**Tema 3: Encriptación Asimétrica**

**3.1. Clave Pública y Clave Privada**

El esquema asimétrico resuelve el problema de necesitar una clave pre-establecida. Se divide la clave en dos partes *par de claves*): la *clave pública* y la *clave privada*. Dichas claves son complementarias: Un mensaje encriptado con la clave pública sólo puede ser desencriptado con la correspondiente clave privada. Por lo tanto, el conocimiento de la clave pública no implica capacidad de desencriptación. Así, una persona podrá difundir su clave pública para que cualquiera pueda enviarle un mensaje cifrado que solamente podrá desencriptarse con la clave privada.

El algoritmo RSA es el más conocido para manejar esquemas asimétricos. Una característica interesante de este método es que las claves son intercambiables a efectos de figurar como públicas o privadas.

En cuanto a la longitud de las claves, esta deberá ser mayor que en el caso simétrco para proporcionar el mismo nivel de seguridad: una clave asimétrica de 1024 bits proporciona la misma seguridad que una simétrica de 128 bits. Esto hace el esquema asimétrico 1000 veces más lento que su homólogo simétrico.

**3.2. Clases e Interfaces en Java**

Para aplicar encriptación asimétrica usaremos básicamente las siguientes clases de **java.security**:

|  |  |
| --- | --- |
| **KeyPair** | Encapsula un par de claves. **getPublic()** devuelve la clave pública y **getPrivate()** la privada |
| **PublicKey** | Interface para claves públicas. En el package **java.security.interfaces** está la sub-interface **RSAPublicKey** que define las claves para el algoritmo RSA y permite acceder a información de las claves RSA. |
| **PrivateKey** | Similar a la anterior, para la clave privada. Hay que considerar las sub-interfaces **RSAPrivateKey** y **RSAPrivateCrtKey** que contienen métodos extra para coger parámetros de estas claves. |
| **KeyPairGenerator** | Las claves pública y privada siempre se generan juntas con el método **genKeyPair().** |

**3.3. Encriptación de Clave-de-Sesión**

La encriptación asimétrica, lenta por naturaleza, es especialmente útil para encriptar claves simétricas. En el modelo de clave-de-sesión (*session-key*) un mensaje se encripta con una clave secreta y ésta a su vez es encriptada con la clave con la clave pública del receptor del mensaje. Cuando éste lo recibe solo tiene que utilizar su clave privada para desencriptar la clave secreta y ésta a su vez para desencriptar el mensaje.

Veamos como realizar este tipo de estrategia en el ejemplo [RSA.java](http://www.jtech.ua.es/j2ee/2002-2003/modulos/seguridad/apuntes/sesion2_archivos/RSA.java). En primer lugar creamos una clave simétrica, tipo Blowfish de 128 bits, para cifrar el texto.

System.out.println("Generando clave Blowfish...");

KeyGenerator generador = KeyGenerator.getInstance("Blowfish");

generador.init(128);

Key claveBlowfish = generador.generateKey();

System.out.println("Formato: "+claveBlowfish.getFormat());

A continuación generamos el par de claves RSA (publica y privada).

System.out.println("Generando par de claves RSA...");

**KeyPairGenerator** generadorRSA = KeyPairGenerator.getInstance("RSA");

generadorRSA.initialize(1024);

**KeyPair** claves = generadorRSA.genKeyPair();

System.out.println("Generada la clave asimétrica.");

Ya podemos crear e inicializar el cifrador RSA que se va a encargar de encriptar la clave Blowfish con la parte pública del par RSA.

>

Cipher cifradorRSA= Cipher.getInstance("RSA/ECB/PKCS1Padding");

cifradorRSA.init(Cipher.ENCRYPT\_MODE, claves.**getPublic()**);

Una vez tenemos este cifrador cogemos los byte de la clave Blowfish y los encriptamos

byte[] bytesClaveBlowfish = claveBlowfish.**getEncoded();**

byte[] claveBlowfishCifrada = cifradorRSA.**doFinal(bytesClaveBlowfish);**

Desencriptamos la clave Blowfish con la parte privada del par RSA.

cifradorRSA.init(Cipher.DECRYPT\_MODE, claves.**getPrivate()**);

byte[] bytesClaveBlowfish2 = cifradorRSA.**doFinal(claveBlowfishCifrada);**

Finalmente recreamos la clave Blowfish.

**SecretKey** nuevaClaveBlowfish = new **SecretKeySpec(bytesClaveBlowfish2, "Blowfish");**

**3.4. Codificación de Claves Públicas y Privadas**

Las claves asimétricas también se pueden codificar, como hicimos con las simétricas, para guardarlas en fichero. Sin embargo, debido a su mayor complejidad, la parte pública de una clave asimétrica se codifica usando X.509 mientras que la parte privada se codifica usando el estándar PKCS#8. Llamando a **getEncoded()** se realiza la codificación con el protocolo adecuado. En cuanto a la decodificación ya no consiste en usar un **SecretKeySpec** sino que se recurre a un *Spec* específico. Por ejemplo, para decodificar la clave pública se usa **X509EncodedKeySpec** y se le pasa a un **KeyFactory**:

X509EncodedKeySpec spec = new **X509EncodedKeySpec(bytesClave);**

KeyFactory factoria = KeyFactory.getInstance("RSA");

PublicKey clavePublica = factoria.**generatePublic(spec);**

Algo similar sucede con la clave privada pero usando **PKCS8EncodedKeySpec** y pasándolo de nuevo a un **KeyFactory**

PKCS8EncodedKeySpec spec = new **PKCS8EncodedKeySpec(bytesClave);**

KeyFactory factoria = KeyFactory.getInstance("RSA");

PublicKey clavePublica = factoria.**generatePrivate(spec);**

**3.5. Encriptación de Ficheros con RSA**

Para ilustrar la aplicación de RSA a ficheros vamos a ver un ejemplo en el que generaremos un par de claves RSA y posteriormente usaremos la parte pública para encriptar un fichero y la parte privada para desencriptarlo.

La primera fase del proceso consiste en generar un par RSA y almacenarlo convenientemente en el sistema de ficheros. Como la parte pública puede ser conocida por otros puede almacenarse de forma débilmente encriptada, esto es codificada con X.509. Sin embargo, la parte privada vamos a encriptarla con un password (con el método **encriptarPBE**) y guardarla codificada con PKCS#8. Esto es lo que hace el método **crearClave()** de [RSAFicheros.java](http://www.jtech.ua.es/j2ee/2002-2003/modulos/seguridad/apuntes/sesion2_archivos/RSAFicheros.java).

La segunda fase del proceso la lleva a cabo el método **encriptar()** usando la clave pública. Se abre el fichero que contiene la clave pública. Como dicha clave está en X.509 lo primero es transformarla en **PublicKey**. Siguiendo el esquema *session-key* dicha clave pública se utilizará para encriptar una clave simétrica (p.e. basada en Rijndael) en la que que a su vez se basará la encriptación propiamente dicha del fichero de entrada. Por seguridad la clave simétrica encriptada se guardará, precedida de su longitud, al principio del fichero de salida.

Seguidamente tiene lugar la encriptación con la clave sesión del fichero de entrada. Como se va a utilizar un cifrador de *stream*, lo primero es generar un IV y guardarlo a continuación de la clave simétrica encriptada en el fichero de salida. Luego creamos un cifrador Rijndael en modo CBC y lo aplicamos para añadir el texto cifrado al fichero de salida. Así, el resultado devuelto es un fichero con la siguiente estructura: *Long\_clave\_sesión\_encriptada + clave\_sesión\_encriptada + iv + texto\_cifrado*.

La segunda fase consiste en llevar a cabo el desencriptado del anterior fichero. Esta tarea la realiza el método **desencriptar()**. Lo primero que hay que tener en cuenta es que dicha clave reside, encriptada por un password en un determinado fichero. Así que procedemos a extraerla y llamamos al método **desencriptarPBE()** para obtener la clave privada original. Sin embargo, como dicha clave está en PKCS#8 hay que traducirla a un objeto **PrivateKey**.

Una vez que tenemos la clave privada, ya podemos utilizarla para desencriptar la clave de sesión que está contenida en la cabecera del fichero a desencriptar. Así pués lo que hay que hacer es leer esta cabecera, inicializar un descifrador RSA, desencriptar la clave y traducirla a un objeto **SecretKey**. La cabecera termina con el IV necesario para inicializar el descrifrador simétrico que trabaja en modo CBC. Aplicando este descifrador conseguimos finalmente desencriptar el texto.