellenar los campos de cada columna usamos SetColL(), pasándole como parámetros la columna y el dato que queremos introducir. Por último actualizamos los cambios usando PutL().

3.2 NOKIA CONECTIVITY FRAMEWORK

Nokia Connectivity Framework 1.2 (NCF) es una herramienta para visualizar, construir, mantener, y modificar emulación o probar entorno de desarrollo que utilizan los emuladores de SDK. El NCF facilita la conectividad entre los emuladores de Nokia Developer Platform SDK, entre un emulador y un dispositivo, entre un emulador y un servidor de aplicación, donde ayuda a la creación y prueba de las aplicaciones conectado para los dispositivos de Nokia. Soporta tecnología de Bluetooth, Short Message Service (SMS), y Multimedia Messaging Service (MMS). El tráfico de SMS y Bluetooth se pueden funcionar a través de tarjetas seleccionadas o adaptadores de Bluetooth.

El NCF monitoriza las comunicaciones entre los componentes en un entorno de prueba, proporcionando un extenso registro de actividades. El NCF resulta muy práctico para probar el envío y recepción de los mensajes antes de probarlo en dispositivos reales.

Instalación de NCF 1.2

En siguiente link se puede descargar el instalador:

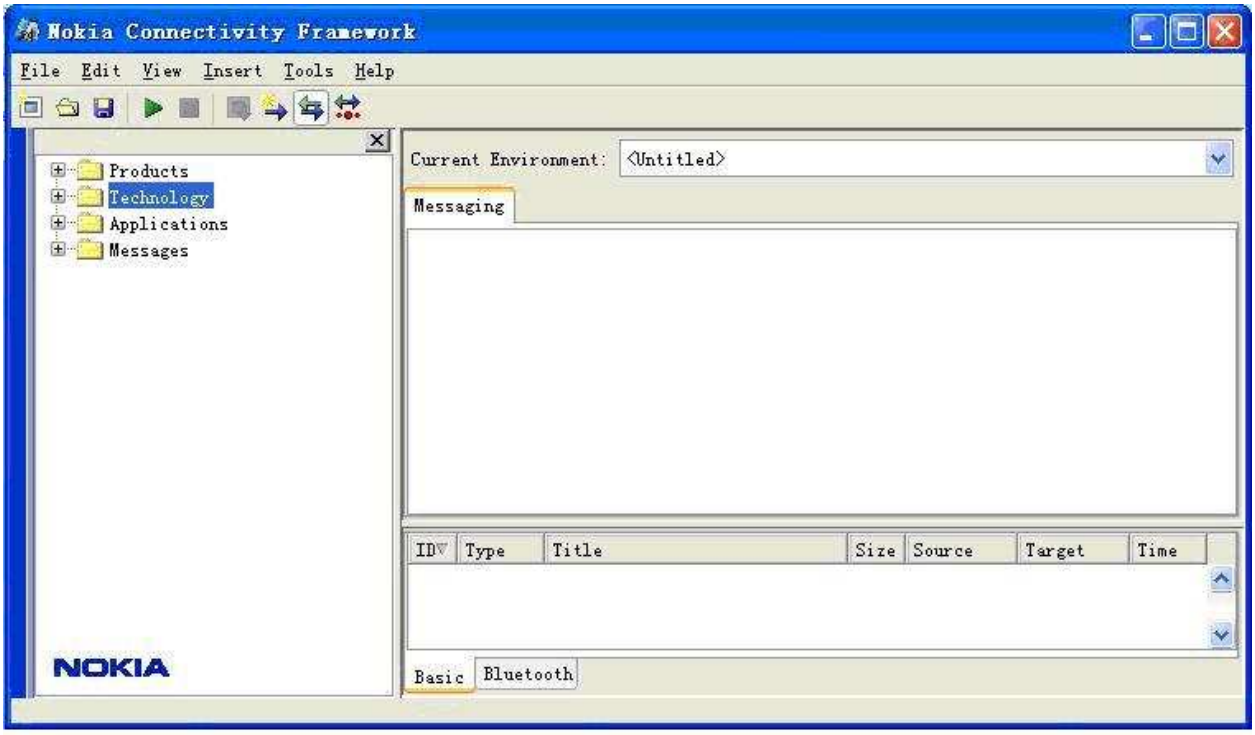
http://sw.nokia.com/id/fb0180d3-5b0b-43e4-8792-147488267c2d/ncf_v1_2.zip

(http://www.forum.nokia.com)

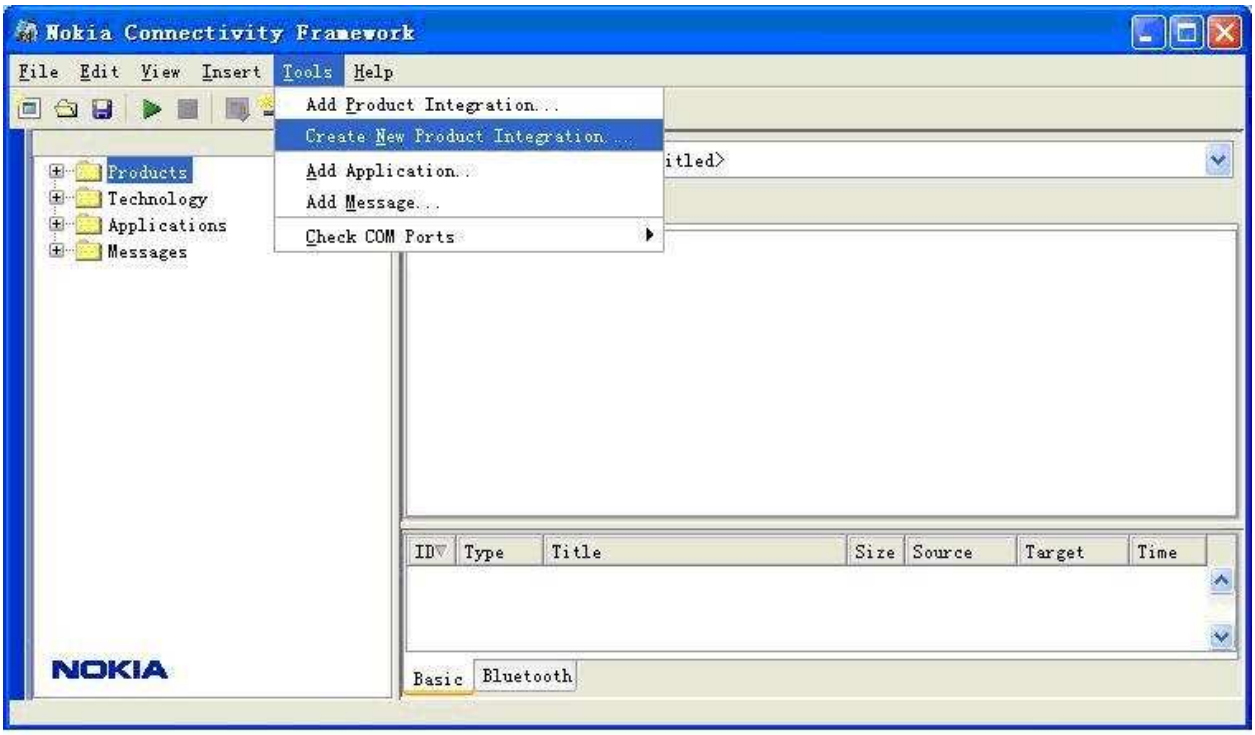
Nota : Es conveniente instalar antes los SDKs y Carbide, deja el NCF como el último elemento a instalar

Funcionamiento de NCF 1.2

NCF detecta automáticamente los SDKs instalados, para poder ejecutar dos emuladores es necesario tener dos instalaciones del SDK (en distintas carpetas).



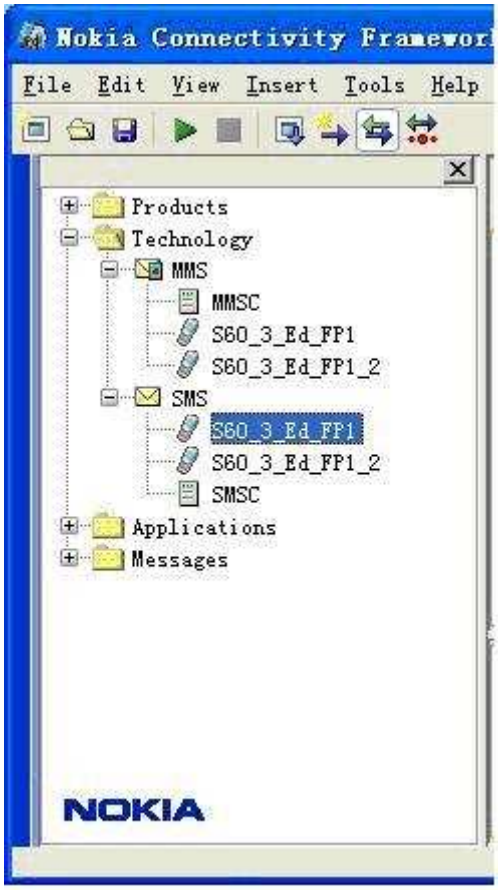
Al ejecutar NCF 1.2 aparece la siguiente ventana. Para ejecutar dos emuladores a la vez, es necesario añadir un nuevo "Product Integration".



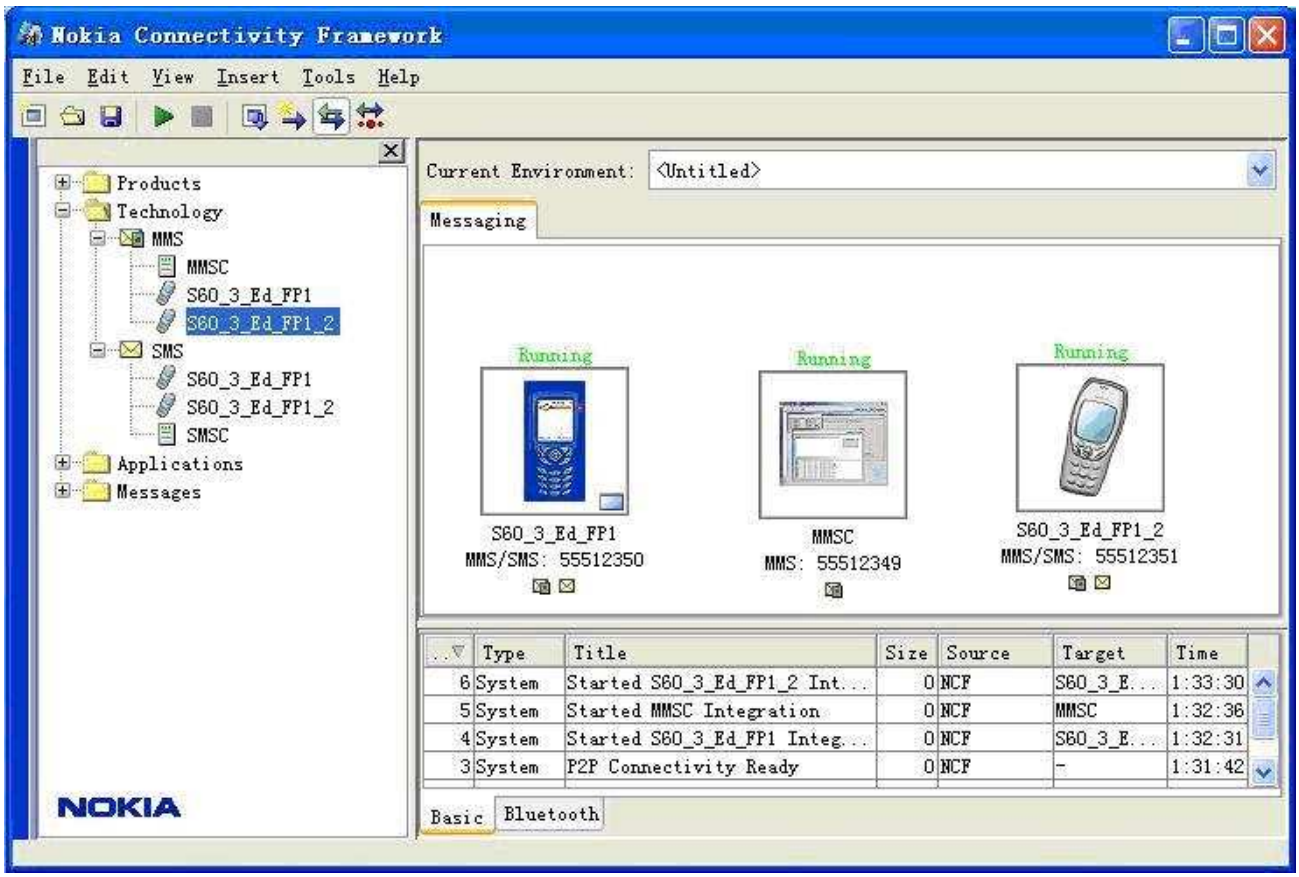
Ajusta las propiedades del nuevo "Product Integration" al ya existente, cambiando solamente los directorios del SDK.

Un ejemplo de simulación

En este ejemplo hemos creado los siguientes terminales : S60_3_Ed_FP1 y S60_3_Ed_FP1_2.



Finalmente para probar el envío y recepción de SMS o MMS, solo tenemos que arrastrar los terminales a la ventana de la derecha y su correspondiente servidor de mensajes SMS o MMS.

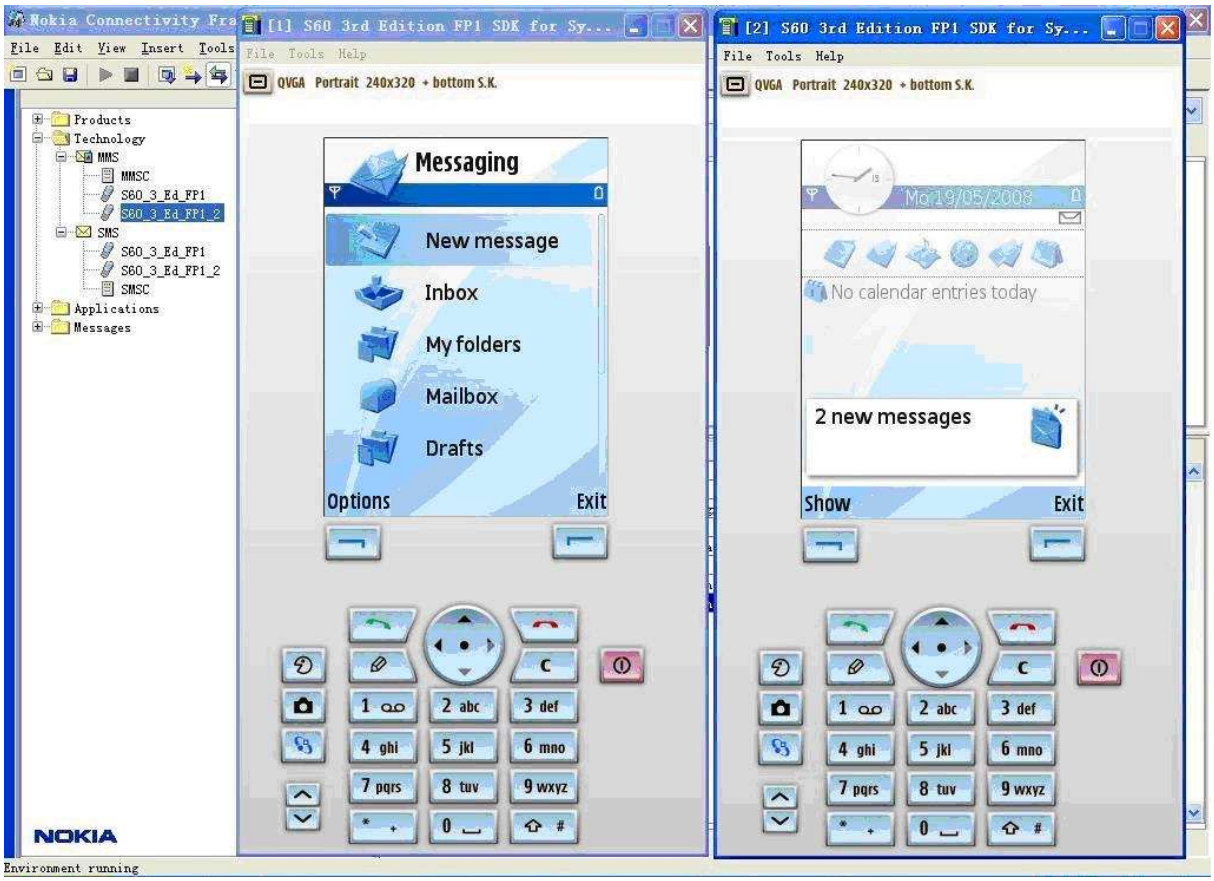


Ponemos en marcha el servidor y los terminales, ya podemos probar SMS o MMS. (Es necesario configurar el centro de mensajes para el correcto funcionamiento del envío y recepción).

En la parte inferior derecha hay log que registra todos los eventos del entorno.

ID	Type	Title	Size	Source	Target	Time
8	MMS	MMS Message	464	55512350	55512351	1:43:23
7	MMS	MMS Message	464	55512350	55512351	1:39:37
6	System	Started S60_3_Ed_FP1_2 Integration	0	NCF	S60_3_Ed_FP1_2	1:33:30
5	System	Started MMSC Integration	0	NCF	MMSC	1:32:36
4	System	Started S60_3_Ed_FP1 Integration	0	NCF	S60_3_Ed_FP1	1:32:31
3	System	P2P Connectivity Ready	0	NCF	-	1:31:42
2	System	NCF Integration Library Connectivity Ready	0	NCF	-	1:31:40
1	System	NCF Integration Library Connectivity Ready	0	NCF	-	1:31:40

Un ejemplo de envío y recepción de MMS.



3.3 FIRMANDO UN SIS

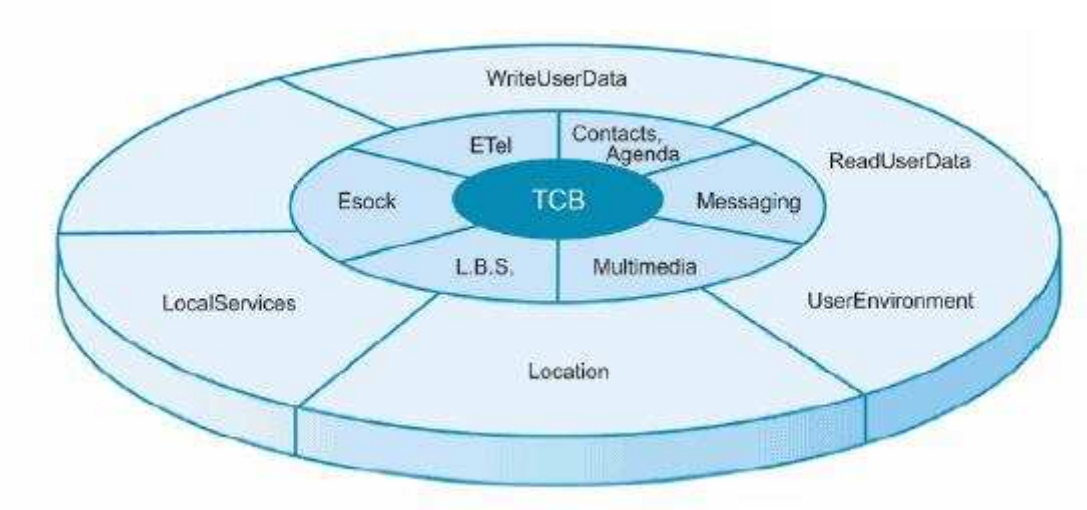
Lo primero que hay que saber es que en symbian se ha organizado el acceso a las API's según el paradigma de las 7 capas: TCB, TCE, Capabilities, SID&VID, DataCaging, SB&SP, Software Installer.

Cada capa tiene una serie de tareas determinadas, TCB se encarga de mantener la integridad del software del sistema, aquí se encuentra el kernel, el servidor de ficheros y el instalador de software; TCE protege las API's sensitivas que manejan recursos del dispositivo.Capabilities controla que puede hacer una aplicación incluyendo las librerías que utilice;SID&VID se utiliza para identificar la aplicación y el vendedor de la aplicación durante la carga en memoria; DataCaging restringe el acceso a los directorios del dispositivo basandose en las capacidades y su identidad, se apoya en TCB; SB&SP es la capa que se encarga de la seguridad de las llamadas entre distintos procesos; y por último Software Installer que verifica la firma de los archivos a instalar así como que el identificador de la aplicación sea único y crea valores Hash en caso de que las aplicaciones sean instaladas en memorias extraíbles del dispositivo para asegurar su integridad.

Lo segundo que se requiere de la firma para poder instalar un .sis en un terminal móvil que use la tercera edicion del SDK de symbian o posterior.

Y lo tercero que hay 2 maneras de firmar digitalmente un paquete de instalación o .sis; la autofirma y la firma a través de una autoridad firmante.

En cuanto al acceso a las API's asi como código que requiera ciertas librerías, aproximadamente el 60% no requiere de permisos el resto se han agrupado en "capacidades" dependiendo de donde acceden y su funcionalidad.



Al intentar instalar la aplicación que queramos en un dispositivo móvil, la capa mas baja, Software Installer, se encarga de comprobar:

1. Si el paquete de instalación está firmado o no (no siempre es necesario).
2. Que capacidades vienen registradas en el .sis (siempre podríamos tener menos de las necesarias, entonces se instalaría pero fallaría al intentar ejecutarlo).Las capacidades se especifican en el .mmp insertando una línea CAPABILITY cap1 cap2 ... NONE ... ALL ... -cap1 -cap2, de tal manera que podemos enumerar una detrás de otra ,conceder todas, y quitar las que no interesen o no conceder ninguna.

Capacidades del sistema:

- TCB: Máximo privilegio, el código TCB puede hacer cualquier cosa en el móvil.

- ALLFILES: Acceso de lectura de todo el sistema de ficheros y permiso de escritura a los directorios privados de otros procesos.
- COMMDD: Acceso directo a todos los dispositivos de comunicación.
- DISKADMIN: Acceso al sistema de ficheros del administrador.

- DRM: Acceso al contenido DRM.
- MULTIMEDIADD: Acceso a funciones multimedia críticas.
- NETWORKCONTROL: Modificar o acceder a protocolos de control de red.
- POWERMGMT: Apagar, encender, hibernar y matar cualquier proceso.
- PROTSERV: permite a un proceso servidor registrarse con un nombre protegido.
- READDEVICEDATA: Permiso de lectura a un operador de red y ajustes del dispositivo.
- SURROUNDIGNSDD: Acceso al controlador de dispositivos que proveen información del entorno del teléfono.
- SWEVENT: Habilidad de emular pulsaciones de teclas y capturar esos eventos desde cualquier programa.
- TRUSTEDUI: Habilidad de crear sesiones UI confiables y mostrar diálogos en un entorno de interfaz seguro.
- WRITEDEVICEDATA: Acceso de escritura para ajustar el control del comportamiento del dispositivo.

Capacidades de usuario:

- LOCALSERVICES: Acceso a servicios como bluetooth o infrarrojos; servicios que normalmente no ocasionan gasto para el usuario.
- LOCATION: Acceso a datos de posición dados por el teléfono.
- NETWORKSERVICES: Acceso a servicios remotos como wifi; servicios que pueden ocasionar gastos para el usuario.
- READUSERDATA: Capacidad de lectura a datos de usuario confidenciales.
- USERENVIROMENT: Acceso a datos en directo del usuario y su entorno.
- WRITEUSERDATA: Acceso de escritura a datos confidenciales de usuario.

En cuanto a las firmas, la autofirma la realiza automáticamente el carbide cuando construimos el .sis generando un certificado y un par de claves además del .sis firmado (.sisx, es el que hay que instalar con el pcsuite). Si queremos hacerlo nosotros "a mano" podemos hacerlo a través de una ventana de comandos.

Hay 3 comandos encargados de la creacion y autofirma del .sis, makesis, makekeys y singsis; makesis se encarga de leer el .pkg y construir el paquete de instalación, makekeys crea un certificado y un par de claves publica y privada, singsis firma el .sis con un certificado y un par de claves . Hay otro comando mas, createsis que lo único que hace es llamar a los anteriores, por si no queremos ejecutar cada uno independientemente.

La autofirma sólo es válida siempre que se se utilicen capacidades que puedan ser concedidas por usuario, de esta manera en la instalación aparecerán consultas de autorización a los servicios que se requieran.

Las capacidades que pueden usarse con la autofirma son: ninguna, LocalServices, ReadUserData, WriteUserData, NetworkServices y UserEnviroment. El resto requieren de una firma por autoridad firmante de otra manera aparecerá un mensaje en la instalación y el Software Installer no instalará la aplicación.

Para la firma por autoridad firmante hay que generar un certificado de programador y solicitar la firma.

Hay otro método de firmar el .sis con symbian signed website:"https://www.symbiansigned.com/app/page" Te permite firmar un .sis para un IMEI determinado de tal manera que te envían el firmado a una dirección de correo electrónico. Lo único que solo sirve para las capacidades:LocalServices, Location, NetworkServices, PowerMgmt, ProtServ, ReadDeviceData, ReadUserData, SurroundingsDD, SwEvent, TrustedUI, UserEnvironment, WriteDeviceData y WriteUserData.

4. FUTURAS VERSIONES

Al comienzo de la implementación del proyecto, decidimos utilizar el algoritmo asimétrico RSA para enviar las claves entre los contactos, ya que nuestra API (Symbian Crypto Lib) disponía de metodos que implementaban este algoritmo.

Para guardar estos mensajes una vez leídos, decidimos utilizar el algoritmo simétrico de encrptación AES.

Posteriormente, con el fin de simplificar el envio de mensajes, estamos utilizando el algoritmo AES para ambas cosas.

Por consiguiente en futuras versiones se deberia ampliar:

- Uso del algoritmo de curvas elípticas en el envio de las claves.
- Uso del algoritmo AES para guardar los mensajes en el terminal y no para envio de claves.

4.1 ALGORITMO DE CURVAS ELÍPTICAS

La Criptografía de Curva Elíptica (CCE) es una variante de la criptografía asimétrica o de clave pública basada en las matemáticas de las curvas elípticas.

Una curva elíptica es una curva plana definida por una ecuación de la forma $y^2 = x^3 + ax + b$.

Con el conjunto de puntos G que forman la curva (i.e., todas las soluciones de la ecuación más un punto O, llamado punto en el infinito) más una operación aditiva +, se forma un grupo abeliano. Si las coordenadas x e y se escogen desde un campo finito, entonces estamos en presencia de un grupo abeliano finito. El problema del logaritmo discreto sobre este conjunto de puntos (PLDCE) se cree que es más difícil de resolver que el correspondiente a los campos finitos (PLD). De esta manera, las longitudes de claves en criptografía de curva elíptica pueden ser más cortas con un nivel de seguridad comparable.

Introducción Matemática

Sea $p > 3$ primo. La curva elíptica E $y^2 = x^3 + ax + b$ sobre $\mathbb{Z}_p$ es el conjunto de soluciones $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$ en la congruencia $y^2 = x^3 + ax + b \pmod p$, donde $a, b \in \mathbb{Z}_p$ son constantes tal que $4a^3 + 27b^2 \neq 0 \pmod p$.

Se define una operación aditiva como sigue:

Considerando que $P = (x_1,y_1)$ y $Q = (x_2,y_2)$ son puntos en E y $\mathcal{O}$ es un punto en el infinito.

Si $x_2 = x_1$ e $y_2 = -y_1$, entonces $P + Q = \mathcal{O}$; de lo contrario $P + Q = (x_3,y_3)$, donde

- $x_3 = \lambda^2 - x_1 - x_2$,
- $y_3 = \lambda(x_1 - x_3) - y_1$,

y

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & \text{si } P \neq Q \\ \dfrac{3x_1^2 + a}{2y_1}, & \text{si } P = Q \end{cases}.$$

Finalmente, definimos $P + \mathcal{O} = \mathcal{O} + P = P \ \forall P \in E$.

Con esto se puede mostrar que *E* es un grupo abeliano con elemento identidad $\mathcal{O}$. Cabe notar que la inversa de (x, y) (que se escribe como -(x, y) ya que la operación es aditiva) es (x, -y), para todo $(x, y) \in E$.

De acuerdo al teorema de Hasse, el numero de puntos #E que contiene *E* es cercano a *p*. Más precisamente se satisface la siguiente desigualdad $p + 1 - 2\sqrt{p} \leq \#E \leq p + 1 + 2\sqrt{p}$.

Como se sabe que cualquier grupo de orden primo es cíclico, lo que se requiere es encontrar un subgrupo de *E* de orden q (q primo) para tener un isomorfismo con $\mathbb{Z}_q$ donde el problema del logaritmo discreto sea intratable. En este caso, siendo α un generador del grupo cíclico (el cual puede ser cualquier elemento del grupo distinto de $\mathcal{O}$, la identidad), se pueden calcular las "potencias" de α(las que se escriben como múltiplos de α, debido a que la operación del grupo es aditiva).

Ejemplo:

Sea E la curva elíptica $y^2 = x^3 + x + 6$ sobre $\mathbb{Z}_{11}$. Se calculan los puntos sobre E verificando los posibles valores de $x \in \mathbb{Z}_{11}$, y luego verificando si $z = x^3 + x + 6 \pmod{11}$ es un residuo cuadrático. Los valores se muestran en la siguiente Tabla,

x	$x^3 + x + 6 \pmod{11}$	*Y*
0	6	
1	8	
2	5	4, 7
3	3	5, 6

4	8	
5	4	2, 9
6	8	
7	4	2, 9
8	9	3, 8
9	7	
10	4	2, 9

Como E tiene 13 puntos, sigue que es cíclico e isomorfo a $\mathbb{Z}_{13}$.. Considerando el generador α = (2,7), entonces :2α = (2,7) + (2,7)

$$\lambda = (3 \times 2^2 + 1)(2 \times 7)^{-1} \pmod{11}$$

$$= 2 \times 3^{-1} \pmod{11}$$

$$= 2 \times 4 \pmod{11}$$

= 8

Entonces tenemos $x_3 = 8^2 - 2 - 2 \pmod{11} = 5$ y

$y_3 = 8(2 - 5) - 7 \pmod{11} = 2$

Por lo tanto 2α = (5,2)

Uso aplicado a la criptografía:

En el uso criptográfico, se elige un punto base G específico y publicado para utilizar con la curva E(q). Se escoge un número entero aleatorio k como clave privada, y entonces el valor P = k*G se da a conocer como clave pública (nótese que la supuesta dificultad del PLDCE implica que k es difícil de deducir a partir de P). Si A y B tienen las claves privadas kA y kB, y las claves públicas PA y PB, entonces A podría calcular kA*PB = (kA*kB)*G; y B puede obtener el mismo valor dado que kB*PA = (kB*kA)*G.

Esto permite establecer un valor "secreto" que tanto A como B pueden calcular fácilmente, pero que es muy complicado de derivar para una tercera persona. Además, B no consigue averiguar nada nuevo sobre kA durante ésta transacción, de forma que la clave de A sigue siendo privada.

Los métodos utilizados en la práctica para cifrar mensajes basándose en este valor secreto consisten en adaptaciones de antiguos criptosistemas de logaritmos discretos originalmente diseñados para ser usados en otros grupos. Entre ellos se podrían incluir Diffie-Hellman, ElGamal y DSA.

ECIES

ECIES (Elliptic Curve Integrated Encryption Scheme, Esquema Integrado de Encriptación por curva Elíptica) es un esquema de encriptación con clave pública. Propuesto por Bellare y Rogaway colocar aquí referencia a la bibliografía, es una variable del algoritmo de llave pública ElGamal. Es descrito por los estándares ANSI X9.63, ISO/IEC 15946-3 y IEEE P1363a.

Algoritmo de encriptación:

Para que A pueda enviar un mensaje a B usando ECIES, A necesita la siguiente información:

Información Criptográfica:

1. **KDF:** Key Derivation Funciton.
2. **MAC:** message authentication code algorithm
3. **Esquema de encriptación simétrica,** como TDES o el esquema de encriptación TDES.

    - Parámetros de la curva elíptica (p,a,b,G,n,h)
    - La clave pública de B K_B (con $K_B = k_B G$, donde k_B es la llave privada que B a elegido y $k_B \in [1, n-1]$).
    - Información compartida opcional: S_1 y S_2.

Para encriptar un mensaje *m,* A hace lo siguiente:

1. Genera un número aleatorio $r \in [1, n-1]$ y calcula $R = rG$;
2. Crea un secreto compartido: $S = P_x$, donde $P = (P_x, P_y) = rK_B$ con P distinto de 0.
3. Usa KDF para derivar una encriptación simétrica y unas llaves MAC: $k_E \| k_M = \mathrm{KDF}(S \| S_1)$
4. Encripta el mensaje: $c = E(k_E; m)$;
5. Calcula el final del mensaje encrpitado y S_2: $d = \mathrm{MAC}(k_M; c \| S_2)$
6. Devuelve $R \| c \| d$

Algoritmo de desencriptación:

Para desencriptar el texto cifrado $R \| c \| d$ B hace lo siguiente:

1. Obtiene el secreto compartido: $S = P_x$, donde $P = (P_x,P_y) = k_BR$ (es el mismo que obtuvo A, porque $P = k_BR = k_BrG = rk_BG = rK_B$), o devuelve fallo si $P = O$;
2. Deriva las llaves de la manera que lo hizao A: $k_E \| k_M = \mathrm{KDF}(S \| S_1)$
3. Usa MAC para chequear el tag o final del mensaje encriptado, y el algoritmo falla si $d \neq \mathrm{MAC}(k_M; c \| S_2)$;
4. Usa el esquema de encriptación simétrico para desencriptar el mensaje mediante la siguiente ecuación: $m = E^{-}1(k_E;c)$.

4.2 CRIPTOLOGÍA SIMÉTRICA PARA LOS DATOS ALMACENADOS EN EL TERMINAL

Advanced Encryption Standard (aes)

Es un esquema de cifrado por bloques adoptado como un estándar de cifrado por el gobierno de los Estados Unidos.

Opera como una red de sustitución-permutación.

Es rápido tanto en software como en hardware, es relativamente fácil de implementar y requiere poca memoria. Como nuevo estándar de cifrado, se está utilizando actualmente a gran escala.

Descripción del cifrado:

AES tiene un tamaño de bloque fijo de 128 bits y tamaños de llave de 128, 192 ó 256 bits.La mayoría de los cálculos del algoritmo AES se hacen en un campo finito determinado.

AES opera en una matriz de 4×4 de bytes, llamada. Para el cifrado, cada ronda de la aplicación del algoritmo AES (excepto la última) consta de cuatro pasos:

1. SubBytes - en este paso se realiza una sustitución no lineal donde cada byte es reemplazado con otro de acuerdo a una tabla de búsqueda.

2. ShiftRows - en este paso se realiza un transposición donde cada fila del state es rotado de manera cíclica un número determinado de veces.

3. MixColumns - operación de mezclado que opera en las columnas del «state», combinando los cuatro bytes en cada columna usando una transformación lineal.

4. AddRoundKey - cada byte del «state» es combinado con la clave «round»; cada clave «round» se deriva de la clave de cifrado usando una iteración de la clave.

La ronda final reemplaza la fase MixColumns por otra instancia de AddRoundKey.

ETAPA SubBytes:

Substitución de bits En la etapa SubBytes, cada byte en la matriz es actualizado usando la caja-S de Rijndael de 8 bits. Esta operación provee la no linealidad en el cifrado. La caja-S utilizada proviene de la función inversa alrededor del GF(28), conocido por tener grandes propiedades de no linealidad. Para evitar ataques basados en simples propiedades algebraicas, la caja-S se construye por la combinación de la función inversa con una transformación afín inversible. La caja-S también se elige para evitar puntos estables (y es por lo tanto un derangement), y también cualesquiera puntos estables opuestos.

ETAPA ShiftRows:

Desplazar filas El paso ShiftRows opera en las filas del state; rota de manera cíclica los bytes en cada fila por un determinado offset. En AES, la primera fila queda en la misma posición. Cada byte de la segunda fila es rotado una posición a la izquierda. De manera similar, la tercera y cuarta filas son rotadas por los offsets de dos y tres respectivamente. De esta manera, cada columna del state resultante del paso ShiftRows está compuesta por bytes de cada columna del state inicial. (variantes de Rijndael con mayor tamaño de bloque tienen offsets distintos).

ETAPA MixColumns:

Mezclar columnas En el paso MixColumns, los cuatro bytes de cada columna del state se combinan usando una transformación lineal inversible. La función MixColumns toma cuatro bytes como entrada y devuelve cuatro bytes, donde cada byte de entrada influye todas las salidas de cuatro bytes. Junto con ShiftRows, MixColumns implica difusión en el cifrado. Cada columna se trata como un polinomio GF(28) y luego se multiplica el módulo x4 + 1 con un polinomio fijo c(x). El paso MixColumns puede verse como una multiplicación matricial en el campo finito de Rijndael.

ETAPA AddRoundKey:

Cálculo de las subclaves En el paso AddRoundKey, la subclave se combina con el state. En cada ronda se obtiene una subclave de la clave principal, usando la iteración de la clave; cada subclave es del mismo tamaño del state. La subclave se agrega combinando cada byte del state con el correspondiente byte de la subclave usando XOR.

5. BIBLIOGRAFÍA

Libros sobre criptografía

- Guide to elliptic curve cryptography / D. Hankerson, A. Menezes S. Vanstone (2004)
- Introducción a la criptografía / Pino Caballero Gil (2002)
- Criptografía digital: Fundamentos y Aplicaciones / José Pastor Franco, Miquel Ángel Sarasa López (1998)
- A classical introduction to cryptography: applications for communications security / Serge Vaudenay (2006)
- Modern cryptanalysis: techniques for advanced code breaking / Christopher Swenson (2008)
- Introduction to cryptography : principles and applications / H. Delfs, H. Knebl (2007)
- Introduction to cryptography with coding theory / Wade Trappe, Lawrence C. Washington (2006)

Libros sobre Symbian

- Symbian OS Internals / Jane Sales (2005)
- Developing Software for Symbian OS / Steve Babin (2005)
- Wireless Java for Symbian Devices / Jonathan Allin (2001)
- Symbian OS Communications Programming / Michael J Jipping (2002)
- Programming for the Series 60 Platform and Symbian OS / Digia (2002)
- Symbian OS C++ for Mobile Phones, Volume 1 / Richard Harrison (2003)
- Symbian OS C++ for Mobile Phones, Volume 2 / Richard Harrison (2004)
- Programming Java 2 Micro Edition on Symbian OS / Martin de Jode (2004)
- Symbian OS Explained / Jo Stichbury (2004)
- Programming PC Connectivity Applications for Symbian OS / Ian McDowall (2004)
- Rapid Mobile Enterprise Development for Symbian OS / Ewan Spence (2005)
- The Symbian OS Arquitecture Sourcebook / Ben Morris (2007
- Smartphone Operating System Concepts with Symbian OS, A Tutorial Guide / Michael J. Jipping (2007)
- Wiley,.Programming.for.the.Series.60.Platform.and.Symbian.OS

Páginas web

- Forum Nokia - www.forum.nokia.com
- Wikipedia (para el estudio de los algoritmos).
- www.todosymbian.com/
- www.newlc.com/
- www.elrincondesymbian.com/

- www.symbianforever.com
- www.allaboutsymbian.com
- www.majinate.com 251
- www.s60.com
- www.symbian.com
- www.symbiangear.com
- www.symbianone.com
- www.symbiansigned.com
- www.uiq.com 251

Libros electrónicos sobre criptografía (en http://www.cripto.es/libros.htm)

- The Codebreakers.
- Applied cryptography.
- "Basic Cryptanalysis".Manual de campo del ejército norteamericano.
- Handbook of Applied Cryptography.

6. GLOSARIO

Cifrado: es el proceso de convertir el texto plano en un texto cifrado o criptograma.

Clave publica: La criptografía asimétrica es el método criptográfico que usa un par de claves para el envío de mensajes. Las dos claves pertenecen a la misma persona a la que se ha enviado el mensaje. Una clave es pública y se puede entregar a cualquier persona.

Clave privada: El propietario debe guardarla de modo que nadie tenga acceso a ella. El remitente usa la clave pública del destinatario para cifrar el mensaje, y una vez cifrado, sólo la clave privada del destinatario podrá descifrar este mensaje.

Criptografía: es la ciencia de cifrar y descifrar información utilizando técnicas que hagan posible el intercambio de mensajes de manera segura para que sólo puedan ser leídos por las personas a quienes van dirigidos.

Descifrado: es el proceso inverso que recupera el texto plano a partir del criptograma y la clave.

Emulador: Es un software que permite ejecutar programas de computadora en una plataforma diferente de la cual fueron escritos originalmente. Trata de modelar de forma precisa el dispositivo que se está emulando.

Framework: Es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Protocolo criptográfico: especifica los detalles de cómo se utilizan los algoritmos y las claves para conseguir el efecto deseado.

Prototipo: Un prototipo o prototipado es un modelo del ciclo de vida del Software, que posteriormente sufrirá modificaciones hasta llegar a su versión final y definitiva.

Symbian: es un sistema operativo que fue producto de la alianza de varias empresas de telefonía celular. El objetivo de Symbian fue crear un sistema operativo para terminales móviles que pudiera competir con el de Palm o el Windows Mobile de Microsoft.

SMS: Short Message Service (Servicio de mensajes cortos).

MMS: Multimedia Messaging System (Sistema de mensajería multimedia).

J2ME: Java 2 Micro Edition.

CryptoSMS: Aplicación en Java para el envío de mensajes cifrados**.**

API: Application Programming Interface (Interfaz de Programación de Aplicaciones): conjunto de métodos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

WMA: Wireless Messaging API (API para la mensajería sin cables)**.**

PIM: Personal Information Management (Gestión de la información personal)**.**

JVM: Java Virtual Machine**.**

VM : Virtual Machine (Máquina virtual).

CLDC: Connected Limited Device Configuration.

CDC: Connected Device Configuration**.**

KVM: VM de la configuración CLDC.

SDK: software delevolpment kit (kit de desarrollo de software).

CVM: VM de la configuración CDC**.**

MIDP: Mobile Information Device Profile.

MIDlet: Aplicación para teléfonos móviles.

DBMS: DataBase Management System

GSM:Sistema Globar para las comunicaciones móviles.

3G: denominación para la tercera generación de telefonía móvil.

JRE:Java Runtime Enviroment (entorno en tiempo de ejecución java).

DAO: Database Access Object (Objeto de acceso a datos).

OEM: Original Equipment Manufacturer (Fabricante de equipos originales)

7. AUTORIZACIÓN

Autorizamos a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado".

En Madrid a 30 de Junio de 2008.

Bernardo Chenlo García Marta Garbayo Sesma Elisa Sanz Troyano

8. RESUMEN

Este documento pretende mostrar la portación, diseño e implementación de una herramienta de comunicación sobre terminales móviles, segura, confiable y de fácil manejo, orientada a las necesidades de autenticidad y no repudio de las sociedades de información actuales.

Esta primera versión permite a los usuarios de la aplicación mantener un medio de comunicación privado y fiable mediante mensajes de texto y mensajes multimedia y restringirlo a un grupo conocido de personas.

Utiliza algoritmos de encriptación de tipo simétrico para el envío de mensajes y proporciona mecanismos de intercambio de claves entre contactos que se almacenan en una agenda privada en cada terminal.

El objetivo fundamental es el desarrollo de una API en lenguaje Symbian intuitiva, segura y robusta para permitir la encriptación y desencriptación de mensajes de texto.

Actualmente, las API´s existentes están destinadas a la criptografía en general, no están diseñadas específicamente para el módulo de SMS y MMS. Sería una de las primeras API´s diseñadas para este propósito.

9. SUMMARY

This document aims to show the design and implementation of a communication tool on movile terminals, secure, reliable, and user-friendly, guided to the needs of authenticity and non-repudation of current information societes.

This first version provides a private and reliable communication channel for text messaging and multimedia messaging and restrict it to a group of known people.

This application performs symmetric encryption algorithms (AES algorithm), to send messages and provides mechanisms for exchanging keys between contacts that are stored in a private agenda at each terminal.

The primary objective is the development of an API in Symbian intuitive language, to enable secure and robust encryption and decryption of text messages.

Currently´s existing APIs are intended to cryptography in general, are no designed specifically for release of SMS an MMS. It would probably be one of the first API´s designed for this purpose.

10. KEYWORDS

“Cryptography**”,** *“Symbian”***,** **“***Encryption”, “Decription”, “SMS”, “MMS”, “Symmetric”, “Algorithm”*