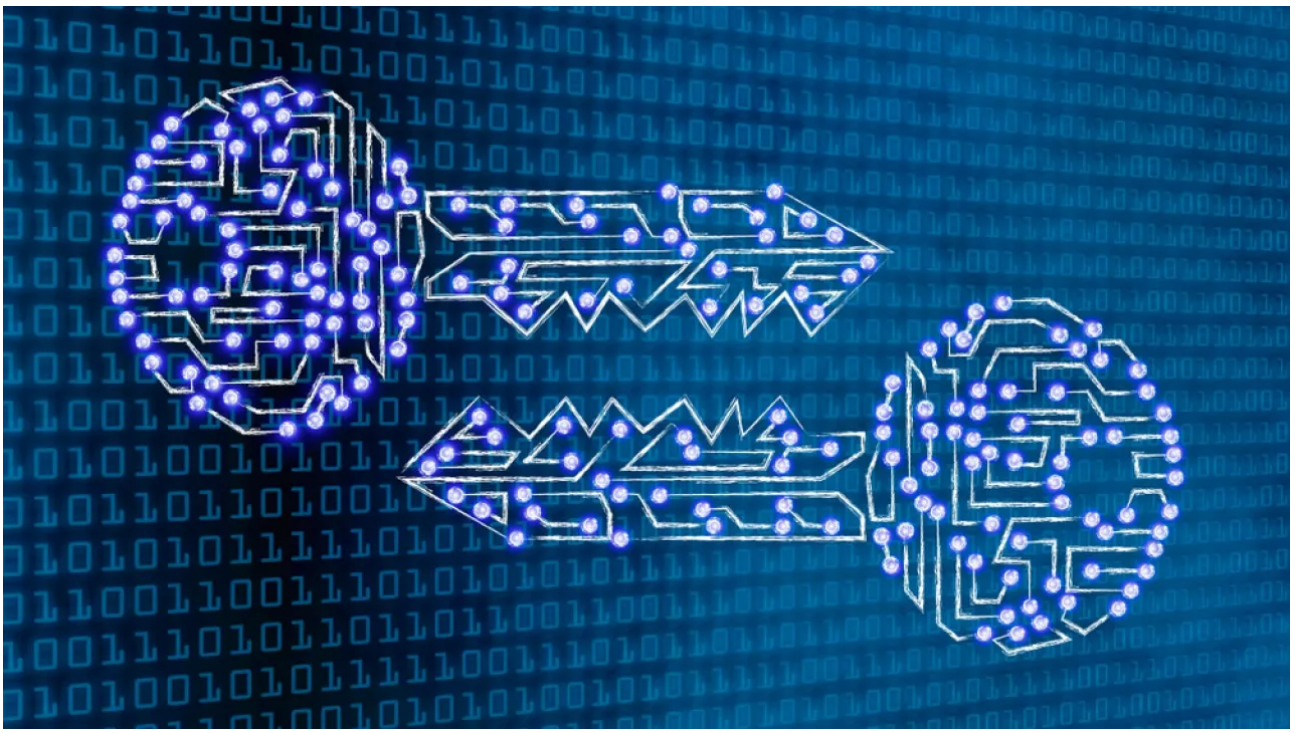
TAREA: CRIPTOGRAFÍA Y COMUNICACIONES SEGURAS



Alumna: Isabel María González Rodríguez

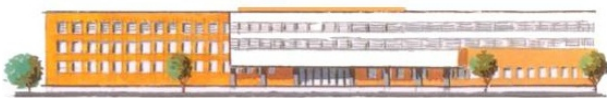
Profesor: José Luis Rodríguez Rodríguez

Módulo: Programación de Servicios y Procesos

Curso: 2022 - 2023

Fecha: 15/02/2023

IES Ramón del Valle-Inclán



ÍNDICE

ACTIVIDAD 1. CONCEPTOS SOBRE CRIPTOGRAFÍA.

1. CRIPTOGRAFÍA.

El término **criptografía** es un derivado de la palabra griega κρυπτός, que significa “oculto”. El objetivo de la criptografía es ocultar el significado de un mensaje mediante el cifrado o codificación del mensaje.

El proceso general de cifrado y descifrado de mensajes es el siguiente:

- Si a un **texto legible** se le aplica un algoritmo de cifrado, que en general depende de una **clave**, esto arroja como resultado un **texto cifrado** que es el que se envía o guarda. A este proceso se le llama **cifrado** o **encriptado**.
- Si a este **texto cifrado** se le aplica el mismo algoritmo, dependiente de la misma clave o de otra clave, se obtiene el **texto legible** original. A este segundo proceso se le llama **descifrado** o **desencriptación**.

Existen tres clases de algoritmos criptográficos:

- **Funciones de una sola vía** (o funciones *hash*).

  Estas funciones permiten mantener la integridad de los datos tanto en almacenamiento como en el tráfico de redes. Se utilizan en mecanismos de **firma digital**. Las funciones de una sola vía tienen un amplio abanico de usos en la seguridad informática. Prácticamente cualquier protocolo las usa para procesar claves, encadenar una secuencia de eventos, o incluso autenticar eventos y son esenciales en la **autenticación por firmas digitales**.

  Los dos algoritmos de una sola vía más utilizados son el **MD5** y **SHA-1**.

- **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, por eso se dice que ésta es privada o secreta y que los algoritmos son simétricos.

  El algoritmo de cifrado simétrico más popular es el **DES**, utiliza claves de 56 bits y un cifrado de bloques de 64 bits. Una variante de este es el **Triple DES** (o 3DES), la clave es de 128 bits. Otro algoritmo muy utilizado es el **AES**, que tiene un tamaño de clave variable siendo el estándar 256 bits.

- **Algoritmos de clave pública o de criptografía asimétrica**.

  El emisor de un mensaje emplea una clave pública, difundida previamente por el receptor, 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** y otra la **clave privada**.

  El algoritmo asimétrico más popular es el **RSA**. Su uso es prácticamente universal como método de autenticación y **firma digital**, y es componente de protocolos y sistemas como IPSec, SSL, PGP, etc.

1.1. FUNCIONAMIENTO DE LA FIRMA DIGITAL.

Una **firma digital** está compuesta por una serie de datos asociados a un mensaje, estos datos nos permiten asegurar la identidad del firmante y la integridad del mensaje. El método más extendido es el **RSA**.

- El emisor genera un *hash* del mensaje mediante una función acordada.
- Este *hash* es cifrado con su clave privada. El resultado es lo que se conoce como “firma digital” que se envía adjunta al mensaje.
- El emisor envía el mensaje y su firma digital al receptor, es decir, el mensaje firmado.

El receptor realiza las siguientes operaciones:

- Separa el mensaje de la firma.
- Genera el *hash* del mensaje recibido usando la misma función que el emisor.
- Descifra la firma mediante la clave pública del emisor obteniendo el *hash* original.
- Si los dos *hashes* coinciden se puede afirmar que el mensaje ha sido enviado por el propietario de la clave pública utilizada y que no fue modificado en el transcurso de la comunicación.

Todo sistema criptográfico de firma digital descansa sobre un pilar fundamental: la autenticidad de la clave pública de cada participante.

1.2. PUNTOS FUERTES Y PUNTOS DÉBILES.

1.2.1. PUNTOS FUERTES.

- CRIPTOGRAFÍA SIMÉTRICA
    - Cifran más rápido que los algoritmos de clave pública.
    - Sirven habitualmente como base para los sistemas criptográficos basados en hardware.
- CRIPTOGRAFÍA ASIMÉTRICA
    - 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.

1.2.2. PUNTOS DÉBILES.

- CRIPTOGRAFÍA SIMÉTRICA
    - Requieren un sistema de distribución de claves muy seguro.
    - 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
    - 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 hardware.
    - Para una gran red de usuarios y/o máquinas se requiere un sistema de certificación de la autenticidad de las claves públicas.

2. CERTIFICADOS DIGITALES.

Un **certificado digital** es un documento que certifica que una entidad determinada, como puede ser un usuario, una máquina, un dispositivo en red o un proceso, tiene una clave pública determinada. El certificado tiene que ser capaz de enlazar una clave pública junto al nombre del titular, ya sea una empresa, una persona u otra entidad.

Para certificar estos documentos se acude a las **Autoridades de Certificación (AC)**, que son entidades que se encargan de emitir y gestionar tales certificados y que tienen una propiedad muy importante: que se puede confiar en ellas. La forma en la que la AC hace válido el certificado es firmándolo digitalmente. Al aplicar el algoritmo de firma digital al documento se obtiene un texto, una secuencia de datos que permiten asegurar que el titular de ese certificado ha “firmado electrónicamente” el texto y que éste no ha sido modificado.

Los certificados digitales se pueden solicitar a través de la aplicación web de la AC. Por ejemplo, una persona física para solicitar un certificado a la Fábrica de Nacional de Moneda y Timbre (FNMT), accede a la web www.ceres.fnmt.es y sigue una serie de pasos.

ACTIVIDAD 2. CRIPTOGRAFÍA CON JAVA.

EJERCICIO 1. OBTENER PROVEEDORES DE ALGORITMOS CRIPTOGRÁFICOS.

JCA es una parte importante de la plataforma Java y contiene una arquitectura de “proveedor” y un conjunto de APIs para firmas digitales, resúmenes de mensajes (hash), validación de certificados, cifrado, generación y administración de claves, y generación segura de números aleatorios. Estas APIs permiten a los desarrolladores integrar fácilmente la seguridad en el código de su aplicación.

El API **JCA**, incluida dentro del paquete JDK, incluye dos componentes de software:

- El marco que define y soporta los servicios criptográficos para que los proveedores faciliten implementaciones. Este marco incluye paquetes como java.security, java.security.cert, java.security.spec, java.security.interfaces, javax.crypto, javax.crypto.spec y javax.crypto.interfaces.
- El proveedor es el encargado de proporcionar la implementación de uno o varios algoritmos al programador. Los proveedores de seguridad se definen en el fichero java.security localizado en la carpeta *java.home**conf**security*\; forman una lista de entradas con un número que indican el orden de búsqueda cuando en los programas no se especifica un proveedor.

JCA está estructurado en torno a algunas clases e interfaces centrales de propósito general. La funcionalidad real detrás de estas interfaces es proporcionada por los proveedores. Por lo tanto, podemos usar la clase **Clipher** para cifrar y descifrar algunos datos, pero la implementación de cifrado concreto depende del proveedor concreto utilizado.

JCA define el concepto de proveedor mediante la clase **Provider** del paquete java.security. Se trata de una clase abstracta que debe ser redefinida por clases proveedor especificas. Tiene métodos para acceder al nombre del proveedor, el número de versión y otras informaciones sobre las implementaciones de los algoritmos, para la generación, conversión y gestión de claves y la generación de firmas y resúmenes.

EJERCICIO 2. GENERAR RESÚMENES DE MENSAJES.

Un *message digests* o resumen de mensaje (hash) es una marca digital de un bloque de datos. Existe un gran número de algoritmos diseñados para procesar estos *message digests*, siendo los dos más conocidos 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 resumen de mensaje criptográficamente seguro toma una entrada de tamaño arbitrario y genera un resultado de tamaño fijo, llamado digest o hash.

Un digest o hash tiene dos propiedades:

- Es computacionalmente inviable encontrar dos mensajes que tengan el mismo valor.
- El digest 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. A veces se llaman “checksums” o “huellas dactilares” de los datos. Los cambios en solo un bit del mensaje deberían producir un valor de resumen diferente.

EJERCICIO 3. GENERANDO Y VERIFICANDO FIRMAS DIGITALES.

El resumen de un mensaje no nos da un alto nivel de seguridad, ya que se podría cambiar el texto del fichero y el resumen y no podríamos estar seguros de que sea el original o está modificado.

Las firmas digitales pueden autenticar un mensaje y asegurar que éste 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.

En algunos casos, el par de claves están disponibles en ficheros. En ese caso, el programa puede importar y utilizar la clave privada para firmar. En otros casos, el programa necesita generar el par de claves utilizando la clase **KeyPairGenerator**, cuya clase soporte es **KeyPair**.

PrivateKey y **PublicKey** son interfaces que agrupan todas las interfaces de clave privada y pública respectivamente.

Pasos a seguir para generar el par de claves:

1. Obtener un objeto generador.
2. Inicializar el generador de par de claves. Se necesita el método *initialize()* y le pasaremos dos argumentos (el tamaño de la clave y un generador de números aleatorio).
3. Generar el par de claves y almacenarlas en los objetos **PrivateKey** y **PublicKey**.

Una vez creadas las claves, se pueden firmar los datos. Para ello, se usa la clase **Signature**, que proporciona la funcionalidad de un algoritmo de firma digital criptográfica como SHA-256 con DSA o SHA-512 con RSA. Un algoritmo de firma criptográfica seguro toma una entrada de tamaño arbitrario y una clave privada y genera una *firma*, con las siguientes propiedades:

- Solo el propietario del par de claves puede crear una firma.
- Dada la clave pública correspondiente a la clave privada utilizada para generar la firma, debería ser posible verificar la autenticidad e integridad de la entrada.

Un objeto **Signature** se inicializa para firmar con una clave privada y se le asignan los datos que se deben firmar. Los bytes de firma resultantes generalmente se guardan con los datos firmados. Cuando se necesita verificación, se crea e inicializa otro objeto del mismo tipo para su verificación y se le da la clave pública correspondiente. Los bytes de datos y de firma se envían al objeto de firma, y si coinciden los datos y la firma, el objeto **Signature** informa de éxito.

Existen tres fases en el uso de un objeto **Signature**:

1. Inicialización (ya sea con clave pública, *initVerify()*, o privada, *initSign()*).
2. Actualización (*update()*).
3. Firma (*sign()* o *verify()*).

Al especificar el nombre del algoritmo de firma, también se debe incluir el nombre del algoritmo de resumen de mensajes utilizado por el algoritmo de firma. *SHA1withDSA* es una forma de especificar el algoritmo de firma DSA, usando el algoritmo de resumen SHA-1.

ALMACENAR LAS CLAVES PÚBLICA Y PRIVADA EN FICHEROS

Para **almacenar la clave privada en disco** es necesario codificarla en formato PKCS8 usando la clase **PKCS8EncodedKeySpec**. Para **almacenar la clave pública en disco** es necesario codificarla en formato X.509 usando la clase **X509EncodedKeySpec**. Ambas clases están definidas en el paquete java.security.spec.

RECUPERAR LAS CLAVES PÚBLICA Y PRIVADA DE FICHEROS

Para recuperar las claves de los ficheros necesitamos la clase **KeyFactory** que proporciona métodos para convertir claves de forma criptográfico (PKCS8, X.509) a especificaciones de claves y viceversa.

EJERCICIO 4. CIFRAR MENSAJES.

1. ENCRIPTACIÓN CON CLAVE SECRETA.

Cuando la información que se envía es encriptada, su contenido no es visible. Sólo se puede desencriptar con una clave coincidente. La autenticación es suficiente para firmar la información, no hay necesidad de ocultarla. La encriptación es necesaria cuando las aplicaciones transfieren información confidencial como números de tarjetas de crédito y otros datos personales.

Java proporciona la clase **Cipher** para encriptar y desencriptar información, esta clase forma el núcleo de la extensión criptográfica de Java (JCE). El cifrado de un texto legible consiste en transformarlo con ayuda de una clave en un texto ilegible; el descifrado es el proceso inverso, se toman los datos ilegibles y la clave y se produce texto legible.

Para crear un objeto **Cipher** se llama al método *getInstance()* pasando como argumento el algoritmo y opcionalmente, se puede especificar el nombre de un proveedor, como pueden ser, por ejemplo:

- AES/CBC/PKCS5Padding
- AES/ECB/PKCS5Padding
- DES/CBC/PKCS5Padding
- DES/ECB/PKCS5Padding
- DESede/CBD/PKCS5Padding
- DESede/ECB/PKCS5Padding
- RSA/ECB/PKCS5Padding
- RSA/ECB/OAEPWithSHA-1AndMGF1Padding
- RSA/ECB/OAEPWithSHA-256WithMGF1Padding

Para proporcionar una clave al método *init()* del objeto Cipher usamos la clase **KeyGenerator** que proporciona funcionalidad para generar claves secretas para usarse en algoritmos simétricos.

PASOS PARA ENCRIPTAR Y DESENCRIPTAR CON CLAVE SECRETA

1. Creamos la clave secreta usando el algoritmo AES o DES y definimos un tamaño de clave de 128 bits.
2. Creamos un objeto Cipher con el algoritmo AES/ECB/PKCS5Padding, lo inicializamos en modo encriptación con la clave creada anteriormente.
3. Realizamos el cifrado de la información con el método *doFinal()*.
4. Configuramos el objeto Cipher en modo desencriptación con la clave anterior para desencriptar el texto, usamos el método *doFinal()*.

Muchos modos de algoritmos requieren un vector de inicialización que se especifica cuando se inicializa el objeto Cipher en modo desencriptación. En estos casos, se debe pasar al método *init()* el vector de inicialización. La clase **IvParameterSpec** se usa para hacer esto en el cifrado DES.

2. ENCRIPTAR Y DESENCRIPTAR CON CLAVE PÚBLICA.

En el cifrado asimétrico o de clave pública se resuelve el problema del simétrico ya que la clave para encriptar se puede compartir sin problemas y la clave para desencriptar sólo la tiene que poseer el receptor del mensaje.

El esquema básico de cifrado y descifrado con clave pública es el siguiente:

1. Se crea el par de claves pública y privada.
2. Se crea un objeto Cipher con algoritmo RSA y se inicializa en modo encriptación con la clave pública.
3. Realizamos el cifrado de la información con el método *doFinal()* como antes.
4. Configuramos el objeto Cipher en modo desencriptación con la clave privada para desencriptar el texto, usamos el método *doFinal()*.

El cifrado mediante clave pública es más lento que el cifrado mediante clave privada. No es práctico utilizar este cifrado para encriptar grandes cantidades de información. Este problema se puede solucionar combinando cifrado de clave pública con cifrado simétrico.

El concepto de **clave de sesión** es un término medio entre el cifrado simétrico y el asimétrico, que permite combinar las dos técnicas. Consiste en generar una clave de sesión y cifrarla usando la clave pública del receptor. El receptor descifra la clave de sesión usando su clave privada. El emisor y el receptor comparten una clave que sólo ellos conocen y pueden cifrar sus comunicaciones usando la misma clave de sesión.

3. ENCRIPTAR Y DESENCRIPTAR FLUJOS DE DATOS.

JCA proporciona un conjunto de clases que encriptan o desencripta automáticamente flujos de datos, estas son **CipherOutputStream** y **CipherInputStream**.

CipherOutputStream es un FilterOutputStream que encripta o descifra los datos que pasan a través de él. Está compuesto por un OutputStream y un objeto Cipher. Los métodos de escritura de esta clase primero procesan los datos con el objeto Cipher incrustado antes de escribirlos en el OutputStream subyacente. Es necesario inicializar el objeto Cipher antes de ser utilizado por un CipherOutputStream.

CipherInputStream es un FilterInputStream que encripta o descifra los datos que lo atraviesan. Está compuesto por un InputStream y un objeto Cipher. Los métodos de lectura de éste devuelven datos que se leen del InputStream subyacente que además han sido procesados por el objeto Cipher incrustado. Este objeto debe estar inicializado antes de ser utilizado por un CipherInputStream.

Estas clases manipulan de forma transparente las llamadas *update()* y *doFinal()*.

ACTIVIDAD 3. COMUNICACIONES EN RED SEGURAS.