CAPÍTULO 5: UTILIZACIÓN DE TÉCNICAS DE PROGRAMACIÓN SEGURA

Programación de Servicios y Procesos

ÍNDICE

- Criptografía
- Características de los servicios de seguridad
- Estructura de un sistema secreto
- Modelo de clave privada
  - Función hash
- Modelo de clave pública
  - Firma digital
- Servicios en red seguros
- Control de acceso

Alberto Sánchez Campos
Jesús Montes Sánchez

Criptografía

CRIPTOGRAFÍA

Arte de escribir con clave secreta o de un modo enigmático

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.

ENCRIPTAR

Acción de proteger la información mediante su modificación utilizando una clave.

Alberto Sánchez Campos
Jesús Montes Sánchez

Ra-Ma®

Criptografía

- Aplicaciones de la Criptografía:
  - Identificación y autentificación. Identificar a un individuo o validar el acceso a un servidor.
  - Certificación. Esquema mediante el cual agentes fiables validan la identidad de agentes desconocidos (como usuarios reales).
  - Seguridad de las comunicaciones. Permite establecer canales seguros para aplicaciones que operan sobre redes que no son seguras.
  - Comercio electrónico. El empleo de canales seguros y mecanismos de identificación posibilita permite que las empresas y los usuarios tengan garantías de que las operaciones no van a ser espiadas ni modificadas, reduciéndose el riesgo de fraudes.

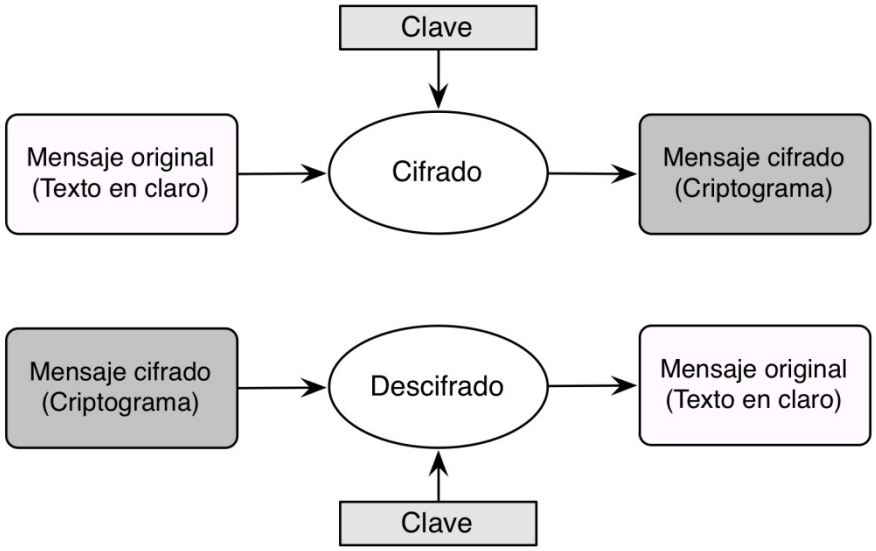
Características de los servicios de seguridad

- **Confidencialidad**: se trata de asegurar que la comunicación solo pueda ser vista por los usuarios autorizados, evitando que ningún otro pueda leer el mensaje. 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.
- **Autenticación**: se trata de asegurar el origen, autoría y propiedad de la información de quien envía el mensaje.
- **No repudio**: se trata de evitar que la persona que envía el mensaje o realiza una acción niegue haberlo hecho ante terceros. Al igual que la característica de confidencialidad, necesita de la autenticación del origen de la información.

Alberto Sánchez Campos
Jesús Montes Sánchez

Estructura de un sistema secreto

- Un sistema secreto actual se encuentra definido por dos funciones:
  - Función de **cifrado.**
  - Función de **descifrado**.
- La **clave** es el parámetro que especifica una transformación concreta dentro de todas las posibles sustituciones que se podrían realizar con la función de cifrado.



Alberto Sánchez Campos
Jesús Montes Sánchez

Criterios de Shannon

- Todos los cifrados se deben construir en base a:
  - **Confusión:** distribuir las propiedades estadísticas (redundancia) de todos los elementos del mensaje sobre el texto cifrado.
  - **Difusión:** dificultar la relación entre la clave y el texto cifrado mediante algoritmos para dificultar su descifrado.

- Características deseables que se deben cumplir las funciones:
  1. 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.

Criterios de Shannon

- 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** ($\mathbf{K_s}$): 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** ($\mathbf{K_p}$ y $\mathbf{K_c}$): las claves de cifrado y descifrado son diferentes y están relacionadas entre sí de algún modo. Dan lugar al **modelo de clave pública**.

Alberto Sánchez Campos
Jesús Montes Sánchez

Tipos de claves

- La seguridad depende tanto del algoritmo de cifrado como del nivel de secreto que se le dé a la clave.
- Tipos de claves:
  - Claves **simétricas** (**K_s**)
    - Las claves de cifrado y descifrado son la misma.
    - El problema 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**.

Alberto Sánchez Campos
Jesús Montes Sánchez

Programación de cifrado

- El lenguaje Java proporciona herramientas para el manejo de claves y mecanismos de cifrado.
- Los componentes básicos para el cifrado en Java son:
  - La interfaz *Key*.
  - La Interfaz *KeySpec*.
  - La clase *Cipher*.

Interfaz *Key*

- Representa una clave, que se puede emplear para operaciones de cifrado y descifrado.
- En Java toda clase que represente una clave debe implementar esta interfaz y tener tres partes fundamentales:
  - El algoritmo: El nombre de la función de cifrado y descifrado.
  - La forma codificada: Es una representación de la clave.
  - El formato: Es la forma en la que se encuentra codificada la clave.

Alberto Sánchez Campos
Jesús Montes Sánchez

Interfaz *KeySpec*

- Las clases que implementan la interfaz Key almacenan las claves de forma **opaca**: No se proporciona acceso a sus componentes internos.
- Por el contrario, la interfaz *KeySpec*, sirve para representar claves en formato **transparente** (sus componentes internos son accesibles), lo que se emplea para poder distribuirlas.

Generadores y factorías de claves

- Generadores de claves: Se utilizan para crear objetos de clases que implementan la interfaz *Key*, es decir, para crear claves nuevas.
- Factorías de claves: Se emplean para obtener versiones transparentes (*KeySpec*) a partir de de claves opacas (*Key*) y viceversa.

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.

Ejemplo de cifrado

```
import javax.crypto.*;

public class CipherExample {

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

         KeyGenerator keygen = KeyGenerator.getInstance("DES");
         SecretKey key = keygen.generateKey();
         Cipher desCipher = Cipher.getInstance("DES");
         desCipher.init(Cipher.ENCRYPT_MODE, key);
         String mensaje = "Mensaje de prueba";
         String mensajeCifrado = new String(desCipher.doFinal(mensaje.getBytes()));

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

Modelo de clave privada

- Tanto el emisor como el receptor del mensaje utilizan la misma clave K_s.
- Presenta un buen rendimiento
- Problemas:
  - Cómo el emisor y el receptor obtienen la misma clave
  - Necesidad de emplear claves distintas para comunicarse con cada uno de los posibles receptores de la información.
  - 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.

Alberto Sánchez Campos
Jesús Montes Sánchez

Algoritmos de cifrado simétrico

- DES
  Estándar por el Gobierno de los EEUU para comunicaciones desde 1976
  Cifra un texto de una longitud fija en otro texto cifrado de la misma longitud.
  Tres fases:
    1. Permutación inicial: para dotar de confusión y difusión al algoritmo.
    2. Sustitución mediante cajas-S previamente definidas que comprimen la información, la permutan y la sustituyen.
    3. Permutación final inversa a la inicial.

  Clave de 56 bits para modificar las transformaciones realizadas..

- AES
  Sustituto del DES
  Estándar de cifrado por el Gobierno de EEUU desde el año 2000.
  Su desarrollo se llevó a cabo de forma pública y abierta
  Sistema de cifrado por bloques
  Maneja longitudes de clave y de bloque variables, entre 128 y 256 bits.
  Rápido y fácil de implementar.

Alberto Sánchez Campos
Jesús Montes Sánchez

Programación de cifrado simétrico

- La interfaz *SecretKey*, que implementa a su vez la interfaz *Key,* representa claves simétricas (opacas).
- La clase *KeyGenerator* se usa para generar claves simétricas.
- 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.

Ejemplo de cifrado y descifrado simétrico

```
import javax.crypto.*;

public class CipherExample2 {

   public static void main(String[] args) {
      try {
         KeyGenerator keygen = KeyGenerator.getInstance("AES");
         SecretKey key = keygen.generateKey();
         Cipher aesCipher = Cipher.getInstance("AES");
         aesCipher.init(Cipher.ENCRYPT_MODE, key);
         String mensaje = "Mensaje que se cifrará con AES";
         String mensajeCifrado = new String(aesCipher.doFinal(mensaje.getBytes()));

         aesCipher.init(Cipher.DECRYPT_MODE, key);
         String mensajeDescifrado = new String(aesCipher.doFinal(mensajeCifrado.getBytes()));
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

Alberto Sánchez Campos
Jesús Montes Sánchez

Función hash

- Una **función *hash*** es un algoritmo matemático que resume el contenido de un mensaje en una cantidad de información fija menor.
- Requisitos:
  - La descripción de la función de descifrado debe ser pública.
  - El texto en claro puede tener una longitud arbitraria. Sin embargo, el resultado debe tener una longitud fija (resumen).
  - Resistente a **colisiones.**
  - **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*.
  - Aun cuando se conozca un gran número de pares (texto en claro, resumen) debe ser difícil determinar la clave.
  - Rápida de calcular.

Alberto Sánchez Campos
Jesús Montes Sánchez

Función hash

- Una **función *hash*** es un algoritmo matemático que resume el contenido de un mensaje en una cantidad de información fija menor.
- Sirven para proporcionar pruebas de la **integridad** de la transferencia de información. Denominadas MAC (en inglés, *Message Authentication Codes*).
- Requisitos:
  - La descripción de la función de descifrado debe ser pública.
  - El texto en claro puede tener una longitud arbitraria. Sin embargo, el resultado debe tener una longitud fija (resumen).
  - Resistente a **colisiones.**
  - **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*.
  - Aun cuando se conozca un gran número de pares (texto en claro, resumen) debe ser difícil determinar la clave.
  - Rápida de calcular.

Alberto Sánchez Campos
Jesús Montes Sánchez

Programación con funciones hash

- En Java se emplea la clase *MessageDigest* (*java.security.MessageDigest*).
- Para generar un hash se siguen los siguientes pasos:
  1. Obtención de una instancia de la clase *MessageDigest*.
  2. Introducción de datos en la instancia.
  3. Cómputo del resumen.

Ejemplo de cálculo de resumen

```
import java.security.MessageDigest;

public class MessageDigestExample {
   public static void main(String[] args) {
      try {
         MessageDigest md = MessageDigest.getInstance("MD5");
         byte[] c1 = "Primera cadena".getBytes();
         byte[] c2 = "Segunda cadena".getBytes();
         byte[] c3 = "Tercera cadena".getBytes();
         md.update(c1);
         md.update(c2);
         md.update(c3);
         byte[] resumen = md.digest();
         md.reset();
         byte[] c4 = "Cuarta cadena".getBytes();
         md.update(c4);
         resumen = md.digest();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

Modelo de clave pública

- Diseñado para evitar el problema de la distribución de las claves entre emisor y receptor.
- Dos claves diferentes:
  - **Clave privada** K_c: que solo el usuario conoce.
  - **Clave pública** K_p: publicada para todo el mundo que lo desee.
  - Relacionadas entre sí por una función de sentido único. Debe ser muy difícil calcular una clave a partir de la otra
- Sistema asimétrico, 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.
  - La única forma de desencriptarlo es utilizando la clave privada del receptor

Algoritmo RSA

- Debe su nombre a sus tres inventores: Ronald Rivest, Adi Shamir y Leonard Adleman, del MIT.
- Algoritmo asimétrico de cifrado por bloques, que utiliza:
  - Una clave pública, conocida por todos.
  - Una clave otra privada, guardada en secreto por su propietario.
  - La relación de claves se basa en el producto de dos números primos muy grandes elegidos al azar. No hay maneras rápidas conocidas de factorizar un número grande en sus factores primos.

Alberto Sánchez Campos
Jesús Montes Sánchez

Algoritmo RSA

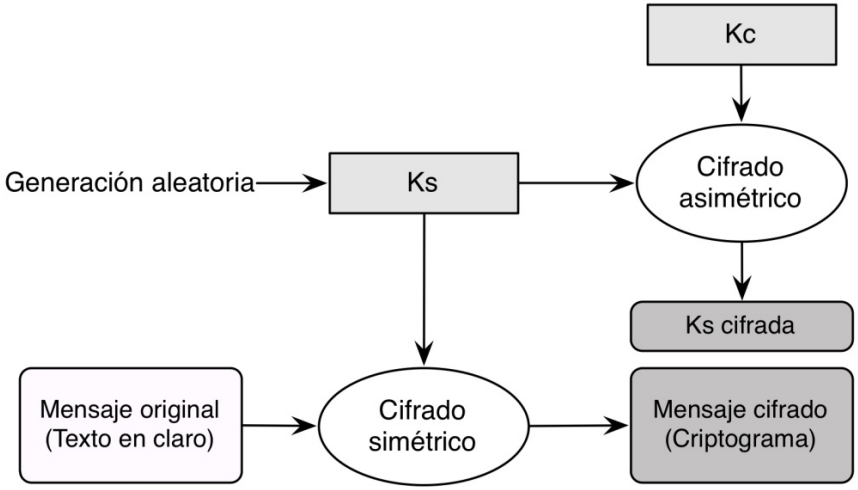
- Proceso:
  - Se eligen al azar y en secreto dos números primos *p* y *q* lo suficientemente grandes y se calcula:
    - $n = p \cdot q$
    - $\Phi(n) = (p-1) \cdot (q-1)$
  - Se eligen dos exponentes *e* y *d* tales que:
    - *e* y $\Phi(n)$ no tengan números en común que los dividan, es decir $mcd(e, \Phi(n)) = 1$
    - $e \cdot d = 1 \bmod \Phi(n)$, es decir, *e* y *d* son inversos multiplicativos en $\Phi(n)$
  - La clave pública es el par (*e*,*n*).
  - La clave privada (*d*,*n*) o (*p*,*q*).
  - Para cifrar M lo único que se debe hacer es $M^e \bmod n = M'$, siendo el descifrado $M'^d \bmod n$. Como se puede observar, conmuta.

Alberto Sánchez Campos
Jesús Montes Sánchez

Ra-Ma®

Modelo híbrido

- Los sistemas de clave pública son computacionalmente mucho más costosos que los sistemas de clave privada.
- Se emplea una combinación de ambos sistemas
  - En el intercambio de información por rapidez se codifican los mensajes mediante algoritmos simétricos y una única clave K_s.
  - Dicha clave se suele crear aleatoriamente por el emisor.
  - Se utilizan sistemas de clave pública para codificar y enviar la clave simétrica K_s
  - El resto de mensajes intercambiados durante esa sesión cifrados con la clave K_s



Alberto Sánchez Campos
Jesús Montes Sánchez

Ra-Ma®

Programación de cifrado asimétrico

- La interfaz *PublicKey* (*java.security.PublicKey*). Implementa a su vez la interfaz *Key*, y se emplea para representar claves públicas.
- La interfaz *PrivateKey* (*java.security.PrivateKey*). Implementa a su vez la interfaz *Key*, y se emplea para representar claves privadas.
- La clase *KeyPair* (*java.security.KeyPair*). Los objetos de esta clase se utilizan para almacenar pares de claves.
- La clase *KeyPairGenerator* (*java.security.KeyPairGenerator*). Se utiliza para generar pares de claves.
- La clase *KeyFactory* (*java.security.KeyFactory*). Se usa como factoría de claves.

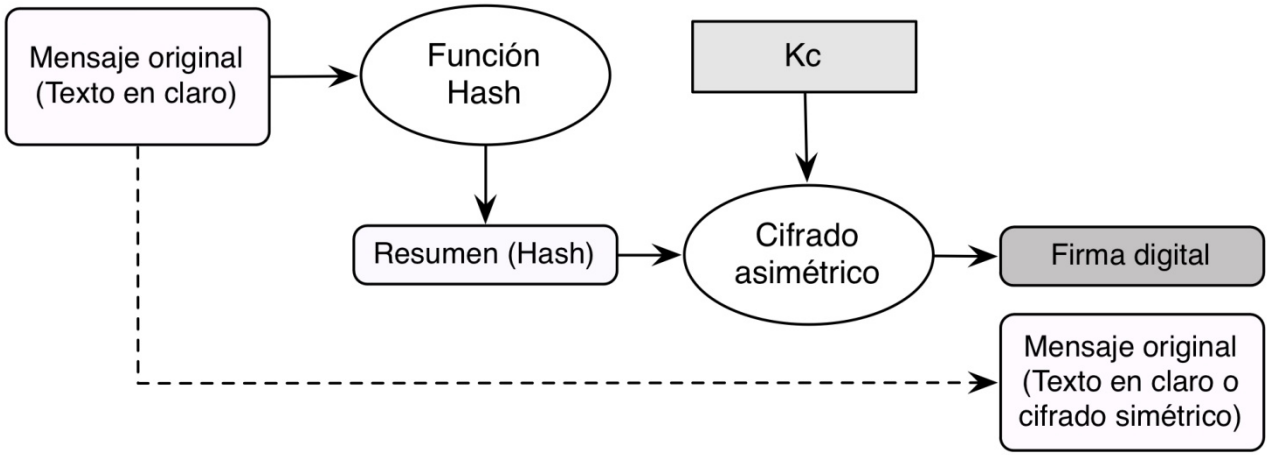
Ejemplo de cifrado asimétrico

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

public class AsymetricExample {

   public static void main(String[] args) {
      try {
         KeyPairGenerator keygen = KeyPairGenerator.getInstance("RSA");
         KeyPair keypair = keygen.generateKeyPair();
         Cipher rsaCipher = Cipher.getInstance("RSA");
         rsaCipher.init(Cipher.ENCRYPT_MODE, keypair.getPrivate());
         String mensaje = "Mensaje de prueba del cifrado asimétrico";
         String mensajeCifrado = new String(rsaCipher.doFinal(mensaje.getBytes()));
         rsaCipher.init(Cipher.DECRYPT_MODE, keypair.getPublic());
         String mensajeDescifrado = new String(rsaCipher.doFinal(mensajeCifrado.getBytes()));
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

Alberto Sánchez Campos
Jesús Montes Sánchez

Firma digital

- Asocia la identidad del emisor al mensaje que se transmite a la vez que comprueba la integridad del mensaje, mediante:
  1. Aplicación de una función *hash*.
  2. Empleo de 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**



Alberto Sánchez Campos
Jesús Montes Sánchez

Programación de firmado digital

- La biblioteca de Java incluye la clase *Signagture* (*java.security.Signature*) para realizar operaciones de firmado digital.
- Requiere de un par de claves.
- Se usa de forma similar al cálculo de funciones hash.
- Sirve además para realizar verificación de firma.

Ejemplo de firmado digital

```
import java.security.*;

public class SignatureExample {
   public static void main(String[] args) {
      try {
         KeyPairGenerator keygen = KeyPairGenerator.getInstance("DSA");
         KeyPair keypair = keygen.generateKeyPair();
         Signature signature = Signature.getInstance("DSA");
         signature.initSign(keypair.getPrivate());
         String mensaje = "Mensaje para firmar";
         signature.update(mensaje.getBytes());
         byte[] firma = signature.sign();

         signature.initVerify(keypair.getPublic());
         signature.update(mensaje.getBytes());
         if (signature.verify(firma))
            System.out.println("El mensaje es auténtico :-)");
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

Alberto Sánchez Campos
Jesús Montes Sánchez

Certificado

- El modelo de calve pública se basa en el conocimiento de las claves públicas K_p de los diferentes usuarios.
- Un **certificado** digital o certificado electrónico es un documento firmado electrónicamente por alguien en quien se confía, que confirma la identidad vinculándolo con su correspondiente clave pública K_p.
- Permite a los usuarios y servicios identificarse, estando confirmado por una **autoridad de certificación**.
- Un certificado electrónico sirve, por tanto, para:
  - Firmar digitalmente para garantizar la integridad de los datos trasmitidos y su procedencia, y autenticar la identidad del usuario de forma electrónica ante terceros.
  - Cifrar datos para que solo el destinatario del documento pueda acceder a su contenido.

Alberto Sánchez Campos
Jesús Montes Sánchez

Servicios en red seguros

- 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.
- El proceso es seguro gracias a un modelo de clave mixta, en el que se emplea clave asimétrica para establecer la sesión, y cifrado simétrico para las comunicaciones posteriores.
- En Internet la mayoría de estas comunicaciones hace uso de los protocolos SSL (*Secure Sockets Layer*) y su sucesor TLS (*Transport Layer Security*).

Alberto Sánchez Campos
Jesús Montes Sánchez

SSL y TLS

- SSL y TLS ofrecen una interfaz de programación basada en *sockets stream*, muy similar a la vista en el Capítulo 3.
- 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.

SSL y TLS

- SSL y TLS garantizan el establecimiento y mantenimiento de una **sesión segura**:
  - La identidad del elemento servidor está garantizada (**autenticación**). Esto se consigue gracias a un certificado de servidor y al cifrado asimétrico
  - La privacidad de las comunicaciones está garantizada (**confidencialidad**). Esto se consigue gracias a una clave de sesión, simétrica.
  - Los mensajes que se intercambian no pueden ser alterados de ninguna manera (**integridad**). Esto se consigue mediante cálculo de resúmenes.

Programación con sockets seguros

- *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*.

Autoridades de certificación

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

Alberto Sánchez Campos
Jesús Montes Sánchez

Control de acceso

- 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.

Alberto Sánchez Campos
Jesús Montes Sánchez

Control de acceso

- Ejemplos típicos de aspectos que se gestionan usando control de acceso son:
  - Operaciones sobre ficheros y directorios.
  - Acceso a dispositivos hardware.
  - 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.