Universidad Nacional Autónoma de México

Facultad de Ingeniería

Criptografía

Protocolos Seguros con criptografía asimétrica y firmas digitales.

Integrantes:

- Contreras Mejía David
- López Castillón Jonathan Jhosua.
- Sánchez Pérez Omar Alejandro

Profesor:

- Padron Godinez Alejandro

Número de Equipo:

- Equipo 7

Grupo:

- Grupo 1

Fecha de Entrega:

- 16/06/2023

Introducción

En diversas organizaciones, tanto públicas como comerciales, el uso de Internet se ha convertido en un foro principal para hacer negocios, sirviendo como medio principal para la publicidad, el marketing, las ventas y la atención al cliente. Ha permitido que las empresas crezcan y a las grandes corporaciones, expandir su dominio.

Junto a las oportunidades que ofrece Internet se encuentran riesgos de seguridad importantes. Los servidores Web pueden ser sustituidos y a veces, las páginas Web han sido desconfiguradas. Los datos privados de los consumidores podrían ser revelados. Las transacciones financieras pueden ser falsificadas. Los cortafuegos de las empresas pueden ser infringidos y las redes de la empresa saboteadas. Todos estos escenarios conllevan a un uso seguro de Internet.

La comunicación segura dentro de una red debe cumplir con las siguientes características: confidencialidad, autenticación, integridad del mensaje, disponibilidad y control de acceso. Las 3 primeras características se han considerado componentes claves de una red segura, sin embargo, se han añadido en las últimas décadas la necesidad de mantener la red operando. El concepto de firma digital fue introducido por Diffie y Hellman en 1976, siendo un bloque de caracteres que acompaña a un documento acreditando quién es su autor (autenticación) y que no ha existido ninguna manipulación posterior de los datos (integridad).

El proceso de firmado digital se inicia cuando el autor de un documento utiliza su clave secreta dentro del esquema de cifrado asimétrico, a la que sólo él tiene acceso, esto impide que pueda negar su autoría (revocación o no repudio). De esta forma el autor es vinculado al documento de la firma. El software del autor aplica un algoritmo hash sobre el texto a firmar, obteniendo un extracto de longitud fija y absolutamente específico del mensaje. Un cambio mínimo en el mensaje dará lugar a una cadena hash distinta. El extracto tiene una longitud de 128 a 160 bits, dependiendo del algoritmo utilizado, entre los que se encuentran: MD5 o SHA-1. El algoritmo más utilizado en el proceso de encriptación asimétrica es el RSA.

La validez de la firma es probada por cualquier persona que disponga de la clave pública del autor.

Objetivos

- Realizar una aplicación que permita el proceso de firma de un documento a través de la infraestructura de claves asimétricas, en el lenguaje orientado a objetos Java.

Metodología

Se hará uso del lenguaje de programación “Java” para la realización de este proyecto. La aplicación Java a realizar se compone de dos programas, el primero hará el firmado del documento y el segundo verificará la firma de dicho documento.

El programa que realiza el primer procedimiento debe cumplir con los siguientes requerimientos:

1.- Solicitar mediante la entrada estándar, el nombre del archivo que contiene la información a

2.- firmar (Archivo A).

3.- La salida del programa que firma el documento, deben ser dos archivos, el primero contendrá

4.- La clave pública generada y el segundo la firma del documento (Archivo B, Archivo C).

5.- El programa que recibe el Archivo A debe generar internamente el par de claves.

El programa que realiza el segundo procedimiento debe cumplir con los siguientes requerimientos:

1.- Solicitar mediante la entrada estándar, el nombre del archivo que contiene la clave pública

2.- Generada en el primer punto, además del nombre del archivo que contiene la información

3.- Firmada (Archivo B, Archivo C).

4.- Indicar mediante una cadena si la firma es correcta

Resultados y Ejercicios

Se programó un algoritmo que permite crear una firma digital para un archivo con información dentro de este. Su implementación se realizó usando el lenguaje de programación Java, se desarrollaron dos clases, la primera es **GeneraFirma.java** y la segunda es **VerificaFirma.java**. A continuación, se describe brevemente el funcionamiento de las clases y se muestran los códigos.

GeneraFirma.java

El código proporcionado es un programa en Java que genera una firma digital utilizando el algoritmo de cifrado RSA. El código fue implementado de la siguiente forma:

1. Se importan las bibliotecas necesarias para el programa, que incluyen java.io para operaciones de entrada/salida y java.security para funcionalidades de seguridad.

2. Se declara la clase principal llamada "GeneraFirma".

3. El método principal "main" se inicia.

4. Se imprime un mensaje de título en la consola.

5. Se verifica si se proporcionó un argumento al programa. Si no se proporciona, se muestra un mensaje que indica cómo ejecutar el programa correctamente.

6. Si se proporciona un argumento, se intenta generar una clave privada y pública utilizando el algoritmo de cifrado RSA.

7. Se crea una instancia de KeyPairGenerator con el algoritmo RSA y se inicializa con un tamaño de clave de 1024 bits y una fuente aleatoria segura.

8. Se generan un par de claves (privada y pública) utilizando el generador de claves.

9. Se crea una instancia de Signature con el algoritmo "MD5withRSA" para realizar la firma digital.

10. Se inicia la firma con la clave privada.

11. Se abre el archivo especificado en el argumento y se lee su contenido en un búfer. Luego, se envía el contenido al objeto de firma.

12. Se cierra el flujo de entrada del archivo.

13. Se genera la firma digital de los datos utilizando el método "sign()" de la instancia de firma.

14. Se crea un archivo llamado "firma.txt" y se escribe la firma digital en él.

15. Se cierra el flujo de salida del archivo.

16. Se imprime un mensaje de éxito en la consola.

17. Se crea un archivo llamado "clavepublica.txt" y se escribe la clave pública en él.

18. Se cierra el flujo de salida del archivo.

19. Se imprime un mensaje de éxito en la consola.

20. Si se produce alguna excepción durante la ejecución del programa, se captura y se muestra un mensaje de error en la consola.

En resumen, este programa genera una firma digital utilizando el algoritmo RSA y guarda la firma en un archivo llamado "firma.txt". También guarda la clave pública utilizada en la firma en un archivo llamado "clavepublica.txt".

Implementación

```
/*
* Universidad Nacional Autónoma de México
* Facultad de Ingeniería
*
* Generación de las claves (pública y privada) y de la firma digital
*/

// Se importan las bibliotecas
import java.io.*;
import java.security.*;

// Clase general
public class GeneraFirma {
    public static void main(String[] args) {
        // Título
        System.out.println("\t\tCreando la firma digital...");

        if(args.length != 1) {
            System.out.println("Para ejecutar el programa se deben
seguir los siguientes pasos: ");
            System.out.println("\t1) Se debe generar el archivo class
--> 'javac GeneraFirma.java'");
            System.out.println("\t2) Se ejecuta el programa --> 'java
GeneraFirma <archivo>'");
    } else try {
        // Se determina una clave privada
        KeyPairGenerator genClave =
KeyPairGenerator.getInstance("RSA");

        // Se inicia el objeto genClave a través del método
initialize, el cual tiene dos argumentos:
        // a. Tamaño de la clave es de 1024 bits.
        // b. Fuente aleatorio.
```

```
        SecureRandom aleatorio =
SecureRandom.getInstance("SHA1PRNG","SUN");
        genClave.initialize(1024, aleatorio);

        // Se genera el par de claves
        // PrivateKey --> Clave privada
        // PublicKey --> Clave pública
        KeyPair pardeClaves = genClave.generateKeyPair();
        PrivateKey privada = pardeClaves.getPrivate();
        PublicKey publica = pardeClaves.getPublic();

        // Se crea la firma digital
        Signature firma = Signature.getInstance("MD5withRSA");
        firma.initSign(privada);

        // Se abre el archivo, guarda el contenido en un buffer y
posteriormente se lo hace llegar
        // al objeto firma
        FileInputStream archivo = new FileInputStream(args[0]);
        BufferedInputStream bufferIn = new
BufferedInputStream(archivo);
        byte[] buffer = new byte[1024];
        int longitud;

        while(bufferIn.available() != 0) {
            longitud = bufferIn.read(buffer);
            firma.update(buffer, 0, longitud);
        }
        bufferIn.close();

        // Generando la firma digital de los datos
        byte[] firmaReal = firma.sign();

        // Se guardan los datos firmados en un archivo
        FileOutputStream archivoFirma = new
FileOutputStream("firma.txt");
        archivoFirma.write(firmaReal);
        archivoFirma.close();
        System.out.println("Se ha creado el archivo 'firma.txt'");

        // Se guarda la clave pública en un archivo
        FileOutputStream clavePublica = new
```

```
FileOutputStream("clavepublica.txt");
        clavePublica.write(publica.getEncoded());
        clavePublica.close();
        System.out.println("Se ha creado el archivo
'clavepublica.txt'");
    } catch(Exception error) {
        System.out.println("Hubo un error al ejecutar el programa: "
+ error);
    }
}
```

VerificaFirma.java

El código proporcionado es un programa en Java que realiza la verificación de una firma digital utilizando el algoritmo de cifrado RSA. Aquí hay una descripción paso a paso del código:

1. Se incluyen comentarios iniciales que indican la universidad y la facultad a la que pertenece el código, así como una descripción general del propósito del programa.

2. Se importan las bibliotecas necesarias para el programa, que incluyen java.io para operaciones de entrada/salida, java.security para funcionalidades de seguridad y java.security.spec para especificaciones de claves.

3. Se declara la clase principal llamada "VerificaFirma".

4. El método principal "main" se inicia.

5. Se imprime un mensaje de título en la consola.

6. Se verifica si se proporcionaron los argumentos necesarios para el programa. Si no se proporcionan, se muestra un mensaje que indica cómo ejecutar el programa correctamente.

7. Si se proporcionan los argumentos necesarios, se intenta verificar la firma digital.

8. Se lee el archivo de clave pública especificado y se guarda su contenido en un arreglo de bytes.

9. Se crea una instancia de X509EncodedKeySpec utilizando los bytes de la clave pública leída.

10. Se crea una instancia de KeyFactory con el algoritmo RSA.

11. Se genera un objeto de tipo PublicKey utilizando la fábrica de claves y la especificación de la clave pública.

12. Se lee el archivo firmado especificado y se guarda su contenido en un arreglo de bytes.

13. Se crea una instancia de Signature con el algoritmo "MD5withRSA" para realizar la verificación de la firma.

14. Se inicia la firma con la clave pública.

15. Se abren los datos a verificar desde el archivo especificado y se leen en un búfer. Luego, se envía el contenido al objeto de firma.

16. Se cierra el flujo de entrada de datos.

17. Se realiza la verificación de la firma utilizando el método "verify()" de la instancia de firma y se guarda el resultado en una variable booleana.

18. Se imprime el resultado de la verificación en la consola.

19. Si se produce alguna excepción durante la ejecución del programa, se captura y se muestra un mensaje de error en la consola.

En resumen, este programa verifica la firma digital de un conjunto de datos utilizando el algoritmo RSA y la clave pública especificada en el archivo "clavepublica.txt". Los datos a verificar se leen desde el archivo "datosaFirmar.txt". El resultado de la verificación se muestra en la consola.

Implementación

```
/*
* Universidad Nacional Autónoma de México
* Facultad de Ingeniería
*
* Se realiza la verificación de la firma digital
*/

// Se importan las bibliotecas
import java.io.*;
import java.security.*;
import java.security.spec.*;

public class VerificaFirma {
```

```
    public static void main(String[] args) {
        System.out.println("\t\tVerificación de la firma
digital...");

        if(args.length != 3) {
            System.out.println("Para ejecutar el programa se deben
seguir los siguientes pasos: ");
            System.out.println("\t1) Se debe generar el archivo class
--> 'javac VerificaFirma.java'");
            System.out.println("\t2) Se ejecuta el programa --> 'java
VerificaFirma clavepublica.txt datosaFirmar.txt'");
        } else try {
            // Se importan los bytes codificados de la clave pública
del archivo que lo contiene y
            // y los convierte en un objeto de tipo PublicKey
            FileInputStream clavePublica = new
FileInputStream(args[0]);
            byte[] clave = new byte[clavePublica.available()];
            clavePublica.read(clave);
            clavePublica.close();

            // Se obtiene el valor de la clave pública
            X509EncodedKeySpec publicKeySpec = new
X509EncodedKeySpec(clave);
            // Se requiere para realizar la conversión
            KeyFactory keyFactory = KeyFactory.getInstance("RSA");

            // Se genera un objeto PublicKey
            PublicKey clvPublica =
keyFactory.generatePublic(publicKeySpec);

            // Se introducen los bytes firmados desde el archivo
especificado
            FileInputStream archivoFirmado = new
FileInputStream(args[1]);
            byte[] firmaVerificada = new
byte[archivoFirmado.available()];
            archivoFirmado.read(firmaVerificada);
archivoFirmado.close();

            // Se define los algoritmos usados en este proceso
            Signature firma = Signature.getInstance("MD5withRSA");
```

```
            firma.initVerify(clvPublica);

            // Se suministran los datos para los cuales se generó la
lista
            FileInputStream datos = new FileInputStream(args[2]);
            BufferedInputStream bufferIn = new
BufferedInputStream(datos);
             byte[] buffer = new byte[1024];
             int longitud;

             while(bufferIn.available() != 0) {
                 longitud = bufferIn.read(buffer);
                 firma.update(buffer, 0, longitud);
             }
             bufferIn.close();

             // Se reporta el resultado
             boolean verifica = firma.verify(firmaVerificada);
             System.out.println("\tVerificación de la firma: " +
verifica);

        } catch(Exception error) {
            System.err.println("Hubo un error en la verificación: " +
error);
        }
    }
}
```

Funcionamiento

A continuación, se muestra el funcionamiento del programa usando la terminal de Debian 11 con los comandos proporcionados por la máquina virtual de Java.

Se compilan las clases usando el comando **javac**:

```
$ javac GeneraFirma.java
$ javac VerificaFirma.java
```

Primero se corre la clase **GeneraFirma** usando el comando java, tal como se muestra a continuación:

```
alejandro@SM-J410G:~/Documentos/Criptografia/Criptografia/Proyecto$ java GeneraFirma datosaFirmar.txt
                Creando la firma digital...
Se ha creado el archivo 'firma.txt'
Se ha creado el archivo 'clavepublica.txt'
```

Al ejecutar el programa, muestra en pantalla tres mensajes indicando que se ha creado la firma digital para el archivo datosaFirmar.txt, este proceso crea dos archivos más, uno con la firma digital y el otro con la clave pública.

Para comprobar que la firma digital se ha efectuado correctamente, se ejecuta el programa **VerificaFirma**, tal como se muestra a continuación:

```
alejandro@SM-J410G:~/Documentos/Criptografia/Criptografia/Proyecto$ java VerificaFirma clavepublica.txt firma.txt datosaFirmar.txt
              Verificación de la firma digital...
       Verificación de la firma: true
```

Si la firma digital se ha creado correctamente, este programa devolverá un **true**, de lo contrario devolverá un **false**.

Discusión

Podemos decir que la criptografía asimétrica tiene una gran importancia en todos los temas relacionados a seguridad, puesto que sin la criptografía, toda la información sería más vulnerable o prácticamente las comunicaciones inseguras, y una aplicación de ella es en las comunicaciones electrónicas, específicamente en el envío y recepción de documentos o mensajes.

Secure Socket Layer (SSL)

El Secure Socket Layer (SSL) es un protocolo de seguridad que fue desarrollado originalmente por Netscape en la década de 1990. utiliza criptografía asimétrica (o de clave pública) y simétrica para lograr sus objetivos de seguridad. El protocolo SSL/TLS se utiliza principalmente para cifrar las comunicaciones entre un cliente (como un navegador web) y un servidor web, asegurando la confidencialidad y la integridad de los datos transmitidos. Pasos del funcionamiento:

1. Establecimiento de una conexión segura:

- El cliente envía una solicitud al servidor para iniciar una conexión segura mediante SSL/TLS.

- El servidor responde con su certificado digital, que contiene su clave pública y está firmado por una entidad de certificación confiable.

- El cliente verifica la autenticidad del certificado del servidor utilizando una lista de autoridades de certificación confiables y comprueba si el nombre de dominio en el certificado coincide con el servidor al que está intentando acceder.

- Si el cliente confía en el certificado del servidor, genera una clave de sesión aleatoria y la cifra con la clave pública del servidor.

- El cliente envía la clave de sesión cifrada al servidor.

2. Apretón de manos (Handshake):

- El servidor utiliza su clave privada para descifrar la clave de sesión enviada por el cliente.

- A partir de este punto, el cliente y el servidor generan claves de cifrado y descifrado simétricas a partir de la clave de sesión.

- A través del apretón de manos, el cliente y el servidor acuerdan un conjunto de algoritmos criptográficos que se utilizarán para cifrar y descifrar los datos transmitidos.

3. Transmisión segura de datos:

- Una vez establecida la conexión segura, el cliente y el servidor pueden intercambiar datos cifrados utilizando las claves simétricas generadas durante el apretón de manos.

- Los datos se cifran en el cliente y se descifran en el servidor, y viceversa, asegurando la confidencialidad de la información transmitida.

4. Verificación de integridad:

- Durante la transmisión de datos, SSL/TLS también se encarga de verificar la integridad de los datos para detectar cualquier modificación no autorizada.

- Se utilizan funciones hash y firmas digitales para generar y verificar resúmenes de los datos transmitidos.

- El servidor puede firmar digitalmente los datos que envía al cliente, y el cliente puede verificar la autenticidad de la firma utilizando la clave pública del servidor.

Ejemplo de funcionamiento de SSL/TLS:

Imaginemos que un usuario está navegando por un sitio web seguro (por ejemplo, https://www.ejemplo.com):

1. El usuario abre su navegador web e ingresa la URL del sitio web seguro.

2. El navegador envía una solicitud al servidor de www.ejemplo.com para establecer una conexión segura.

3. El servidor responde con su certificado digital, que incluye su clave pública y está firmado por una autoridad de certificación confiable.

4. El navegador verifica la autenticidad del certificado, comprueba si el nombre de dominio coincide y confía en la autoridad de certificación.

5. Si la verificación es exitosa, el navegador genera una clave de sesión aleatoria y la cifra con la clave pública del servidor.

6. El navegador envía la clave de sesión cifrada al servidor.

7. El servidor utiliza su clave privada para descifrar la clave de sesión.

8. El cliente y el servidor genera claves de cifrado y descifrado simétricas a partir de la clave de sesión.

9. A partir de este punto, el cliente y el servidor intercambian datos cifrados utilizando las claves simétricas generadas.

10. Los datos se cifran en el cliente y se descifran en el servidor, y viceversa, asegurando la confidencialidad de la información transmitida.

11. Durante la transmisión, se verifica la integridad de los datos y las firmas digitales para detectar cualquier modificación no autorizada.

12. La conexión segura se mantiene hasta que el usuario cierra la sesión o navega a otro sitio.

El Protocolo de Seguridad de la Capa de Transporte (TLS) es la sucesión del Secure Socket Layer (SSL) y ha reemplazado gradualmente a SSL en la mayoría de las implementaciones. TLS presenta varias mejoras y actualizaciones en comparación con SSL. A continuación, se detallan algunas de las mejoras clave del TLS en relación con SSL y las razones por las que se ha preferido su adopción:

1. Mejoras en seguridad: TLS ha mejorado las debilidades y vulnerabilidades identificadas en versiones anteriores de SSL. Ha corregido problemas de seguridad, como ataques de intercambio de claves, ataques de hombre en el medio y vulnerabilidades en los algoritmos de cifrado. TLS incluye protocolos y cifrados más seguros, y se actualiza periódicamente para abordar nuevas amenazas.

2. Flexibilidad en los algoritmos de cifrado: TLS permite una mayor flexibilidad en la elección de los algoritmos de cifrado y los métodos de autenticación. SSL estaba limitado a un conjunto de algoritmos y métodos específicos, mientras que TLS admitía una variedad más amplia de opciones criptográficas, lo que permite una mayor adaptabilidad y seguridad en función de las necesidades y las mejores prácticas actuales.

3. Compatibilidad con versiones anteriores: Aunque TLS es una evolución de SSL, está diseñado para ser compatible con versiones anteriores de SSL. Esto significa que los clientes y servidores que utilizan versiones anteriores de SSL aún pueden comunicarse con sistemas que utilizan TLS. Esta compatibilidad permite una transición más suave y un uso progresivo de TLS sin afectar la interoperabilidad con sistemas existentes.

5. Amplia adopción y estandarización: TLS ha ganado una amplia adopción en la industria y ha sido estandarizado por organizaciones como el Internet Engineering Task Force (IETF). Esto ha llevado a una mayor confianza y soporte por parte de proveedores de software, navegadores web y otros sistemas, lo que ha fomentado su adopción generalizada.

Cuando un sitio web utiliza HTTPS, significa que la comunicación entre el cliente (navegador web) y el servidor web está cifrada y protegida mediante SSL/TLS. Esto garantiza que los datos transmitidos, como contraseñas, información de tarjetas de crédito u otra información confidencial, estén protegidos contra posibles ataques de intercepción y manipulación.

Ejemplos

Ejemplo de funcionamiento de SSL/TLS:

Imaginemos que un usuario está navegando por un sitio web seguro (por ejemplo, https://www.ejemplo.com):

El usuario abre su navegador web e ingresa la URL del sitio web seguro.

El navegador envía una solicitud al servidor de www.ejemplo.com para establecer una conexión segura.

El servidor responde con su certificado digital, que incluye su clave pública y está firmado por una autoridad de certificación confiable.

El navegador verifica la autenticidad del certificado, comprueba si el nombre de dominio coincide y confía en la autoridad de certificación.

Si la verificación es exitosa, el navegador genera una clave de sesión aleatoria y la cifra con la clave pública del servidor.

El navegador envía la clave de sesión cifrada al servidor.

El servidor utiliza su clave privada para descifrar la clave de sesión.

El cliente y el servidor generan claves de cifrado y descifrado simétricas a partir de la clave de sesión.

A partir de este punto, el cliente y el servidor intercambian datos cifrados utilizando las claves simétricas generadas.

Los datos se cifran en el cliente y se descifran en el servidor, y viceversa, asegurando la confidencialidad de la información transmitida.

Durante la transmisión, se verifica la integridad de los datos y las firmas digitales para detectar cualquier modificación no autorizada.

La conexión segura se mantiene hasta que el usuario cierra la sesión o navega a otro sitio.

Conclusiones

David Contreras Mejía:

En conclusión, la criptografía asimétrica y los protocolos como SSL y TLS desempeñan un papel fundamental en la seguridad de las comunicaciones y la protección de nuestros datos en línea. La criptografía asimétrica nos brinda un sistema de clave pública y privada que permite el intercambio seguro de información sin necesidad de compartir claves secretas. En un mundo cada vez más conectado y dependiente de la transmisión de datos, la criptografía asimétrica y los protocolos como SSL y TLS son elementos esenciales para garantizar la seguridad y la privacidad en nuestras comunicaciones en línea.

López Castillón Jonathan Jhosua:

En conclusión, tanto la criptografía asimétrica y los protocolos como SSL y TLS, como la implementación de aplicaciones que utilizan claves asimétricas para el proceso de firma digital, desempeñan un papel vital en la seguridad y protección de nuestras comunicaciones y datos en línea. Estos sistemas y protocolos nos brindan la capacidad de intercambiar información de manera segura sin tener que compartir claves secretas, lo que es fundamental en un mundo cada vez más conectado y dependiente de la transmisión de datos. La criptografía asimétrica y la firma digital nos permiten garantizar la autenticidad, integridad y confidencialidad de la información en entornos digitales, previniendo la falsificación y manipulación de datos en transacciones en línea. Adoptar medidas sólidas de seguridad, como el uso de algoritmos de cifrado asimétrico y protocolos de seguridad, se vuelve imprescindible para salvaguardar la confidencialidad y la confianza en nuestras transacciones electrónicas.

Sánchez Pérez Omar Alejandro:

En conclusión, la implementación de una aplicación en Java que permita el proceso de firma digital de documentos a través de claves asimétricas ha servido como ejemplo para entender su funcionamiento y cómo se aplica para garantizar la autenticidad e integridad de la información en entornos digitales. La firma digital, junto con los algoritmos de cifrado asimétrico como RSA y los protocolos de seguridad como SSL y TLS, proporcionan una capa adicional de protección para prevenir la falsificación y manipulación de datos en transacciones en línea. En un mundo cada vez más dependiente de las comunicaciones digitales, es crucial adoptar medidas de seguridad sólidas para salvaguardar la confidencialidad y la confianza en las transacciones electrónicas.

Referencias

Internet Society. (2022, August 24). What is TLS & How Does it Work? | ISOC Internet Society. https://www.internetsociety.org/deploy360/tls/basics/

(S/f). Cloudflare.com. Recuperado el 11 de junio de 2023, de https://www.cloudflare.com/es-es/learning/ssl/transport-layer-security-tls/

Cobb, M., & Loshin, P. (2021, julio 20). What is Secure Sockets Layer? Security; TechTarget. https://www.techtarget.com/searchsecurity/definition/Secure-Sockets-Layer-SSL

Qué es un certificado SSL: definición y explicación. (2023, abril 19). latam.kaspersky.com. https://latam.kaspersky.com/resource-center/definitions/what-is-a-ssl-certificate