




Process and Service Programming

6.1 Security



I.E.S.
Doctor Balmis

PSP class notes (https://psp2dam.github.io/psp_sources) by Vicente Martínez is licensed under
CC BY-NC-SA 4.0  (<http://creativecommons.org/licenses/by-nc-sa/4.0/?ref=chooser-v1>)

6.1 Security

- 6.1.1. Digital security
- 6.1.2. Seguridad en Java
- 6.1.2. Security in Java
 - JCA: Engines, algorithms and providers
- 6.1.3. Providers
 - 6.1.4. Engines
 - 6.1.5 Algorithms

6.1.1. Digital security

Fundamental aspects of security in digital communications are:

- **Integrity:** It allows to ensure that the data received by a receiver are identical to those sent by the sender. That is, it has not been modified at any intermediate point in **the channel, which as we know, is a shared and therefore, insecure channel**. Modifications can be caused by failures in transmission through the channel or by an intentional action of a third party.
- **Confidentiality:** It ensures that the transmitted data is intelligible only to the recipient of the message. Due to the characteristics of the medium, we cannot prevent the message from reaching other recipients, but what we can prevent is that they can see the original content of the message. This is achieved by encrypting the message.
- **Authentication:** It allows to ensure to the receiver of a message that the sender of the message is who he says he is and not any other. This is achieved with certificates and digital signature.
- **Non-repudiation:** It is a consequence of the previous characteristic, since a receiver can demonstrate that the message was sent by a sender unequivocally.

6.1.2. Seguridad en Java

6.1.2. Security in Java

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

From the point of view of security, the set of security classes distributed with the Java 2 SDK can be divided into two subsets:

- Classes related to access control and permission management.
- Classes related to Cryptography.

Java includes APIs for accessing general-purpose cryptographic functions, known as the **Java Cryptography Architecture (JCA)** and the **Java Cryptography Extension (JCE)**.

The JCA is made up of the basic classes related to cryptography distributed with the JDK and the support for encryption is provided by the JCE extension package.

Java also includes a set of packages for secure Internet communication, known as the **Java Secure Socket Extension (JSSE)**. It implements a Java version of the SSL and TLS protocols, and also includes features such as data encryption, server authentication, message integrity, and client authentication.

Finally, Java includes an interface that allows Java applications to access authentication and access control services, the **Java Authentication and Authorization Service (JAAS)**. It can be used for two purposes: user authentication to know who is running Java code; and user authorization to ensure that whoever runs it has the necessary permissions to do so.

JCA: Engines, algorithms and providers

Java has a Provider Architecture, which allows multiple implementations of cryptographic algorithms to coexist (i.e. multiple implementations of the JCE). The Java 2 platform substantially extends the JCA, among other things the certificate management infrastructure has been improved to support X.509 V3 certificates.

To understand how the JCA works we have to define some basic terms:

Engine

In the context of the JCA we use the term engine to refer to an abstract representation of a cryptographic service that does not have a concrete implementation. A cryptographic service is always associated with an algorithm or type of algorithm and can have any of the following functions:

- Provide cryptographic operations (such as those used in signing and digesting messages)
- Generate or provide the cryptographic material (keys or parameters) necessary to perform the operations.
- Generate objects (key stores or certificates) that group cryptographic keys in a secure manner.

Algorithm

An algorithm is an implementation of an engine. For example, the MD5 algorithm is an implementation of the message digest engine. The internal implementation may vary depending on the code provided by the MD5 class.

Provider

A provider is responsible for providing the implementation of one or more algorithms to the programmer (i.e. giving him access to a specific internal implementation of the algorithms).

6.1.3. Providers

The JCA defines the concept of provider through the Provider class of the java.security package. It is an abstract class that must be redefined by specific provider classes.

The constructor of a provider class adjusts the values of several properties that the Java security API needs to locate the algorithms or other facilities implemented by the provider.

The Provider class has methods to access the name of the provider, the version number and other information about the implementations of the algorithms for key generation, conversion and management and the generation of signatures and digests.

If a programmer wants to know the available providers, he can use the methods

- `getProvider("name")` to know if a specific provider is installed
- `getProviders()` that returns a vector of strings with the names of the providers



java.security file

`%JAVA_HOME%/conf/security/java.security` is the file that contains the security configuration information used by the JCA classes.

All the providers and algorithms that are available are declared there, as well as the order in which the classes will look for them.

To understand how providers work we will give an example. Suppose a program needs an implementation of the MD5 algorithm. To obtain it the programmer needs to create an instance of it and will do so by writing the following line of code:

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

Internally, the `getInstance()` method requests the `java.security.Security` class to provide it with the requested object. Since no provider has been specified, the `Security` class queries all the available providers, requesting an implementation of the "MD5" algorithm, until it finds one or runs out of providers. The query is made according to the list of providers in the `java.security` file, which by default only contains the entry:

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

6.1.4. Engines

In the JDK the JCA defines the following Engine classes

JCA Class	Function
<code>java.security.MessageDigest</code>	Calculation of message summary (hash).
<code>java.security.Signature</code>	Data signing and signature verification.
<code>java.security.KeyPairGenerator</code>	Generate key pairs (public and private) for an algorithm.
<code>java.security.KeyFactory</code>	Convert cryptographic key formats, key specifications and vice versa
<code>java.security.cert.CertificateFactory</code>	Create public key certificates and revocation lists (CRLs).
<code>java.security.KeyStore</code>	Create and manage a key store (keystore).
<code>java.security.AlgorithmParameters</code>	Manage the parameters of an algorithm, including encoding and decoding.
<code>java.security.AlgorithmParameterGenerator</code>	Generate a set of parameters for an algorithm.
<code>java.security.SecureRandom</code>	Generate random or pseudo-random numbers.

To instantiate an engine class you must invoke the static method `getInstance()`, if you pass an algorithm name it will try to obtain an implementation of some provider.

6.1.5 Algorithms

As with command line tools, we need to know which algorithms are available for use by applications on our virtual machine.

The following program allows us to know which providers and algorithms we have installed on our system.

In addition, if we invoke it with the `-l` option it will tell us which algorithms they implement (reading the provider's properties)

All the information shown is extracted from the `java.security` file

```
1 class U6_S1_1_InfoProvedoresJCA {
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 }

```

The following program allows us to check the properties of algorithms available in our system.

```

1     class U6_S1_2_ProbarAlgoritmosJCA {
2
3         public static void main(String[] args) {
4             if (args.length != 1) {
5                 System.out.println("Uso: java ProbarAlgoritmosJCA <algoritmo>");
6                 System.exit(1);
7             }
8             try {
9                 MessageDigest md = MessageDigest.getInstance(args[0]);
10                System.out.println("Algoritmo: " + md.getAlgorithm());
11                System.out.println("Proveedor: " + md.getProvider().getName());
12                System.out.println("Info      : " + md.toString());
13                System.out.println("Tamaño   : " + md.getDigestLength());
14                System.out.println("Bloque   : " + md.getBlockSize());
15                System.out.println("Entrada  : " + md.getInputSize());
16                System.out.println("Salida   : " + md.getOutputSize());
17                System.out.println("Implement: " + md.getClass().getName());
18            } catch (NoSuchAlgorithmException e) {
19                System.out.println("Algoritmo no disponible");
20            }
21        }
22    }

```

java