UNIDAD 4

TÉCNICAS DE PROGRAMACIÓN SEGURA

Aunque la criptografía se define en el Diccionario de la Real Academia de la Lengua Española como "el arte de escribir con clave secreta o de un modo enigmático". Así, la criptografía puede verse más como la ciencia que trata de conservar los secretos o hasta el arte de enviar mensajes en clave secreta aplicándose a todo tipo de información, tanto escrita como digital

Se denomina encriptar a la acción de proteger la información mediante su modificación utilizando una clave.

La criptografía es una disciplina con multitud de aplicaciones, sobre todo en el área de Internet. Entre otras se pueden destacar:

- Identificación. Identificar a un individuo o validar el acceso a un servidor con más garantías que los sistemas de usuario y clave tradicionales.
- Certificación. Esquema mediante el cual agentes fiables validan la identidad de agentes desconocidos
- Seguridad de las comunicaciones. Permite establecer canales seguros para aplicaciones que operan sobre redes que no son seguras (Internet).

Un sistema muy conocido fue el empleado por el mismísimo Julio César para enviar mensajes secretos a sus tropas. El sistema, denominado "método de Cesar", consiste en sumar un número determinado al número de orden de cada letra sustituyendo esa letra en el mensaje por la letra resultado que se obtiene de la operación. Si dicho número fuera un 3, la palabra “Hola” quedaría como “KROD”

Todos los algoritmos criptográficos son vulnerables a los ataques de fuerza bruta (probar sistemáticamente con todas y cada una de las posibles claves de encriptación).

CARACTERÍSTICAS DE LOS SERVICIOS DE SEGURIDAD

-Confidencialidad:

Se trata de asegurar que la comunicación solo pueda ser vista por los usuarios autorizados, evitando que ningún otro pueda leer el mensaje. Esto se requiere siempre que la información tenga que ser privada y la autenticación de los usuarios que participan sea obligatoria

-Integridad:

Se trata de asegurar que el mensaje no haya sido modificado de ningún modo por terceras personas durante su transmisión. La información recibida debe ser igual a la que fue emitida.

-Autenticación:

Se trata de asegurar el origen, autoría y propiedad de la información.

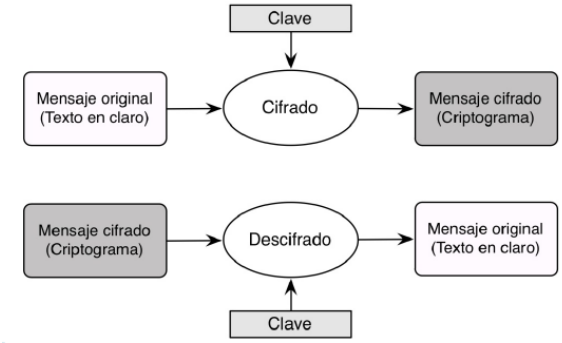
-No repudio:

Se trata de evitar que la persona que envía el mensaje o realiza una acción niegue haberlo hecho ante terceros.

ESTRUCTURA DE UN SISTEMA SECRETO

Un sistema secreto actual se encuentra definido por dos funciones:
- Función de cifrado.
- Función de descifrado.

La clave es el parámetro que especifica una transformación concreta dentro de todas las posibles sustituciones que se podrían realizar con la función de cifrado.



Todos los cifrados se deben construir en base a:
- Difusión: distribuir las propiedades de todos los elementos
- Confusión: dificultar la relación entre clave y texto cifrado

Características deseables:
- El análisis estadístico del texto cifrado para descifrarlo debe suponer tal cantidad de trabajo que no sea rentable hacerlo por el envejecimiento de la propia información contenida en él.
- Las claves deben ser de fácil construcción y sencillas.
- Los sistemas secretos, una vez conocida la clave, deben ser simples
- Los errores de transmisión no deben originar ambigüedades.
- La longitud del texto cifrado no debe ser mayor que la del texto en claro.

Algoritmos de clave secreta o de criptografía simétrica:

El emisor y el receptor comparten el conocimiento de una clave que no debe ser revelada a ningún otro. La clave se utiliza tanto para cifrar como para descifrar el mensaje. En estos algoritmos lo esencial es que la clave K, que sirve tanto para cifrar como para descifrar sea compartida sólo entre los participantes del sistema, Por eso se dice que la clave es privada o secreta y que los algoritmos son simétricos, pues la misma clave cifra y descifra. Los algoritmos de cifrado simétrico más populares son es el DES y el AES.

Algoritmos de clave pública o de criptografía asimétrica:

El emisor de un mensaje (Participante B) emplea una clave pública, difundida previamente por el receptor (Participante A), para encriptar el mensaje. El receptor emplea la clave privada correspondiente para desencriptar el mensaje, sólo el receptor puede desencriptar el mensaje gracias a su clave privada. Estos algoritmos se basan en que cada participante genera una pareja de claves relacionadas entre sí. Una es la clave pública que todo el mundo puede conocer y otra la clave privada, conocida solo por el receptor. El algoritmo asimétrico más popular es el RSA, usado para la firma digital actualmente.

CRIPTOGRAFÍA SIMÉTRICA O DE CLAVE SECRETA	
Puntos fuertes	**Puntos débiles**
Cifran más rápido que los algoritmos de clave pública. Sirven habitualmente como base para los sistemas criptográficos basados en hardware.	Requieren un sistema de distribución de claves muy seguro (si se conoce la clave se pueden conocer todos los mensajes cifrados con ella). En el momento en que la clave cae en manos no autorizadas, todo el sistema deja de funcionar. Esto obliga a llevar una administración compleja. Si se asume que es necesaria una clave por cada pareja de usuarios de una red, el número total de claves crece rápidamente con el número de usuarios.
CRIPTOGRAFÍA ASIMÉTRICA O DE CLAVE PÚBLICA	
Puntos fuertes	**Puntos débiles**
Permiten conseguir autenticación y no repudio para muchos protocolos criptográficos. Suelen emplearse en colaboración con cualquiera de los otros métodos criptográficos. Permiten tener una administración sencilla de claves al no necesitar que haya intercambio de claves seguro.	Son algoritmos más lentos que los de clave secreta, con lo que no suelen utilizarse para cifrar gran cantidad de datos. Sus implementaciones son comúnmente hechas en sistemas software. Para una gran red de usuarios y/o máquinas se requiere un sistema de certificación de la autenticidad de las claves públicas.

CRIPTOGRAFÍA CON JAVA

JCA (Java Cryptography Architecture), es una parte importante de la plataforma Java y contiene un conjunto de APIs para poder trabajar con firmas digitales, resúmenes de mensajes (hash), validación de certificados, cifrado, generación y administración de claves, etc. Estas APIs permiten a los desarrolladores integrar fácilmente la seguridad en el código de su aplicación.

CLASES JCA / JCE	FUNCIÓN
java.security.MessageDigest	Calcular el resumen del mensaje (hash)
java.security.Signature	Firmar datos y verificar firmas digitales
java.security.KeyPairGenerator	Generar pares de claves (pública y privada) para un algoritmo.
java.security.KeyFactory	Convertir claves de formato criptográfico a especificaciones de claves y viceversa
java.security.KeyStore	Crear y gestionar un almacén de claves (keystore)
java.security.cert.CertificateFactory	Crear certificados de clave pública y listas de revocación de certificados (CRLs)
java.security.cert.CertStore	Recuperar Certificados y CRLs de un repositorio
java.security.cert.CertPathBuilder	Se utiliza para crear cadenas de certificados (también conocidas como rutas de certificación)
java.security.cert.CertPathValidator	Validar cadenas de certificados
java.security.AlgorithmParameters	Almacenar los parámetros de un algoritmo, incluyendo codificación y descodificación
java.security.AlgorithmParameterGenerator	Generar un conjunto de AlgorithmParameters para un algoritmo específico
java.security.SecureRandom	Generar números aleatorios o pseudo aleatrorios
java.crypto.Cipher	Se utiliza para cifrar y descifrar datos
java.crypto.KeyAgreement	Utilizado por dos o más partes para acordar y establecer una clave específica para usar en una operación criptográfica particular
java.crypto.KeyGenerator	Generar nuevas claves secretas para usar con un algoritmo específico
java.crypto.Mac	Proporciona un algoritmo de autentificación de mensajes
java.crypto.SecretKeyFactory	Se utiliza para convertir claves criptográficas opacas existentes de tipo SecretKey en especificaciones clave, y viceversa. Los SecretKeyFactorys son KeyFactorys especializados que crean claves secretas (simétricas) solamente

MESSAGE DIGESTS (HASH)

Un message digests o resumen de mensaje (también se le conoce como función hash) es una marca digital de un bloque de datos. Existe un gran número de algoritmos diseñados para procesar estos message digests, los dos más conocidos son SHA-1 y MD5. La clase MessageDigest permite a las aplicaciones implementar algoritmos de resumen de mensajes criptográficamente seguros como SHA-256 o SHA-512.

Un digest o hash tiene dos propiedades:

- Es computacionalmente inviable encontrar dos mensajes que tengan el mismo valor.
- El hash no debe revelar nada sobre la entrada que se utilizó para generarlo.

Los resúmenes de mensajes se usan para generar identificadores de datos únicos y confiables.

MÉTODOS	MISIÓN
public static MessageDigest getInstance(String algoritmo) **public static MessageDigest getInstance(String algoritmo, String proveedor)**	Devuelve un objeto **MessageDigest** que implementa el algoritmo de resumen especificado. En el primer caso, los proveedores de seguridad se buscan según el orden establecido en el fichero **java.security**. En el segundo caso se busca el proveedor dado. Nombres válidos para el proveedor de seguridad predeterminado de *Sun* son SHA, SHA-1 y MD5. Puede lanzar la excepción ***NoSuchAlgorithmException*** si no hay proveedor que implemente el algoritmo dado. Si el nombre de proveedor no se encuentra se produce ***NoSuchProviderException***.
void update(byte input)	Realiza el resumen del byte especificado.
void update(byte[] input)	Realiza el resumen del array de bytes especificado.
byte[] digest()	Completa el cálculo del valor hash, devuelve el resumen obtenido.
byte [] digest (byte [] entrada)	Realiza una actualización final sobre el resumen utilizando el array de bytes indicado en el argumento, y luego completa el cálculo de resumen.
void reset()	Reinicializa el objeto resumen para un nuevo uso.
int getDigestLength()	Devuelve la longitud del resumen en bytes, o 0 si la operación no está soportada por el proveedor
String getAlgorithm()	Devuelve un String que identifica el algoritmo
Provider getProvider()	Devuelve el proveedor del objeto
static boolean isEqual(byte[] digesta, byte[] digestb)	Comprueba si dos mensajes resumen son iguales. Devuelve *true* si son iguales y *false* en caso contrario

```java
import java.security.*;
public class Ejemplo4 {
    public static void main(String[] args) {
        MessageDigest md;
        try {
            md = MessageDigest.getInstance("SHA-256");
            String texto = "Esto es un texto plano.";
            byte dataBytes[] = texto.getBytes();
            md.update(dataBytes);// SE INTRODUCE TEXTO A RESUMIR
            byte resumen[] = md.digest(); // SE CALCULA EL RESUMEN
            System.out.println("Mensaje original: " + texto);
            System.out.println("Número de bytes: " + md.getDigestLength());
            System.out.println("Algoritmo: " + md.getAlgorithm());

            System.out.println("Mensaje resumen: " + new String(resumen));
            System.out.println("Mensaje en Hexadecimal: "
                    + Hexadecimal(resumen));
            Provider proveedor=md.getProvider();
            System.out.println("Proveedor: " + proveedor.toString());
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }// main
    // CONVIERTE UN ARRAY DE BYTES A HEXADECIMAL
    static String Hexadecimal(byte[] resumen) {
        String hex = "";
        for (int i = 0; i < resumen.length; i++) {
            String h = Integer.toHexString(resumen[i] & 0xFF);
            if (h.length() == 1)
                hex += "0";
            hex += h;
        }
        return hex.toUpperCase();
    }// Hexadecimal
}// ..
```

FIRMAS DIGITALES

Las firmas digitales pueden autenticar un mensaje y asegurar que el mensaje no ha sido alterado y procede del emisor correcto. Para crear una firma digital se necesita una clave privada y la clave pública correspondiente con el fin de verificar la autenticidad de la firma. La clase KeyPairGenerator nos permite generar el par de claves. El método de firma digital más extendido es el RSA.

MÉTODOS	MISIÓN
static KeyPairGenerator getInstance(String algoritmo) **static KeyPairGenerator getInstance(String algoritmo, String provider)**	Devuelve un objeto **KeyPairGenerator** que genera un par de claves púbica/privada para el algoritmo especificado. Puede lanzar la excepción ***NoSuchAlgorithmException***. En el segundo método se especifica el proveedor. Si el nombre de proveedor no se encuentra se produce ***NoSuchProviderException***.
void initialize(int keysize, SecureRandom random)	Inicializa el generador de par de claves para un determinado tamaño de clave y un generador de números aleatorios.
KeyPair generateKeyPair() **KeyPair genKeyPair()**	Genera el par de claves.

La clase KeyPair es una clase soporte para generar las claves pública y privada. Dispone de dos métodos:

MÉTODOS	MISIÓN
PrivateKey getPrivate()	Devuelve una referencia a la clave privada del par de claves.
PublicKey getPublic()	Devuelve una referencia a la clave pública del par de claves.

Una vez creadas las claves se pueden firmar los datos. Para firmar los datos se usa la clase Signature**.**

MÉTODOS	MISIÓN
static Signature getInstance(String algoritmo) **static Signature getInstance(String algoritmo, String provider)**	Devuelve un objeto **Signature** que implementa el algoritmo especificado. Puede lanzar la excepción ***NoSuchAlgorithmException*** En el segundo método se especifica el proveedor. Si el nombre de proveedor no se encuentra se produce ***NoSuchProviderException***
void initSign(PrivateKey privateKey, SecureRandom random)	Inicializa el objeto para la firma. Se especifica la clave privada de la identidad cuya firma se va a generar y la fuente de aleatoriedad. Si la clave no es válida puede lanzar la excepción ***InvalidKeyException***
void update(byte b)	Actualiza los datos a firmar o verificar usando el byte especificado
void update(byte[] data)	Actualiza los datos a firmar o verificar usando el array de bytes especificado
void update(ByteBuffer data)	Actualiza los datos a firmar o verificar usando el *ByteBuffer* especificado
byte[] sign()	Devuelve en un array de bytes la firma de los datos.
void initVerify(PublicKey publicKey)	Inicializa el objeto para la verificación de la firma. Necesita como parámetro la clave pública. Si la clave no es válida puede lanzar la excepción ***InvalidKeyException***.
boolean verify(byte[] signature)	Verifica la firma que se pasa como parámetro.

Ejemplo7.java