uc3m | Universidad Carlos III de Madrid

PRÁCTICA: CRIPTOGRAFÍA SIMÉTRICA Y ASIMÉTRICA

CURSO CRIPTOGRAFÍA Y SEGURIDAD INFORMÁTICA

Ana I. González-Tablas Ferreres
José María de Fuentes García-Romero de Tejada
Lorena González Manzano
Pablo Martín González
UC3M | GRUPO COMPUTER SECURITY LAB (COSEC)

HERRAMIENTAS

En esta práctica se va a hacer uso de un programa en Java creado para los alumnos. En él se hace uso de la librería Bouncy Castle. La librería escogida es Bouncy Castle, que consta de una serie de Clases y APIs de criptografía con versiones para Microsoft .NET, Java y JavaME que además ha adquirido gran relevancia debido a que es la elegida por Google como implementación de algoritmos criptográficos por defecto en su sistema Android.

INTRODUCCIÓN

Actualmente es necesario en cualquier entorno cliente-servidor actual el envío, recepción (e incluso el almacenamiento de los datos) en forma cifrada, bien sea por motivos de seguridad (información sensible o secreta), de privacidad de acceso (no es información pública y sólo los usuarios registrados deben poder conectar al servidor), o bien por seguridad ante ataques.

Para ello, prácticamente todos los lenguajes de programación incorporan o disponen de una serie de APIs o librerías de criptografía. Hay que prestar una especial atención a la hora del desarrollo en el caso de sistemas mixtos (por ejemplo, aplicaciones móviles que se conectan a un servidor Linux en internet), ya que ambas implementaciones de los métodos criptográficos deben posibilitar la comunicación (deben implementar el mismo tipo de cifrado, de modo de bloque, de tamaño de éste, de tamaño de clave, etc.).

En esta práctica utilizaremos las librerías Bouncy Castle sobre el lenguaje Java.

Java incorpora su propia arquitectura de criptografía llamada JCA (Java Cryptography Architecture) diseñada para poder incorporar una serie de proveedores de criptografía (Bouncy Castle puede ser uno de ellos) y ofrece un conjunto de APIs estándar para interactuar con ellos independientemente de cual sea. De esta manera podríamos cambiar de proveedor de criptografía (librerías criptográficas) sin que dicho cambio afectase al código fuente de la aplicación. Si el alumno desea profundizar en JCA, puede comenzar en http://es.wikipedia.org/wiki/Java_Cryptography_Architecture

En esta práctica se ha optado por usar las librerías Bouncy Castle no como proveedor criptográfico en JCA, sino como librería independiente (Lightweight API) con el objetivo de que el alumno se familiarice con el uso de este tipo de librerías con independencia del lenguaje de programación escogido (no todos los lenguajes de programación incorporan APIs y Frameworks como el JCA de Java).

Estructura de la librería Bouncy Castle

La librería Bouncy Castle no sólo implementa algoritmos de cifrado, sino también algoritmos de firma, generación de certificados, control de acceso, validación de datos, autenticación por DNS, etc.

Está organizada en forma de paquetes Java, teniendo como raíz el paquete org.bouncycastle. En esta práctica se va a hacer uso del paquete org.bouncycastle.crypto, que contiene la implementación de los algoritmos de cifrado y de función resumen que usaremos.

En el paquete org.bouncycastle.crypto.test el alumno encontrará código Java que contiene ejemplos de implementación de distintos cifradores. Puede consultar la documentación online en formato Javadoc contenido dentro del archivo docs1.5on, que se incluye en el directorio "doc" que acompaña a la práctica. También puede consultar dicha documentación on-line en el siguiente enlace:

https://www.bouncycastle.org/docs/docs1.5on/index.html

https://www.bouncycastle.org/specifications.html

Preste atención a la documentación y al código incluido en la práctica, que orientará al alumno en la forma de uso de la librería. Por ejemplo, en BouncyCastle el proceso normal para crear y utilizar un cifrador simétrico es:

1. Instanciar un objeto de una clase Engine (por ejemplo, AESEEngine)
2. Elegir un modo de operación para cifradores de bloque (por ejemplo CBCBlockCipher) e instanciarlo sobre el objeto Engine.
3. Crear un cifrador a partir del de bloque (por ejemplo PaddedBufferedBlockCipher) que especifique las características que va a tener el cifrador (por ejemplo, padding usando PKCS5).

Posteriormente tenemos que crear una clave. En este caso de ejemplo, como necesitamos un vector de inicialización IV debido al modo de operación elegido, creamos un objeto KeyParameter y a partir de él un objeto KeyParameterWithIV, el cual es usado en la inicialización del cifrador obtenido en el paso 3.

Ejecución de la práctica

Debe ejecutar las demostraciones de cifrado y función resumen que se ofrecen en el código proporcionado para la práctica. Para ejecutar las demostraciones de cifrado y

función resumen de la práctica se debe ejecutar el archivo Demos.java, contenido en el paquete p2.demos. Posteriormente, debe acometer las cuestiones que se le planteen en cada apartado.

El código de la práctica consta de un proyecto Eclipse, por lo que se puede ejecutar de dos maneras, desde el propio IDE o desde la consola de comandos. En el caso de ejecutarse desde la consola de comandos debe asegurarse de que se ejecuta desde el directorio raíz de la práctica y que está incluida en el ClassPath el archivo lib/bcprov-ext-jdk15on-151.jar, que contiene las librerías Bouncy Castle.

Con el objetivo de facilitar al alumno el uso de las demostraciones todo el almacenamiento y lectura de archivos se realiza en el directorio raíz "files" desde la propia aplicación.

A continuación, se enumeran las distintas opciones disponibles durante la ejecución de la demostración así como sugerencias de cambio del código y una serie de preguntas que permitan al alumno profundizar en la comprensión del uso de la librería:

EJERCICIOS

1. Crear archivo de texto

i) Crear un fichero de texto

Permite crear un archivo de texto dentro de la carpeta files para poder usarlo posteriormente en las demás opciones de la demostración

2. Sección DES

ii) Generar clave para el algoritmo de cifrado DES

Genera una clave simétrica válida para el algoritmo DES y la almacena en el directorio files con la extensión "deskey".

Para ello hace uso de la clase SecureRandom de Java para generar un número "lo más aleatorio posible" a partir de una semilla secreta y un generador de claves para el algoritmo DES a partir de dicho número aleatorio.

La clave se almacena en el archivo en forma de cadena de texto en formato Hexadecimal para facilitar la lectura por parte del alumno.

Ejercicio 1 :

a) Consulte la siguiente página web y evalúe con la información proporcionada si hubiese sido más adecuado utilizar la clase java.util.Random de Java.

b) Observe el método “generateKey” de la clase “DES.java”. ¿Por qué se multiplica la longitud preestablecida de una clave DES por 8? Cambie dicha longitud por otra, ¿qué ocurre durante la ejecución?

Solución:

a) Con la clase java.util.Random, dos instancias que utilicen la misma semilla van a crear la misma secuencia aleatoria. Así, si un atacante aprende la semilla, sería capaz de generar la secuencia. Además de considerar que las semillas pueden ser reutilizadas. Por tanto, java.util.Random no se debe utilizar en aplicaciones en las que la seguridad sea un aspecto crítico o en las que se protejan datos. Este problema lo resuelve el uso de la clase java.security.SecureRandom.

b) Se multiplica por 8 porque el constructor de KeyGenerationParameters necesita la longitud en bits y la constante DESParameters.DES_KEY_LENGTH la expresa en bytes.

Si cambia la longitud de la clave generada se produce una excepción debido a que la clave DES debe ser de 64 bits.

iii) Cifrar/Descifrar un archivo usando DES

Permite cifrar y descifrar un archivo de texto con una clave creada anteriormente (ambos se deben encontrar dentro del directorio “files” de la práctica).

Para ello se crea un “motor” de cifrado de tipo DES y se construye sobre él un cifrador en modo Bloque CBC con padding. Posteriormente se inicializa dicho cifrador con la clave elegida y se realiza el procesamiento.

Ejercicio 2 :

a) Observe los métodos encrypt y decrypt de la clade DES.java. ¿pueden unificarse ambos métodos en uno sin modificar el resultado de la ejecución?

b) Consulte la documentación de Bouncy Castle y cambie el modo de Bloque del cifrador al modo CFB y OFB. Genere un archivo cifrado para cada modo de bloque a partir de un mismo texto en claro. Si cambiamos algún byte del archivo cifrado (es decir, introducimos errores), ¿qué ocurre al intentar descifrarlo en cada uno de los modos?

Solución:

a) Al ser un cifrado simétrico se podría utilizar el método encrypt tanto para cifrar como para descifrar y sería 100% operativo. Sólo es necesario cambiar la inicialización del método.

b) Para cambiar los modos de bloque basta cambiar la clase CBCBlockCipher por otras que implementan otros modos (CFBBlockCipher, OFBBlockCipher, etc..) e inicializarlos según lo que pone en su documentación. Verá como al cifrar y descifrar con los distintos modos de operación la única diferencia es que los textos cifrados son distintos, pues el modo es distinto.

3. Sección AES

iv) Generar clave para el algoritmo de cifrado AES

Genera una clave y vector de inicialización válidos para el algoritmo AES y los almacena concatenados en el directorio files con la extensión “aeskey”.

Ejercicio 3 :

a) Observe el método “generateKey” de la clase “AES.java”. ¿Por qué se añade blocksize a la longitud de la clave AES en la generación del número aleatorio? Cambie dicha longitud por otra, ¿qué ocurre durante la ejecución?

Solución:

a) Se añade a la longitud de la clave AES la longitud del bloque del cifrador, ya que la parte del IV debe tener, como es lógico, la misma longitud que un bloque.

Si cambia la longitud de la clave generada pueden pasar dos cosas: si la clave resultante es mayor de la longitud deseada se almacenará en el archivo únicamente la parte que corresponde a las longitudes necesarias.

Si la longitud de la nueva clave es menor se producirán excepciones en el código a la hora de intentar grabar/leer la clave AES a/desde el archivo.

Si se consiguiese leer/grabar la clave+IV con distintas longitudes se generarían excepciones de ejecución a la hora de cifrar/descifrar, ya el tamaño del IV sería distinto al tamaño del bloque usado.

v) Cifrar/Descifrar un archivo usando AES

Permite cifrar y descifrar un archivo de texto con una clave creada anteriormente (ambos se deben encontrar dentro del directorio "files" de la práctica).

Ejercicio 4 :

a) Observe los métodos encrypt y decrypt de la clade AES.java. ¿pueden unificarse ambos métodos en uno sin modificar el resultado de la ejecución?

b) El algoritmo Rjindael es un modo "rápido" de cifrado AES que permite distintos tamaños de clave mientras que ésta sea múltiplo de 32 bits. Modifique el método encrypt para que se cifre mediante el algoritmo Rjindael que sea compatible con la función decrypt ya existente. La implementación de Rjindael en BouncyCastle se realiza en la clase RjindaelEngine.

Solución:

a) AES no es reversible como DES, por lo que no se puede utilizar el mismo método para cifrar y descifrar. No obstante, la única diferencia en ambo métodos para cifrar/descifrar es la inicialización del cifrador (método init). Si ponemos el primer parámetro a true, el cifrador se configura para cifrar, y si lo ponemos a false, para descifrar. Por tanto, sí es posible unificar ambos métodos (y recomendable), pero sólo a efectos de código, no porque el algoritmo AES sea simétrico.

b) Habría que especificar el mismo tamaño de clave y tamaño/modo bloque para Rjindael que lo haga compatible con el estándar AES utilizado). Ejemplo de cifrador Rijndael:

```
BlockCipher engine = new RijndaelEngine(256);
BufferedBlockCipher cipher = new PaddedBufferedBlockCipher(new
CBCBlockCipher(engine), new ZeroBytePadding());
byte[] keyBytes = "0123456789abcdef0123456789abcdef".getBytes();
cipher.init(true, new KeyParameter(keyBytes));
byte[] input = "value".getBytes();
byte[] cipherText = new byte[cipher.getOutputSize(input.length)];
int cipherLength = cipher.processBytes(input, 0, input.length, cipherText, 0);
cipher.doFinal(cipherText, cipherLength);
String result = new String(Base64.encode(cipherText));
log.debug("result : " + result);
```

4. *Sección Funciones Resumen*

En la práctica se implementan las funciones resumen MD5 y SHA1 en el fichero Hash.java. En bouncyCastle las funciones resumen (digest) derivan de una clase general (GeneralDigest) que contiene las funciones básicas para implementar una función resumen.

Ejercicio 5 :

a) Consulte la documentación de BouncyCastle e implemente una nueva opción que genere resúmenes usando SHA512.

Solución:

a) Un ejemplo de código es el siguiente:

```
public MessageDigest() throws NoSuchAlgorithmException {
   Security.addProvider(new BouncyCastleProvider());
   messageDigest = new SHA512Digest();
}
```

5. *Sección RSA*

vi) Generación de par de claves RSA

Observe el método “generateKey” de la clase “RSA.java”. En este caso, y para que el alumno esté familiarizado con el formato, la información de las claves se almacena en archivos en Base64, no en Hexadecimal, usando la clase Base64Encoder

El formato Base64 es un formato de representación de bytes en forma de caracteres ASCII que permite intercambiar información entre distintos ordenadores, sistemas operativos sin que la representación de la información dependa del modo de almacenamiento de ésta entre origen destino (Big-Endian / Little-Endian, etc.). Este formato es especialmente útil cuando se desea transmitir información entre, por ejemplo, dispositivos móviles y servidores Unix en internet. Adicionalmente, esta forma de representación es la estándar para el intercambio de Certificados y Claves públicas y privadas en criptografía

vii) Cifrado/Descifrado RSA

Para el cifrado y descifrado se hace uso del estándar PKCS1, que define la forma de implementar el algoritmo RSA (las propiedades matemáticas de sus claves, las operaciones primitivas a usar en el cifrado y firma, etc.).

Ejercicio 6 :

a) Realice el cifrado de dos ficheros de texto, uno de ellos con una longitud menor en bytes que la longitud de clave usada y otro mayor. ¿Qué ocurre cuando se intenta cifrar un texto mayor que la longitud de la clave?
b) Aumente el tamaño de las claves generadas y compruebe las opciones de generación de claves, cifrado y descifrado. Pruebe con longitudes de 1548 y 2048 bytes. ¿qué ocurre?
c) ¿Es posible unificar el método encrypt y decrypt de RSA en uno sólo?

Solución:

a) La implementación general del cifrado RSA no permite cifrar elementos mayores que el tamaño de la clave. Esto se debe a que el cifrado RSA (y normalmente cualquier cifrado asimétrico) no está diseñado para cifrado “en bloques”, sino para cifrar, por ejemplo, claves de sesión AES (de un único uso) que serán las usadas para cifrar todo un texto en bloques.

b) Pero lo lógico es que en el caso de 1528 bytes no funcione y provoque excepción de longitud de clave incorrecta y en el caso de 2048 funcione perfectamente.

c) Aunque es un sistema de clave asimétrica, la implementación de RSA permite "compartir" el código para cifrar y descifrar siempre que se inicialize con la clave correcta (en un caso usando PrivateKeyFactory y en el otro PubliceyFactory). Por todo lo indicado, la respuesta es sí, mientras que la "carga" de las claves se desplace a los métodos doEncrypt y doDecrypt