

**MP\_0490.**

**Programación de servicios y procesos  
UF4. Técnicas de programación segura**

**4.3. Firma digital y  
control de acceso**

# Índice

---

☰	Objetivos	3
☰	Hash: funciones de resumen	4
☰	Algoritmo MD5	6
☰	Algoritmo SHA	9
☰	¿Qué son las colisiones?	12
☰	¿Qué es una firma digital?	13
☰	Algoritmo DSA para firma digital	17
☰	Verificación de la firma	23
☰	Certificado digital (algoritmo SHA1withRSA)	28
☰	Control de acceso a usuarios	40
☰	Resumen	42

# Objetivos

---

En esta lección perseguimos los siguientes objetivos:

- 1 Comprender el significado y objetivo de los resúmenes *hash*.
  - 2 Comprobar la integridad de un mensaje utilizando resúmenes *hash*.
  - 3 Crear archivos de firma digital.
  - 4 Verificar mensajes utilizando el archivo de firma digital.
  - 5 Utilizar los resúmenes *hash* y la firma digital para control de acceso de usuarios con contraseña segura.
  - 6 Comprender el significado del certificado digital.
  - 7 Crear documentos con certificado digital.
- 

¡Ánimo y adelante!

# Hash: funciones de resumen

Las funciones de resumen, o funciones *hash*, disponen de un algoritmo capaz de crear, a partir de una determinada información de longitud variable (*entrada*), una cadena de longitud fija (*salida*) que resume dicha información.

Imagínate el siguiente texto original:

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor. Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados, lentejas los viernes, algún palomino de añadidura los domingos, consumían las tres partes de su hacienda.

Aplicándole un algoritmo de resumen hash podríamos obtener la siguiente cadena:

① QUÅΠ-º™ ñë#Z%+C£

Los algoritmos *hash* garantizan que:

1

La cadena de salida obtenida representa un resumen cifrado de los datos suministrados.

2

Para la misma información de entrada, se obtendrá siempre el mismo resumen hash siempre y cuando se utilice el mismo tipo de algoritmo.

Además, un resumen *hash* tiene varias **aplicaciones**:

- **Comprobar que un archivo no ha sido modificado.**
- **Comprobar contraseñas de manera segura.** Un usuario que desea acceder a una aplicación distribuida introduce sus credenciales (usuario y *password*). Dichas credenciales tendrán que navegar por la red y llegar al proceso servidor, que realizará las comprobaciones oportunas contra la base de datos. En la transmisión de dicha información, la contraseña podría ser interceptada por terceras personas o procesos indeseados. Una solución segura es transmitir a través de la red el resumen *hash* y no las credenciales originales.
- **Garantizar la integridad de una información transmitida.** Cuando hay que transmitir mucha información a través de la red, dicha información puede transmitirse junto con el resumen y el tipo de algoritmo utilizado. Una vez que la información llega al destino, puede volverse a generar el resumen con el mismo algoritmo y comprobar si ha habido algún cambio durante la transmisión.

**(i)** Este sistema es utilizado por los sitios web que proporcionan descargas de archivos grandes, para comprobar que, tras la transmisión, el fichero sigue intacto.

# Algoritmo MD5

---

**MD5 (Message-Digest Algorithm 5) es un algoritmo de resumen criptográfico creado por el profesor Ronald Rivest del Instituto Tecnológico de Massachusetts en 1991.**

El resumen generado por el algoritmo MD5 tiene una longitud de 128 bits.

Las librerías estándar de Java (*JRE System Library*) cuentan con una clase específica que implementa los algoritmos para la generación de resúmenes *hash*. Se trata de la clase *MessageDigest*.

La clase *MessageDigest* es capaz de unir varios mensajes en uno solo y crear un resumen *hash* con la unión de todos ellos. Veamos un **ejemplo** muy simple:

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class Resumen {

    public static void main(String[] args) throws NoSuchAlgorithmException {
        byte[] cancion1 = "Bohemian Rhapsody".getBytes();
        byte[] cancion2 = "Imagine".getBytes();
        byte[] cancion3 = "We Are the Champions".getBytes();
        byte[] cancion4 = "Hotel California".getBytes();
        byte[] cancion5 = "Yesterday".getBytes();
    }
}
```

```
MessageDigest md = MessageDigest.getInstance("SHA-256");
md.update(cancion1);
md.update(cancion2);
md.update(cancion3);
md.update(cancion4);
md.update(cancion5);

byte[] resumen = md.digest();
System.out.println("Resumen hash: " + new String(resumen));
}
```

Vamos a analizar el ejemplo:

```
byte[] cancion1 = "Bohemian Rhapsody".getBytes();
byte[] cancion2 = "Imagine".getBytes();
byte[] cancion3 = "We Are the Champions".getBytes();
byte[] cancion4 = "Hotel California".getBytes();
byte[] cancion5 = "Yesterday".getBytes();
```

La clase *MessageDigest* trabaja la información como un conjunto de bytes, así que creamos varias cadenas de texto y las convertimos a un *array* de *bytes* con el método *getBytes()*.

```
MessageDigest md = MessageDigest.getInstance("MD5");
md.update(cancion1);
md.update(cancion2);
md.update(cancion3);
md.update(cancion4);
md.update(cancion5);
```

Creamos un objeto *MessageDigest* a través del método estático *getInstance()* al que se le pasa el tipo de algoritmo que vamos a utilizar. En este caso, el algoritmo *MD5*. Luego, actualizamos nuestro objeto con todos los mensajes que queremos unir para crear el resumen *hash*.

```
byte[] resumen = md.digest();
System.out.println("Resumen hash: " + new String(resumen));
```

El último paso será ejecutar el método *digest()* de nuestro objeto *MessageDigest* para obtener el resumen, que también será una cadena de bytes.

Hay que tener en cuenta que no se puede descifrar el mensaje original con la cadena resumen obtenida, ya que no es ese el objetivo que se persigue.

---

**El objetivo es garantizar la integridad del mensaje,  
comprobando más tarde que no ha sufrido cambios.**

---

# Algoritmo SHA

---

**El algoritmo de resumen SHA (*Secure Hash Algorithm*), cuya primera versión fue lanzada en 1993 por el Instituto Nacional de Normas y Tecnología de EEUU, genera una salida de 160 bits.**

Este algoritmo ha ido evolucionando para su mejora según se han ido detectando vulnerabilidades, y dando lugar a las siguientes versiones:

- **SHA:** la versión original de 1993.
- **SHA-1:** segunda versión de 1995.
- **SHA-2:** tercera versión de 2001. Esta versión está compuesta por cuatro algoritmos distintos que permiten el cifrado del resumen con 224, 256, 384 o 512 bits.
- **SHA-3:** cuarta versión de 2012.

Si quieres modificar el anterior programa para utilizar el algoritmo SHA en lugar de MD5, solo tendrás que cambiar la línea de la llamada al método `getInstance()`, pasando como argumento el nuevo tipo de algoritmo.

```
MessageDigest md = MessageDigest.getInstance("SHA");
```

Observa cómo la cadena de resumen obtenida es distinta y algo más larga que la obtenida con el algoritmo MD5.

---

**También puedes probar con las distintas evoluciones del algoritmo SHA.**

```
MessageDigest md = MessageDigest.getInstance("SHA-1");
```

Con este algoritmo, aunque difiere la forma de obtener el resumen, el resultado es el mismo.

Para usar el algoritmo SHA-2, dado que existen cuatro alternativas, habrá que indicar el número de bits del resumen que se desea obtener. Prueba a poner en práctica las siguientes variantes:

```
MessageDigest md = MessageDigest.getInstance("SHA-224");
```

Algoritmo SHA-2 con 224 bits.

```
MessageDigest md = MessageDigest.getInstance("SHA-256");
```

Algoritmo SHA-2 con 256 bits.

```
MessageDigest md = MessageDigest.getInstance("SHA-384");
```

Algoritmo SHA-2 con 384 bits.

```
MessageDigest md = MessageDigest.getInstance("SHA-512");
```

Algoritmo SHA-2 con 512 bits.



Las librerías estándar de Java no han incorporado todavía el algoritmo SHA-3, aunque podría utilizarse importando otras librerías de terceros.

## ¿Qué son las colisiones?

En los algoritmos *hash*, se denomina *colisión* a la ocurrencia de dos mensajes de entrada distintos que generan la misma cadena de salida de resumen.

La detección de colisiones en los distintos algoritmos ha provocado la necesidad de crear las siguientes versiones:

- 1 MD5 nació en 1991, después de ser descubiertas colisiones a su predecesor MD4.
- 2 En 1996 el criptógrafo alemán Hans Dobbertin encontró una colisión en el algoritmo HD5, lo que puso en entredicho la eficacia de dicho algoritmo.
- 3 El algoritmo SHA es considerado más resistente a colisiones que el algoritmo MD5. Aun así, distintos hallazgos de vulnerabilidades han provocado que este algoritmo tuviera que ir evolucionando hasta la versión más moderna.

**(i) ¡Recuerda!** Las funciones **hash** se consideran resistentes a colisión cuando varios hackers de todo el mundo han intentado crear colisiones sin poder conseguirlo.

# ¿Qué es una firma digital?

---

La firma digital está estrechamente relacionada con el concepto de resumen hash anteriormente estudiado.

---

**El resumen *hash* solo permite comprobar la integridad del mensaje (si ha sufrido cambios), pero no permite verificar la identidad del organismo o proceso que envió dicho mensaje.**

Aquí es donde entra en juego la firma digital:

---

Firma digital = resumen *hash* + verificación de identidad

## Objetivos de la firma digital

La firma digital puede cubrir los siguientes objetivos:

- Comprobar la **integridad** de los datos transmitidos mediante el resumen *hash*.
- **Identificar** el origen de dicho mensaje. Si la firma contiene credenciales (usuario y clave), se puede utilizar para verificar dichos credenciales.
- Garantía de **no repudio**, es decir, que el autor del mensaje no puede negar su autoría.

Los algoritmos de firma digital requieren de un par de claves (pública y privada) del mismo tipo que utilizan los procesos de cifrado y descifrado de mensajes de tipo asimétrico o de clave pública.

Cada algoritmo debe implementar dos métodos distintos:

- Un método para la **creación de la firma digital**, que utilizará la **clave privada**.
- Un método para la **verificación de la firma digital**, que utilizará la **clave pública** asociada con la clave privada con la que se creó la firma.

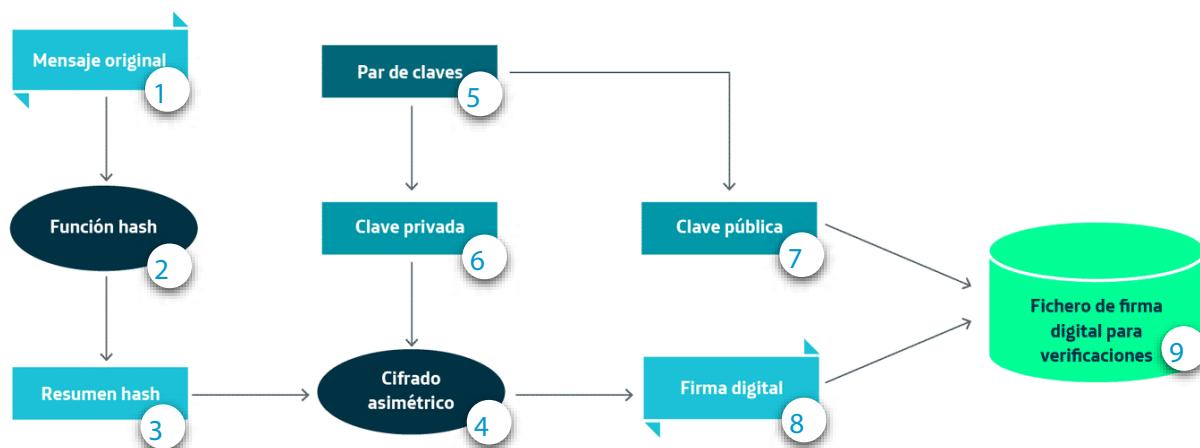


La clase Java **Signature** localizada en el paquete `java.security` es la encargada de la **creación y verificación de las firmas electrónicas**.

## ¿Cómo se crea la firma digital?

1

Funcionamiento del método de creación de la firma digital.



**1. Mensaje original.**

Conjunto de bytes del mensaje original que deseamos incluir en la firma.

**2. Aplicación de la función hash.**

Versión resumida del mensaje original que servirá para posteriormente verificar su integridad.

**3. Resumen hash.**

Versión resumida del mensaje original que servirá para posteriormente verificar su integridad.

**4. Cifrado asimétrico.**

Cifrado asimétrico del resumen *hash* utilizando la clave privada que servirá para acreditar la autoría del mensaje.

**5. Par de claves.**

Privada y pública, utilizadas en el cifrado asimétrico.

**6. Clave privada.**

Única para el proceso que creó la firma digital. De esta manera, sirve como sello o carnet de identidad para dicho proceso.

**7. Clave pública.**

La misma para todos los procesos que tengan acceso a la firma digital. Necesaria en el proceso de verificación de la firma.

**8. Firma digital.**

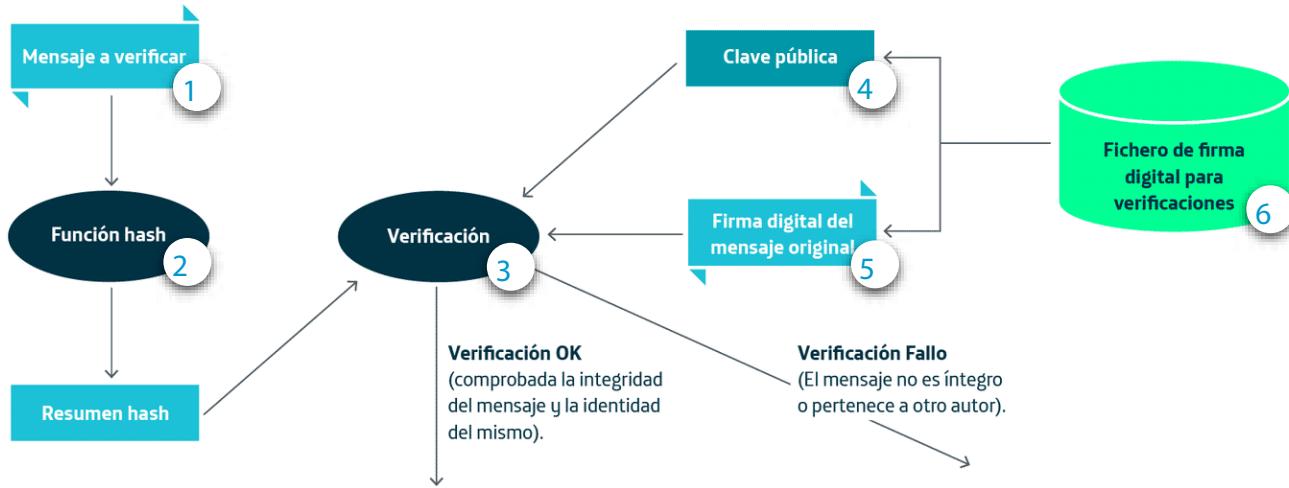
La firma digital es el resumen *hash* cifrado con la clave privada.

**9. Fichero de firma digitalestará compuesto por:**

- a. La firma digital, creada con la clave privada usando un algoritmo asimétrico.
- b. La clave pública asociada a la clave privada, que servirá para posteriores verificaciones.

2

## Verificación con el fichero de firma digital.



### 1. Mensaje a verificar.

Mensaje cuyo contenido queremos comparar con el original. Se comparará el resumen hash del mensaje con el resumen hash del mensaje original almacenado en el fichero de firma digital.

### 2. Función hash.

Función que resume el mensaje a verificar.

### 3. Verificación.

Procedimiento que tendrá como resultado OK si se cumplen estas dos condiciones:

- El mensaje a verificar es igual que el mensaje original (comprobación de integridad).
- La clave pública utilizada en el proceso de verificación está relacionada con la clave privada con la que se creó la firma digital (verificación de identidad).

### 4. Clave pública.

Clave pública guardada en el fichero de firma digital y que se relaciona con la clave privada con la que se creó la firma digital.

### 5. Firma digital.

Firma digital almacenada en el fichero de firma digital.

### 6. Fichero de firma digital.

Guarda un objeto con la firma digital y la clave pública (elementos necesarios para realizar verificaciones).

# Algoritmo DSA para firma digital

**DSA (Digital Signature Algorithm) es un algoritmo para firma digital que permite asociar un mensaje con el emisor que lo ha transmitido, certificando su autoría.**

El algoritmo DSA fue aceptado como estándar para firma digital por el Gobierno Federal de los Estados Unidos a partir de 1991.

El procedimiento para firmar un mensaje requiere de las siguientes tareas:

- 1 Obtención de una clave privada (*PrivateKey*) a partir de un par de claves (objeto de tipo *KeyPair*).
- 2 Creación de un objeto de tipo *Signature* (gestor de firmas).
- 3 Configuración del objeto *Signature* para firmar (*initSign*), asignándole la clave privada.
- 4 Creación del mensaje que se añadirá a la firma.
- 5 Actualización del objeto *Signature* (*update*) con los datos del mensaje que se añadirá a la firma.
- 6 Creación de la firma digital (*sign*) que estará compuesta por una secuencia de bytes con el resumen *hash* del mensaje cifrado con la clave privada.

**Veamos un ejemplo, donde se firma un mensaje secreto utilizando la clave privada y, finalmente, se crea un archivo en disco con la firma digital que servirá para realizar posteriores verificaciones.**

El archivo generado tendrá que contener los siguientes elementos:

- La cadena (mensaje secreto) que contendrá la **firma**.
- **La clave pública** asociada a la clave privada con la que se creó la firma, ya que será necesaria para realizar verificaciones.

Estos elementos serán guardados en un objeto de la clase *MensajeFirmado* que posteriormente se guardará en disco.

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.security.GeneralSecurityException;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.Signature;

public class FirmaMensaje {
    public static void main(String[] args) {

        System.out.println("VAMOS A CREAR UNA FIRMA DIGITAL");
        try {
            KeyPairGenerator generadorDeClaves = KeyPairGenerator.getInstance("DSA");
            KeyPair claves = generadorDeClaves.generateKeyPair();
            PrivateKey clavePrivada = claves.getPrivate();

            Signature firmadorVerificador = Signature.getInstance("DSA");
            firmadorVerificador.initSign(clavePrivada);

            byte[] mensajeSecreto = "CriptaMagica".getBytes();
            firmadorVerificador.update(mensajeSecreto);

            byte[] firmaDigital = firmadorVerificador.sign();
```

```
        System.out.println("Firma: " + new String(firmaDigital));

        MensajeFirmado mf = new MensajeFirmado(claves.getPublic(), firmaDigital);
        guardar(mf);

    } catch (GeneralSecurityException e) {
        System.out.println("Error al crear la firma digital");
        System.out.println("Excepción de tipo: " + e.getClass().getName());
        System.out.println(e.getMessage());
    }
}

private static void guardar(MensajeFirmado mf) {
    try {
        FileOutputStream fichero = new FileOutputStream("firma.obj");
        ObjectOutputStream buffer = new ObjectOutputStream(fichero);
        buffer.writeObject(mf);
        buffer.close();
        fichero.close();
        System.out.println("El fichero de firma digital se ha creado con éxito");
    } catch (IOException e) {
        System.out.println("Error al grabar fichero de firma digital");
        System.out.println("Excepción de tipo: " + e.getClass().getName());
        System.out.println(e.getMessage());
    }
}
}
```

Programa que genera la firma digital con el mensaje secreto *CriptaMagica*.

```
import java.io.Serializable;
import java.security.PublicKey;

public class MensajeFirmado implements Serializable {
    private static final long serialVersionUID = 8700992515811830138L;

    private PublicKey clavePublica;
    private byte[] firma;

    public MensajeFirmado(PublicKey clavePublica, byte[] firma) {
        this.clavePublica = clavePublica;
        this.firma = firma;
    }

    public PublicKey getClavePublica() {
        return clavePublica;
    }
    public byte[] getFirma() {
        return firma;
    }
}
```

Clase *MensajeFirmado* utilizado en el programa anterior de creación de firmas de mensajes.

**Una vez ejecutado el programa, el resultado será similar a este:**

- i** VAMOS A CREAR UNA FIRMA DIGITAL  
Firma: 0,000\A;q?™ &†1ÁÇ"q\$ñ³¶¶¶hü¶"fä,ûdeöÇÝÝ?ð^pe  
El fichero de firma digital se ha creado con éxito

## Análisis paso a paso del programa:

- 1 Creamos un par de claves y obtenemos la clave privada.

```
KeyPairGenerator generadorDeClaves = KeyPairGenerator.getInstance("DSA");
KeyPair claves = generadorDeClaves.generateKeyPair();
PrivateKey clavePrivada = claves.getPrivate();
```

- 2 Creamos un objeto de tipo *Signature* (gestor de firma digital) que usará el algoritmo DSA.

```
Signature firmadorVerificador = Signature.getInstance("DSA");
```

- Configuramos nuestro objeto *Signature* para que utilice la clave privada durante el proceso de cifrado asimétrico del resumen *hash*.

```
firmadorVerificador.initSign(clavePrivada);
```

4

Creamos el mensaje secreto que vamos a incluir en la firma.

```
byte[] mensajeSecreto = "CriptaMagica".getBytes();
```

5

Actualizamos el objeto *Signature*, añadiendo los datos del mensaje secreto. A partir de estos datos se creará el resumen *hash* que después será cifrado.

```
firmadorVerificador.update(mensajeSecreto);
```

6

Con el método *sign()* obtenemos la firma digital, que está formada por el resumen *hash* del mensaje cifrado con la clave privada.

```
byte[] firmaDigital = firmadorVerificador.sign();
System.out.println("Firma: " + new String(firmaDigital));
```

7

Construimos el objeto *mf* de tipo *MensajeFirmado* que contendrá todo lo necesario para poder realizar verificaciones de nuevos mensajes contra la firma digital.

```
MensajeFirmado mf = new MensajeFirmado(claves.getPublic(), firmaDigital);
guardar(mf);
```

8

Por último, el método *guardar()* guarda el objeto *mf* en disco.

```
private static void guardar(MensajeFirmado mf) {  
    try {  
        FileOutputStream fichero = new FileOutputStream("firma.obj");  
        ObjectOutputStream buffer = new ObjectOutputStream(fichero);  
        buffer.writeObject(mf);  
        buffer.close();  
        fichero.close();  
        System.out.println("El fichero de firma digital se ha creado con  
éxito");  
    } catch (IOException e) {  
        System.out.println("Error al grabar fichero de firma digital");  
        System.out.println("Excepción de tipo: " + e.getClass().getNa-  
me());  
        System.out.println(e.getMessage());  
    }  
}
```

## Verificación de la firma

**Vamos a crear un programa que restringirá el acceso a los usuarios que no conozcan el mensaje secreto o que no dispongan del archivo de firma digital.**

Para verificar una firma es necesario realizar las siguientes tareas:

- 1 Obtención de la clave pública (*PublicKey*); la que se relaciona con la clave privada con la que se creó la firma. Esta clave está guardada en el fichero que habrá que leer.
- 2 Creación de un objeto *Signature* (gestor de firmas).
- 3 Configuración del objeto *Signature*, esta vez, para verificar (*initVerify*) y lo asociamos a la clave pública.
- 4 Configurar el mensaje (mensaje secreto) que se desea verificar.
- 5 Actualización del objeto *Signature* (*update*) incluyendo el mensaje que se desea verificar.
- 6 Verificar el mensaje con la firma digital (*verify*). La firma digital, junto con la clave pública, se encuentran en el fichero que hemos leído previamente.

El programa que vamos a desarrollar solicitará al usuario que introduzca el mensaje secreto. El acceso al programa solo estará autorizado para los usuarios que conozcan el mensaje secreto y dispongan del archivo de firma digital, ya que se validará el mensaje introducido con el que esté guardado en la firma digital.

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.security.GeneralSecurityException;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.Signature;

public class FirmaMensaje {
    public static void main(String[] args) {

        System.out.println("VAMOS A CREAR UNA FIRMA DIGITAL");
        try {
            KeyPairGenerator generadorDeClaves = KeyPairGenerator.getInstance("DSA");
            KeyPair claves = generadorDeClaves.generateKeyPair();
            PrivateKey clavePrivada = claves.getPrivate();

            Signature firmadorVerificador = Signature.getInstance("DSA");
            firmadorVerificador.initSign(clavePrivada);

            byte[] mensajeSecreto = "CriptaMagica".getBytes();
            firmadorVerificador.update(mensajeSecreto);

            byte[] firmaDigital = firmadorVerificador.sign();
            System.out.println("Firma: " + new String(firmaDigital));

            MensajeFirmado mf = new MensajeFirmado(claves.getPublic(), firmaDigital);
            guardar(mf);

        } catch (GeneralSecurityException e) {
            System.out.println("Error al crear la firma digital");
            System.out.println("Excepción de tipo: " + e.getClass().getName());
            System.out.println(e.getMessage());
        }
    }

    private static void guardar(MensajeFirmado mf) {
        try {
            FileOutputStream fichero = new FileOutputStream("firma.obj");
            ObjectOutputStream buffer = new ObjectOutputStream(fichero);
            buffer.writeObject(mf);
            buffer.close();
            fichero.close();
            System.out.println("El fichero de firma digital se ha creado con éxito");
        } catch (IOException e) {
            System.out.println("Error al grabar fichero de firma digital");
            System.out.println("Excepción de tipo: " + e.getClass().getName());
            System.out.println(e.getMessage());
        }
    }
}
```

Teniendo en cuenta que el mensaje secreto almacenado en la firma era *CriptaMagica*, veamos dos ejemplos de ejecución.

- ⓘ ¿Cuál es el mensaje secreto?

**Perico de los palotes**

La verificación ha fallado, Acceso denegado

- ⓘ ¿Cuál es el mensaje secreto?

**CriptaMagica**

Verificación OK, Bienvenido al sistema

## Análisis del ejemplo paso a pas

1

Obtenemos el objeto de la clase *MensajeFirmado* que previamente guardamos en el fichero *firma.obj*. Recuerda que el objeto guardado está compuesto por la firma digital y la clave pública necesaria para realizar verificaciones con dicha firma.

```
MensajeFirmado mf;
try {
    FileInputStream fichero = new FileInputStream("firma.obj");
    ObjectInputStream buffer = new ObjectInputStream(fichero);
    mf = (MensajeFirmado) buffer.readObject();
    buffer.close();
    fichero.close();
} catch (IOException | ClassNotFoundException e) {
    System.out.println("Error al leer fichero de firma digital");
    System.out.println("Excepción de tipo: " + e.getClass().getName());
    System.out.println(e.getMessage());
    return;
}
```

2

Creamos un objeto *Signature* y lo configuramos para verificar con la clave pública guardada en el fichero (ahora disponible en el objeto *mf*).

```
Signature firmadorVerificador = Signature.getInstance("DSA");
firmadorVerificador.initVerify(mf.getClavePublica());
```

3

Creamos una cadena para verificar (**mensaje secreto**). Es el usuario quien tendrá que introducir por teclado esta cadena, para lo que utilizaremos el objeto *lector* de tipo *Scanner*.

```
System.out.println("¿Cuál es el mensaje secreto?");
String mensajeSecreto = lector.nextLine();
```

4

Actualizamos el objeto *Signature* con los datos a verificar (mensaje secreto introducido por el usuario).

```
firmadorVerificador.update(mensajeSecreto.getBytes());
```

5

Realizamos la verificación con el método *verify*.

```
boolean ok = firmadorVerificador.verify(mf.getFirma());
if (ok) {
    System.out.println("Verificación OK, Bienvenido al sistema");
}
else {
    System.out.println("La verificación ha fallado, Acceso denegado");
}
```

Si el usuario utiliza otro archivo de firma digital distinto, aunque este contenga el mismo mensaje secreto, no tendrá acceso al sistema porque la clave pública utilizada no será la correcta.

## Certificado digital (algoritmo SHA1withRSA)

---

**Un certificado digital permite firmar digitalmente documentos, acreditando su identidad. Además, permite cifrar mensajes que solo podrán ser descifrados por el creador de la firma digital, único conocedor de la clave privada.**

En el apartado anterior utilizaste el algoritmo DSA (*Digital Signature Algorithm*) para la creación de la firma digital, pero hay que tener en cuenta que dicho algoritmo solo permite crear o verificar la firma, pero no cifrar y descifrar mensajes.

**¿Y si queremos que nuestro programa verificador sea, además, capaz de aprovechar la clave pública que le ha sido otorgada para cifrar mensajes?** Con el algoritmo DSA no sería posible.

---

Vamos a recordar los conceptos más importantes sobre firma digital para que comprendas mejor el siguiente paso:

---

## El creador y dueño de la firma digital realiza las siguientes acciones:

1

Para construir la firma digital crea un par de claves (pública/privada) y utiliza la clave privada para generar dicha firma.

2

Entrega la firma digital junto con la clave pública a todas las personas o procesos en los que confía.

---

## Los poseedores de la firma digital y de la clave pública realizan las siguientes acciones:

1

Verificar la firma con la clave pública.

2

Con el resultado de la verificación, restringir el acceso a los usuarios que conozcan el mensaje guardado en la firma.

### Un paso más

Ahora queremos que el poseedor de la firma pueda dar un paso más, como, por ejemplo:

- Aprovechar la clave pública que le ha sido otorgada para cifrar mensajes utilizando el algoritmo RSA, mensajes que después solo podrán ser descifrados por el poseedor de la clave privada, es decir, por el creador de la firma.

- ⓘ Recuerda que el algoritmo RSA realiza el cifrado con la clave pública y el descifrado con la clave privada.

Esta mejora es la que convierte la firma digital en un certificado digital. Además de la verificación, abre una vía de comunicación cifrada con el emisor del certificado.

Para lograrlo, el objeto *Signature* debe utilizar el algoritmo *SHA1withRSA* en lugar del DSA.

Y modifica la clase *FirmaMensaje*, dejándola así:

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.security.GeneralSecurityException;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.Signature;

public class FirmaMensaje {
    public static void main(String[] args) {

        System.out.println("VAMOS A CREAR UNA FIRMA DIGITAL");
        try {
            KeyPairGenerator generadorDeClaves = KeyPairGenerator.getInstance("RSA");
            KeyPair claves = generadorDeClaves.generateKeyPair();
            PrivateKey clavePrivada = claves.getPrivate();

            Signature firmadorVerificador = Signature.getInstance("SHA1withRSA");

            firmadorVerificador.initSign(clavePrivada);

            byte[] mensajeSecreto = "CriptaMagica".getBytes();
            firmadorVerificador.update(mensajeSecreto);

            byte[] firmaDigital = firmadorVerificador.sign();
            System.out.println("Firma: " + new String(firmaDigital));

            MensajeFirmado mf = new MensajeFirmado(claves.getPublic(), firmaDigital);
            guardar(mf);

        } catch (GeneralSecurityException e) {
            System.out.println("Error al crear la firma digital");
            System.out.println("Excepción de tipo: " + e.getClass().getName());
            System.out.println(e.getMessage());
        }
    }

    private static void guardar(MensajeFirmado mf) {
        try {
```

```
    FileOutputStream fichero = new FileOutputStream("firma.obj");
    ObjectOutputStream buffer = new ObjectOutputStream(fichero);
    buffer.writeObject(mf);
    buffer.close();
    fichero.close();
    System.out.println("El fichero de firma digital se ha creado con éxito");
} catch (IOException e) {
    System.out.println("Error al grabar fichero de firma digital");
    System.out.println("Excepción de tipo: " + e.getClass().getName());
    System.out.println(e.getMessage());
}
}
```

Observa que **solo se han modificado un par de líneas:**

```
KeyPairGenerator generadorDeClaves = KeyPairGenerator.getInstance("RSA");
```

Creamos el par de claves compatibles con el algoritmo criptográfico RSA. Esto nos permitirá utilizar dichas claves para el cifrado y descifrado de mensajes.

```
Signature firmadorVerificador = Signature.getInstance("SHA1withRSA");
```

Creamos el objeto *signature* de tipo *SHA1withRSA*, lo que significa que utilizará el algoritmo *SHA1* para generar el resumen *hash* del mensaje y *RSA* para su cifrado.

Ejecuta el programa y verás que **el resultado es similar, solo que ahora la clave es más grande:**



#### VAMOS A CREAR UNA FIRMA DIGITAL

Firma: FÀiNÒùÀTYGö-àIÝù·ùDš{ùOàòUz;]W\_?'™ èùùE t\$ÓCÙÙ„¼ àír -  
tþ¹^UåšÙÐÙ‰ ÈjÈ7ìà+`?ÅòÄ‰%½ôý?U]Nd] iÃ4Sä?D-x?`-þM‰ ÙÑqoÍÙÐ¹a9,;

El fichero de firma digital se ha creado con éxito

## A continuación, vamos a modificar el programa verificador, dejándolo así:

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.security.GeneralSecurityException;
import java.security.Signature;
import java.util.Scanner;

public class AccesoConFirmaDigital {

    public static void main(String[] args) {

        MensajeFirmado mf;
        try {
            FileInputStream fichero = new FileInputStream("firma.obj");
            ObjectInputStream buffer = new ObjectInputStream(fichero);
            mf = (MensajeFirmado) buffer.readObject();
            buffer.close();
            fichero.close();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error al leer fichero de firma digital");
            System.out.println("Excepción de tipo: " + e.getClass().getName());
            System.out.println(e.getMessage());
            return;
        }

        Scanner lector = new Scanner(System.in);

        try {
            Signature firmadorVerificador = Signature.getInstance("SHA1withRSA");
            firmadorVerificador.initVerify(mf.getClavePublica());

            System.out.println("¿Cuál es el mensaje secreto?");
            String mensajeSecreto = lector.nextLine();

            firmadorVerificador.update(mensajeSecreto.getBytes());

            boolean ok = firmadorVerificador.verify(mf.getFirma());
            if (ok) {
                System.out.println("Verificación OK, Bienvenido al sistema");
            } else {
                System.out.println("La verificación ha fallado, Acceso denegado");
            }

            lector.close();
        } catch (GeneralSecurityException e) {
            System.out.println("Error en la verificación de la firma digital");
            System.out.println("Excepción de tipo: " + e.getClass().getName());
            System.out.println(e.getMessage());
            lector.close();
            return;
        }
    }
}
```

Como ves, solo hemos modificado la siguiente línea:

```
Signature firmadorVerificador = Signature.getInstance("SHA1withRSA");
```



Si ejecutas, verás que funciona exactamente igual que antes, pero ahora podemos completar el programa grabando un texto cifrado que solo el dueño de la clave privada podrá descifrar.

## Continuamos adaptando el código de la clase *FirmaMensaje*.

Ya has modificado el código de la clase *FirmaMensaje*, de manera que ahora guardará dos ficheros:

1

El fichero *firma.obj* con la firma digital y la clave pública, igual que antes, pero ahora solo se grabará la primera vez que se ejecute el programa. Es decir, que si el fichero de firma ya existe, no se creará de nuevo.

2

El fichero *clave.obj* que se creará también una sola vez, al mismo tiempo que el fichero *firma.obj*. En este archivo se guardará la clave privada que servirá para posteriormente leer mensajes cifrados.

## Ahora, la clase *FirmaMensaje* funcionará de la siguiente forma:

- Comprueba si existe el fichero *clave.obj*, y si no existe lo crea a la vez que se crea también el fichero *firma.obj*.
- Si existe el fichero *clave.obj* lo abre para obtener la clave privada.
- Solo si existe previamente el fichero *clave.obj* comprueba también si existe un fichero llamado *mensaje.obj* (mensaje cifrado por la clase *AccesoConFirmaDigital*), en cuyo caso lo descifra y lo muestra en pantalla.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.security.GeneralSecurityException;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.Signature;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;

public class FirmaMensaje {
    public static void main(String[] args) throws BadPaddingException, General-
    SecurityException, IOException {

        File fichero = new File("clave.obj");
        if (fichero.exists()) {
            PrivateKey clavePrivada = leerClavePrivada();
            String texto = decifrarTexto(clavePrivada);
            System.out.println("Mensaje descifrado: " + texto);
        }
        else {
            crearCertificado();
        }
    }

    private static void crearCertificado() {
        System.out.println("VAMOS A CREAR EL CERTIFICADO");
        try {
            KeyPairGenerator generadorDeClaves = KeyPairGenerator.getInstance("RSA");
            KeyPair claves = generadorDeClaves.generateKeyPair();
        }
    }
}
```

```
PrivateKey clavePrivada = claves.getPrivate();

Signature firmadorVerificador = Signature.getInstance("SHA1withRSA");

firmadorVerificador.initSign(clavePrivada);

byte[] mensajeSecreto = "CriptaMagica".getBytes();

firmadorVerificador.update(mensajeSecreto);

byte[] firmaDigital = firmadorVerificador.sign();
System.out.println("Firma: " + new String(firmaDigital));

MensajeFirmado mf = new MensajeFirmado(claves.getPublic(), firmaDigital);
guardar(mf, clavePrivada);

} catch (GeneralSecurityException e) {
    System.out.println("Error al crear la firma digital");
    System.out.println("Excepción de tipo: " + e.getClass().getName());
    System.out.println(e.getMessage());
}
}

private static void guardar(MensajeFirmado mf, PrivateKey clavePrivada) {
try {
    FileOutputStream fichero = new FileOutputStream("firma.obj");
    FileOutputStream ficheroClavePrivada = new FileOutputStream("clave.obj");
    ObjectOutputStream buffer = new ObjectOutputStream(fichero);
    ObjectOutputStream bufferClavePri = new ObjectOutputStream(ficheroClavePrivada);
    buffer.writeObject(mf);
    bufferClavePri.writeObject(clavePrivada);
    buffer.close();
    bufferClavePri.close();
    fichero.close();
    ficheroClavePrivada.close();
    System.out.println("El fichero de firma digital se ha creado con éxito");
} catch (IOException e) {
    System.out.println("Error al grabar fichero de firma digital");
    System.out.println("Excepción de tipo: " + e.getClass().getName());
    System.out.println(e.getMessage());
}
}

public static PrivateKey leerClavePrivada() {
    PrivateKey pk;
    try {
        FileInputStream fichero = new FileInputStream("clave.obj");
        ObjectInputStream buffer = new ObjectInputStream(fichero);
        pk = (PrivateKey) buffer.readObject();
        buffer.close();
        fichero.close();
        return pk;
    } catch (IOException | ClassNotFoundException e) {
```

```
        System.out.println("Error al leer fichero de firma digital");
        System.out.println("Excepción de tipo: " + e.getClass().getNa-
me());
        System.out.println(e.getMessage());
        return null;
    }
}

public static String decifrarTexto(PrivateKey clavePrivada) {
    try {
        File ficheroCifrado = new File("mensaje.obj");
        if (ficheroCifrado.exists()) {
            Cipher descifrador = Cipher.getInstance("RSA");
            descifrador.init(Cipher.DECRYPT_MODE, clavePrivada);

            FileInputStream fichero = new FileInputStream("mensa-
je.obj");
            ObjectInputStream buffer = new ObjectInputStream(fichero);
            byte[] mensajeCifrado = (byte[]) buffer.readObject();
            buffer.close();
            fichero.close();

            byte[] bytesMensajeDescifrado = descifrador.doFinal(mensaje-
Cifrado);
            return new String(bytesMensajeDescifrado);
        }
        else {
            return "No hay mensaje que descifrar";
        }
    } catch (GeneralSecurityException | IOException | ClassNotFoundException e) {
        System.out.println("Error al descifrar el mensaje");
        System.out.println("Excepción de tipo: " + e.getClass().getNa-
me());
        System.out.println(e.getMessage());
        return null;
    }
}
}
```

---

A continuación, adaptaremos el programa *AccesoConFirmaDigital* para que, además de verificar el mensaje secreto, solicite al usuario la introducción de un texto que será cifrado y guardado en un fichero con el nombre *mensaje.obj*.

```
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.security.GeneralSecurityException;
import java.security.PublicKey;
import java.security.Signature;
import java.util.Scanner;

import javax.crypto.Cipher;

public class AccesoConFirmaDigital {

    public static void main(String[] args) {

        MensajeFirmado mf;
        try {
            FileInputStream fichero = new FileInputStream("firma.obj");
            ObjectInputStream buffer = new ObjectInputStream(fichero);
            mf = (MensajeFirmado) buffer.readObject();
            buffer.close();
            fichero.close();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error al leer fichero de firma digital");
            System.out.println("Excepción de tipo: " + e.getClass().getName());
            System.out.println(e.getMessage());
            return;
        }

        Scanner lector = new Scanner(System.in);

        try {
            Signature firmadorVerificador = Signature.getInstance("SHA1withRSA");
            firmadorVerificador.initVerify(mf.getClavePublica());

            System.out.println("¿Cuál es el mensaje secreto?");
            String mensajeSecreto = lector.nextLine();

            firmadorVerificador.update(mensajeSecreto.getBytes());

            boolean ok = firmadorVerificador.verify(mf.getFirma());
            if (ok) {
                System.out.println("Verificación OK, Bienvenido al sistema");
                crearMensajeCifrado(mf.getClavePublica(), lector);
            }
            else {
                System.out.println("La verificación ha fallado, Acceso denegado");
            }

            lector.close();
        } catch (GeneralSecurityException e) {
            System.out.println("Error en la verificación de la firma digital");
            System.out.println("Excepción de tipo: " + e.getClass().getName());
            System.out.println(e.getMessage());
            lector.close();
            return;
        }
    }
}
```

```
private static void crearMensajeCifrado(PublicKey clavePublica, Scanner lector) {  
    System.out.println("Escribe un mensaje: ");  
    String mensaje = lector.nextLine();  
    try {  
        Cipher cifrador = Cipher.getInstance("RSA");  
        cifrador.init(Cipher.ENCRYPT_MODE, clavePublica);  
        byte[] bytesMensajeCifrado = cifrador.doFinal(mensaje.getBytes());  
  
        FileOutputStream fichero = new FileOutputStream("mensaje.obj");  
        ObjectOutputStream buffer = new ObjectOutputStream(fichero);  
        buffer.writeObject(bytesMensajeCifrado);  
        buffer.close();  
        fichero.close();  
  
        System.out.println("Mensaje cifrado: " + new String(bytesMensajeCifrado));  
    } catch (GeneralSecurityException | IOException e) {  
        System.out.println("Error al grabar el fichero mensaje.obj");  
        System.out.println("Excepción de tipo: " + e.getClass().getName());  
        System.out.println(e.getMessage());  
    }  
}  
}  
}
```

## Cómo funcionan ambos programas

```
<terminated> FirmaMensaje [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (23 sept. 2018 23:32:24)  
VAMOS A CREAR EL CERTIFICADO  
Firma: {vÔ%@B@8!)?i@qñZ÷qYJþ£³±à"t6ß8sÝ# G.»ýÈ~íl@@:í#Ýidæö@ab?IyË4R,dI^ž#7VÖ?ùØÓ`nø¥U"IrP&êñ Øç?aØ  
!ž=ðÛ"@@Vñ9@-¹Å),àýÜß,¶Ø  
Rý«  
El fichero de firma digital se ha creado con éxito
```

Primera ejecución del programa *FirmaMensaje* desde Eclipse. Se crean los archivos *firma.obj* y *clave.obj*.

```
<terminated> FirmaMensaje [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (23 sept. 2018 23:35:03)  
Mensaje descifrado: No hay mensaje que descifrar
```

Segunda ejecución del programa *FirmaMensaje*: todavía no existe el fichero *mensaje.obj* (mensaje cifrado), por lo que no hay nada que descifrar.

```
<terminated> AccesoConFirmaDigital [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (23 sept. 2018 23:37:26)
¿Cuál es el mensaje secreto?
Cocodrilo
La verificación ha fallado, Acceso denegado
```

Ejecución del programa *AccesoConFirmaDigital* con verificación fallida: se ha introducido un mensaje secreto que no corresponde al de la firma digital.

```
<terminated> AccesoConFirmaDigital [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (23 sept. 2018 23:39:32)
¿Cuál es el mensaje secreto?
CriptaMagica
Verificación OK, Bienvenido al sistema
Escribe un mensaje:
El perro de san Roque no tiene rabo
Mensaje cifrado: #...í~:6?f^~.ñD2àt$ÀÍÓ$dgjQ?;...!@¹,Í'øpf}çÖa!c">EÍm6@VÒCj?ç‘|Y?ø^+?%"EÀfjyδ_WøaQ'š6]L      %7'E
El mensaje se ha grabado en el fichero mensaje.obj
```

Ejecución del programa *AccesoConFirmaDigital* con verificación OK: se ha introducido el mensaje secreto correcto, permitiendo el acceso al sistema. Se solicita al usuario la introducción por teclado de un mensaje que será cifrado y guardado en un archivo.

```
<terminated> FirmaMensaje [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (23 sept. 2018 23:41:45)
Mensaje descifrado: El perro de san Roque no tiene rabo
```

Tercera ejecución del programa *FirmaMensaje*: Los ficheros *firma.obj* y *clave.obj* ya existen, así que no ha sido necesario crearlos. Se ha leído el contenido cifrado del archivo *mensaje.obj* y se ha descifrado con la clave privada guardada.

Ahora puedes seguir ejecutando varias veces el programa *AccesoConFirmaDigital* y después el **FirmaMensaje**, y comprobar cómo este último **siempre es capaz de descifrar el mensaje**.

# Control de acceso a usuarios

En los apartados anteriores has podido comprobar cómo las técnicas de resumen *hash* y de firma digital sirven para controlar el acceso a los usuarios.

Recordemos cómo:

1

## Con resumen *hash*.

Los usuarios que desean acceder a una aplicación en red deben introducir sus credenciales (usuario y clave). Dichas credenciales tendrán que navegar a través de la red hasta llegar al programa servidor. Enviar el resumen *hash* en lugar de las credenciales originales asegura que no sea fácil secuestrar dichos datos durante la transferencia.

2

## Con firma digital.

Una determinada entidad podrá emitir firmas digitales a los usuarios en los que confía. Los usuarios propietarios de la firma digital tendrán acceso a las aplicaciones de dicha entidad. En este caso, el mensaje almacenado en la firma podrían ser *login* y *password*.

3

## Con DNI electrónico.

El DNI electrónico (DNIE) es el mejor ejemplo que podemos mencionar para hablar sobre certificados digitales y control de acceso. El Cuerpo Nacional de Policía define el DNIE de la siguiente manera:

“El Documento Nacional de Identidad electrónico (DNIe) acredita física y digitalmente la identidad personal de su titular, permite la firma electrónica de documentos y otorga la posibilidad a su portador de utilizar la identidad electrónica en cuantos servicios digitales estén disponibles.”

- Cuerpo Nacional de Policía-

La Dirección General de la Policía es el único organismo autorizado a emitir los certificados digitales para el DNI, que contendrán los mismos datos que el DNI convencional pero en formato cifrado.

---

**Sitios web de organismos como la Agencia Tributaria, Dirección General de Policía, Tesorería de la Seguridad Social, etc. permiten el acceso a determinadas áreas restringidas por medio de la presentación del DNIe donde, además, se puede enviar documentos certificados digitalmente.**

---

## Resumen

---

Has terminado la lección, repasemos los puntos más importantes que hemos tratado.

- Las funciones de **resumen**, o funciones **hash**, disponen de un algoritmo capaz de crear, a partir de una determinada información de longitud variable (entrada), una cadena de longitud fija (salida) que resume dicha información. Se utiliza principalmente para garantizar la integridad de los datos transmitidos.
- La **firma digital** está compuesta por el resumen hash de un mensaje secreto, cifrado con clave privada, junto a la clave pública que será necesaria para realizar la verificación de identidad. La firma digital va más allá que el resumen hash, garantizando, no solo la integridad de la información transmitida, sino la comprobación de la identidad del emisor.



**PROEDUCA**