



# Process and Service Programming

## 6.2 Hash functions

---



I.E.S.  
Doctor Balmis

PSP class notes ([https://psp2dam.github.io/psp\\_sources](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>)

## 6.2 Hash functions

- [6.2.1. Hash functions](#)
- [6.2.2. MessageDigest](#)
- [6.2.3. MessageDigest with GnuPG](#)

### 6.2.1. Hash functions

A **message digest**, better known as **hash functions**, is a digital mark of a block of data. There are a large number of algorithms designed to process these summaries, the two best known are SHA-1 and MD5.

From a digest we can highlight the following characteristics:

- For the same algorithm, the digest always has the same size, regardless of the size of the data used to generate it.
- It is impossible to recover the original information from a digest.
- The digest should not reveal anything about the data used to generate it.
- It is computationally unfeasible to find two messages that have the same digest value. Mathematically it is highly unlikely, but not impossible.
- A small change in the summarized data generates a completely different digest.

Digests are used to generate unique and reliable identifiers. Sometimes they are called *checksum*, since they are used to check if a download has been done correctly, generating its summary and comparing it with the one generated by the original file.



#### A hash is not used to encrypt

It is important to note that, because it is impossible to obtain the data that generated a digest from the digest itself, the digest cannot be used to encrypt information.

On the other side, it is a mechanism that is used to compare. Its most widespread use is with passwords, since in the databases a summary is saved instead of the password in clear. In this way, when a password is received, its digest is generated and compared with the stored value.

### 6.2.2. MessageDigest

The *MessageDigest* class allows applications to implement cryptographically secure summary algorithms such as SHA-256 or SHA-512

To generate a hash with JCA, proceed as follows:

1. An object of the *MessageDigest* class is created with the static method *getInstance()* of the same class, specifying the name of the algorithm. Optionally, the name of the provider can be specified.
2. Data is added with the *update()* method. A byte or byte array can be added. This method can be invoked several times to add new data.
3. The hash value is obtained with the *digest()* method.
4. If you wanted to calculate a new hash, the *reset()* method would be invoked to start the process again.

Next we can see an example

```
1 public class U6S2_MessageDigest {  
2
```

java

```

3 public static void main(String[] args) {
4     String plaintext = "Esto es un texto plano.";
5     try {
6         // Obtenemos un ENGINE que implementa el algoritmo especificado
7         // Se puede indicar cualquier algoritmo disponible en el sistema
8         // SHA-224, SHA-512, SHA-256, SHA3-224, ...
9         MessageDigest m = MessageDigest.getInstance("SHA-256");
10
11        // Opcional - Reinicia el objeto para un nuevo uso
12        // Por si queremos poner este código en un bucle y procesar más
13        // de un mensaje
14        m.reset();
15
16        // Realiza el resumen de los datos pasados por parámetro
17        // Si queremos procesar la información poco a poco,
18        // debemos ir llamando al método update para cada bloque de datos
19        m.update(plaintext.getBytes());
20
21        // Completa el cálculo del valor del hash y devuelve el resumen
22        byte[] digest = m.digest();
23
24        // Mensaje de resumen
25        System.out.println("Resumen (raw data): " + new String(digest));
26
27        // Mensaje en formato hexadecimal
28        System.out.println("Resumen (hex data): " + toHexadecimal(digest));
29
30
31        // Información del proceso
32        System.out.println("=> Algoritmo: " + m.getAlgorithm() + ", Provider: " + m.getProvider().getName());
33    } catch (NoSuchAlgorithmException e) {
34        System.err.println("No se ha encontrado la implementación del algoritmo MD5 en ningún Provider");
35    }
36 }
37
38 static String toHexadecimal(byte[] hash) {
39     String hex = "";
40     for (int i = 0; i < hash.length; i++) {
41         String h = Integer.toHexString(hash[i] & 0xFF);
42         if (h.length() == 1) {
43             hex += "0";
44         }
45         hex += h;
46     }
47     return hex.toUpperCase();
48 }
49 }

```

and this would be the output provided

```
1 Resumen (raw data): Y"3 bbs?;~E
2 Resumen (hex data): FB59D3112291331411B92CD60628ED7E7DE62733F3B10EDADAF303AAABE57E45
3 => Algoritmo: SHA-256, Provider: SUN 11
```

sh

### 6.2.3. MessageDigest with GnuPG

With the GnuPG suite we can generate summaries of files using the algorithms provided by the suite.



### Algorithms available for GnuPG

To see the list of available algorithms we have to show the help of the command

```
gpg --help
```

and at the top we see the information of the algorithms available for each type of service. Specifically, of summaries, in my installed version:

Summary: SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224

To generate a summary of a file, we run the command as follows

1

```
gpg --print-md SHA256 filename.ext
```

sh