CRIPTOGRAFÍA

Conceptos básicos

- **Mensaje:** Son los datos que actualmente conciernen. Normalmente será un texto plano. Denotado por M

- **Mensaje cifrado:** Es el mensaje M encriptado. Llamamos al mensaje cifrado C

- **Espacio de mensajes:** Son todos los mensajes posibles, en la mayoría de los casos suele ser infinito, pero en otros la longitud puede estar limitada.

- **Espacio de mensajes cifrado:** Lo mismo que el punto anterior pero para mensajes cifrados.

- **Espacio de claves:** Son el conjunto de todas las claves posibles. Se representa por la letra K (mayúscula, la k minúscula representa una clave en concreto)

Matemáticamente, la encriptación no es más que una aplicación desde el dominio de M al rango de C, y la desencriptación es la función inversa.

Expresamos la encriptación como C = E (M)

La desencriptación como M = D(C)

Y el mensaje se obtiene con la siguiente ecuación: M= D(E(M))

Un ejemplo de lo anterior: Un mensaje M = "hola", una función de encriptación escribir la siguiente letra del alfabeto, Y una función de desencriptado escribir la letra anterior del alfabeto nos quedaría.
M ="hola", C= E(M) ="ipmb", M = D(E(M)) = "hola".

Es importante notar la diferencia entre un protocolo y un algoritmo en términos de criptografía.
Un algoritmo es el conjunto de pasos necesarios para transformar un dato en otro dato.
Un protocolo es el conjunto de todos los pasos entorno al transporte de la actividad criptográfica.
Los protocolos usan necesariamente uno o más algoritmos criptográficos.

Criptosistemas de clave privada (algoritmos simétricos).

Normalmente no se suele usar un método tan sencillo de cifrado, ya que este es lineal y solo depende de un factor (Una entrada siempre produce la misma salida). Un paso más allá es usar una clave para realizar el cifrado, denotándolo así

Expresamos la encriptación como C = E {k}(M)
La desencriptación como M = D{k}(C)
M= D{k} (E {k} (M))
Pero no es lo mismo cifrar y descifrar con dos claves distintas.
Entonces M != D{k1}(E{k2}(M), donde k1 != k2

Un ejemplo moderno de este tipo de sistema es el algoritmo DES.

Criptosistemas de clave publica (algoritmos asimétricos).

Podemos dividir el tipo de encriptaciones según usen o no la misma contraseña, para encriptar y desencriptar.

Decimos que la encriptación es simétrica que para desencriptar, M=D{k}(E{k}(M))

Si usamos dos claves distintas entonces nos hallamos ante encriptación asimétrica.

M != D{k1}(E{k2}(M).

A la llave que usamos para encriptar (k1) la llamamos "clave pública" y la clave que se usa para el descifrado (k2) es la "clave privada".

Computacionalmente no debe ser posible transformar una clave pública en privada, veamos por qué:

1. Sea k-privada la clave privada
2. Sea k-publica la clave pública
3. Sea X ( ) una función computacionalmente factible que transforme cualquier clave pública en una privada.
4. Sea D(k-privada) {} la función de desencriptación
5. Por definición, M= D{k-privada}(E{k-publica}(M))
6. X(k-publica) = k-privada, entonces ---> D'{k-publica} ( ) = D{X(k-publica)}() = D{kpriv}() , esto quiere decir que obtenemos una función de desencriptación a partir de la clave publica. Lo que nos lleva a
7. M = D'{k-pub}(E{k-pub}(M)). Por lo cual hemos reducido a una encriptación simétrica !!!!!

El uso de las claves, esto nos permite hacer públicas la clave para cifrar, con la que los usuarios podrán cifrar, pero solo aquellos que posean la clave privada podrán descifrar el mensaje. Por lo cual podemos recibir mensajes secretos sin tener que compartir la clave.

Criptografía con Java

Desde la aparición de JDK 1.4 , java nos ofrece la posibilidad de trabajar con un framework para criptografía incluido en el núcleo de la JVM. El framework JCE (Java Cryptography Extension) nos ofrece las siguientes características:

- Soporte para cifrado simétrico (DES, RC2, y IDEA)
- Soporte para cifrado simétrico (RSA)
- Cifrado por bloques o por flujo
- Algoritmos MAC (Message Authentication Code)
- Funciones de resumen (funciones hash como MD5 y SHA1)
- Generación de claves
- Encriptación Basada en Password (PBE), transforma un password en una clave robusta mediante procesos aleatorios
- Acuerdo de claves: Es un protocolo, para transmisiones entre dos o más partes, mediante el cual se pueden establecer un acuerdo sobre las claves de cifrado sin intercambiar información secreta

JCE fue un framework opcional hasta la versión 1.4 y apareció por primera vez con la llegada de Java 2.

Las clases básicas de JCE son

- Cipher
- Las clases Cipher Stream (CipherInputStream, CipherOutputStream)
- KeyGenerator
- SecretKeyFactory
- KeyAgreement
- Mac

Distribuidas en los siguientes paquetes

- javax.crypto
- javax.crypto.interfaces
- javax.crypto.spec

Integridad de los mensajes

Veamos como podemos defender nuestros mensajes, mediante el uso de funciones de resumen (hash), para ello estudiaremos las diferentes especificaciones que nos ofrece JDK 1.4 en JCE.

Anteriormente ya vimos en que consistían y qué condiciones debían tener las funciones de resumen para ser útiles para su uso en programas de cifrado. Ahora veremos el uso de funciones estándar como MD5. JDK1.4 nos ofrece los siguientes algoritmos de resumen:

- MD2 y MD5 (que son algoritmos de 128-bits)
- SHA-1 (160-bit)
- SHA-256, SHA-383, y SHA-512, (256, 383, y 512 bits, respectivamente)

Los más comunes de usar son MD5 o SHA1

La clase MessageDigest maneja el resumen de los mensajes. Esta clase tiene un constructor protegido, por lo que debemos acceder a ella mediante un método llamado **getInstance(String algorithm)**, donde el parámetro representa el nombre del algoritmo con el que deseamos resumir.

```
import java.security.*;

public class ClaseCifrado01
{
        public static void main( String args[] ) throws Exception
        {
                if (args.length !=1)
                {
                        System.exit(1);
                }

                System.out.println( "El argumento de la linea de comandos es:" + args[0] );
                byte[] textoPlano = args[0].getBytes("UTF8");
                System.out.println( "El argumento de la linea de comandos pasado a byte[] es:" +
textoPlano );
                MessageDigest messageDigest = MessageDigest.getInstance("MD5");
                System.out.println(   "messageDigest.getProvider().getInfo()   da   el   siguiente
resultado:" );
```

```
                System.out.println( "\n" + messageDigest.getProvider().getInfo() );
                System.out.println( "Ahora con messageDigest MD5 actualizamos el texto plano:" );
                messageDigest.update( textoPlano);
                System.out.println( "\nresumen: " );
                System.out.println( new String( messageDigest.digest(), "UTF8") );
                System.out.println("\n ------------------------------------\n");
                MessageDigest md2= MessageDigest.getInstance("SHA");
                System.out.println( "Ahora con messageDigest SHA actualizamos el texto plano:" );
                md2.update( textoPlano);
                System.out.println( "\nresumen:" );
                System.out.println( new String( md2.digest(), "UTF8") );
        }
}
```

Veamos más a fondo este ejemplo. primero nos fijamos en que creamos un String con el parámetro UFT8, esto es porque algunos caracteres del resumen pueden dar problemas al salir por pantalla.

- messageDigest.getProvider().getInfo() nos da la información del Provider
- Con getInstance obtenemos dos instancias una del algoritmo MD5 y otra SHA, con las que podremos hace el resumen del mensaje.
- con update, introducimos el mensaje que queremos resumir
- digest() realizamos la acción de resumen.

Una posible salida del programa es la siguiente:

java ClaseCifrado01 "Hola Mundo"

SUN (DSA key/parameter generation; DSA signing; SHA-1, MD5 digests; SecureRandom; X.509 certificates; JKS keystore; PKIX CertPathValidator; PKIX CertPathBuilder; LDAP, Collection CertStores)
resumen:
?Z?5h?n?m*+Eh_?
---
resumen:
???,4?? ?w?aK$7??tb?

Vemos que las salidas son diferentes tanto en contenido como en longitud, 128 bits que codifican 16 caracteres para el MD5 y 160 bits que generan los 20 caracteres, si necesitásemos un resumen mayor, por ejemplo para evitar una colisión, usaríamos SHA 256 o un SHA de mayor tamaño.

Para generar claves adecuadas podemos usar el siguiente ejemplo:

```
import javax.crypto.*;

public class MessageAuthenticationCodeExample
{
        public static void main (String[] args) throws Exception
        {
                if (args.length !=1)
                {
                        System.exit(1);
```

```
                }
                byte[] textoPlano = args[0].getBytes("UTF8");
                System.out.println( "\n Generando la clave.................." );
                KeyGenerator keyGen = KeyGenerator.getInstance("HmacMD5");
                SecretKey MD5key = keyGen.generateKey();
                System.out.print( "Clave generada" );
                Mac mac = Mac.getInstance("HmacMD5");
                mac.init(MD5key);
                mac.update(textoPlano);
                System.out.println( "\nMAC: " );
                System.out.println( new String( mac.doFinal(), "UTF8") );
                System.out.println( "\n------------------------------\n" );
                System.out.println( "\n Generando la clave.................." );
                KeyGenerator keyGen2 = KeyGenerator.getInstance("HmacSHA1");
                SecretKey SHA1key = keyGen.generateKey();
                System.out.print( "clave generada" );
                Mac mac2 = Mac.getInstance("HmacSHA1");
                mac2.init(SHA1key );
                mac2.update(textoPlano);
                System.out.println( "\nMAC: " );
                System.out.println( new String( mac2.doFinal(), "UTF8") );
        }
}
```

Este ejemplo tarda más de un minuto en ejecutarse, esto se debe a que se generan números seudo-aleatorios a través de capturas del reloj interno La salida de consola esta vez nos da el siguiente resultado.

java MessageAuthenticationCodeExample "Probando el generador"

Generando la clave.....Clave generada
MAC:
*??? @Z¶ ? n?2

Generando la clave.....Clave generada
MAC:
??? ? ??H??o?????

Analicemos las clases Java que se utilizaron en el ejemplo anterior:

La clase KeyGenerator

La clase KeyGenerator es usada para generar claves secretas para usarse en algoritmos simétricos.
Como mucha de las clases de esta API, KeyGenerator se instancia a través de una factoría, a través del método getInstance

public static KeyGenerator getInstance(String algorithm) especifica el algoritmo de cifrado
public static KeyGenerator getInstance(String algorithm, String provider) especifica el proveedor y el algoritmo de cifrado.

Los algoritmos para los que generar claves está disponible son

- AES
- Blowfish

- DES
- DESede
- HmacMD5
- HmacSHA1

Todos los algoritmos comparten el concepto de aleatorio y de tamaño de la clave para especificar estos atributos tenemos los método init. los tres primeros casos son útiles para todos, y los dos siguientes se usan para configurar parámetros específicos del algún algoritmo.

- public void init(SecureRandom random);
- public void init(int keysize);
- public void init(int keysize, SecureRandom random);
- public void init(AlgorithmParameterSpec params);
- public void init(AlgorithmParameterSpec params, SecureRandom random);

para crear la clave debemos crear un objeto de la clase SecretKey (para almacenar la clave) y usar el método generateKey( ); Por ejemplo

```
SecretKey clave = mikeyGenerator.generateKey();
```

La clase Mac

La clase Mac, nos proporciona las funcionalidades para autentificar los mensajes.

En la clase Mac, también se crea a través de una factoría.

- public static Mac getInstance(String algorithm);
- public static Mac getInstance(String algorithm, String provider);

Iniciamos la clase Mac con los métodos

- public void init(Key key);
- public void init(Key key, AlgorithmParameterSpec params);

Confidencialidad del mensaje con clave privada.

Introducción

Ahora que hemos asegurado el mensaje de posibles ataques, es importante que el atacante no pueda ver su contenido. Veremos cómo cifrar con algoritmos de clave simétrica. JDK 1.4 soporta los siguientes algoritmos:

- DES. DES (Data Encryption Standard) desarrollado por IBM en los 70. Es un cifrado de bloque de 56-bit
- TripleDES. Consiste en aplicar el algoritmo DES tres veces (encriptar desencriptar encriptar) con dos claves dando un resultado de 112 bits.
- AES. Es el algoritmo que reemplazo a DES. Creado por Joan Daemen y Vincent Rijmen. Es un cifrado por bloque de 128-bit con claves de longitud 128, 192 o 256 bits.
- RC2, RC4 y RC5.
- Blowfish. Fue creado por Bruce Schneier y hace un cifrado por bloques con claves de longitud variable desde 32 a 448 bits (en múltiplos de 8), Diseñado para ser eficiente en computadoras.

- PBE. PBE (Password Based Encryption) puede ser usado con algoritmos de clave privada y funciones de resumen.

La clase Cipher

La clase Cipher, se usa para cifrar mediante algoritmos de clave simétrica. Como ya vimos es muy normal instanciar clases a través de una factoría. Este es otro ejemplo, aquí también usaremos getInstance( ):

public static Cipher getInstance(String transformation);
public static Cipher getInstance(String transformation, String provider);

Una transformación tiene la forma:

- ***"algoritmo / modo / relleno "***
- ***"algoritmo"***

Ejemplos:

Cipher c1 = Cipher.getInstance("*DES/ECB/PKCS5Padding*");

o

Cipher c1 = Cipher.getInstance("*DES*");

Los modos son la forma de trabajar del algoritmo. Existen varios modos:

- Ninguno
- ECB (Electronic Code Book)
- CBC (Cipher Block Chaining)
- CFB (Cipher Feedback Mode)
- OFB (Output Feedback Mode)
- PCBC (Propagating Cipher Block Chaining)

Como hemos visto, un algoritmo puede cifrar por bloques de una longitud determinada, cuando el mensaje es un múltiplo de dicha longitud no existe ningún problema. Pero cuando el mensaje no es un múltiplo, el último bloque es menor que la longitud necesaria para realizar el cifrado, entonces se realiza un relleno de ese bloque. A este relleno se le llama ***padding***, Existen varias formas de rellenar los bloques de menor tamaño.

- No rellenar
- PKCS5
- OAEP
- SSL3

Para más información sobre el padding, consulte la API.

Esquema básico de cifrado con clave privada.

Veamos un esquema para guiarnos.

1. **Primero crearemos la clave**
   a) **KeyGenerator.getInstance("DES")**
   b) **.init(56)**
   c) **.generateKey()**

2. **Creamos un objeto Cipher y lo configuramos con los parámetros deseados**
   a) **Cipher.getInstance("DES/ECB/PKCS5Padding"):**
   b) **.init(Cipher.ENCRYPT_MODE, clave):**

3. **Realizamos el cifrado**
   a) **.doFinal(textoPlano):**

4. **Configuramos otra vez el objeto Cipher**
   a) **.init(Cipher.DECRYPT_MODE, clave)**

5. **Y desciframos**
   a) **.doFinal(cipherText)**

Generar claves es muy sencillo, solo tenemos que indicar con que algoritmo deseamos que sea compatible nuestra clave el tamaño y llamar al método generateKey().

Ejemplo:

```
import java.security.*;
import javax.crypto.*;

public class PrivateExample
{
	public static void main (String[] args) throws Exception
	{
		if (args.length !=1)
		{
			System.exit(1);
		}
		System.out.println( "El argumento original de la linea de comandos es:" + args[0]
);
		byte[] textoPlano = args[0].getBytes("UTF8");
		System.out.println( "El argumento de la linea de comandos pasado a bytes y a bits
es:" );
		bytesToBits( textoPlano );
		System.out.println( "-----------------------------------------------------" );
		System.out.println( "Generando la llave..." );
		KeyGenerator keyGen = KeyGenerator.getInstance("DES");
		keyGen.init(56);
		Key clave = keyGen.generateKey();
		Cipher cifrar = Cipher.getInstance("DES/ECB/PKCS5Padding");
		cifrar.init(Cipher.ENCRYPT_MODE, clave);
		byte[] cipherText = cifrar.doFinal(textoPlano);
		System.out.println( "El argumento ENCRIPTADO es:" );
		System.out.println( new String(cipherText, "UTF8") );
		System.out.println( "El argumento ENCRIPTADO pasado a bytes y a bits es:" );
		bytesToBits( cipherText );
		System.out.println( "-----------------------------------------------------" );
		cifrar.init(Cipher.DECRYPT_MODE, clave);
		byte[] newPlainText = cifrar.doFinal(cipherText);
		System.out.println( "El argumento DESENCRIPTADO es:" );
		System.out.println( new String(newPlainText, "UTF8") );
		System.out.println( "El argumento DESENCRIPTADO pasado a bytes y a bits es:" );
		bytesToBits( newPlainText );
	}

	public static void bytesToBits( byte[] texto )
```

```
        {
                StringBuilder stringToBits = new StringBuilder();
                for( int i=0; i < texto.length; i++ )
                {
                        StringBuilder binary = new StringBuilder();
                        byte b = texto[i];
                        int val = b;
                        for( int j = 0; j < 8; j++ )
                        {
                                binary.append( (val & 128) == 0 ? 0 : 1 );
                                val <<= 1;
                        }
                        System.out.println( (char)b + " \t " + b + " \t " + binary );
                        stringToBits.append( binary );
                }
                System.out.println( "El mensaje completo en bits es:" + stringToBits );
        }
}
```

java PrivateExample "Hola Mundo"
El argumento original de la linea de comandos es:Hola Mundo
El argumento de la linea de comandos pasado a bytes y a bits es:
H 72 01001000
o 111 01101111
l 108 01101100
a 97 01100001
32 00100000
M 77 01001101
u 117 01110101
n 110 01101110
d 100 01100100
o 111 01101111
El mensaje completo en bits es:01001000011011110110110001100001001000000010011010111010101101110011001000110111

Generando la llave...
El argumento ENCRIPTADO es:
??]#?▲????¶?♠?
El argumento ENCRIPTADO pasado a bytes y a bits es:
? -69 10111011
? -11 11110101
] 93 01011101
 # 35 00100011
? -100 10011100
▲ 30 00011110
? -78 10110010
? -96 10100000
? -17 11101111
? -88 10101000

? 63 00111111
¶ 20 00010100
? -32 11100000
♠ 6 00000110
? -7 11111001
? -85 10101011
El mensaje completo en bits es:1011101111110101010111010010001110011100000111010110001010100000111011111101010000011111000101001110000000000110111110011010101011

El argumento DESENCRIPTADO es:
Hola Mundo
El argumento DESENCRIPTADO pasado a bytes y a bits es:
H 72 01001000
o 111 01101111
l 108 01101100
a 97 01100001
32 00100000
M 77 01001101
u 117 01110101
n 110 01101110
d 100 01100100
o 111 01101111
El mensaje completo en bits es:0100100001101111011011000110000100100000010011010111010101110011100101000110111