Bloque 4. Criptografía en Android

José A. Montenegro

Dpto. Lenguajes y Ciencias de la Computación
ETSI Informática. Universidad de Málaga
monte@lcc.uma.es twitter

22 de noviembre de 2011

Números Aleatorios

Java nos ofreces dos posibilidades:

java.util.Random

- Algoritmo usado[1] produce una secuencia predecible de números.
- Utiliza el valor del reloj del sistema (predecible) si no aportamos semilla.

java.security.SecureRandom

- Algoritmo usado es SHA1PRNG basado en SHA-1 (Secure Hash Algorithm).
- Por defecto, la semilla utilizada utilizan una fuente interna de entropía, /dev/urandom. Esta semilla no es predecible y es apropiada para un uso seguro.

[1] The Art of Computer Programming, Volume 2: Seminumerical Algorithms, section 3.2.1

`SecureRandom()` Constructor crea una instancia con semilla generada automáticamente.

`SecureRandom(byte[] seed)` Constructor crea una instancia con semilla dada por usuario

`setSeed(byte[] seed)` Método actualiza estado interno de SecureRandom. Los datos no reemplazan la semilla original, complementa.

`nextBytes(byte[] bytes)` Método nos proporciona la información pseudo-aleatorias.

```
1 SecureRandom sr = new SecureRandom();
2 byte[ ] pseudoRandom = new byte[100];
3 sr.nextBytes(pseudoRandom);
```

Código 1: Generación Número Aleatorio Seguro SHA-1

Administración claves

java.security.Key Interfaz encapsula clave criptográfica.

- `String getAlgorithm()`. Método devuelve el nombre del algoritmo que utilizara la clave.
- `byte[] getEncoded()`. Obtenemos el valor de la clave a un array de bytes.
- `String getFormat()`. Devuelve el formato utilizado para codificar la clave (X.509)

Interfaces que extienden java.security.Key:

java.security.PublicKey Interfaz representa clave pública de un par de claves

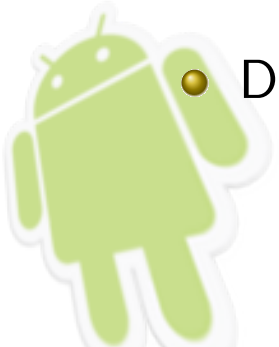
java.security.PrivateKey Interfaz representa clave privada de un par de claves

java.security.SecretKey Interfaz representa un secreto (privado o sesión) utilizado para un algoritmo simétrico

java.security.KeyPair Encapsula un par de claves

Generación Claves

- ¿Cómo creamos las claves?
  - Utilizamos una clase especial, denominada key generator, utilizada para crear claves aleatorias nuevas
- Tres pasos están involucradas para crear claves:
  1. Establecer un objeto "generador claves" para el algoritmo determinado
  2. Inicializar el generador de claves
  3. Solicitar al generador de claves una clave o par de claves
- Dos variedades de generadores de claves.
  - Algoritmos asimétricos
  - Algoritmos simétricos



`java.security.KeyPairGenerator`

KeyPairGenerator.getInstance("RSA") Crea una instancia del algoritmo seleccionado.

initialize (int strength) Inicializa claves dando longitud

initialize(int strength, SecureRandom random) Inicializa claves dando longitud y semilla

KeyPair genKeyPair() Genera el par de claves

```
1 KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
2 kpg.initialize(1024);
3 KeyPair kp = kpg.genKeyPair();
```

Código 2: Generación Clave 1024 bits RSA

`java.security.KeyGenerator`

`KeyPairGenerator.getInstance(‘‘DES’’)` Crea una instancia del algoritmo seleccionado.

`init(SecureRandom random)` Proporciona generador de números aleatorios.

`init(int strength)` Establece longitud.

`init(int strength, SecureRandom random)` Unión de los dos anteriores.

`SecretKey generateKey()` Generador de claves.

```
1 KeyGenerator kg = KeyGenerator.getInstance("DES");
2 kg.init(new SecureRandom());
3 SecretKey key = kg.generateKey();
```

`javax.crypto.spec.SecretKeySpec`. Implementa el interfaz SecretKey

`SecretKeySpec(byte[] key, String algorithm)` Crea SecretKeySpec usando un byte array y el algoritmo para su uso.

`SecretKeySpec(byte[] key, int offset, int len, String algorithm)` Idem anterior, estableciendo un offset y una longitud.

```
1 SecureRandom sr = new SecureRandom();
2 byte[] keyBytes = new byte[20];
3 sr.nextBytes(keyBytes);
4 SecretKey key = new SecretKeySpec(keyBytes, "HmacSHA1");
```

Código 4: Ejemplo Creación Clave Secreta

`java.security.KeyFactory`

`KeyFactory getInstance(String algorithm)` Creamos un KeyFactory para un algoritmo asimétrico dado.

`PublicKey generatePublic(KeySpec keySpec)` Convertir de KeySpec a PublicKey

`PrivateKey generatePrivate(KeySpec keySpec)` Convertir de KeySpec a PrivateKey

`KeySpec getKeySpec(Key key, Class keySpec)` Convertir de Clave a KeySpec

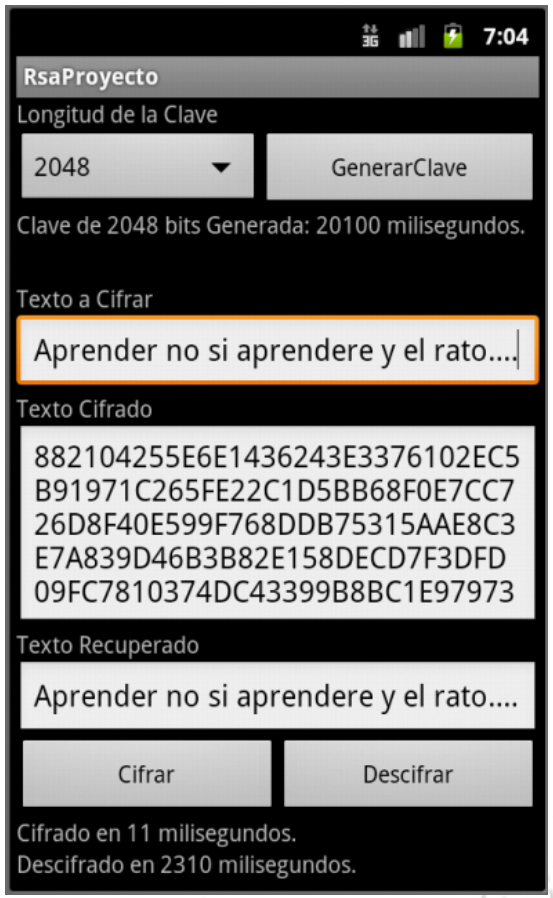


Figura 1: Aplicación que cifra y descifra utilizado RSA

Práctica 1

Modifique la aplicación mostrada en la imagen 1 para que sea posible firmar y verificar un mensaje mediante el algoritmo RSA. Recuerde que la firma es realizada con la clave privada y la verificación con la clave pública.
Las funciones de cifrado y descifrado son definidas como sigue:

$$E_{n,e}(m) = m^e \ (m \in \mathbb{Z}_n),$$
$$D_{n,d}(c) = c^d \ (c \in \mathbb{Z}_n)$$

Mientras que las funciones de firma y verificación son:

$$E_{n,d}(m) = m^d \ (m \in \mathbb{Z}_n),$$
$$D_{n,e}(c) = c^e \ (c \in \mathbb{Z}_n)$$

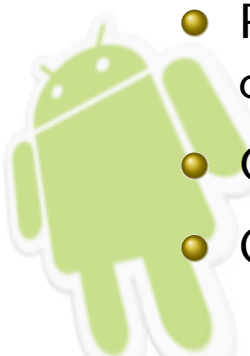
Cifrado Simétrico

La principal clasificación de los Algoritmos Simétricos:

- Algoritmos Bloque: cifran y descifran bloques fijos de datos, normalmente 64 bits.
- Algoritmos Flujo: operan sobre un flujo de bits o bytes.

Los Algoritmos de Bloque tienen distintos modos de operación:

- ECB (electronic code book). Bloque claro $\rightarrow$ Bloque cifrado.
- CBC (cipher block chaining). Bloque claro Xor Bloque cifrado.
- PCBC (Propagating cipher block chaining). Bloque cifrado Xor Bloque claro Xor Bloque anterior cifrado.
- CFB (Cipher feedback)
- OFB (Output feedback)



`javax.crypto.Cipher`

Esta clase proporciona acceso para implementar los algoritmos criptográficos para cifrado y descifrado.

El principal método es `Cipher.getInstance` que permite seleccionar el algoritmo a utilizar:

```
Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
```

Algunos métodos de la clase a destacar son:

`init(int opmode, Key key)` Este método inicializa el algoritmo para cifrar o descifrar. Opmode puede ser ENCRYPT_MODE o DECRYPT_MODE.

`update(byte [ ] input)` Proporciona información al algoritmo.

`byte[] doFinal( )` Finaliza el proceso de cifrado o descifrado.

```
1 Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
2
3 cipher.init (Cipher.ENCRYPT_MODE, key);
4 byte[ ] textoclaro    = "criptografia en Android".getBytes();
5 byte[ ] textocifrado = cipher.doFinal(plaintext);
```

Código 5: Ejemplo de cifrado DES

`javax.crypto.CipherInputStream` y `javax.crypto.CipherOutputStream`

Usado para cifrar y descifrar fácilmente información.

```
1   FileOutputStream fileOut = new FileOutputStream("archivo.txt");
2   CipherOutputStream out = new CipherOutputStream(fileOut, cipher);
3
4 byte[] buffer = new byte[kBufferSize];
5 int length;
6
7 while ((length = in.read(buffer)) != -1)
8 out.write(buffer, 0, length);
```

Código 6: Ejemplo de cifrado con flujo datos

Práctica 2

Modifique la aplicación anterior para que permita cifrar y descifrar con el algoritmo simétrico DES.
Realice los cambios visuales y computacionales oportunos para realizar la nueva aplicación.

Hash

`java.security.MessageDigest`

`MessageDigest getInstance(String algorithm)` Generador.

`update(byte input)` Proporciona la información a la función hash.

`digest()` Realiza la función hash.

```
1 MessageDigest md = MessageDigest.getInstance("MD5");
2 md.update(inputData);
3 byte[ ] digest = md.digest();
```

Código 7: Aplicación función hash MD5

Hash

`java.security.MessageDigest`

`MessageDigest getInstance(String algorithm)` Generador.

`update(byte input)` Proporciona la información a la función hash.

`digest()` Realiza la función hash.

```
1 MessageDigest digester = MessageDigest.getInstance("MD5");
2   byte[] bytes = new byte[8192];
3   int byteCount;
4   while ((byteCount = in.read(bytes)) > 0) {
5     digester.update(bytes, 0, byteCount);
6   }
7   byte[] digest = digester.digest();
```

Código 8: Aplicación función hash MD5

java.security.DigestInputStream

```
1  MessageDigest md = MessageDigest.getInstance("MD5");
2  FileInputStream in = new FileInputStream(args[0]);
3  b
4  yte[] buffer = new byte[8192];
5  int length;
6
7  while ((length = in.read(buffer)) != -1)
8        md.update(buffer, 0, length);
9
10 byte[] raw = md.digest();
```

Código 9: Ejemplo DigestInputStream

java.security.DigestOutputStream

```
1 MessageDigest md = MessageDigest.getInstance("MD5");
2
3  DigestInputStream in = new DigestInputStream(
4 new FileInputStream(args[0]), md);
5
6   byte[] buffer = new byte[8192];
7   while (in.read(buffer) != -1);
8
9  byte[] raw = md.digest();
```

Código 10: Ejemplo DigestOutputStream

MAC: Hash+Clave

`javax.crypto.Mac`

Realiza la función Hash añadiendo una clave.

`Mac getInstance(String algorithm)` Generador.

`init(Key key)` Inicializamos la función con la clave que vamos a utilizar.

`update(byte input)` Añadimos información a la función.

`digest()` Realiza la función MAC.

```
1  SecureRandom sr = new SecureRandom();
2  byte[] keyBytes = new byte[20];
3  sr.nextBytes(keyBytes);
4  SecretKey key = new SecretKeySpec(keyBytes, "HmacSHA1");
5
6   Mac m = Mac.getInstance("HmacSHA1");
7   m.init(key);
8   m.update(inputData);
9   byte[] mac = m.doFinal();
```

Código 11: Ejemplo MAC

Firmas

`java.security.Signature`

`Signature getInstance(String algorithm))` Generador.

`initSign(PrivateKey privateKey)` Inicializa la firma con una clave privada dada.

`initVerify(PublicKey publicKey)` Inicializa la verificación con una clave pública dada.

`update(byte input)` Aporta información para hacer la firma

`byte[] sign( )` Realiza la firrma.

`boolean verify(byte[] signature)` Verifica la firma.

Certificados de Identidad

En este caso nos encontramos con dos paquetes que contienen la clase Certificado.

- `javax.security.cert`. Este paquete es incluido por razones de compatibilidad. Contiene una versión simplificada de JSSE.
- `java.security.cert`. Todas las aplicaciones que no tengan por que ser compatibles con las versiones de JSSE utilizarán este paquete que es una versión limitada de la librería Bouncy Castle.

`java.security.cert.X509Certificate`

Esta clase representa los certificados de identidad[2] que vinculan la clave pública con su identidad, y es el "formato" estándar, como es el caso del DNI electrónico y el protocolo SSL/TLS.

La versión que se incluye en Android solamente contiene solamente los métodos para realizar consulta de los elementos del certificado y quedan excluidos los métodos que permiten crear un certificado.

Para la creación de certificados, así como el tratamiento de otros elementos criptográficos, es necesario incluir en nuestro proyecto la librería Spongy Castle que veremos a continuación.

[2] Información detallada en RFC 2459 "Internet X.509 Public Key Infrastructure Certificate and CRL Profile"

checkValidity ( ) : Verifica la validez del certificado, si ha expirado o no.

getSubjectDN( ) : Obtiene el usuario que firma el certificado, para certificar su validez.

getIssuerDN( ) : Obtiene el propietario del certificado, sería el portal web en el caso SSL y el nombre/apellidos en el caso DNI.

getIssuerAlternativeNames( ) : Obtener nombres alternativos, a veces utilizado para establecer varios nombres distintos al propietario del certificado.

getPublicKey( ) : (Heredada de Certificado). Obtener Clave Pública del certificado.

verify (PublicKey key) : (Heredada de Certificado). Verificar la firma del certificado con la clave pública de la autoridad que lo firmó.

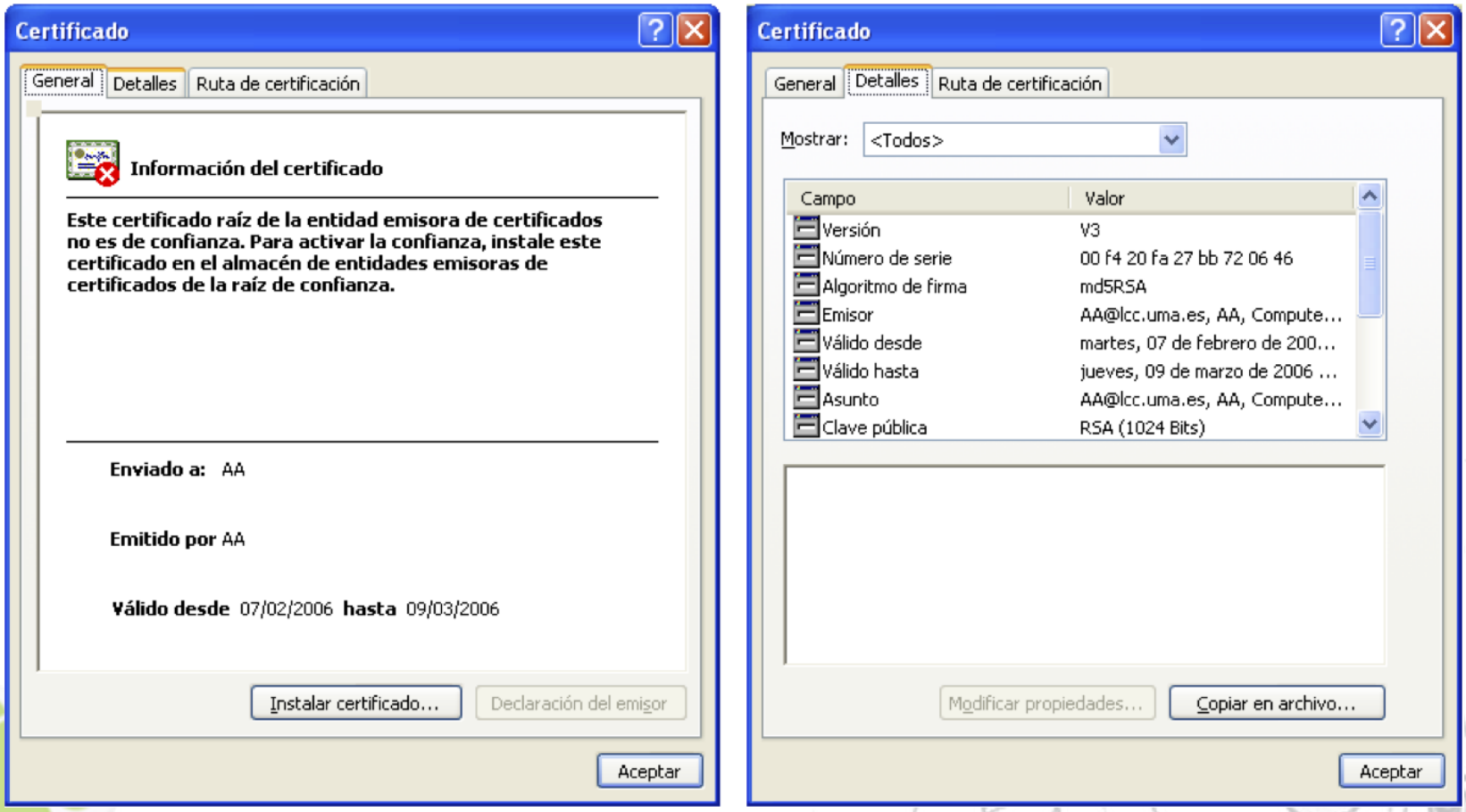


Figura 2: Certificados X509 en Windows

Protocolo SSL

El protocolo SSL (Secure Socket Layer) o TLS (Transport Layer Security) permite establecer una capa de seguridad sobre otros protocolos que carecen de ella. El protocolo que más lo utiliza en la práctica es el protocolo http.

La figura 3 detalla el protocolo SSL donde inicialmente el servidor envía la cadena de certificados necesarias para identificarse con el cliente y posteriormente, una vez autenticado, se realiza los pasos necesarios para crear un canal seguro.

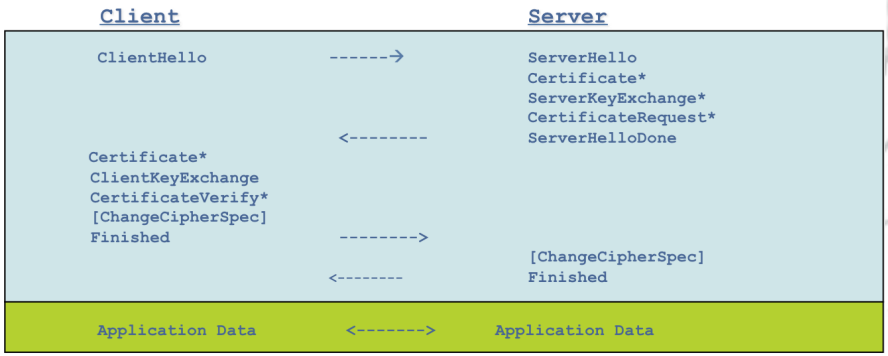


Figura 3: Descripción Protocolo SSL

El siguiente ejemplo muestra como realizar la conexión SSL con un servidor y como obtener los certificados del servidor para identificar correctamente la conexión.

```
1 Certificate[] peerCertificates;
2 SSLSocketFactory factory = HttpsURLConnection.getDefaultSSLSocketFactory();
3
4 /* Conecta SSL al servidor y puerto (443 normalmente) */
5     SSLSocket socket = (SSLSocket)factory.createSocket(host, port);
6     sslSocket.startHandshake();
7
8 /* Obtenemos la cadena de Certificados del Servidor */
9     peerCertificates = sslSocket.getSession().getPeerCertificates();
```

Código 12: Ejemplo Obtención Certificados en Conexión SSL

Práctica 3

Modifique la aplicación mostrada en la figura 4 de forma que verifique la cadena de certificados obtenidas del servidor. Los elementos concretamente a realizar son los siguientes:

- *Verificar las firmas de cada uno de los certificados.*
- *Verificar la cadenas de certificados los nombres de usuarios (Issuer) y certificadoes (Subject) coinciden.*
- *Incluir en la verificación de nombres IssuerAlternativeName.*
- *Verificar si los certificados no están expirados.*
- *Mostrar visualmente una lista de certificados.*

Figura 4: Obtener Certificados de una Conexión SSL

Librería Spongy Castle

Las clases relativas a criptografía en Android están basada en una versión reducida de la librería Bouncy Castle.

Por tanto algunas funcionalidades, así como algunos algoritmos criptográficos no puede ser utilizados dentro de nuestra aplicación Android.

Una solución es incluir la librería original, pero establecería conflictos con las clases instaladas por defecto. Por ello nace, la librería Spongy Castle, la cual es una versión de Bouncy Castle renombrada para evitar los posibles conflictos.

Ejemplo inclusión librería Eclipse

El siguiente ejemplo nos muestra como incluir la librería en Eclipse (link).

La figura 5 muestra la inclusión de la librería en el árbol del proyecto, así como en las propiedades del proyecto.

La aplicación de ejemplo (ver figura6) muestra información de dos claves privadas (RSA:`id_rsa_sample` y Curva Elíptica:`id_ecdsa_sample`) almacenadas en el proyecto dentro de la carpeta `assets`.

Con la inclusión de esta librería tendremos a disposición algoritmos criptográficos como es el caso de los basados en Curvas Elíptica, así como la creación de certificados y un largo etcétera. Más información aquí.

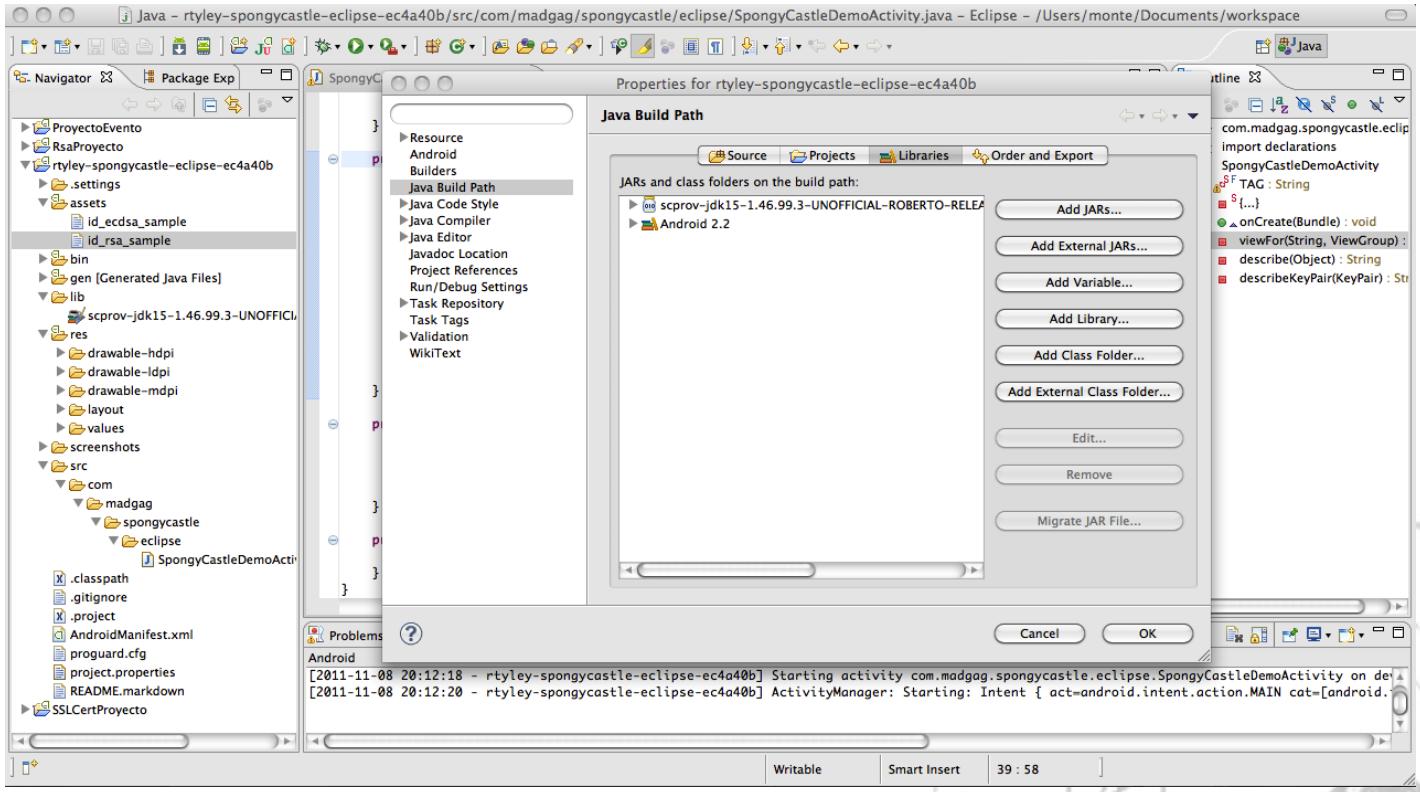


Figura 5: Inclusión de la librería SpongyCastle en Eclipse

Figura 6: Ejecución del Ejemplo de Instalación de la Librería

¿Qué es Openssl?

- OpenSSL es un conjunto de herramientas que implementan los protocolos (SSL v2/v3) y (TLS v1[3]).
- Además es una librería que proporciona rutinas criptográficas, basada en un desarrollo anterior, la librería SSLeay.
- Distribuciones en www.openssl.org.
- Actualmente existen multitud de proyectos que utilizan esta librería como base para incluir algoritmos criptográficos en sus aplicaciones. Una relación de aplicaciones que hacen uso de OpenSSL puede ser consultada en este link.

[3] RFC 5246: The Transport Layer Security (TLS) Protocol

Compilando Openssl en Android

- Para compilar Openssl en Android es necesario tener instalado Android-ndk. La compilación ha sido realizada utilizando `android-ndk-r6b`.
- La distribución de openssl "adaptada" para Android puede encontrarse en este link.
- Ahora solo nos queda ejecutar `ndk-build` en el directorio de openssl.
- Al finalizar la compilación nos encontraremos en el subdirectorio de openssl `obj/local/armeabi`:
  - Dos librerias `libcrypto.so` y `libssl.so`.
  - El ejecutable openssl.

```
1 cd eighthave-openssl-android-7834d21 /* Directorio donde esta openssl */
2 ../android-ndk-r6b/ndk-build   /* Directorio donde esta ndk, ejectuto ndk-build */
```

Instalando Openssl en Android

- Primero copiamos el ejecutable y el archivo de configuración en el dispositivo o emulador:

```
1       ./adb push openssl /sdcard/
2       ./adb push CAss.cnf /sdcard/
```

- Ejecutamos la línea de comandos y creamos un directorio dentro del directorio `/data/data`.
- En el directorio `/data/data` tenemos permisos de ejecución y escritura.
- Finalmente establecemos los permisos necesarios para su ejecución con la sentencia `chmod`.

Solamente debemos copiar el ejecutable en caso que `openssl` sea compatible con las librerias ya instaladas en Android `/system/lib`. En caso que no sean compatibles es necesario instalar las dos librerías creadas.

```
1     ./adb shell  /* Establezco una consola */
2
3  /* Ya estoy en el emulador o dispositivo */
4     cd /data/data
5     mkdir openssl
6     cd openssl
7
8     cd  /data/data/openssl
9     cat /sdcard/openssl > openssl
10    cat /sdcard/CAss.cnf > CAss.cnf
11
12 /* Permiso de ejecucion */
13    chmod 755 openssl
```

Para verficar que funciona la instalación vamos a instalar un servidor SSL en el dispositivo o emulador Android.

```
1  /* 1- Creamos certificados y claves del Servidor SSL */
2
3        openssl req -config /sdcard/CAss.cnf -x509 -nodes -days 365
4                -subj "/C=ES/ST=Andalucia/L=Malaga/CN=www.uma.es"
5                -newkey rsa:1024 -keyout /sdcard/server.pem
6                -out /sdcard/server.pem
7
8   /* 2- Ejecutamos el servidor SSL con la clave y certificado creado */
9
10       openssl s_server -cert /sdcard/server.pem -www -verify 1
```

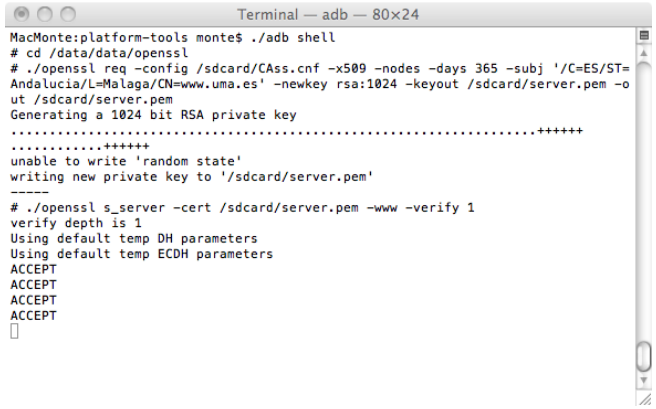


Figura 7: Ejecución Servidor SSL con OpenSSL

Finalmente el navegador se conectará al servidor ssl que hemos instalado y mostrará primero (figura 8) un certificado no confiable y si aceptamos conexión la información SSL (figura 9).

```
1 /* 3- Finalmente ejecuto el navegador. */
2
3  adb shell am start https://localhost:4433
```

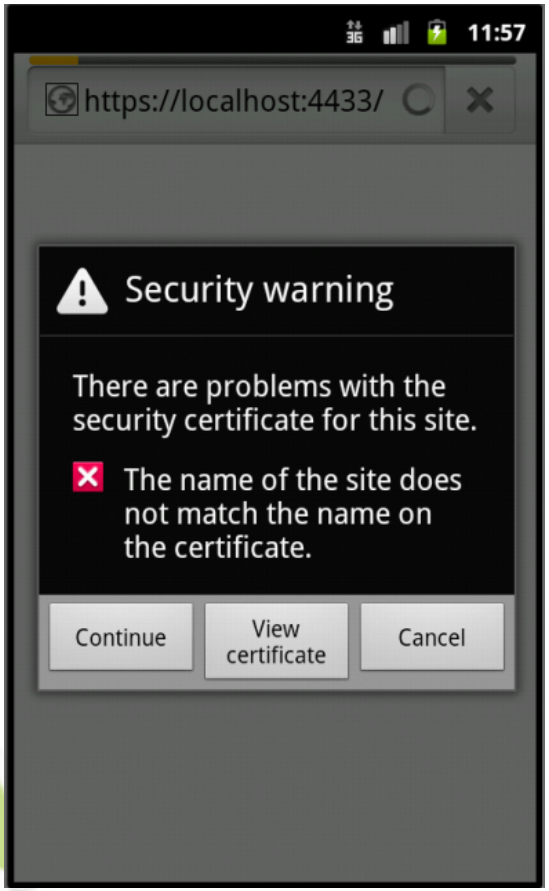


Figura 8: SSL Navegador

Figura 9: Configuracion SSL

Ejecución SSL desde Java

Existe una posibilidad de instalar el ejecutable Openssl desde java. El siguiente código copia de la carpeta Assets del proyecto el archivo binario al directorio /data/data.

```
1  /*Openssl esta almacenado carpeta Assets del proyecto*/
2        InputStream in= getAssets().open("openssl");
3        FileOutputStream out = new FileOutputStream(localPath);
4        int read;
5      byte[] buffer = new byte[4096];
6      while ((read = in.read(buffer)) > 0) {
7      out.write(buffer, 0, read);
8      }
9        out.close(); in.close();
10 /*Establezco los permisos de Openssl*/
11     exec("/system/bin/chmod 744 " + localPath);
```

Este código permite ejecutar un proceso del Dispositivo o Emulador, obteniendo la salida del resultado de la ejecución del programa. /data/data.[4].

```
1  Process process = Runtime.getRuntime().exec(command);
2              BufferedReader reader = new BufferedReader(
3                      new InputStreamReader(process.getInputStream()));
4              int read; char[] buffer = new char[4096];
5              StringBuffer output = new StringBuffer();
6              while ((read = reader.read(buffer)) > 0) {
7                  output.append(buffer, 0, read);
8              }
9              reader.close();
10           process.waitFor();
```

[4] El proyecto es una modificación del proyecto android-native-exe-demo. Más información aquí

Figura 10: Ejecución OpenSSL desde Java

Comandos OpenSSL

Como hemos visto además de ser una librería que nos "libra" de programar los algoritmos criptográficos, OpenSSL nos proporciona un comando para realizar diversas operaciones criptográficas.

A continuación veremos algunos de los comandos disponibles.

- Cifrar y Descifrar con Crip. Simétrica, ej. DES.
- Generación de Claves Crip. Asimétrica, ej. RSA.
- Generación de Certificados Identidad X590.

Más información en el siguiente link.

Definición 1 (ENC)

El comando `enc` permite cifrar – descifrar archivos.

Ejemplo 1

Cifra un archivo usando triple DES en modo CBC usando un solicitando un "password".

```
des3 -salt -in file.txt -out file.des3
```

Descifra un archivo proporcionando el password

```
des3 -d -salt -in file.des3 -out file.txt -k mypassword
```

Definición 2 (GENRSA)

El comando `genrsa` permite crear claves rsa.

Ejemplo 2

Genera una clave RSA de 1024 bits y la almacena en la tarjeta cifrada con Des

```
openssl genrsa -out /mnt/sdcard/miclave.cif -des 1024
```

Definición 3 (GENDSA)

El comando `gendsa` permite crear claves dsa.

Definición 4 (RSA)

El comando `rsa` permite manipular una clave ya generada.

Ejemplo 3

Elimina la palabra de paso en una clave privada, previa petición

```
rsa -in key.pem -out keyout.pem
```

Imprime información de una clave privada en pantalla

```
rsa -in key.pem -text -noout
```

Obtiene la parte pública de la clave

```
rsa -in key.pem -pubout -out pubkey.pem
```

Definición 5 (DSA)

El comando `dsa` permite manipular una clave ya generada.

Definición 6 (REQ)

El comando `req` genera una petición de certificado o PKCS_10

Ejemplo 4

Crea una petición de certificado y la clave a la vez.

```
req -newkey rsa:1024 -keyout key.pem -out req.pem
```

Crea un Certificado Autofirmado y la clave a la vez.

```
req -x509 -newkey rsa:1024 -keyout key.pem -out req.pem
```

Definición 7 (x509)

El comando x509 permite generar y manipular certificados.

Ejemplo 5

CA (ca.crt, ca.key) firma una petición de certificado y muestra por pantalla.

```
X509 -req -in peticion.usr -CA ca.crt -CAkey ca.key -Cacreatesertial
```

Muestra información del certificado en modo texto.

```
X509 -in certif.crt -noout -text
```

Muestra el número de serie del certificado.

```
X509 -in certif.crt -noout -serial
```

Implementación propietaria algoritmos criptográficos

Hasta ahora hemos visto tres opciones para incluir elementos criptográficos en nuestras aplicaciones:

1. Las posibilidades que nos ofrece la SDK de Android para implementar algoritmos criptográficos.
2. Hemos ampliado las posibilidades incluyendo una librería externa de critografía.
3. La posibilidad de ejecutar el comando OpenSSL.

Y nos queda una opción más que es la inclusión de implementaciones propias de los algoritmos criptográficos.

Algoritmos Criptografía Clásica

El proyecto cryptoid es un ejemplo de aplicación del cifrado y descifrado utilizando matrices (link).

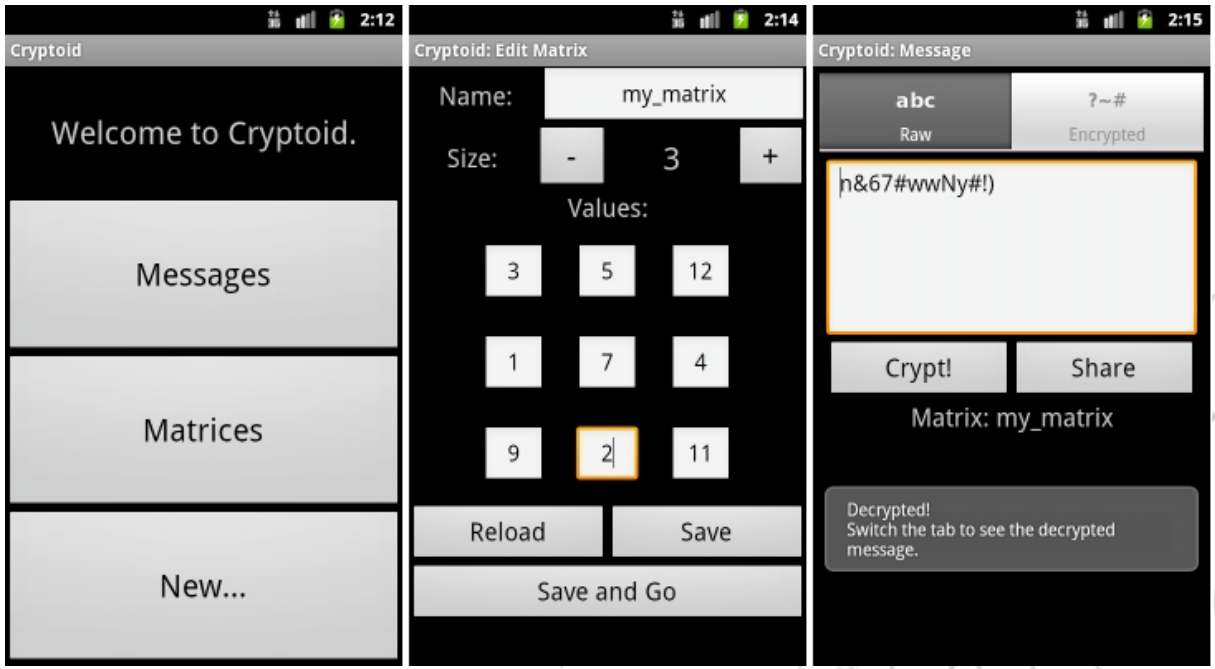


Figura 11: Ejecución Aplicación CryptoID

Implementación RSA

```
/** Constructor de la clase RSA */
   public RSA(int tamPrimo) {
       this.tamClave = tamPrimo;
      generaPrimos();                //Genera p y q
      generaClaves();                //Genera e y d
   }

   /* Genera los Primos   */
   public void generaPrimos()
   {
       p = new BigInteger(tamClave/2, 10, new Random());

       do q = new BigInteger(tamClave/2, 10, new Random());
           while(q.compareTo(p)==0);
   }
```

```
/*** Genera claves privada y publica ***/
    public void generaClaves()
    {
        // n = p * q
        n = p.multiply(q);
        // toltient = (p-1)*(q-1)
        totient = p.subtract(BigInteger.valueOf(1));
        totient = totient.multiply(q.subtract(BigInteger.valueOf(1)));

        // Elegimos un e coprimo de y menor que n
        do e = new BigInteger(tamClave, new Random());
        while((e.compareTo(totient) != -1) ||
              (e.gcd(totient).compareTo(BigInteger.valueOf(1)) != 0));
        // d = e^1 mod totient
        d = e.modInverse(totient);
    }
```

```
/** Cifra el texto usando la clave publica
     * @param   mensaje    Mensaje a cifrar
     * @return   El mensaje Cifrado  */

    public BigInteger[] cifrar(String mensaje)
    {
        int i; byte[] temp = new byte[1];
        byte[] digitos = mensaje.getBytes();
        BigInteger[] bigdigitos = new BigInteger[digitos.length];

        for(i=0; i<bigdigitos.length;i++){
            temp[0] = digitos[i];
            bigdigitos[i] = new BigInteger(temp);
        }
        BigInteger[] cifrado = new BigInteger[bigdigitos.length];
        for(i=0; i<bigdigitos.length; i++)
            cifrado[i] = bigdigitos[i].modPow(e,n);

        return(cifrado);
    }
```

```java
/**
 * Descifra el texto cifrado usando la clave privada
 *
 * @param   Array objetos
 * @return  El texto en claro
 */
public String descifrar(BigInteger[] cifrado) {
    BigInteger[] descifrado = new BigInteger[cifrado.length];

    for(int i=0; i<descifrado.length; i++)
        descifrado[i] = cifrado[i].modPow(d,n);

    char[] charArray = new char[descifrado.length];

    for(int i=0; i<charArray.length; i++)
        charArray[i] = (char) (descifrado[i].intValue());

    return(new String(charArray));
}
```

José A. Montenegro Montes
Dpto. Lenguajes y Ciencias de la Computación
ETSI Informática. Universidad de Málaga
monte@lcc.uma.es
twitter