Unidad 6. Técnicas de programación segura

Descargar estos apuntes

Índice

Introducción

El término criptografía es un derivado de la palabra griega *kryptos* que significa oculto y *grafos* que significa escritura. El objetivo de la criptografía es ocultar el significado de un mensaje mediante el cifrado o codificación del mensaje.

De todas las técnicas de programación segura, nosotros vamos a estudiar en este tema

- Las funciones de resumen (Hash)
- El **cifrado simétrico** o de clave oculta
- El **cifrado asimétrico** o de clave privada

También estudiaremos cómo usar canales seguros de comunicación para nuestras aplicaciones, mediante el uso de sockets seguros empleando SSL/TLS.

Además, veremos una herramienta de seguridad que incorpora Java, las políticas de seguridad.

Objetivos

Objetivos de esta unidad:

- Identificar principios y prácticas de programación segura.
- Analizar técnicas y prácticas criptográficas.
- Definir políticas de seguridad.
- Emplear algoritmos criptográficos.
- Utilizar sockets seguros para la transmisión de información.
- Estudiar el funcionamiento de TLS/SSL sobre el protocolo TCP
- Conocer herramientas de uso genérico en la criptografía

Seguridad

Seguridad digital

Los aspectos fundamentales de la seguridad en las comunicaciones digitales son los siguientes

- **Integridad**: Permite asegurar que los datos que recibe un receptor son idénticos a los que ha enviado el emisor. Es decir, no se ha modificado en ningún punto intermedio en **el canal, que como sabemos, es un canal compartido y por tanto, inseguro**. Las modificaciones pueden ser causadas por fallos en la transmisión a través del canal o bien por una acción intencionada de un tercero.
- **Confidencialidad**: Nos asegura que los datos transmitidos son inteligibles sólo para el receptor del mensaje. Por las características del medio no podemos evitar que el mensaje llegue a otros destinatarios, pero lo que sí podemos evitar es que estos puedan ver el contenido original del mensaje. Esto se consigue cifrando el mensaje.
- **Autenticación**: Permite asegurar al receptor de un mensaje que el emisor del mensaje es quien dice ser y no cualquier otro. Esto se consigue con los certificados y la firma digital.
- **No repudio**: Es una consecuencia de la característica anterior, ya que un receptor puede demostrar que el mensaje fue enviado por un emisor de forma inequívoca.

Seguridad en Java

oracle Security Developers Guide

Desde el punto de vista de la seguridad, el conjunto de clases de seguridad distribuidas con el SDK de Java 2 pueden dividirse en dos subconjuntos:

- Clases relacionadas con el control de acceso y la gestión de permisos.
- Clases relacionadas con la Criptografía.

Java incluye APIs de acceso a funciones criptográficas de propósito general, conocidas como la **Arquitectura Criptográfica de Java o Java Cryptography Architecture (JCA)** y la **Extension Criptográfica de Java o Java Cryptography Extension (JCE)**.

El JCA está formado por las clases básicas relacionadas con criptografía distribuidas con el JDK y el soporte para la encriptación lo proporciona el paquete de extensión JCE.

Java también incluye un conjunto de paquetes para la comunicación segura en Internet, conocidos como la **Extensión de Sockets Seguros de Java o Java Secure Socket Extension (JSSE)**. Implementa una versión Java de los protocolos SSL y TLS, además incluye funcionalidades como cifrado de datos, autenticación del servidor, integridad de mensajes y autenticación del cliente.

Por último Java incluye una interfaz que permite a las aplicaciones Java acceder a servicios de control de autenticación y acceso, el **Servicio de Autentificación y Autorización de Java o Java Authentication and Authorization Service (JAAS)**. Puede usarse con dos fines: la autenticación de usuarios para conocer quién

está ejecutando código Java; y la autorización de usuarios para garantizar que quién lo ejecuta tiene los permisos necesarios para hacerlo.

JCA: Engines, algoritmos y proveedores

Java tiene una Arquitectura de Proveedores, que permite que coexistan multiples implementaciones de algoritmos criptográficos (es decir multiples implementaciones del JCE). La plataforma Java 2 extiende substancialmente la JCA, entre otras cosas se ha mejorado la infraestructura de gestión de certificados para soportar los certificados X.509 V3.

Para comprender el funcionamiento del JCA tenemos que definir algunos términos básicos:

Engine

En el contexto del JCA utilizamos el término motor (engine) para referirnos a una representación abstracta de un servicio criptográfico que no tiene una implementación concreta. Un servicio criptográfico siempre está asociado con un algoritmo o tipo de algoritmo y puede tener alguna de las siguientes funciones:

- Proporcionar operaciones criptográficas (como las empleadas en el firmado y el resumen de mensajes)- Generar o proporcionar el material criptográfico (claves o parámetros) necesario para realizar las operaciones.
- Generar objetos (almacenes de claves o certificados) que agrupen claves criptográficas de modo seguro.

Algoritmo

Un algoritmo es una implementación de un motor. Por ejemplo, el algoritmo MD5 es una implementación del motor de algoritmos de resumen de mensajes. La implementación interna puede variar dependiendo del código que proporcione la clase MD5.

Proveedor

Un proveedor es el encargado de proporcionar la implementación de uno o varios algoritmos al programador (es decir, darle acceso a una implementación interna concreta de los algoritmos).

Proveedores

La JCA define el concepto de proveedor mediante la clase Provider del paquete java.security. Se trata de una clase abstracta que debe ser redefinida por clases proveedor específicas.

El constructor de una clase proveedor ajusta los valores de varias propiedades que necesita el API de seguridad de Java para localizar los algoritmos u otras facilidades implementadas por el proveedor.

La clase Provider tiene métodos para acceder al nombre del proveedor, el número de versión y otras informaciones sobre las implementaciones de los algoritmos para la generación, conversión y gestión de claves y la generación de firmas y resúmenes.

Si un programador desea saber los proveedores disponibles puede emplear los métodos

- getProvider("nombre") para saber si un proveedor concreto está instalado

- getProviders() que retorna un vector de cadenas con los nombres de los proveedores

> **Archivo java.security**
>
> %JAVA_HOME%/conf/security/java.security es el archivo que contiene la información de la configuración de seguridad que utilizan las clases de la JCA.
>
> Ahí están declarados todos los proveedores y algoritmos que están disponibles, así como el orden en el que las clases los buscarán.

Para entender como funcionan los proveedores daremos un ejemplo. Supongamos que un programa necesita una implementación del algoritmo MD5. Para obtenerla el programador necesita crear una instancia del mismo y lo hará escribiendo la siguiente línea de código:

```
MessageDigest m = MessageDigest.getInstance("MD5");
```

Internamente, el método getInstance() solicita a la clase java.security.Security que le proporcione el objeto solicitado. Como no se ha especificado proveedor la clase Security consulta a todos los proveedores disponibles, solicitando una implementación del algoritmo "MD5", hasta que encuentra una o se queda sin proveedores. La consulta se realiza según la lista de proveedores del archivo java.security, que por defecto sólo contiene la entrada:

```
Security.provider.1=sun.security.provider.Sun
```

Engines

En el JDK el JCA define las siguientes clases Engine

Clase JCA	Función
java.security.MessageDigest	Calculo de resumen de mensajes (hash).
java.security.Signature	Firma de datos y verificación firmas.
java.security.KeyPairGenerator	Generar pares de claves (pública y privada) para un algoritmo.
java.security.KeyFactory	Convertir claves de formato criptográfico, especificaciones de claves y viceversa
java.security.certificate.CertificateFactory	Crear certificados de clave pública y listas de revocación(CRLs).
java.security.KeyStore	Crear y gestionar un almacén de claves (keystore).

Clase JCA	Función
java.security.AlgorithmParameters	Gestionar los parámetros de un algoritmo, incluyendo codificación y descodificación.
java.security.AlgorithmParameterGenerator	Generar un conjunto de parámetros para un algoritmo.
java.security.SecureRandom	Generar números aleatorios o pseudo aleatorios.

Para instanciar una clase motor se debe invocar el método estático *getInstance()*, si se le pasa un nombre de algoritmo se intentará obtener una implementación de algún proveedor.

Algoritmos

Al igual que pasa con las herramientas de línea de comandos, debemos saber qué algoritmos están disponibles para su uso por las aplicaciones en nuestra máquina virtual.

El siguiente programa nos permite saber que proveedores y algoritmos tenemos instalados en nuestro sistema.

Además, si lo invocamos con la opción -l nos dirá que algoritmos implementan (leyendo las propiedades del proveedor)

Toda la información mostrada se extrae del archivo *java.security*

```java
class U6S1_1_InfoProveedoresJCA {

    public static void main(String[] args) {
        boolean listarProps = false;
        if (args.length > 0 && args[0].equals("-l")) {
            listarProps = true;
        }
        System.out.println("-----------------------------------");
        System.out.println("Proveedores instalados en su sistema");
        System.out.println("-----------------------------------");
        int i = 0;
        for (Provider proveedor: Security.getProviders()) {
            System.out.println("Núm. proveedor : " + (i + 1));
            System.out.println("Nombre         : " + proveedor.getName());
            System.out.println("Versión        : " + proveedor.getVersion());
            System.out.println("Información    :\n  " + proveedor.getInfo());
            System.out.println("Propiedades    :");
            if (listarProps) {
                Enumeration propiedades = proveedor.propertyNames();
                while (propiedades.hasMoreElements()) {
                    String clave = (String) propiedades.nextElement();
                    String valor = proveedor.getProperty(clave);
                    System.out.println("  " + clave + " = " + valor);
                }
            }
            System.out.println("-----------------------------------");
        }
    }
}
```

El siguiente programa nos permite comprobar las propiedades de los algoritmos disponibles en nuestro sistema.

```java
class U6_S1_2_ProbarAlgoritmosJCA {

    public static void main(String[] args) {
        if (args.length != 1) {
            System.out.println("Uso: java ProbarAlgoritmosJCA <algoritmo>");
            System.exit(1);
        }
        try {
            MessageDigest md = MessageDigest.getInstance(args[0]);
            System.out.println("Algoritmo: " + md.getAlgorithm());
            System.out.println("Proveedor: " + md.getProvider().getName());
            System.out.println("Info     : " + md.toString());
            System.out.println("Tamaño   : " + md.getDigestLength());
            System.out.println("Bloque   : " + md.getBlockSize());
            System.out.println("Entrada  : " + md.getInputSize());
            System.out.println("Salida   : " + md.getOutputSize());
            System.out.println("Implement: " + md.getClass().getName());
        } catch (NoSuchAlgorithmException e) {
            System.out.println("Algoritmo no disponible");
        }
    }
}
```