



Programación de Servicios y Procesos

6.1 Seguridad



I.E.S.
Doctor Balmis

Apuntes de PSP (https://psp2dam.github.io/psp_sources/es/) creados por Vicente Martínez bajo licencia
CC BY-NC-SA 4.0  (<http://creativecommons.org/licenses/by-nc-sa/4.0/deed.es>)

6.1 Seguridad

- 6.1.1. Seguridad digital
- 6.1.2. Seguridad en Java
 - JCA: Engines, algoritmos y proveedores
- 6.1.3. Proveedores
- 6.1.4. Engines
 - 6.1.5. Algoritmos

6.1.1. 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.

6.1.2. Seguridad en Java

oracle Security Developers Guide (<https://docs.oracle.com/en/java/javase/11/security/java-cryptography-architecture-jca-reference-guide.html>)

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 Autenticación 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 múltiples implementaciones de algoritmos criptográficos (es decir múltiples 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).

6.1.3. 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:

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

java

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
```

6.1.4. Engines

En el JDK el JCA define las siguientes clases Engine

Clase JCA	Función
<code>java.security.MessageDigest</code>	Calculo de resumen de mensajes (hash).
<code>java.security.Signature</code>	Firma de datos y verificación firmas.
<code>java.security.KeyPairGenerator</code>	Generar pares de claves (pública y privada) para un algoritmo.
<code>java.security.KeyFactory</code>	Convertir claves de formato criptográfico, especificaciones de claves y viceversa
<code>java.security.cert.CertificateFactory</code>	Crear certificados de clave pública y listas de revocación(CRLs).
<code>java.security.KeyStore</code>	Crear y gestionar un almacén de claves (keystore).
<code>java.security.AlgorithmParameters</code>	Gestionar los parámetros de un algoritmo, incluyendo codificación y decodificación.
<code>java.security.AlgorithmParameterGenerator</code>	Generar un conjunto de parámetros para un algoritmo.
<code>java.security.SecureRandom</code>	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.

6.1.5. 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*

```
1  class U6S1_1_InfoProveedoresJCA {
2
3      public static void main(String[] args) {
4          boolean listarProps = false;
5          if (args.length > 0 && args[0].equals("-l")) {
6              listarProps = true;
7          }
8          System.out.println("-----");
9          System.out.println("Proveedores instalados en su sistema");
10         System.out.println("-----");
11         int i = 0;
12         for (Provider proveedor: Security.getProviders()) {
13             System.out.println("Núm. proveedor : " + (i + 1));
14             System.out.println("Nombre       : " + proveedor.getName());
15             System.out.println("Versión      : " + proveedor.getVersion());
16             System.out.println("Información  :\n" + proveedor.getInfo());
17             System.out.println("Propiedades  :");
18             if (listarProps) {
19                 Enumeration propiedades = proveedor.propertyNames();
20                 while (propiedades.hasMoreElements()) {
21                     String clave = (String) propiedades.nextElement();
22                     String valor = proveedor.getProperty(clave);
23                     System.out.println("  " + clave + " = " + valor);
24                 }
25             }
26             System.out.println("-----");
27         }
28     }
29 }
```

java