

5

Utilización de técnicas de programación segura

CONTENIDOS

- ✓ Aprender los conceptos básicos relacionados con la seguridad de la información y la historia de la criptografía.
- ✓ Familiarizarse con los modelos criptográficos más importantes, como el modelo de clave privada y el modelo de clave pública.
- ✓ Aprender a programar usando los mecanismos de cifrado de la información más importantes.
- ✓ Conocer los algoritmos de cifrado más usados en la actualidad.
- ✓ Aprender los fundamentos de la programación de aplicaciones distribuidas que utilizan comunicaciones seguras.

5.1

CONCEPTOS BÁSICOS

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”, esta definición no encaja en los tiempos actuales. 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, la cual se puede almacenar en un ordenador o enviar a través de la red.

Se denomina **encriptar** a la acción de proteger la información mediante su modificación utilizando una clave. En informática también se usan los términos codificar/descodificar y cifrar/descifrar como sinónimos de encriptar/desencriptar.



EJEMPLO 5.1

Supongamos que Pedro desea enviar un mensaje a Ana. Sin embargo, Pedro quiere enviarlo de forma privada, ya que es una carta amorosa, a través de Internet y no se fía, ya que piensa que el medio de comunicación no es seguro. En concreto, Pedro quiere evitar que el mensaje pueda ser leído por terceras personas que incluso podrían llegar a modificarlo. El objetivo de la criptografía consiste en proporcionar métodos para prevenir tales ataques.

Pedro tiene una opción sencilla, codificará el mensaje y obtendrá un mensaje cifrado el cual puede enviar tranquilamente a Ana (este mensaje cifrado es ilegible y difícilmente descifrable sin tener la clave correcta). Ana empleará un método de descodificación para conseguir obtener el mensaje original a partir del mensaje cifrado. Para ello, habitualmente necesitará alguna información secreta, por ejemplo una clave. Por supuesto, alguien podría interceptar el mensaje, pero sin la clave no podría obtener información útil de él.



APLICACIONES DE LA CRIPTOGRAFÍA

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

- **Identificación y autenticación.** 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 (como usuarios reales). El sistema de certificación es la extensión lógica del uso de la criptografía para identificar y autenticar cuando se emplea a gran escala.
- **Seguridad de las comunicaciones.** Permite establecer canales seguros para aplicaciones que operan sobre redes que no son seguras (Internet).



EJEMPLOS

SSL y TLS son los protocolos más utilizados en la actualidad para proporcionar versiones seguras de protocolos de red (por ejemplo son los protocolos utilizado por HTTPS para proporcionar seguridad o un canal seguro a HTTP, como se verá posteriormente). SSL permite gestionar qué algoritmos se van a emplear e intercambiar las claves de encriptación y la autenticación de clientes y ordenadores, mientras que TLS es una evolución del anterior.

Comercio electrónico. El empleo de canales seguros y mecanismos de identificación posibilita el comercio electrónico o *e-commerce* a través de Internet, ya que tanto las empresas como los usuarios tienen garantías de que las operaciones no van a ser espiadas ni modificadas, reduciéndose el riesgo de fraudes.

A lo largo del capítulo se analizará cómo conseguir cada una de ellas.



HISTORIA DE LA CRIPTOGRAFÍA

Aunque se puede pensar que la utilización de la criptografía es algo relativamente nuevo, se lleva utilizando desde la Antigüedad. Los egipcios y mesopotámicos ya utilizaban sistemas de transformación deliberada de la información desde antes del año 1500 a. C. utilizando símbolos no habituales con significados menos comunes para dificultar la comprensión de la misma. Los chinos, griegos y hebreos años más tarde, a partir del 700 a. C., empezaron a introducir métodos de protección de la información en el envío de sus mensajes a través de mensajeros. Así, se introdujeron métodos como la valija diplomática, donde se sellaba el contenedor de información y el sello indicaba si se había abierto, la ocultación de información en bolas de papel o de seda tragadas por el propio mensajero en China, el alfabeto hebreo invertido *Atbash*, o la escritura en tatuajes en la cabeza afeitada del mensajero en Grecia, esperando hasta el crecimiento de su pelo antes del envío al cual antes de partir se esperaba que le creciera el pelo. Como ejemplo, se puede destacar el procedimiento empleado por los espartanos en el siglo v a. C., denominado *escitalo lacedomonio* por Plutarco. Era un sistema muy sencillo y se basaba en enmascarar el significado real de un texto alterando el orden de los signos que lo forman. Para ello se escribía el mensaje sobre una tela envuelta en una vara. Una vez escrito y retirada la vara, el mensaje podía enviarse sin miedo a que fuera comprendido. Así, el mensaje solo podía leerse cuando se enrollaba sobre un bastón del mismo grosor, bastón que poseía el destinatario. Los generales espartanos lo utilizaron para enviarse mensajes en la guerra contra Atenas.

Otro 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.



En el “método Cesar”, si dicho número fuera el 3, cada letra A (orden 1) del mensaje se sustituye por la letra de orden 4, la D. La B se sustituye por la E y así sucesivamente. En este sentido, un simple mensaje como HOLA quedaría codificado como LROD.

En este sistema, la clave es el número y debe ser conocido por el emisor y el receptor del mensaje. Aunque este sistema se ha utilizado hasta nuestra era, es un sistema demasiado simple, ya que para descifrarlo (conseguir conocer el mensaje), basta con probar todos los posibles números (desde 1 hasta el número máximo de letras [27 en castellano]) hasta que el mensaje decodificado tenga sentido.



¿SABÍAS QUE...?

Para dificultar el descifrado del mensaje, el ejército de Felipe II utilizó un sistema de cifrado con más de 500 símbolos y 6 tablas de códigos. Las letras, sílabas y trigramas más utilizados se sustituían por letras, números o hasta trazos especiales. De la misma forma, los nombres y palabras se sustituían por códigos numéricos y alfabéticos. Tal era el grado de confianza en el sistema que, cuando el ejército francés consiguió descifrarlo, la corte española llevó al rey de Francia ante el papa de Roma acusado de emplear magia negra.

La criptografía moderna, tal como se conoce hoy en día, se desarrolló durante la Segunda Guerra Mundial, coincidiendo con el desarrollo de las computadoras en los años 40 con el ordenador *Colossus*. Durante la Primera Guerra Mundial los ejércitos eran capaces de descubrir la mayoría de las claves utilizadas para cifrar los mensajes enemigos. Para ello bastaba con conseguir suficiente texto cifrado con una determinada clave. Empleando estos textos y mediante un análisis estadístico se podían reconocer patrones y deducir la clave. Sin embargo, durante la Segunda Guerra mundial, el ejército alemán empleó un mecanismo de cifrado mucho más complejo y difícil de descifrar. Se trataba de un cifrado rotatorio implementado sobre un dispositivo llamado "Enigma". Dicho dispositivo estaba constituido por un teclado parecido al de las máquinas de escribir. Sin embargo, sus teclas eran interruptores que activaban una luz que pasaba a través de varios rotores conectados entre sí, los cuales iban girando a cada tecla pulsada. Esto producía que mismas letras en el mensaje original tuvieran diferente resultado en el mensaje cifrado. Además, todo el mensaje obtenido variaba dependiendo de la configuración inicial de los rotores. Para hacerse una idea, el mensaje:

nczvvusxpnyminhzxmqxsfwxlkjahshnmcoccakujpmkcsmhkseinjusblkiosxckubhmlxcsjusrrdvkohulxwc
cbgvliyxeaoahxrhkkfvdrewezlxbafgyujqukgrtvukameurbveksuhhvoyhabcjwmaklfklmyfunrizrvurtkofdanjm
olbgffleoprgtflvrhowopbekvwmuqfmpwparmfhagkxiibg

significaba:

"Señal de radio 1132/19. Contenido: Forzados a sumergirnos durante ataque, cargas de profundidad. Última localización enemiga: 8:30 h, cuadrícula AJ 9863, 220 grados, 8 millas náuticas. Siguiendo. Cae 14 milibares. NNO 4, visibilidad 10".

Para una información más detallada sobre el funcionamiento de Enigma se puede consultar la página: [http://es.wikipedia.org/wiki/Enigma_\(máquina\)](http://es.wikipedia.org/wiki/Enigma_(máquina))



¿SABÍAS QUE...?

Debido a la importancia del descifrado de Enigma, su historia ha sido contada en multitud de libros. Entre otros muchos, se pueden destacar las novelas *Criptónomicón* de Neal Stephenson y *Enigma* de Robert Harris, que cuenta como fue llevado a cabo el descifrado del código por las tropas inglesas y gran parte de sus dificultades. Con posterioridad, una adaptación del libro *Enigma* con el mismo nombre fue llevada al cine en 2001.

En definitiva, Enigma era una máquina que automatizaba considerablemente los cálculos que era necesario realizar para las operaciones de cifrado y descifrado de mensajes, convirtiéndola en prácticamente indescifrable. Los científicos aliados necesitaron obtener una de las máquinas Enigma y realizar titánicos esfuerzos para conocer el sistema de cifrado alemán. La máquina alemana se convirtió así en el talón de Aquiles del régimen nazi, un topo en el que el ejército alemán confiaba ciegamente y que, en definitiva, trabajaba para el enemigo. La posibilidad de leer los mensajes enviados entre las fuerzas alemanas es considerada por numerosos historiadores como una de las principales causas de la victoria aliada final.



¿SABÍAS QUE...?

Alan Turing fue un científico británico considerado como uno de los padres de la informática. Trabajó en descifrar los códigos alemanes provenientes de la máquina Enigma durante la Segunda Guerra Mundial, y a él se debe gran parte de la victoria aliada. Debido a sus investigaciones, en las cuales era necesario conseguir poder computacional para descifrar los códigos, diseñó uno de los primeros ordenadores, la máquina *Colossus*, supercalculadora de la época realizada con más de 1.500 tubos de vacío.

Además, Turing trató el problema de la inteligencia artificial, formalizando el funcionamiento de los ordenadores mediante la máquina de Turing y proponiendo un experimento estándar para averiguar si una máquina es inteligente. Dicho experimento se conoce como "test de Turing" y consiste en un juez situado en una habitación distinta a donde se encuentran una máquina y un ser humano. El juez debe descubrir cuál es la persona y cuál es la máquina, estando permitido mentir. Si la máquina es suficientemente inteligente, el juez no debería distinguir cuál es humano y cuál no. Sesenta años después ninguna máquina ha podido pasar dicho test.

A pesar de ser uno de los científicos más influyentes en la sociedad, fue procesado por ser homosexual, siguiendo las anticuadas leyes de esa época. Se suicidó en 1954, dos años después de ser condenado por este motivo. En 2009 el primer ministro británico, Gordon Brown, pidió públicamente disculpas en nombre del Gobierno de su país por la "lamentable forma en la que Turing había sido tratado" en los últimos años de su vida.

Tras la conclusión de la Segunda Guerra Mundial, la criptografía vivió un desarrollo teórico importante. A mediados de los años 70 del pasado siglo, la autoridad de estándares estadounidense NBS (National Bureau of Standards), ahora denominada National Institute of Standards and Technology publicó el primer diseño lógico de un cifrador que estaría llamado a ser el principal sistema criptográfico de finales de siglo: el DES (*Data Encryption Standard*). En esas mismas fechas ya se empezaba a gestar lo que sería la, hasta ahora, última revolución de la

criptografía teórica y práctica: los sistemas asimétricos. Desde entonces hasta hoy en día ha habido un crecimiento espectacular de la tecnología criptográfica, aunque, como es normal, la mayoría de estos avances se han mantenido en secreto. Muchas de las investigaciones en este campo se han tratado como secretos militares, si bien en los últimos años el interés del mundo académico e internauta por el análisis criptográfico está sacando a la luz nuevas aplicaciones y desarrollos teóricos.

Aunque se han mostrado diferentes formas de cifrar mensajes a través de la historia, es importante señalar que los algoritmos criptográficos tienden a degradarse con el tiempo. Es decir, los algoritmos caducan al igual que la fruta fresca. 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). A medida que pasa el tiempo, es más fácil romper los algoritmos debido al avance en la potencia de los computadores. Sin embargo, al mismo tiempo que avanza el poder de los computadores, también avanzan los nuevos métodos y técnicas criptográficas.



¿SABÍAS QUE...?

El análisis de contraseñas es un proceso altamente paralelizable. Frente a los procesadores más rápidos disponibles en el mercado han aparecido los nuevos chips de tarjetas gráficas o GPU (*Graphics Processing Unit*, en inglés, por similitud con el nombre de CPU, *Central Processing Unit*) que permiten realizar operaciones masivamente paralelas, ya que cuentan con varios múltiples núcleos (del orden de más de 100), lo que permite ejecutar un elevado número de hilos en paralelo. La utilización de esta nueva arquitectura ofrece un rendimiento una centena de veces superior al de una CPU para aquellos algoritmos criptográficos que puedan ejecutarse de forma eficiente en una GPU. Esto reduce el tiempo necesario para romper una clave de forma muy significativa.



CARACTERÍSTICAS DE LOS SERVICIOS DE SEGURIDAD

Para proteger la información, tanto para su envío como para su almacenamiento se deben utilizar métodos criptográficos. Las claves de acceso a los sistemas y la gestión de permisos que proporcionan los sistemas operativos ayudan en el proceso pero no son suficientes para proteger la información de manera adecuada. Es necesario el uso de la criptografía.

Existe una serie de características que se deben cumplir para proporcionar la seguridad necesaria para un caso en particular:

- **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 esta característica suele ir acompañada de la autenticación de los usuarios que participan en la misma.

■ **Integridad** de la información: 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, ya que si no, por ejemplo, en acuerdos comerciales se podrían modificar productos, precios, etc.

■ **Autenticación**: se trata de asegurar el origen, autoría y propiedad de la información de quien envía el mensaje. Esto se requiere siempre que sea importante conocer cuál es la fuente del mensaje.



■ EJEMPLO 5.1

Es fácilmente modificable el remite de una carta. Por ejemplo, Carlos podría escribir una carta a Juan indicando en el remite que la ha escrito Pedro. Si necesitamos conocer con exactitud quién envía el mensaje, se deben establecer métodos que permitan asegurar con exactitud quién lo realizó.

■ **No repudio**: se trata de evitar que la persona que envía el mensaje o realiza una acción niegue haberlo hecho ante terceros. Esto es muy importante en acuerdos comerciales ya que si se cumple, el emisor, por ejemplo, no podrá decir que no envió el mensaje correspondiente pidiendo los productos indicados. Al igual que la característica de confidencialidad, necesita de la autenticación del origen de la información.

Habitualmente, el objetivo de un sistema seguro es que se cumplan todas, aunque esto queda en función de la necesidad concreta del usuario.

■ ESTRUCTURA DE UN SISTEMA SECRETO

Un sistema secreto actual se encuentra definido por dos funciones: la **función de cifrado** y la 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.

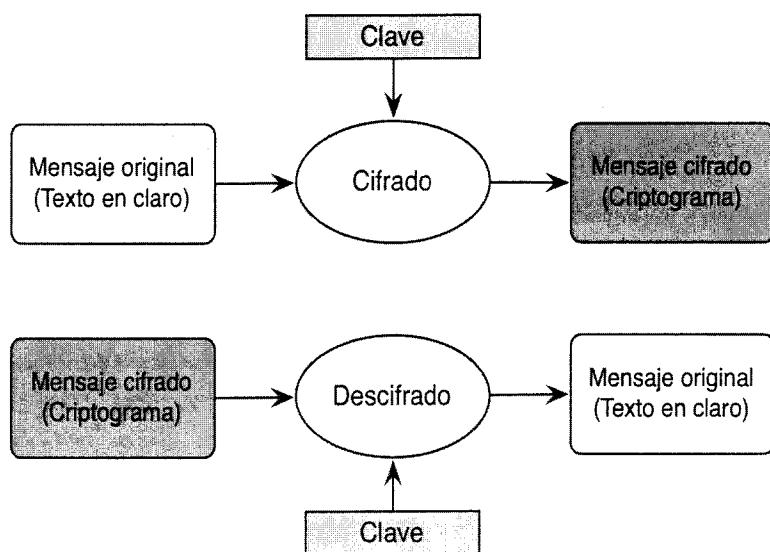


Figura 5.1. Estructura de un sistema secreto

La función de cifrado transforma, de forma reversible (mediante la función de descifrado), todos los posibles mensajes en claro en sus correspondientes mensajes cifrados o **criptogramas** (uno a uno) sabiendo que si se modifica la clave, el mensaje cifrado correspondiente debe ser diferente. Más aún, para dos mensajes similares, pero no iguales, la encriptación utilizando la misma clave debe dar mensajes cifrados no correlacionados, es decir, muy diferentes. Y de la misma forma, el mismo mensaje cifrado utilizando dos claves similares debe dar resultados no correlacionados. Además, debe ser imposible deducir la clave utilizada aunque se conozca la versión en claro y cifrada de cualquier mensaje.

Todo ello es necesario para dificultar la obtención de la clave mediante análisis estadístico. Shannon proponía que todos los cífrados se construyeran sobre la base de una confusión y difusión máximas. La **confusión** se refiere a distribuir las propiedades estadísticas (redundancia) de todos los elementos del mensaje (por ejemplo, alterar posición de caracteres, etc.) sobre el texto cifrado. La **difusión** se refiere a dificultar la relación entre la clave y el texto cifrado mediante complejos algoritmos de sustitución para dificultar su descifrado.



¿SABÍAS QUE...?

Shannon fue un ingeniero electrónico y matemático estadounidense, conocido por ser el padre de la teoría de la información, y al igual que Alan Turing investigó durante la Segunda Guerra Mundial, en su caso en los laboratorios Bell. Muchos de sus estudios, pensados inicialmente para todas las fuentes de información (telégrafo, teléfono, radio, etc.) han sido dirigidos hacia la criptografía y hacia el mundo de los ordenadores.

Según Shannon las características deseables que se deben cumplir en las funciones de cifrado y descifrado para proporcionar un sistema secreto serían:

1. El grado de protección determinará el trabajo y tiempo requeridos para poder vulnerar el sistema. 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.
2. Las claves deben ser de fácil construcción y sencillas.
3. Los sistemas secretos, una vez conocida la clave, deben ser simples pero han de destruir la estructura del mensaje en claro para dificultar su análisis.
4. Los errores de transmisión no deben originar ambigüedades.
5. La longitud del texto cifrado no debe ser mayor que la del texto en claro.

Sin embargo, dichas funciones se suponen conocidas por un atacante. Suelen ser públicas para que la gente pueda analizarlas y ver si les proporcionan la confianza suficiente para cifrar con ellas (para evitar, por ejemplo, puertas traseras que permitan descifrar mensajes al que diseñó el algoritmo). Así, la mayor parte del secreto no reside en ellas mismas sino en la clave utilizada. En general, los algoritmos criptográficos utilizan claves con un elevado número de bits (ceros y unos puestos uno detrás del otro). Normalmente se mide la calidad de un algoritmo por el esfuerzo requerido para descifrarlo. Como hemos visto anteriormente, se puede realizar un análisis estadístico de los mensajes

para ayudar a conseguir la clave pero también se puede tratar de encontrar la clave mediante fuerza bruta, es decir, probando todas y cada una de las posibles claves.



¿SABÍAS QUE...?

En febrero de 2013, una investigación interna de la agencia antidrogas norteamericana DEA afirmó que "es totalmente imposible interceptar los mensajes transmitidos a través de la aplicación iMessage entre dos dispositivos, incluso con la orden de un juez" debido al mecanismo de cifrado utilizado por la misma en los dispositivos Apple. Resulta impactante que hasta la fecha ni los Gobiernos sean capaces de descifrarlo teniendo recursos más que suficientes para probar por fuerza bruta, y más si lo comparamos con otras alternativas, como WhatsApp o Skype, las cuales son fácilmente rompibles. Esto da una idea de lo importantes que son los algoritmos de cifrado.

En este sentido, la seguridad depende tanto del algoritmo de cifrado como del nivel de secreto que se le dé a la clave. Si se pierde una clave, o es fácilmente averiguable (dependiente de los datos del poseedor de la clave: fecha de nacimiento, palabra relacionada, nombre relacionado, o hasta conjunción de todo ello) se pone en peligro todo el sistema. Saber elegir una clave y establecer métodos para cuidarla es un requisito muy importante para proporcionar un sistema seguro.

Además del nivel de seguridad de la clave, existen diferentes tipos de claves, cada una con sus ventajas e inconvenientes:

- **Claves simétricas (K_s):** las claves de cifrado y descifrado son la misma. El problema que se plantea con su utilización es cómo transmitir la clave para que el emisor (que cifra la información) y el receptor de la información (descifra) tengan ambos la misma clave. Dan lugar a lo que se denomina **modelo de clave privada**.
- **Claves asimétricas (K_p y K_c):** las claves de cifrado y descifrado son diferentes y están relacionadas entre sí de algún modo. Dan lugar al **modelo de clave pública**.



HERRAMIENTAS DE PROGRAMACIÓN BÁSICAS PARA EL CIFRADO

Al igual que para todo el resto de técnicas que se han visto hasta ahora, el lenguaje Java proporciona herramientas para el manejo de claves y mecanismos de cifrado. La mayoría de estas herramientas son clases e interfaces predefinidas, incluidas en los paquetes *java.security* y *javax.crypto*. Como se verá a lo largo de este capítulo, el uso de estas depende mucho de los sistemas de claves y mecanismos de cifrado empleados. No obstante, existe una serie de componentes básicos que es importante conocer desde el principio, ya que sirven como punto de partida. Son los siguientes:

- La interfaz *Key*.
- La interfaz *KeySpec*.
- La clase *Cipher*.

5.2.2.1 Interfaz Key

La interfaz *Key* (*java.security.Key*) representa una clave, que se puede usar para funciones de cifrado y descifrado. Al ser una interfaz, no implementa una funcionalidad concreta, sino que define las operaciones fundamentales que todo objeto que se use para almacenar claves debe implementar. Toda clave representada por un objeto de una clase que implemente la interfaz *Key* consta de tres partes fundamentales:

- **El algoritmo:** es el nombre de la función de cifrado y descifrado para la que está diseñada la clave. A lo largo de este capítulo se verán los ejemplos más comunes de algoritmos de este tipo.
- **La forma codificada:** es una representación de la clave. Esta representación está codificada siguiendo un formato específico, como el X.509, que se verá más adelante.
- **El formato:** es el formato en el que se encuentra la forma codificada de la clave.

Esta interfaz se utiliza para implementar clases que codifican claves de forma **opaca**. Esto significa que no proporcionan acceso a los componentes de la clave. Las claves en forma opaca se usan de forma directa por los algoritmos de cifrado y descifrado.

5.2.2.2 Interfaz KeySpec

La interfaz *KeySpec* (*java.security.spec.KeySpec*) se utiliza para representar claves en forma **transparente**. En oposición a la forma opaca, las claves en forma transparente sí ofrecen acceso a sus componentes, y se usan para poder diseminarlas (intercambiarlas entre diferentes entidades, como usuarios o aplicaciones).

5.2.2.2.1 Generadores y factorías de claves

A diferencia de la mayoría de objetos en Java, los objetos de las clases que implementan la interfaz *Key* no se suelen crear usando la operación *new*. En Java, las claves se generan usando objetos **generadores de claves y factorías de claves**. Los generadores de claves se utilizan para crear objetos de clases que implementan la interfaz *Key* (claves opacas). El ejemplo más típico de estas clases es *KeyGenerator* (*javax.crypto.KeyGenerator*). Las factorías de claves se utilizan para pasar de *Key* (clave opaca) a *KeySpec* (clave transparente) y viceversa. Análogamente, los generadores y factorías tampoco se crean usando *new*. Sus clases suelen disponer de un método estático llamado *getInstance()*, que sirve para crear instancias de estas clases.



Creación de una clave. Se usa el algoritmo DES, que se verá más adelante. En este ejemplo se usa además *SecretKey*, que implementa la interfaz *Key*, y la clase *SecretKeyFactory*, que está diseñada para operar con objetos *SecretKey*.

```
import java.security.spec.KeySpec;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.DESKeySpec;
```



EJEMPLO 5.5 (cont.)

```

public class CipherExample {

    public static void main(String[] args) {
        try {

            System.out.println("Obteniendo generador de claves con cifrado DES");
            KeyGenerator keygen = KeyGenerator.getInstance("DES");
            System.out.println("Generando clave");
            SecretKey key = keygen.generateKey();
            System.out.println("Obteniendo factoría de claves con cifrado DES");
            SecretKeyFactory keyfac = SecretKeyFactory.getInstance("DES");
            System.out.println("Generando keyspec");
            KeySpec keyspec = keyfac.getKeySpec(key, DESKeySpec.class);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

5.2.2.3 Clase Cipher

Los objetos de la clase *Cipher* (*javax.crypto.Cipher*) representan funciones de cifrado o descifrado. Se crean especificando el algoritmo que se desea utilizar. Posteriormente pueden ser configurados para realizar tanto operaciones de cifrado como descifrado, indicando en el proceso la clave necesaria. Sus métodos más importantes son:

Método	Tipo de retorno	Descripción
getInstance(String transformation)	Static Cipher	Método estático para crear objetos de clase <i>Cipher</i> . Recibe como parámetro el nombre del algoritmo de cifrado/descifrado que se desea emplear
init(int opmode, Key key)	void	Configura el objeto para que realice operaciones. Los modos de funcionamiento más habituales son <i>Cipher.ENCRYPT_MODE</i> para encriptar información y <i>Cipher.DECRYPT_MODE</i> para desencriptar. Recibe además como parámetro la clave que se desea utilizar
dofinal(byte[] input)	byte[]	Realiza la operación para la que ha sido configurado. Recibe como parámetro la secuencia de bytes que se desea cifrar/descifrar y devuelve el resultado de realizar la transformación correspondiente



EJEMPLOS

Este ejemplo muestra el cifrado de un mensaje, usando la clase *Cipher*. Para ello, se emplea el algoritmo DES, que se verá más adelante.

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;

public class CipherExample {

    public static void main(String[] args) {
        try {

            System.out.println("Obteniendo generador de claves con cifrado DES");

            KeyGenerator keygen = KeyGenerator.getInstance("DES");

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

            SecretKey key = keygen.generateKey();

            System.out.println("Obteniendo objeto Cipher con cifrado DES");

            Cipher desCipher = Cipher.getInstance("DES");

            System.out.println("Configurando Cipher para encriptar");

            desCipher.init(Cipher.ENCRYPT_MODE, key);

            System.out.println("Preparando mensaje");

            String mensaje = "Mensaje de prueba";

            System.out.println("Mensaje original: "+mensaje);

            System.out.println("Cifrando mensaje");

            String mensajeCifrado = new
String(desCipher.doFinal(mensaje.getBytes()));

            System.out.println("Mensaje cifrado: "+mensajeCifrado);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

5.3

MODELO DE CLAVE PRIVADA

La encriptación empleada habitualmente para transmitir mensajes entre ordenadores a través de la red se conoce con el nombre de “criptografía de clave secreta”, “simétrica” o “privada”. Tanto el emisor como el receptor del mensaje utilizan la misma clave K_s . Sin embargo, aunque este método presenta un buen rendimiento (se tarda poco en cifrar el mensaje) presenta algunos inconvenientes que son resueltos con la criptografía de clave pública, como veremos en la siguiente sección. En primer lugar está el cómo el emisor y el receptor obtienen la misma clave (sin enviarla sin cifrar por la red, por supuesto, ya que todo el mundo podría obtenerla). Y en segundo lugar está la necesidad de emplear claves distintas para comunicarse con cada uno de los posibles receptores de la información. En este sentido, un mismo emisor necesitaría tantas claves como posibles receptores tuviera, incrementándose la dificultad que supone gestionar esto. Además, la validez de una clave se pone en entredicho a medida que se va utilizando. Cuantos más mensajes se cifren con la misma clave, más expuesta estará a un análisis estadístico, por lo que es necesario cambiar las claves cada cierto tiempo.

Aun así, debido a su buen rendimiento se utiliza por un lado como apoyo a los sistemas de clave pública para asegurar confidencialidad de la información de forma rápida (mediante algoritmos de cifrado simétrico, como DES o AES, entre otros) y por otro lado para proporcionar integridad (mediante funciones *hash*).

5.3.1 ALGORITMOS DE CIFRADO SIMÉTRICO

5.3.1.1 Algoritmo DES

Es el algoritmo más extendido de clave simétrica. Basado en un sistema existente de IBM (con el nombre de Lucifer, con una clave de 128 bits), fue adoptado como estándar por el Gobierno de los Estados Unidos para comunicaciones no clasificadas en 1976, se usó hasta 1999. DES es un algoritmo que tomaba un texto de una longitud fija y lo transforma mediante una serie de complicadas operaciones en otro texto cifrado de la misma longitud. Dicho proceso sigue tres fases, como en la mayor parte de algoritmos simétricos:

- Permutación inicial: para dotar de confusión y difusión al algoritmo.
- Dieciséis (16) etapas en las que se aplica la misma función. La función es realizada mediante unas cajas de sustitución o cajas-S previamente definidas que comprimen la información, la permutan y la sustituyen. En su diseño radica la robustez del algoritmo.
- Permutación final inversa a la inicial.

Emplea para ello una clave de 56 bits para modificar las transformaciones realizadas. La reducción de 128 bits a 56 bits, la corta longitud de la clave y las misteriosas cajas-S definidas por la propia agencia de inteligencia estadounidense NSA (National Security Agency) mediante criterios de diseño clasificados hicieron que recibiera muchas críticas, ya que se sospechaba que el algoritmo fue debilitado para que pudieran leer mensajes cifrados. Aun así, nunca pudo ser roto más que por fuerza bruta.



Ataque de fuerza bruta

En 1998 se demostró que un ataque de fuerza bruta a un texto cifrado con el algoritmo DES era viable debido a la escasa longitud que emplea en su clave.

El número de 56 bits, con dos posibilidades por cada bit (0 o 1), implica 2^{56} posibles claves (72.057.594.037.927.936 posibilidades). Aunque más de 72.000 billones de posibilidades pueden parecer muchas, hoy en día un ordenador puede realizar un número enorme de operaciones por segundo, lo que hace que en un tiempo no demasiado elevado pruebe todas las claves posibles del algoritmo DES.

Por ese motivo se recomienda utilizar siempre cifrado mayor de 128 bits (por ejemplo para las conexiones Wi-Fi domésticas) para evitar que se pueda obtener la clave mediante fuerza bruta.

Debido a la rotura del algoritmo DES, IBM propuso el algoritmo 3DES en 1998 como solución. Como una clave 56 bits no era suficiente para evitar un ataque de fuerza bruta, 3DES hace un triple cifrado del DES alargando la clave hasta 192 bits sin necesidad de cambiar de algoritmo de cifrado. La mayoría de las tarjetas de crédito y otros medios de pago electrónicos utilizan el algoritmo 3DES para cifrar la información.

5.3.1.2 Algoritmo AES

El algoritmo **AES** (*Advanced Encryption Standard*), también llamado *Rijndael*, fue adoptado como estándar de cifrado por el Gobierno de los Estados Unidos en el año 2000 como sustituto del DES. Su desarrollo se llevó a cabo de forma pública y abierta y no de modo secreto como en el caso del DES. Al igual que el DES, AES es un sistema de cifrado por bloques, pero maneja longitudes de clave y de bloque variables, ambas comprendidas entre los 128 y los 256 bits.

AES es un algoritmo muy rápido y es fácil de implementar. Por ello se está utilizando a gran escala.

ACTIVIDADES 5.1

- » Busca información acerca del algoritmo AES y entiende su funcionamiento interno.
- » Además de DES, 3DES y AES, existen multitud de algoritmos de cifrado simétrico entre los que podemos destacar CAST, IDEA, Blowfish, etc. Elige uno de ellos y busca las diferencias existentes (número de bits en la clave, robustez, etc.) con DES y AES.



PROGRAMACIÓN DE CIFRADO SIMÉTRICO

La mayoría de las herramientas del lenguaje Java que se han visto hasta ahora para el uso de claves y cifrado se emplean en el modelo de cifrado simétrico.

Las más importantes son:

- La interfaz *SecretKey*, que implementa a su vez la interfaz *Key* y representa claves simétricas (opacas).
- La clase *KeyGenerator* se usa para generar claves simétricas. El método estático *getInstance()* sirve para obtener objetos de esta clase, indicando el algoritmo de cifrado como parámetro. Una vez obtenido un objeto, el método *generateKey()* crea claves de manera aleatoria. La clase de estas claves depende del algoritmo de cifrado especificado, pero todas ellas implementan la interfaz *SecretKey*.
- La clase *SecretKeyFactory* se emplea como clase de factoría de claves basadas en la interfaz *SecretKey*.
- La clase *SecretKeySpec* implementa la interfaz *KeySpec* y sirve como representación transparente de las claves simétricas.



Las siguientes líneas de código se emplean para crear una clave simétrica que se pueda usar junto con el algoritmo AES:

```
KeyGenerator keygen = KeyGenerator.getInstance("AES");  
SecretKey key = keygen.generateKey();
```

En este caso, la llamada al método *generateKey()* produce un objeto de una clase que implementa la interfaz *SecretKey* y representa claves del algoritmo AES.



Otro ejemplo de cifrado y descifrado simétrico de un mensaje, en este caso el algoritmo AES. Como se puede ver, se utiliza la misma clave para ambas operaciones.

```
import javax.crypto.Cipher;  
import javax.crypto.KeyGenerator;  
import javax.crypto.SecretKey;  
import javax.crypto.SecretKeyFactory;
```



```
public class CipherExample2 {  
    public static void main(String[] args) {  
        try {  
            System.out.println("Obteniendo generador de claves con cifrado AES");  
            KeyGenerator keygen = KeyGenerator.getInstance("AES");  
            System.out.println("Generando clave");  
            SecretKey key = keygen.generateKey();  
            System.out.println("Obteniendo objeto Cipher con cifrado AES");  
            Cipher aesCipher = Cipher.getInstance("AES");  
            System.out.println("Configurando Cipher para encriptar");  
            aesCipher.init(Cipher.ENCRYPT_MODE, key);  
            System.out.println("Preparando mensaje");  
            String mensaje = "Mensaje que se cifrará con AES";  
            System.out.println("Mensaje original: "+mensaje);  
            System.out.println("Cifrando mensaje");  
            String mensajeCifrado = new  
                String(aesCipher.doFinal(mensaje.getBytes()));  
            System.out.println("Mensaje cifrado: "+mensajeCifrado);  
            System.out.println("Configurando Cipher para desencriptar");  
            aesCipher.init(Cipher.DECRYPT_MODE, key);  
            System.out.println("Descifrando mensaje");  
            String mensajeDescifrado = new  
                String(aesCipher.doFinal(mensajeCifrado.getBytes()));  
            System.out.println("Mensaje descifrado: "+mensajeDescifrado);  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

ACTIVIDADES 5.2

- ▶ Busca información acerca de la clase *SecretKeySpec*. ¿De qué métodos dispone para acceder a los componentes fundamentales de la clave?
- ▶ Modifica el ejemplo anterior para construir una representación transparente (clase *SecretKeySpec*) de la clave simétrica empleada (objeto de clase *SecretKey*). Utiliza los métodos identificados anteriormente para obtener los componentes fundamentales de la clave e imprimirlas por pantalla.

RESUMEN DE INFORMACIÓN (FUNCIÓN HASH)

Una **función hash** es un algoritmo matemático que resume el contenido de un mensaje en una cantidad de información fija menor. Las funciones *hash* se emplean en multitud de campos.



Las funciones hash se emplean para identificar archivos independientemente de su nombre o ubicación (necesario, por ejemplo, para el intercambio de archivos en redes P2P, como eMule, o por *torrent* mediante BitTorrent). Si no existieran funciones *hash* habría que buscar en cada posición de una base de datos, una por una. Para evitarlo, aplicamos una función *hash* al dato y lo almacenamos en la posición dada por dicha función. Cuando queremos buscar el dato, no hace falta recorrer toda la base de datos. Simplemente, volvemos a aplicar la función *hash* al dato y accedemos a la posición indicada.

Para que sea de utilidad, la función *hash* debe satisfacer varios requisitos:

1. La descripción de la función de descifrado debe ser pública. Podrían utilizar una clave simétrica K_s para realizar el resumen, aunque en la mayoría de los casos, las funciones hash utilizadas son sin clave, para evitar tener que distribuirla entre emisor y receptor.
2. El texto en claro puede tener una longitud arbitraria. Sin embargo, el resultado debe tener una longitud fija (resumen). La longitud depende del algoritmo utilizado, pero en todos los casos será muy pequeño: 64 bits, 128 bits, etc.
3. Frente a las funciones habituales de cifrado que son uno a uno, como se está generando un resumen de menor tamaño no es posible que para un mensaje de entrada haya un único mensaje cifrado de resumen. A esto se le denomina **colisión**. Sin embargo, debe ser muy difícil encontrar dos documentos (texto de entrada) cuyo valor final para la función *hash* sea el mismo o presenten una colisión.
4. **Función de un único sentido:** dado uno de estos valores cifrados, debe ser imposible producir un documento con sentido que dé lugar a ese *hash*.
5. Aun cuando se conozca un gran número de pares (texto en claro, resumen) debe ser difícil determinar la clave.
6. Rápida de calcular.

En definitiva, la función *hash* ideal ha de aproximarse a la idea de función aleatoria ideal. Existen varios algoritmos para implementar funciones *hash*, entre los que se pueden destacar MD5 y SHA-1, los cuales no requieren clave. MD5 genera resúmenes de 128 bits mientras que los de SHA-1 son de 160 bits.

Todos estos requisitos hacen que las funciones *hash* sirvan para proporcionar pruebas de la **integridad** de la transferencia de información. Debido a esto se las llama “códigos de autenticación de mensajes” o MAC (en inglés, *Message Authentication Codes*).



¿SABÍAS QUE...?

Las funciones *hash* como MD5 sirven, entre otras cosas, para comprobar si la descarga de un fichero a través de Internet ha sido correcta. Se descarga dicho archivo y el MD5 correspondiente (siempre ocupa 128 bits), que contiene el resumen del fichero origen calculado por quien colgó el fichero. Para comprobar si el fichero descargado es correcto se aplica la misma función MD5, y si el resultado es el mismo que el MD5 publicado, la descarga se ha realizado de forma correcta.

5.3.3.1 Programación de resúmenes (*HASHES*)

La biblioteca de Java incluye una clase específica para generación de resúmenes de información, llamada *MessageDigest* (*java.security.MessageDigest*). Los objetos de esta clase se usan para crear resúmenes de secuencias de bytes, usando algoritmos como MD5 y SHA-1. El modo de uso de esta clase es el siguiente:

- 1. Obtención de una instancia:** de forma similar a los generadores y factorías de claves, los objetos de la clase *MessageDigest* se crean usando el método estático *getInstance()*. Al crear una instancia se debe especificar el algoritmo de resumen que se desea emplear.
- 2. Introducción de datos:** una vez obtenida la instancia, se actualiza el contenido de esta con los datos de los que se desea calcular el resumen.
- 3. Cómputo del resumen:** una vez actualizada la instancia con la información necesaria, se calcula el resumen usando el algoritmo seleccionado.

5.3.3.2 Clase *MessageDigest*

Los métodos más importantes de la clase *MessageDigest* se resumen en la siguiente tabla:

Método	Tipo de retorno	Descripción
<i>getInstance(String algorithm)</i>	static MessageDigest	Método estático para crear objetos de clase <i>MessageDigest</i> . Recibe como parámetro el nombre del algoritmo de resumen que se desea emplear
<i>update(byte[] input)</i>	void	Actualiza el contenido del objeto, incluyendo la información pasada como parámetro. Si este método se invoca varias veces va acumulando toda la información suministrada
<i>reset()</i>	void	Reinicia el objeto, eliminando toda la información introducida
<i>digest()</i>	byte[]	Realiza la operación de resumen sobre toda la información almacenada



EJEMPLOS

Un ejemplo de uso de la clase *MessageDigest*, usando el algoritmo MD5.

```
import java.security.MessageDigest;  
  
public class MessageDigestExample {  
  
    public static void main(String[] args) {  
  
        try {  
  
            System.out.println("Obteniendo instancia");  
  
            MessageDigest md = MessageDigest.getInstance("MD5");  
  
            System.out.println("Actualizando contenido de la instancia");  
  
            byte[] c1 = "Primera cadena".getBytes();  
            byte[] c2 = "Segunda cadena".getBytes();  
            byte[] c3 = "Tercera cadena".getBytes();  
  
            md.update(c1);  
            md.update(c2);  
            md.update(c3);  
  
            System.out.println("Calculando resumen");  
  
            byte[] resumen = md.digest();  
  
            System.out.println("Resumen: " + new String(resumen));  
  
            System.out.println("Reiniciando instancia");  
  
            md.reset();  
  
            System.out.println("Actualizando contenido de la instancia");  
  
            byte[] c4 = "Cuarta cadena".getBytes();  
  
            md.update(c4);  
  
            System.out.println("Calculando resumen");  
  
            resumen = md.digest();  
  
            System.out.println("Resumen: " + new String(resumen));  
  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

ACTIVIDADES 5.3

- Busca información acerca de la clase *MessageDigest*. ¿Qué otros algoritmos implementa?
- Modifica el ejemplo anterior para generar resúmenes usando al menos dos algoritmos distintos. Haz que el programa imprima ambos resúmenes por pantalla, y comprueba cómo estos cambian al seleccionar un algoritmo distinto.

54

MODELO DE CLAVE PÚBLICA

El modelo de clave pública se desarrolló en la década de 1970 para evitar los problemas derivados de la criptografía de clave simétrica, sobre todo el que tenía que ver con la distribución de las claves entre emisor y receptor. Para solucionarlo, en este sistema cada usuario tiene dos claves diferentes, las cuales están relacionadas:

- **Clave privada** K_c : que solo él mismo conoce. Como suele estar formada por multitud de bits, por ejemplo 1.024 bits, no suele ser recordable. Habitualmente se almacena en un fichero protegido con contraseña, la cual se pide al usuario cuando desee firmar.
- **Clave pública** K_p : publicada para todo el mundo que lo deseé.

Los sistemas de clave pública están basados en la existencia de funciones de sentido único relacionadas entre sí, lo que permite establecer la relación entre el par de claves. Las características de estas funciones son:

- ✓ Es fácil calcular la función directa $y = f(x)$
- ✓ Existe una función inversa f^{-1}
- ✓ Es computacionalmente imposible obtener f^{-1} a partir de f
- ✓ Si $f^{-1}(f(x)) = f(f^{-1}(x))$ se dice que comutan.

En este sentido, el sistema de clave pública es asimétrico, frente al sistema simétrico de clave secreta, pues se emplean claves diferentes para encriptar y desencriptar la información. Si alguien desea enviar un mensaje, buscará la clave pública de aquel al que desea enviárselo, y lo cifrará con dicha clave. Esta clave, conocida por todos, no permite desencriptar el mensaje. La única forma de desencriptarlo es utilizando la clave privada del receptor, la cual él únicamente conoce asegurando la **confidencialidad** de la información. Además, esto también puede funcionar a la inversa si las claves comutan, es decir, se puede encriptar un mensaje con la clave privada y desencriptarlo con la clave pública. Este procedimiento se denomina **firma digital**, y se estudiará en detalle en la siguiente sección. Sin embargo, este método no sirve para asegurar la confidencialidad del mensaje, aunque sirve para **autenticar** y **no repudiar** el mensaje, indicando que el que lo ha enviado es quien dice ser, ya que es el único que conoce la clave privada. De este modo, una vez producido el descifrado podemos estar seguros de que el mensaje ha sido enviado por la persona adecuada. Como ambas claves están relacionadas, para que estos sistemas sean seguros tiene que ser muy difícil (por no decir imposible) calcular una clave a partir de la otra.

Los sistemas de clave pública son computacionalmente mucho más costosos que los sistemas de clave privada. Por ello y para no poner en riesgo la clave privada cifrando con ella mucha información que podría ser posteriormente analizada, generalmente se emplea una combinación de ambos sistemas, clave privada y pública. Así, en el intercambio de información utilizando, por ejemplo, HTTPS, por rapidez se codifican los mensajes mediante algoritmos simétricos y una única clave K_s . Dicha clave se suele crear aleatoriamente por el emisor en función de la hora del sistema. Luego se utilizan sistemas de clave pública para codificar y enviar la clave simétrica K_s sin miedo a que sea vista por otro usuario, siendo el resto de mensajes intercambiados durante esa sesión cifrados con la clave K_s .

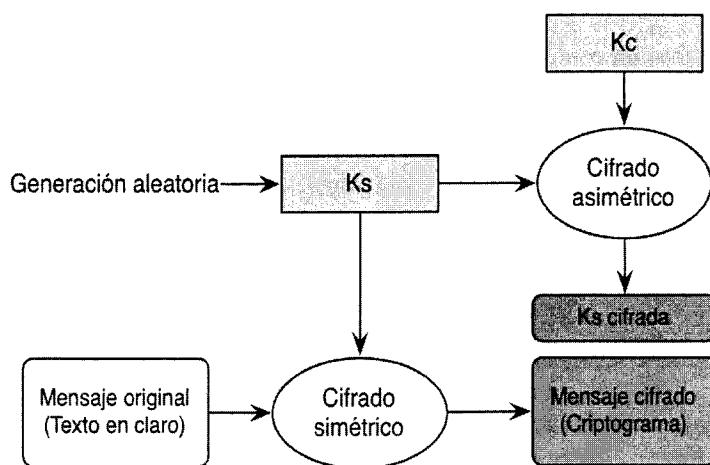


Figura 5.2. Modelo híbrido de comunicación segura

Los algoritmos de clave pública han demostrado su utilidad en redes de comunicación inseguras como Internet y, por ello, los nuevos DNI electrónicos se basan en esta tecnología. El más popular de los sistemas de clave pública es el RSA, incluido en la mayor parte del software para redes.



ALGORITMO RSA

El más popular de entre los algoritmos de clave pública, por su sencillez, es RSA. Debe su nombre a sus tres inventores: Ronald Rivest, Adi Shamir y Leonard Adleman, del MIT (en inglés, Massachusetts Technological Institute).

Se trata de un algoritmo asimétrico de cifrado por bloques, que utiliza una clave pública, conocida por todos, y otra privada, la cual es guardada en secreto por su propietario. Para conformar las claves el funcionamiento se basa en el producto de dos números primos muy grandes elegidos al azar. La pregunta que debemos responder para atacar este sistema es la siguiente: dado un número muy grande, ¿cuáles son los dos primos que se han multiplicado para obtenerlo? Esta pregunta no es fácil de responder, de hecho, representa un problema computacionalmente intratable siempre y cuando los números elegidos sean lo suficientemente grandes. Por tanto, la seguridad de este algoritmo radica en que no hay maneras rápidas conocidas de factorizar un número grande en sus factores primos. De modo que podemos asegurar que la seguridad de este sistema criptográfico y en consecuencia, de todos aquellos sistemas que lo emplean (tales como seguridad en sistemas operativos, seguridad en redes, comercio electrónico...) radica en la dificultad para determinar los factores primos de un número. Curiosamente, nadie ha conseguido probar o rebatir la seguridad del sistema RSA, por lo que se le considera uno de los algoritmos más seguros.

Para construir el texto cifrado, el proceso que realiza una identidad RSA es el siguiente:

Se eligen al azar y en secreto dos números primos p y q lo suficientemente grandes y se calcula:

$$\begin{aligned} n &= p \cdot q \\ \Phi(n) &= (p-1) \cdot (q-1) \end{aligned}$$

Se eligen dos exponentes e y d tales que:

- e y $\Phi(n)$ no tengan números en común que los dividan, es decir $mcd(e, \Phi(n)) = 1$
- $e \cdot d = 1 \text{ mod } \Phi(n)$, es decir, e y d son inversos multiplicativos en $\Phi(n)$

La clave pública es el par (e, n) y la clave privada (d, n) o (p, q) . Para cifrar M lo único que se debe hacer es $M^e \text{ mod } n = M'$, siendo el descifrado $M'^d \text{ mod } n$. Como se puede observar, comuta.



Elegiremos dos números pequeños para ver el proceso. Sin embargo, hay que tener en cuenta que RSA utiliza números primos muy grandes.

Sea $p=11$, primer número primo (privado), y $q=3$ segundo número primo (privado). Tenemos:

$$\begin{aligned} n &= p \cdot q = 33 \\ \Phi(n) &= (p-1) \cdot (q-1) = 10 \cdot 2 = 20 \end{aligned}$$

Consideremos $e=3$ como exponente público, el cual $mcd(20, 3) = 1$. Calculamos:

$$d = e^{-1} (\text{mod } \Phi(n)) = 3^{-1} \text{ mod } 20 = 7, \text{ exponente privado.}$$

En este sentido, tenemos:

- Clave pública K_p : $(3, 33)$
- Clave privada K_c : $(7, 33)$

Sea el número 7 el mensaje que deseamos cifrar. La función de cifrado con la clave pública $(3, 33)$ es:

$$7^3 \text{ mod } 33 = 13$$

Luego, lo que se envía como mensaje cifrado es 13. La función de descifrado utilizando la clave privada $(7, 33)$ es:

$$13^7 \text{ mod } 33 = 7$$

ACTIVIDADES 5.4

Además de RSA, existen otros algoritmos de clave pública. Busca información sobre el algoritmo Diffie-Helman y descubre en qué se basa su robustez frente a RSA, el cual se basa en la diferente complejidad computacional entre encontrar números primos y factorizar números compuestos.



PROGRAMACIÓN DE CIFRADO ASIMÉTRICO

Al igual que en el caso del cifrado simétrico, la biblioteca de Java ofrece herramientas para trabajar con cifrado asimétrico. Estas herramientas nos permiten generar pares de claves y cifrar la información usándolas. Las clases e interfaces más importantes que se usan para estas tareas son:

- La interfaz *PublicKey* (*java.security.PublicKey*). Implementa a su vez la interfaz *Key*, y se emplea para representar claves públicas en el modelo de cifrado asimétrico.
- La interfaz *PrivateKey* (*java.security.PrivateKey*). Implementa a su vez la interfaz *Key*, y se emplea para representar claves privadas en el modelo de cifrado asimétrico.
- La clase *KeyPair* (*java.security.KeyPair*). Los objetos de esta clase se utilizan para almacenar pares de claves en el modelo de cifrado asimétrico.
- La clase *KeyPairGenerator* (*java.security.KeyPairGenerator*). Se utiliza para generar pares de claves, de forma similar a los generadores de claves simétricas.
- La clase *KeyFactory* (*java.security.KeyFactory*). Se usa como factoría de claves para claves del modelo de cifrado asimétrico.

5.4.2.1 Clase KeyPair

Los objetos de la clase *KeyPair* representan pares de claves. Sus métodos más importantes son:

Método	Tipo de retorno	Descripción
<code>KeyPair(PublicKey publicKey, PrivateKey privateKey)</code>	<code>KeyPair</code>	Constructor de la clase
<code>getPrivate()</code>	<code>PrivateKey</code>	Método para obtener la clave privada almacenada
<code>getPublic()</code>	<code>PublicKey</code>	Método para obtener la clave pública almacenada

5.4.2.2 Clase KeyPairGenerator

Los objetos de la clase *KeyPairGenerator* se utilizan para generar pares de claves. El método estático *getInstance()* se emplea para obtener una instancia del generador (indicando el algoritmo de cifrado asimétrico que se desea emplear). Una vez hecho esto, se pueden generar pares de claves usando *generateKeyPair()*.

Método	Tipo de retorno	Descripción
<code>getInstance(String algorithm)</code>	<code>static KeyPairGenerator</code>	Método estático para crear objetos de clase <i>KeyPairGenerator</i> . Recibe como parámetro el nombre del algoritmo de cifrado asimétrico que se desea emplear
<code>generateKeyPair()</code>	<code>KeyPair</code>	Genera aleatoriamente un par de claves (pública y privada) y las devuelve como un objeto de clase <i>KeyPair</i>



SERVICIOS

Programa de ejemplo que utiliza cifrado asimétrico (algoritmo RSA). Observa cómo se usa una clave del par para cifrar y la otra para descifrar.

```
import javax.crypto.Cipher;
import java.security.KeyPair;
import java.security.KeyPairGenerator;

public class AsymmetricExample {

    public static void main(String[] args) {
        try {

            System.out.println("Obteniendo generador de claves con cifrado RSA");
            KeyPairGenerator keygen = KeyPairGenerator.getInstance("RSA");

            System.out.println("Generando par de claves");
            KeyPair keypair = keygen.generateKeyPair();

            System.out.println("Obteniendo objeto Cipher con cifrado RSA");
            Cipher aesCipher = Cipher.getInstance("RSA");

            System.out.println("Configurando Cipher para encriptar
usando la clave privada");

            aesCipher.init(Cipher.ENCRYPT_MODE, keypair.getPrivate());

            System.out.println("Preparando mensaje");
            String mensaje = "Mensaje de prueba del cifrado asimétrico";
            System.out.println("Mensaje original: "+mensaje);

            System.out.println("Cifrando mensaje");

            String mensajeCifrado = new
String(aesCipher.doFinal(mensaje.getBytes()));

            System.out.println("Mensaje cifrado: "+mensajeCifrado);

            System.out.println("Configurando Cipher para desencriptar
usando la clave pública");

            aesCipher.init(Cipher.DECRYPT_MODE, keypair.getPublic());

            System.out.println("Descifrando mensaje");
```



www.BIBBIO-FAIR.com

```
        String mensajeDescifrado = new
        String(aesCipher.doFinal(mensajeCifrado.getBytes()));
        System.out.println("Mensaje descifrado: "+mensajeDescifrado);

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

ACTIVIDADES 5.5

- Modifica el ejemplo anterior para cifrar el mensaje con la clave pública y descifrarlo con la privada. ¿Qué ocurre?
 - Cuando se cifra un mensaje usando el modelo de clave asimétrica, en unas ocasiones se cifra con la clave pública (y descifra con la privada) y en otras se cifra con la privada (y descifra con la pública). ¿En qué ocasiones estará indicada cada una? ¿Por qué?

5.4.2.3 Clase KeyFactory

La clase *KeyFactory* es la factoría de claves por defecto para claves del modelo de cifrado asimétrico. Funciona de manera muy similar al resto de factorías de claves, permitiendo cambiar entre claves opacas y transparentes, y viceversa. Sus métodos más importantes son:

Método	Tipo de retorno	Descripción
getInstance(String algorithm)	static KeyFactory	Método estático para crear objetos de clase KeyFactory. Recibe como parámetro el nombre del algoritmo de cifrado asimétrico que se desea emplear.
generatePrivate(KeySpec keySpec)		Genera una clave privada opaca, a partir de su versión transparente
generatePublic(KeySpec keySpec)		Genera una clave pública opaca, a partir de su versión transparente
getKeySpec(Key, key, Class<T> keySpec)	<T extends KeySpec>	Genera una clave transparente, a partir de su versión opaca. Se le debe especificar como parámetro además la clase derivada de KeySpec que se desea usar para crear la clave transparente. Esto es necesario porque la estructura interna de las claves depende mucho del algoritmo de cifrado, por lo que sus propiedades fundamentales pueden cambiar



EJEMPLOS

Programa de ejemplo que utiliza *KeyFactory* para obtener la información interna de un par de claves de cifrado asimétrico (algoritmo RSA). Observa cómo se usan las clases *RSAPublicKeySpec* y *RSAPrivateKeySpec*. Estas clases implementan la interfaz *KeySpec* y se usan para representar claves transparentes para el algoritmo RSA.

```
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.spec.RSAPrivateKeySpec;
import java.security.spec.RSAPublicKeySpec;

public class KeyFactoryExample {

    public static void main(String[] args) {
        try {
            System.out.println("Obteniendo generador de claves con cifrado RSA");
            KeyPairGenerator keygen = KeyPairGenerator.getInstance("RSA");
            System.out.println("Generando par de claves");
            KeyPair keypair = keygen.generateKeyPair();
            System.out.println("Obteniendo la factoría de claves con cifrado RSA");
            KeyFactory keyfac = KeyFactory.getInstance("RSA");
            System.out.println("Obteniendo las especificaciones del par de claves");
            RSAPublicKeySpec publicKeySpec = keyfac.getKeySpec(keypair.getPublic(),
                RSAPublicKeySpec.class);
            RSAPrivateKeySpec privateKeySpec =
                keyfac.getKeySpec(keypair.getPrivate(), RSAPrivateKeySpec.class);

            System.out.println("CLAVE PÚBLICA");
            System.out.println("Módulo: "+publicKeySpec.getModulus());
            System.out.println("Exponente: "+publicKeySpec.getPublicExponent());

            System.out.println("CLAVE PRIVADA");
            System.out.println("Módulo: "+privateKeySpec.getModulus());
            System.out.println("Exponente: "+privateKeySpec.getPrivateExponent());

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

ACTIVIDADES 5.6

- 💡 Busca información sobre el algoritmo de cifrado asimétrico DSA. ¿Es similar a RSA?
- 💡 Modifica el ejemplo anterior para que el par de claves se genere utilizando el algoritmo DSA. Modifica además la parte que obtiene la versión transparente del par de claves y los componentes de estas que muestra.

FIRMA DIGITAL

La firma digital hace referencia a un método de criptografía que asocia la identidad del emisor al mensaje que se transmite a la vez que se comprueba la integridad del mensaje. Esta firma digital se consigue mediante la aplicación de una función *hash*. A continuación, se emplea un sistema de clave pública para codificar mediante la clave privada el resumen obtenido anteriormente para autenticar el mensaje. Al resumen cifrado con la clave privada se le denomina **firma digital**.

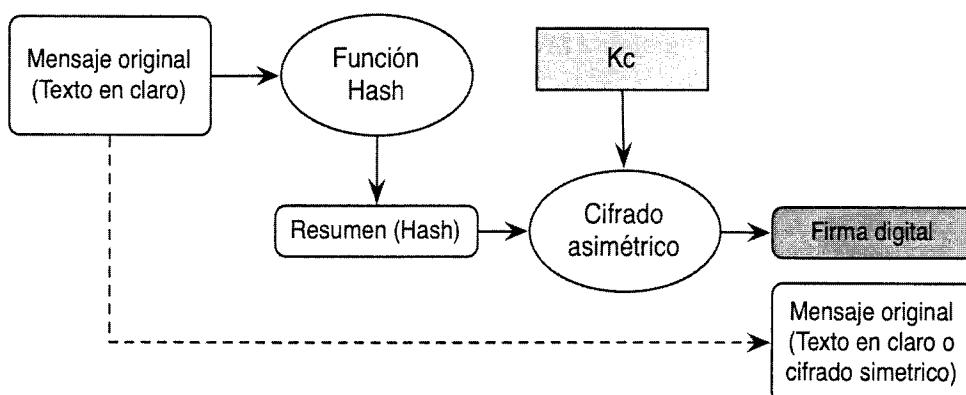


Figura 5.3. Construcción de la firma digital

De esta forma, cualquiera puede comprobar la firma usando la clave pública correspondiente del usuario que lo firmó. Para obtener las claves públicas correspondientes a los diferentes usuarios se utiliza el concepto de **certificado**, que se verá a continuación. Si el documento se modifica, la comprobación de la firma fallará, pero esto es precisamente lo que la verificación se supone que debe descubrir. El documento firmado, del que proviene el resumen, se puede enviar usando cualquier otro algoritmo de cifrado (cifrado simétrico por rapidez), o incluso ninguno si es un documento público.

En definitiva, la firma digital permite saber que la información no se ha modificado (en otro caso, se detecta el cambio porque el valor *hash* almacenado en el mensaje no coincide con el que se calcula al volver a hacer el resumen al descifrarlo) y quién ha enviado el mensaje (ya que está firmado con su clave privada).

5.4.3.1 Programación de firmado digital

La biblioteca de Java incluye la clase *Signature* (`java.security.Signature`) para realizar operaciones de firmado digital. Esta clase se basa en un par de claves asimétricas previamente generadas (usando *KeyPairGenerator* u otro método análogo). Usando este par de claves y la función *hash* correspondiente, los objetos de la clase *Signature* pueden firmar secuencias de bytes, por un procedimiento similar al que se sigue para obtener resúmenes. Además, usando *Signature*, se puede verificar que una firma es válida, esto es, que los datos de partida no han sido modificados. La secuencia de operaciones concreta que se debe llevar a cabo depende de la operación que se desee realizar. Para firmar un mensaje se siguen los siguientes pasos:

1 Obtención de la clave privada. Esto se puede hacer generando un par nuevo de claves (con *KeyPairGenerator*), utilizando uno ya existente, obteniendo la clave opaca a partir de su versión transparente (*KeySpec*), etc.

2 Creación de una instancia de *Signature*. Para ello se debe conocer el algoritmo de firma digital que se desea emplear, y que fue usado para generar la clave privada (y su correspondiente clave pública asociada). Ejemplos típicos de estos algoritmos son DSA (*Digital Signature Algorithm*) o MD5withRSA (Cifrado asimétrico RSA + resumen MD5).

3 Inicialización de la instancia de *Signature* para firmar. Se le debe proporcionar la clave privada en formato opaco.

4 Inserción de datos en la instancia de *Signature*. Se actualiza el contenido del objeto con los datos que se desean firmar.

5 Generación de la firma. Esto genera una secuencia de bytes con la firma asociada a los datos introducidos.

Si se desea verificar una firma, se deben seguir los siguientes pasos:

1 Obtención de la clave pública. Esto se puede hacer generando un par nuevo de claves (con *KeyPairGenerator*), utilizando uno ya existente, obteniendo la clave opaca a partir de su versión transparente (*KeySpec*), etc.

2 Creación de una instancia de *Signature*. Para ello se debe conocer el algoritmo de firma digital que se desea emplear, y que fue usado para generar la clave privada (y su correspondiente clave pública asociada).

3 Inicialización de la instancia de *Signature* para verificar. Se le debe proporcionar la clave pública en formato opaco.

4 Inserción de datos en la instancia de *Signature*. Se actualiza el contenido del objeto con los datos que se desean firmar.

5 Verificación de la firma. Esto produce un resultado lógico (verdadero o falso) indicando si la firma proporcionada verifica la integridad de los datos introducidos.

Los métodos más importantes de la clase *Signature* son:

Método	Tipo de retorno	Descripción
getInstance(String algorithm)	static Signature	Método estático para crear objetos de clase <i>Signature</i> . Recibe como parámetro el nombre del algoritmo de firma digital que se desea emplear
initSign(PrivateKey privateKey)	void	Inicializa el objeto para la operación de firmado. Se le debe pasar como argumento la clave privada
initVerify(PublicKey publicKey)	void	Inicializa el objeto para la operación de verificación de firma. Se le debe pasar como argumento la clave pública
update(byte[] input)	void	Actualiza el contenido del objeto, incluyendo la información pasada como parámetro. Si este método se invoca varias veces va acumulando toda la información suministrada
sign()	byte[]	Firma los datos contenidos en el objeto, usando la clave privada con la que ha sido inicializado. Devuelve la firma resultante. Al finalizar este método el objeto queda reiniciado
verify(byte[] signature)	boolean	Comprueba si la firma pasada como parámetro verifica la integridad de los datos contenidos en el objeto, usando la clave pública con la que ha sido inicializado. Devuelve verdadero o falso, indicando el resultado de la verificación. Al finalizar este método el objeto queda reiniciado



Un ejemplo de uso de la clase *Signature*, usando el algoritmo DSA.

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.Signature;

public class SignatureExample {

    public static void main(String[] args) {
        try {
            System.out.println("Obteniendo generador de claves con cifrado DSA");

            KeyPairGenerator keygen = KeyPairGenerator.getInstance("DSA");

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

            KeyPair keypair = keygen.generateKeyPair();
        }
    }
}
```



EJEMPLO 5.15 (cont.)

```
System.out.println("Creando objeto signature");

Signature signature = Signature.getInstance("DSA");

System.out.println("Firmando mensaje");

signature.initSign(keypair.getPrivate());

String mensaje = "Mensaje para firmar";

signature.update(mensaje.getBytes());

byte[] firma = signature.sign();

System.out.println("Comprobando el mensaje firmado");

signature.initVerify(keypair.getPublic());

signature.update(mensaje.getBytes());

if (signature.verify(firma))
    System.out.println("El mensaje es auténtico :-)");

} catch (Exception e) {
    e.printStackTrace();
}
}
```

ACTIVIDADES 5.7

- Busca más información sobre la clase *Signature* ¿Qué otros algoritmos de firma digital soporta?
- Modifica el ejemplo anterior para que use el algoritmo MD5withRSA.
- La clase *Signature* ofrece un mecanismo muy cómodo para generar y verificar firmas digitales. No obstante, los procesos de firma y verificación de datos mediante firma digital pueden realizarse sin usar esta clase. El resto de las herramientas de programación en Java vistas hasta ahora deberían ser suficientes para realizar el mismo proceso, pero de manera manual y algo más tediosa ¿Se te ocurre cómo? ¿Qué otras clases estarían involucradas?

5.4.3.2 Certificados digitales

Un **certificado** digital o certificado electrónico es un documento firmado electrónicamente por alguien en quien se confía, que confirma la identidad de una persona o servidor vinculándolo con su correspondiente clave pública K_p . En definitiva, un certificado es un documento que permite a los usuarios y servicios identificarse en Internet para realizar trámites, presentando su correspondiente clave pública, confirmada por una **autoridad de certificación** de la cual nos fiamos.



¿SABÍAS QUE...?

El DNI tradicional puede verse como un ejemplo de un certificado en el mundo real no digital. El DNI es un documento que confirma que su portador tiene un determinado nombre, nació en una fecha determinada y vive en un sitio en concreto. Si alguien necesita identificarse puede mostrar el DNI indicando que es esa persona. Sin embargo, cuando se recibe un DNI, no estamos confiando en el documento en sí, sino en la autoridad que lo emitió, en este caso en concreto el Ministerio del Interior, de la cual suponemos que se encargó de comprobar que la persona que lo porta es quien dice el documento que es.

Un certificado electrónico sirve, por tanto, para:

- Firmar digitalmente para garantizar la integridad de los datos trasmítidos y su procedencia, y autenticar la identidad del usuario de forma electrónica ante terceros. Al presentar un certificado, estamos permitiendo que se puedan comprobar las firmas digitales que se emitan con la clave privada relacionada con el certificado.
- Cifrar datos para que solo el destinatario del documento pueda acceder a su contenido. Al tener un certificado, se puede cifrar un mensaje con la clave pública que aparece en él para el dueño del certificado sea el único que lo pueda descifrar con su clave privada.

El formato específico de certificado que se suele utilizar es X.509 el cual incluye:

- Identificación del usuario o servicio. Los certificados no solamente sirven para identificar usuarios, sino también servidores como www.urjc.es.
- Validez de certificado: período en el cual el certificado es válido.
- Clave pública del usuario K_p
- La firma de una autoridad de certificación (CA) en la que se confíe. En concreto es la firma digital (resumen) del certificado firmado por la CA.



Certificado X.509:

• **Identificación de CA:**

Data:

Version: 3 (0x2)
Serial Number: 2 (0x2)
Signature Algorithm: md5WithRSAEncryption
Issuer: O=Test, OU=Test, OU=simpleCA-server.com, CN=Simple CA

• **Validez de certificado:**

Validity

Not Before: Apr 2 09:43:30 2011 GMT
Not After: Apr 2 09:43:30 2012 GMT

• **Identificación de usuario:**

Subject: O=Test, OU=Test, OU=simpleCA-server.com, OU=server.com, CN=Alberto Sanchez Campos

• **Clave pública:**

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit):

00:ac:60:e6:55:3b:a3:d0:a4:7e:3a:4f:dd:04:16:
7d:fc:b3:26:f1:44:a3:d3:06:fc:e7:ae:d1:22:85:
7d:d3:28:fb:6e:57:3e:33:7a:9d:b6:23:d8:dd:52:
38:93:b9:49:af:59:f4:e4:45:dd:b7:e7:c8:a5:8c:
91:ee:3e:67:6c:c5:9c:f2:cd:5e:ff:bf:43:5d:32:
e3:dd:e4:65:39:dc:5c:05:af:4f:7e:d2:59:8c:d3:
28:46:25:c0:9d:84:e1:22:24:b3:f1:33:d1:88:73:
e6:89:9e:7f:72:d2:07:8a:63:d3:30:3b:39:0f:b7:
70:b7:47:43:84:f3:ec:b1:79

Exponent: 65537 (0x10001)

X509v3 extensions:

Netscape Cert Type:

SSL Client, SSL Server, S/MIME, Object Signing

• **Firma de la CA:**

Signature Algorithm: md5WithRSAEncryption
8a:f1:93:7f:33:8e:e6:8b:0e:93:d4:97:6d:bc:07:f0:b8:da:
a7:74:8c:63:3a:c5:ff:6b:59:38:23:df:25:63:d9:f8:07:e1: ...

5.4.3.3 Servicios en red seguros: protocolos SSL, TLS y SSH

Cada vez que accedemos a una página cuya URL empieza por HTTPS, estamos utilizando un protocolo de criptografía para acceder a los contenidos de dichas páginas de forma confidencial. Para ello el servidor presenta su certificado, que debe ser validado por el usuario si no se ha podido comprobar que está firmado de una CA confiable por el navegador. A partir de ese certificado se obtiene la clave pública del servidor, la cual se utiliza para enviarle la clave *simétrica* K_s de sesión que será utilizada para la comunicación. En este sentido, la comunicación se realiza mediante un modelo mixto, se transmite utilizando el modelo de clave pública la clave simétrica para poder enviar y recibir mensajes con el servidor mediante un modelo de clave privada. Este es solo un ejemplo de cómo se articulan la mayoría de comunicaciones seguras a través de Internet y otras redes de comunicaciones. Tal y como se comentó anteriormente, la base de la mayoría de estas comunicaciones son los protocolos SSL (*Secure Sockets Layer*) y sus sucesores TLS (*Transport Layer Security*).

SSL y TLS ofrecen una interfaz de programación basada en *sockets stream*, muy similar a la vista en el Capítulo 3. Además de las funcionalidades proporcionadas por los *sockets stream* (en la pila IP, *sockets TCP*), SSL y TLS agregan las siguientes características de seguridad:

- Uso de criptografía asimétrica para el intercambio de claves de sesión.
- Uso de criptografía simétrica para asegurar la confidencialidad de la sesión.
- Uso de códigos de autenticación de mensajes (resúmenes) para garantizar la integridad de los mensajes.

Como se puede ver, SSL y TLS incorporan la mayoría de mecanismos de seguridad vistos hasta ahora. Estos protocolos se sitúan entre el nivel de aplicación y el de transporte, mejorando de manera transparente la seguridad de los protocolos de nivel de aplicación que los usan (como FTPS o HTTPS). El funcionamiento general de los *sockets stream* SSL/TLS se basa en el establecimiento y mantenimiento de una **sesión segura**. Una sesión segura es una conexión similar a la de un *socket stream*, pero en la que se cumplen tres condiciones básicas:

- La identidad del elemento servidor está garantizada (autenticación).
- La privacidad de las comunicaciones está garantizada (confidencialidad).
- Los mensajes que se intercambian no pueden ser alterados de ninguna manera (integridad).

Para garantizar la autenticación del servidor, SSL y TLS utilizan cifrado asimétrico. El servidor dispone de un **certificado de servidor** que emplea para que los clientes puedan confiar en que se están conectando al elemento adecuado.

Para garantizar la confidencialidad de las comunicaciones, SSL y TLS utilizan cifrado simétrico. Una vez la identidad del servidor ha sido validada usando su certificado, el cliente genera una clave de cifrado simétrico de forma aleatoria y la envía al servidor, cifrada con la clave pública del certificado servidor. Esto garantiza que solo el servidor pueda recibirla y utilizarla. A esta clave simétrica se la denomina **clave de sesión**. Una vez la clave de sesión está en posesión del servidor, ambas partes pueden intercambiar mensajes de manera confidencial.

Por último, para garantizar la integridad de los mensajes se utilizan resúmenes o MAC. Junto con los mensajes se envían sus resúmenes, para que ambas partes puedan comprobar si el mensaje ha sido alterado por el camino.



¿SABÍAS QUE...?

SSL y TLS utilizan una mezcla de cifrado asimétrico y simétrico por motivos de eficiencia. El cifrado asimétrico permite garantizar la autenticación, pero resulta muy costoso computacionalmente, por lo que no resulta viable para cifrar todos los mensajes de la sesión. En su lugar, se utiliza para intercambiar de forma segura un único mensaje con la clave de sesión. Esta clave se usa para cifrado simétrico, mucho menos costoso computacionalmente que el asimétrico.

Otro de los mecanismos más comúnmente usados en Internet para la realización de comunicaciones seguras es el protocolo SSH. Tal y como se explicó en el capítulo anterior, SSH es un protocolo de nivel de aplicación similar a Telnet, cuya función es proporcionar comunicación bidireccional basada en texto plano (caracteres ASCII) entre dos elementos de una red. SSH se usa habitualmente para establecer sesiones remotas de operación por línea de mandatos entre máquinas de una misma red de forma segura. Aunque SSH es un protocolo distinto a SSL y TLS, sus características de seguridad son similares a las de estos. SSH utiliza cifrado asimétrico para garantizar autenticación de las partes (soporta los algoritmos DSA y RSA), cifrado simétrico para garantizar confidencialidad de la sesión y resúmenes para garantizar integridad. SSH comprime la información enviada para aprovechar el ancho de banda de la red.

Existe además un protocolo de nivel de aplicación para transferir ficheros de manera sencilla y segura, llamado SFTP, basada en SSH. SFTP hereda de SSH las características de autenticación, confidencialidad, integridad y compresión de la información.



¿SABÍAS QUE...?

Existen multitud de aplicaciones que permiten establecer sesiones remotas usando SSH y SFTP. En Windows, las más habituales son PuTTY (para SSH) y WinSCP (para SFTP). Ambas son libres y gratuitas, y ofrecen muchas comodidades, como gestión de claves, exploración de archivos de manera visual, etc.

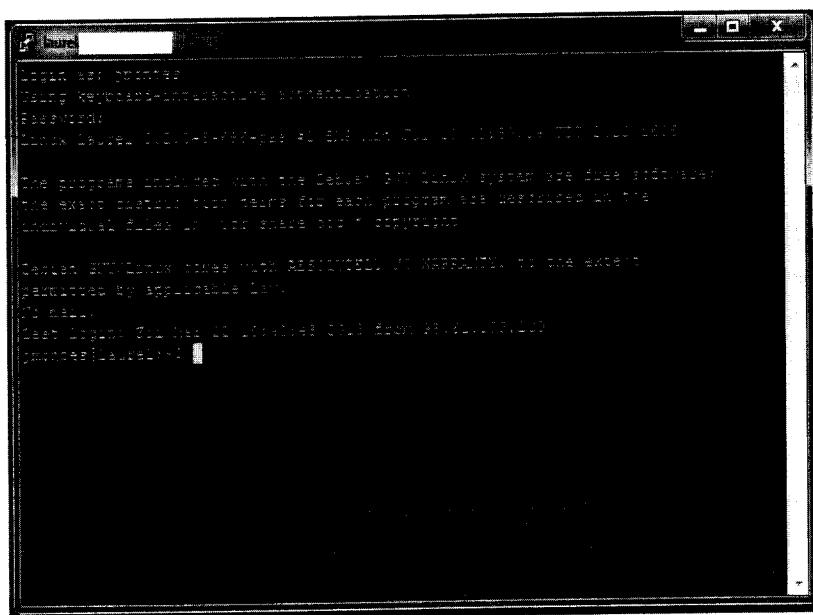


Figura 5.4. Sesión remota por SSH usando PuTTY

ACTIVIDADES 5.8

- Descarga PuTTY (<http://www.chiark.greenend.org.uk/~sgtatham/putty/>) y establece una sesión remota con un servidor a través de SSH.
- Descarga WinSCP (<http://winscp.net>) y establece una sesión remota con un servidor por SFTP. Descarga un fichero usando esta herramienta.

5.4.3.3.1 Programación con *Sockets* seguros

La biblioteca de Java dispone de herramientas para utilizar *sockets* basados en SSL, que incorporan las características de seguridad de dicho protocolo. Las clases más importantes relacionadas son:

- *SSLSocket* (`javax.net.ssl.SSLSocket`). Clase similar a *Socket*, pero incorporando SSL. Sirve para representar *sockets stream* cliente seguros.
- *SSLSocketFactory* (`javax.net.ssl.SSLSocketFactory`). Clase generadora de objetos *SSLSocket*.
- *SSLServerSocket* (`javax.net.ssl.SSLServerSocket`). Clase similar a *ServerSocket*, pero incorporando SSL. Sirve para representar *sockets stream* servidor seguros.
- *SSLServerSocketFactory* (`javax.net.ssl.SSLServerSocketFactory`). Clase generadora de objetos *SSLServerSocket*.

El mecanismo de uso de las clases *SSLSocket* y *SSLServerSocket* es análogo a *Socket* y *ServerSocket*. La única excepción es que estos objetos se crean usando las clases generadoras *SSLSocketFactory* y *SSLServerSocketFactory*.



Ejemplo de un programa que usa *sockets stream* servidor seguros.

```
import java.io.IOException;
import java.io.InputStream;
import java.net.InetSocketAddress;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;

public class SecureServer {

    public static void main(String[] args) {
        try {
            System.out.println("Obteniendo factoría de sockets servidor");

            SSLServerSocketFactory serverSocketFactory =
                (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();
        }
    }
}
```



EJEMPLO 5.17 (cont.)

```
System.out.println("Creando socket servidor");

SSLServerSocket serverSocket =
(SSLServerSocket) serverSocketFactory.createServerSocket();

System.out.println("Realizando el bind");

InetSocketAddress addr = new InetSocketAddress("localhost", 5555);
serverSocket.bind(addr);

System.out.println("Aceptando conexiones");

SSLSocket newSocket = (SSLSocket) serverSocket.accept();

System.out.println("Conexión recibida");

InputStream is = newSocket.getInputStream();

byte[] mensaje = new byte[25];
is.read(mensaje);

System.out.println("Mensaje recibido: "+new String(mensaje));

System.out.println("Cerrando el nuevo socket");

newSocket.close();

System.out.println("Cerrando el socket servidor");

serverSocket.close();

System.out.println("Terminado");

} catch (IOException e) {
    e.printStackTrace();
}
}
```



EJEMPLO 5.18

Ejemplo de un programa que usa *sockets stream* cliente seguros. Este programa sirve de cliente para el servidor del ejemplo anterior.

```
import java.io.IOException;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
```



```
public class SecureClient {  
  
    public static void main(String[] args) {  
        try {  
  
            System.out.println("Obteniendo factoría de sockets cliente");  
  
            SSLSocketFactory socketFactory =  
                (SSLSocketFactory) SSLSocketFactory.getDefault();  
  
            System.out.println("Creando socket cliente");  
  
            SSLSocket clientSocket = (SSLSocket) socketFactory.createSocket();  
  
            System.out.println("Estableciendo la conexión");  
  
            InetSocketAddress addr = new InetSocketAddress("localhost", 5555);  
            clientSocket.connect(addr);  
  
            OutputStream os = clientSocket.getOutputStream();  
  
            System.out.println("Enviando mensaje");  
  
            String mensaje = "mensaje desde el cliente transmitido por SSL";  
            os.write(mensaje.getBytes());  
  
            System.out.println("Mensaje enviado");  
  
            System.out.println("Cerrando el socket cliente");  
  
            clientSocket.close();  
  
            System.out.println("Terminado");  
  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Los ejemplos anteriores muestran un cliente y un servidor usando *sockets SSL* para comunicarse de manera segura. Para que esta comunicación se pueda establecer, se debe disponer de un certificado de servidor válido en el que confie el cliente. Para facilitar la gestión de estos certificados, Java proporciona la herramienta *keytool*.



EJEMPLO 5.9

La herramienta *keytool* se puede usar para crear un certificado de servidor. Véase por ejemplo el siguiente mandato:

```
> keytool -genkey -keystore mySrvKeystore -keyalg RSA
```

Este mandato crea un certificado y lo almacena en un **fichero de almacén de claves**, o *keystore*. En este caso el fichero se llama *mySrvKeystore*. El algoritmo de cifrado con el que se ha generado el certificado es RSA.

Una vez generado el certificado, el programa servidor del ejemplo anterior se lanzaría con el siguiente mandato:

```
> java -Djavax.net.ssl.keyStore=mySrvKeystore  
-Djavax.net.ssl.trustStore=mySrvKeystore  
-Djavax.net.ssl.keyStorePassword=password  
SecureServer
```

Y el cliente con el siguiente mandato:

```
> java -Djavax.net.ssl.keyStore=mySrvKeystore  
-Djavax.net.ssl.trustStore=mySrvKeystore  
-Djavax.net.ssl.keyStorePassword=password  
Secureclient
```

En ambos casos, la opción *-Djavax.net.ssl.keyStore=mySrvKeystore* le indica a la máquina virtual de Java dónde se encuentra el fichero de almacén de claves. La opción *-Djavax.net.ssl.trustStore=mySrvKeystore* le indica a la máquina virtual de Java que los certificados contenidos en *mySrvKeystore* son de confianza. Por último, la opción *-Djavax.net.ssl.keyStorePassword=password* proporciona la contraseña del almacén de claves.

ACTIVIDADES 5.9

- Fijándote en el ejemplo anterior, genera un nuevo almacén de claves que contenga un certificado que utilice cifrado DSA. Copia los ejemplos servidor y cliente que usan *sockets SSL* y ejecútalos usando el nuevo certificado.

5.4.3.4 Autoridades de certificación

Una **autoridad de certificación** (CA por sus siglas en inglés, *Certification Authority*) es una entidad de confianza, responsable de emitir y revocar los certificados electrónicos que se emplean en la criptografía de clave pública.

La firma de una autoridad de certificación (CA) asegura que:

- El certificado procede de la CA.
- Responde de que el nombre corresponde con quien dice que es.
- Afirma la relación de obligación existente entre el nombre y la clave pública.



¿SABÍAS QUE...?

En España, actualmente los certificados electrónicos son emitidos principalmente por entidades públicas que hacen de autoridad de certificación, como la Dirección General de la Policía (DGP), encargada del DNIe o DNI electrónico, o la Fábrica Nacional de Moneda y Timbre (FNMT).

El DNIe español al ser de uso obligado dispone de más de 32 millones de usuarios, lo que facilita su uso y la reducción de costes burocráticos para las Administraciones públicas y los usuarios.

Para conseguir el certificado correspondiente como persona de FNMT es necesario contar con DNI o NIE para poderse identificar en sus instalaciones.

A la hora de solicitar un certificado:

1 Utilizando ciertas funciones del propio navegador y de la página de la CA, se completan ciertos datos identificativos y se genera una pareja de claves pública/privada.

2 Con esa información se compone una petición CSR (*Certificate Signing Request*) que se envía a la CA en formato PKCS (*Public-Key Cryptography Standards*), estándar *de facto* para la utilización del modelo de clave pública.

La CA comprueba la información de identificación aportada, pudiendo ser requerido que vayan a comprobar su identidad a las oficinas de la propia CA en caso de ser una persona física.

3 Se firma el certificado que muestra la identificación junto a la clave pública y se envía al solicitante, para que lo instale en el navegador para su utilización. Habitualmente es necesario descargarlo en el mismo navegador sin actualizar desde donde se hizo la petición. El usuario mantiene siempre su clave privada en su poder, no habiendo sido necesario su envío a la CA.

4 Pueden revocarse certificados (por ejemplo, por pérdida del DNIe) estableciendo una *Certificate Revocation List* (CRL) que debería actualizarse en los navegadores.

Las CA disponen de sus propios certificados públicos, cuyas claves privadas asociadas son empleadas para firmar digitalmente los certificados que emiten. Y el certificado público de la CA, puede a su vez venir firmado por otra CA de rango superior de forma jerárquica. Una jerarquía de certificación consiste en una estructura jerárquica de CA en la que se parte de una CA que se autofirma, y en cada nivel existen una o más CA que pueden firmar certificados.

ACTIVIDADES 5.10

- Solicita un certificado a la FNMT y sigue todo el proceso hasta tener instalado tu propio certificado en el navegador. Intenta acceder a diferentes servicios de la Administración electrónica (Seguridad Social, recursos de multas, etc.) utilizando el certificado para ver su funcionamiento.

Para confiar en una CA hay que instalar en el navegador su certificado correspondiente en el repositorio de CA de confianza, y a partir de ese momento el propio navegador comprobará la validez de los certificados emitidos por esa CA, ya que tiene su clave pública. Será el navegador el responsable de su validación, teniendo que repetirlo por cada uno de los navegadores que existan en el sistema, si utilizamos varios: Google Chrome, Mozilla Firefox, Internet Explorer, etc. Si un certificado recibido no está validado por una CA de confianza, pedirá al usuario su conformidad con el certificado, mostrando el certificado recibido. El propio navegador puede traer por defecto CA de confianza reconocidas internacionalmente, como *VeriSign*, permitiendo que añadan más con posterioridad. Para ello se pueden localizar los certificados correspondientes en las páginas web de las CA o el propio certificado presentado por un servidor o usuario puede contener toda la cadena de certificación necesaria para ser instalado con confianza.

CONTROL DE ACCESO

A lo largo de este capítulo se ha hablado de la criptografía como el principal mecanismo de seguridad que disponen las aplicaciones. Los diferentes modelos criptográficos (clave privada y clave pública) y su tecnología derivada (certificados digitales, *sockets* seguros, etc.) permiten garantizar las características claves de los servicios de seguridad: confidencialidad, integridad, autenticación y no repudio. Estas características se refieren, en su mayor parte, a la forma en que las aplicaciones se comunican y su gestión de la información. Además, muchas veces es importante garantizar la seguridad en lo que respecta a los recursos externos a los que accede una aplicación, como ficheros leídos, dispositivos de la máquina donde se ejecuta la aplicación, etc. El conjunto de mecanismos que gestionan estos aspectos de seguridad se denominan **control del acceso**.

El control de acceso suele ser responsabilidad del sistema operativo sobre el que se ejecuta una aplicación. Usando diferentes mecanismos de control de acceso, el sistema operativo puede controlar lo que puede realizar una aplicación, evitando que ésta lleve a cabo operaciones no autorizadas.

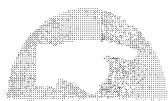
¿SABÍAS QUE...?

El mecanismo de control de acceso más habitual en los sistemas operativos modernos es la gestión de permisos de acceso a ficheros. Este mecanismo de control de acceso permite al sistema operativo determinar si una aplicación tiene permiso para realizar operaciones sobre ficheros como leer, escribir, crear un fichero nuevo, etc. La manera en la que se gestionan estos permisos suele depender, entre otras cosas, del sistema operativo. Por ejemplo, el control de acceso a ficheros en Windows 7 y GNU Linux es bastante diferente.

La función principal de los mecanismos de control de acceso es asegurar que las aplicaciones que ejecutan en una máquina no produzcan violaciones de seguridad que afecten a su integridad o sus usuarios. Ejemplos típicos de aspectos que se gestionan usando control de acceso son:

- Operaciones sobre ficheros y directorios, como lectura, escritura, creación, ejecución, etc.
- Acceso a dispositivos hardware, como periféricos de video, audio, geolocalización (GPS), etc.
- Servicios de sistema, como comunicaciones en red, almacenamiento de datos, etc.
- Operaciones de administración del sistema, como creación de usuarios, instalación de software, etc.

En la mayor parte de sistemas el control de acceso se basa en la autenticación de usuarios y sus políticas de seguridad asociadas. La **autenticación de usuarios** emplea mecanismos criptográficos para identificar a los usuarios que hacen uso del sistema. Un usuario suele identificarse en el sistema usando un nombre y una contraseña, que lo identifican de forma única. Cuando un usuario ejecuta una aplicación, ésta va asociada a él, a efectos de control de acceso. Una **política de seguridad** es un conjunto de reglas que definen qué puede hacer cada usuario del sistema, en términos de control de acceso.



¿SABÍAS QUE...?

En la mayoría de sistemas operativos de la familia UNIX y derivados, como GNU Linux o Mac OS X, la autenticación de usuarios se basa en el cálculo de resúmenes, *hashes* o *checksums*. Cuando se crea una cuenta de usuario, éste proporciona una contraseña. Guardar esa contraseña en el disco duro de la máquina puede ser peligroso, ya que si se produce un acceso no autorizado podría acceder a la misma. En su lugar, se almacena un *checksum* o *hash* de la contraseña, calculado usando un algoritmo estándar como MD5. Como las funciones *hash* de estos algoritmos no son invertibles, la contraseña no se verá comprometida aunque un usuario no autorizado lea el resumen. Cuando un usuario desea acceder al sistema, introducirá de nuevo su contraseña, que será cifrada usando el mismo algoritmo y comparada con el *checksum* almacenado. Si ambos coinciden, el usuario se habrá autenticado correctamente.



¿SABÍAS QUE...?

Android, el sistema operativo para móviles de Google, ofrece de serie mecanismos de control de acceso para gestionar multitud de operaciones, como:

- Acceso a geolocalización (usando GPS o red móvil).
- Acceso al estado de las conexiones de datos (3G, Wi-Fi, etc.).
- Acceso a Internet.
- Acceso al buzón de voz.
- Acceso al dispositivo de *bluetooth*.
- Lectura y envío de SMS.
- Realización de llamadas telefónicas.
- Escritura en la tarjeta de memoria.
- Borrado de archivos y aplicaciones.
- Activación y desactivación del flash/interna.
- Acceso a la cámara (frontal y/o trasera).

Y muchas otras. Como es lógico, estas opciones estarán o no disponibles dependiendo de las características del aparato donde esté instalado el sistema operativo (si, por ejemplo, el móvil o tablet no tiene cámara, las opciones de cámara no estarán disponibles). Además el sistema permite gestionar de manera independiente este control de acceso para cada aplicación instalada.

Como se ha comentado, la mayoría de mecanismos de control de acceso los proporciona el sistema operativo. No obstante, existen mecanismos de programación avanzados que permiten definir políticas de seguridad de manera interna, y gestionar qué operaciones puede realizar una aplicación y cuales no. Estas políticas estarán supeditadas, obviamente, a los propios mecanismos de control de acceso del sistema operativo. En Java existe una clase estandar, denominada *SecurityManager*, que permite definir y gestionar dichas políticas.

ACTIVIDADES 5.11

- Busca en la documentación oficial de Java información sobre la clase *SecurityManager*. ¿Qué tipos de mecanismos de control de acceso permite implementar?

5 6

CASO PRÁCTICO

Se desea programar un servicio en red de descarga de información con transferencia segura. El servicio estará formado por dos aplicaciones:

- Un servidor seguro, que almacenará archivos de texto.
- Un cliente seguro, que podrá descargar archivos de texto.

El servidor almacenará una serie de archivos, identificados por su nombre. Cuando el cliente se conecte, indicará el nombre del archivo que desea descargar y, si existe, el servidor se lo enviará. Los requisitos de seguridad son los siguientes:

- La transferencia de la información se realizará mediante cifrado simétrico. La clave de sesión la generará el cliente.
- Previamente a la transferencia del fichero, se deberá realizar el intercambio de la clave de sesión. Este intercambio se realizará mediante cifrado asimétrico, usando un par de claves generado en el servidor.
- La integridad de la transferencia se verificará usando firma digital. Los mensajes deberán ir firmados por el servidor.

No se debe implementar el sistema usando *sockets SSL*. Todo el proceso de cifrado asimétrico, simétrico y de firma digital debe ser implementado explícitamente usando las bibliotecas de Java vistas a lo largo de este capítulo. Se puede escoger cualquier otra tecnología de comunicaciones vista (*sockets stream*, *sockets datagram*, RMI, etc.). Cualquier otro aspecto de diseño y programación queda a la libre decisión del desarrollador.

RESUMEN DEL CAPÍTULO

El capítulo ha mostrado la utilización de técnicas de programación segura mediante la criptografía. La utilización de la criptografía permite construir comunicaciones seguras obteniendo unas características de seguridad en la comunicación, como confidencialidad en los mensajes intercambiados, autenticación del emisor de los mensajes, integridad en el envío de datos y no repudio. Para su consecución, se ha definido el concepto de encriptación o cifrado como el mecanismo de protección de la información mediante su modificación utilizando una clave. En función del tipo de clave utilizado, simétrica o asimétrica, se establecen dos modelos diferentes de seguridad: de clave privada o de clave pública. El lenguaje Java ofrece una serie de clases básicas para la gestión de claves y cifrado. Las clases *Key* y *Cipher* son la base de estos mecanismos. La primera sirve para representar claves, y la segunda para ejecutar algoritmos de cifrado.

En el modelo de clave privada, las claves de cifrado y descifrado son la misma. Sin embargo, se plantea el problema de cómo transmitirla para que el emisor y el receptor tengan ambos la misma clave y la necesidad de emplear claves distintas para comunicarse con cada uno de los posibles receptores de la información. Para ver su funcionamiento, se han analizado en detalle los algoritmos de clave simétrica DES y AES. De la misma forma, se ha explicado el concepto de función *hash*, como el mecanismo que resume el contenido de un mensaje en una cantidad de datos fija menor sin clave o utilizando para ello una clave simétrica. La biblioteca del lenguaje Java proporciona mecanismos para realizar procesos de generación de claves (generadores y factorías de claves) y generación de resúmenes (clase *MessageDigest*).

En el modelo de clave pública, las claves de cifrado y descifrado son diferentes (asimétricas) y están relacionadas entre sí de alguna forma. Así, por un lado tenemos la clave privada, que solo el usuario conoce, y la clave pública, publicada para todo el mundo mediante un certificado. En este sentido, un certificado es un documento firmado electrónicamente por alguien en quien se confía, denominado “autoridad de certificación”, que confirma la identidad del usuario vinculándolo con su correspondiente clave pública. Esto permite que se cifice la información con una de las claves siendo descifrable únicamente con la otra. Si lo que se cifra es un resumen del mensaje con la clave privada del usuario, se denomina “firma digital”, ya que permite asegurar la integridad, autenticación y no repudio en el envío del mismo. Para mostrar su funcionamiento, se ha analizado el algoritmo RSA en detalle y se han proporcionado herramientas Java para la creación y gestión de pares de claves para cifrado asimétrico (clase *KeyPair* y *KeyPairGenerator*). Los procesos de firma digital pueden igualmente realizarse en Java de forma sencilla, gracias a la clase *Signature*.

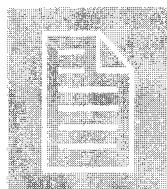
Los sistemas de clave pública son computacionalmente mucho más costosos que los sistemas de clave privada. Por ello habitualmente se suele emplear una combinación mixta de ambos sistemas. Así en los protocolos seguros de comunicación SSL, TLS y SSH se codifican los mensajes a enviar mediante una clave simétrica creada aleatoriamente por el emisor, la cual se envía al receptor mediante sistemas de clave pública. En Java existe un conjunto de clases predefinidas con las que se pueden establecer comunicaciones

usando *sockets stream* que incorporan seguridad mediante el protocolo SSL (sobre TCP). Para este propósito se usan las clases *SSLSocket*, *SSLServerSocket* y *SSLSocketFactory*. También es necesario gestionar los certificados que se usan durante la sesión SSL, para lo que se emplea la herramienta *keytool*.

La seguridad basada en criptografía se complementa con los mecanismos de control de acceso que proporcionan los sistemas operativos. Estos mecanismos permiten controlar aspectos de la interacción de las aplicaciones con el sistema, como el acceso a ficheros, dispositivos hardware, etc.

EJERCICIOS PROPUESTOS

- 1. Cifra el mensaje “HOLA, QUÉ TAL” con un desplazamiento de 6 caracteres, ¿cuál es texto cifrado? ¿Y si lo desplazamos 27? ¿Cómo podríamos atacar un sistema de cifrado tipo César?
- 2. Escribe un programa que lea un fichero de texto, lo cifre usando cifrado DES y lo escriba en un nuevo fichero. Para ello el programa debe generar una clave de usando el objeto generador correspondiente. Una vez el proceso de cifrado haya concluido, el programa debe acceder a la forma transparente de la clave y guardar sus componentes fundamentales en un archivo, de forma que la clave no se pierda y el fichero cifrado pueda ser descifrado en algún momento.
- 3. Escribe un programa que reciba como parámetro el nombre de un algoritmo de resumen, codifique usando dicho algoritmo todo lo que entre por su entrada estándar y devuelva el resultado por su salida estándar. El programa debe funcionar para cualquier algoritmo de resumen soportado por la clase *MessageDigest*.
- 4. Escribe un programa que utilice cifrado asimétrico para cifrar el contenido de un fichero, usando la clave privada. El resultado del cifrado y la clave pública se deben volcar en dos ficheros adicionales de salida. Posteriormente, escribe un segundo programa que cargue dichos ficheros y descifre el archivo cifrado. Utiliza cifrado RSA.
- 5. Escribe un par de programas similares a los del ejercicio anterior, pero que operen con firmas digitales usando el algoritmo DSA. El primer programa generará el par de claves, firmará el archivo de datos y generará dos ficheros de salida con la firma y la clave pública. El segundo programa cargará el fichero de datos, la firma y la clave pública y comprobará que el archivo no ha sido alterado. Verifica el correcto funcionamiento de los programas modificando el contenido del archivo de datos y comprobando si la firma sigue siendo válida.
- 6. Escribe una aplicación cliente/servidor que se comunique mediante *sockets SSL*. El servidor debe recibir mensajes de texto por parte de los clientes, pasarlo a mayúsculas e imprimirlas por pantalla.



TEST DE CONOCIMIENTOS

1 ¿Cuál de las siguientes no es una característica de seguridad?

- (a) Confidencialidad.
- (b) Control de acceso.
- (c) Integridad.
- (d) No repudio.

2 Las funciones *hash*:

- (a) Deben aproximarse a la idea de aleatoriedad máxima posible o ideal.
- (b) Son de sentido único, es decir, se puede producir un documento con sentido que dé lugar a un *hash* especificado previamente.
- (c) Requieren bastante tiempo para producir el cifrado.
- (d) Tanto el texto como el resultado obtenido tienen una longitud fija.

3 ¿Cuál de los siguientes NO es un componente fundamental de una clave?

- (a) Forma codificada.
- (b) Parte pública.
- (c) Algoritmo.
- (d) Formato.

4 Indica cuál de las siguientes afirmaciones sobre el modelo de clave privada es FALSA:

- (a) Las claves de cifrado y descifrado son la misma.
- (b) Para su utilización se requieren claves simétricas.
- (c) El problema que se plantea con su utilización es cómo transmitir la clave para que el emisor y el receptor tengan ambas la misma clave.
- (d) Se tarda bastante más tiempo en cifrar el mensaje si lo comparamos con la criptografía de clave pública.

5 En el modelo de clave pública:

- (a) Las claves de cifrado y descifrado son la misma.
- (b) Las claves de cifrado y descifrado son diferentes y están relacionadas entre sí de algún modo.
- (c) Las claves de cifrado y descifrado son diferentes y no existe relación entre ellas para que un atacante no pueda descubrir una a partir de la otra.
- (d) Se tarda bastante menos tiempo en cifrar el mensaje si lo comparamos con la criptografía de clave privada.

6 ¿Para qué tipos de cifrado se usa la clase *Cipher* de Java?

- (a) Cifrado simétrico y asimétrico.
- (b) Solo cifrado asimétrico.
- (c) Solo cifrado simétrico.
- (d) Ninguna de las anteriores.

7 ¿Qué clase de la biblioteca de Java nos sirve para obtener la versión transparente de un par de claves asimétricas?

- (a) *SecretKeyFactory*.
- (b) *KeyGenerator*.
- (c) *KeyFactory*.
- (d) Ninguna de las anteriores.

8 Para la utilización de la firma digital:

- (a) Dicha firma se obtiene a partir del resumen del mensaje realizado mediante una función *hash* cifrado con la clave pública del usuario.
- (b) La clave pública del usuario se obtiene a partir del certificado.
- (c) El documento firmado, del que proviene el resumen, siempre se debe enviar cifrado mediante un algoritmo de cifrado simétrico.
- (d) El navegador siempre debe conocer la CA que firmó el certificado.

9) ¿Qué clase o clases Java se utilizan para las operaciones de firmado y verificación basadas en firma digital?

- a) *KeyGenerator* para generar las claves y *Signature* para la firma y verificación.
- b) *KeyGenerator* para generar las claves, *Signature* para firmar y *Verification* para verificar.
- c) *Signature* para la generación de claves, firma y verificación.
- d) Ninguna de las anteriores.

10) Indica cuál de las siguientes afirmaciones sobre las autoridades de certificación es FALSA:

- a) Una CA es una entidad de confianza en la cual el usuario confía.
- b) Las CA disponen de sus propios certificados, cuyas claves privadas asociadas son empleadas para firmar digitalmente los certificados que emiten.
- c) El certificado de una CA, puede a su vez venir firmado por otra CA de rango superior de forma jerárquica.
- d) Para confiar en una CA hay que instalar en el navegador su certificado correspondiente en el repositorio de CA de confianza.