

Taller 8 – Cifrado y Control de Integridad

El propósito de este taller es estudiar algunos métodos de cifrado de información disponibles en librerías Java (javax.crypto.*).

Parte A – Cifrado Simétrico

Parte 1: Cifrar y descifrar con cifrado simétrico

1. Escriba una clase llamada **Simetrico** en la cual vamos a incluir los métodos **cifrar()** y **descifrar()**.

Puede utilizar la siguiente constante, la cual determina el mecanismo de **padding** a utilizar para cifrar y descifrar:

```
private final static String PADDING = "AES/ECB/PKCS5Padding";
```

Cree el método **cifrar()** usando el siguiente código.

```
public static byte[] cifrar(SecretKey llave, String texto) {  
    byte[] textoCifrado;  
  
    try {  
        Cipher cifrador = Cipher.getInstance(PADDING);  
        byte[] textoClaro = texto.getBytes();  
  
        cifrador.init(Cipher.ENCRYPT_MODE, llave);  
        textoCifrado = cifrador.doFinal(textoClaro);  
  
        return textoCifrado;  
    } catch (Exception e) {  
        System.out.println("Excepcion: " + e.getMessage());  
        return null;  
    }  
}
```

Cree el método **descifrar()** usando el siguiente código.

```
public static byte [] descifrar(SecretKey llave, byte[] texto) {  
    byte[] textoClaro;  
  
    try {  
        Cipher cifrador = Cipher.getInstance(PADDING);  
        cifrador.init(Cipher.DECRYPT_MODE, llave);  
        textoClaro = cifrador.doFinal(texto);  
    } catch (Exception e) {  
        System.out.println("Excepcion: " + e.getMessage());  
        return null;  
    }  
  
    return textoClaro;  
}
```

2. Escriba una clase `Main` para crear una instancia de la clase `Simetrico`.

En el método **`main()`**:

- Utilice la siguiente constante para indicar que utilizará el algoritmo **AES** para cifrar y descifrar sus mensajes:

```
private final static String ALGORITMO = "AES";
```

- Reciba por teclado la entrada de un texto.
- Imprima el texto recibido por el teclado.
- Imprima texto claro en **`byte []`**. Utilice el método **`getBytes()`** de la clase **`String`** para convertir el mensaje a **`byte[]`**. Para imprimir el contenido del **`byte[]`** utilice el método **`imprimir()`**.

```
public static void imprimir (byte [ ] contenido) {  
    int i = 0;  
    for (; i < contenido.length - 1; i++) {  
        System.out.print(contenido[i] + " ");  
    }  
    System.out.println(contenido[i] + " ");  
}
```

- Genere la llave secreta, la cual será empleada para cifrar y descifrar. Utilice el siguiente fragmento de código:

```
KeyGenerator keygen = KeyGenerator.getInstance(ALGORITMO);  
SecretKey secretKey = keygen.generateKey();
```

- Obtenga un **`byte []`** con el texto cifrado, invocando al método **`cifrar()`** de la clase **`Simetrico`**.
- Imprima el texto cifrado en **`byte []`**. Utilice el método **`imprimir()`**.
- Obtenga un **`byte []`** con el texto descifrado, invocando al método **`descifrar()`** de la clase **`Simetrico`**.
- Imprima el texto descifrado en **`byte []`**. Utilice el método **`imprimir()`**.
- Convierta el **`byte[]`** con el texto descifrado a **`String`**. Utilice el método constructor de la clase **`String`**.

Un ejemplo de la salida en consola de la ejecución del programa se puede ver a continuación:

```
Escriba el texto que desea cifrar: La nueva realidad
La nueva realidad
Mensaje de entrada en texto claro: La nueva realidad
Texto claro: 76 97 32 110 117 101 118 97 32 114 101 97 108 105 100 97 100
Texto cifrado: 84 208 150 15 68 128 38 14 194 74 232 133 253 166 82 139 185 135
109 149 167 81 123 19 251 247 56 179 29 79 225 238
Texto descifrado: 76 97 32 110 117 101 118 97 32 114 101 97 108 105 100 97 100
Texto descifrado: La nueva realidad
```

- Escriba el método **main()**.

3. Responda las siguientes preguntas:

- ¿Qué significa ECB como modo de ejecución del algoritmo de cifrado?

- ¿Cuál es la ventaja de ECB comparado con CBC?

- ¿Qué es y para qué se necesita el padding?

- ¿Cuánto tiempo tarda cifrar y descifrar el mensaje de entrada? Utilice el siguiente fragmento de código:

```
long tiempoInicial = System.nanoTime();
long tiempoFinal = System.nanoTime();

long tiempo = tiempoFinal - tiempoInicial;
```

4. Cree una clase llamada **Main2** y en ella cree el método **main()** de acuerdo con las siguientes modificaciones al método **main()** inicial y responda:

- Genere dos llaves simétricas, **k1** y **k2**.

- Cifre un mensaje de entrada con la llave **k1** y obtenga un objeto de tipo **byte[]** llamado **tc1** con el mensaje de entrada cifrado.
- Cifre un mensaje de entrada con la llave **k2** y obtenga un objeto de tipo **byte[]** llamado **tc2** con el mensaje de entrada cifrado.
- Descifre **tc1** con **k1**. ¿Qué resultados obtiene?

- Descifre **tc1** con **k2**. ¿Qué resultados obtiene?

- Escriba el método **main()**.

5. Cree una clase llamada **Main3** en ella cree el método **main()** de acuerdo con las siguientes modificaciones al método **main()** inicial y responda:

- Genere una llave secreta y guárdela en un archivo. Utilice el siguiente fragmento de código:

```
archivo = new FileOutputStream(nombreArchivo);  
oos = new ObjectOutputStream(archivo);  
  
oos.writeObject(llave);  
oos.close();
```

- Cifre un mensaje de entrada. Almacene el texto cifrado en un archivo. Utilice el siguiente fragmento de código:

```
archivo = new FileOutputStream(nombreArchivo);  
oos = new ObjectOutputStream(archivo);  
  
oos.writeObject(textoCifrado);  
oos.close();
```

6. Cree una clase llamada **Main4** en ella cree el método **main()** de acuerdo con las siguientes modificaciones al método **main()** inicial y responda

- Recupere la llave que está en el archivo. Utilice el siguiente fragmento de código:

```
archivo = new FileInputStream(nombreArchivo);  
ObjectInputStream ois = new ObjectInputStream(archivo);  
textoCifrado = (byte[]) ois.readObject();  
ois.close();
```

- Recupere el texto cifrado que está en el archivo. Utilice el siguiente fragmento de código:

```
archivo = new FileInputStream(nombreArchivo);
ObjectInputStream ois = new ObjectInputStream(archivo);
llave = (SecretKey) ois.readObject();
ois.close();
```

Parte B – Cifrado Asimétrico

Parte 1: Cifrar y descifrar con cifrado asimétrico

7. Escriba una clase llamada **Asimetrico** en la cual vamos a incluir los métodos **cifrar()** y **descifrar()**.

Cree el método **cifrar()** usando el siguiente código.

```
public static byte[] cifrar(Key llave, String algoritmo, String texto) {
    byte[] textoCifrado;

    try {
        Cipher cifrador = Cipher.getInstance(algoritmo);
        byte[] textoClaro = texto.getBytes();

        cifrador.init(Cipher.ENCRYPT_MODE, llave);
        textoCifrado = cifrador.doFinal(textoClaro);

        return textoCifrado;
    } catch (Exception e) {
        System.out.println("Excepcion: " + e.getMessage());
        return null;
    }
}
```

Cree el método **descifrar()** usando el siguiente código.

```
public static byte [] descifrar(Key llave, String algoritmo, byte[] texto) {
    byte[] textoClaro;

    try {
        Cipher cifrador = Cipher.getInstance(algoritmo);
        cifrador.init(Cipher.DECRYPT_MODE, llave);
        textoClaro = cifrador.doFinal(texto);
    } catch (Exception e) {
        System.out.println("Excepcion: " + e.getMessage());
        return null;
    }

    return textoClaro;
}
```

8. Escriba una clase **Main** para crear una aplicación que use los métodos **cifrar()** y **descifrar()** de la clase **Asimetrico**.

En el método **main()**:

- Utilice la siguiente constante para indicar que utilizará el algoritmo **RSA** para cifrar y descifrar sus mensajes:

```
private final static String ALGORITMO = "RSA";
```

- Reciba por teclado la entrada de un texto.
- Imprima el texto recibido por el teclado.
- Imprima texto claro en **byte []**. Utilice el método **getBytes()** de la clase **String** para convertir el mensaje a **byte[]**. Para imprimir el contenido del **byte[]** utilice el método **imprimir()**.

```
public static void imprimir (byte [ ] contenido) {  
    int i = 0;  
    for (; i < contenido.length - 1; i++) {  
        System.out.print(contenido[i] + " ");  
    }  
    System.out.println(contenido[i] + " ");  
}
```

- Genere un par de llaves asimétricas: una pública y otra privada, las cuales empleará para cifrar y descifrar. Utilice el siguiente fragmento de código:

```
KeyPairGenerator generator = KeyPairGenerator  
    .getInstance(ALGORITMO);  
generator.initialize(1024);  
KeyPair keyPair = generator.generateKeyPair();  
PublicKey llavePublica = keyPair.getPublic();  
PrivateKey llavePrivada = keyPair.getPrivate();
```

- Obtenga un **byte []** con el texto cifrado, invocando al método **cifrar()** de la clase **Asimetrico**. Utilice la llave pública para cifrar.
- Imprima el texto cifrado en **byte []**. Utilice el método **imprimir()**.
- Obtenga un **byte []** con el texto descifrado, invocando al método **descifrar()** de la clase **Asimetrico**. Utilice la llave privada para descifrar.
- Imprima el texto descifrado en **byte []**. Utilice el método **imprimir()**.

- Convierta el **byte[]** con el texto descifrado a **String**. Utilice el método constructor de la clase **String**.

Un ejemplo de la salida en consola de la ejecución del programa se puede ver a continuación:

```
Escriba un mensaje de texto:
#QuedateEnTuCasa
Input en texto plano: #QuedateEnTuCasa

Input en bytes[]:
35 81 117 101 100 97 116 101 69 110 84 117 67 97 115 97

Input cifrado en RSA con Llaves de 1024 bits en byte[]:
70 116 101 111 78 56 77 78 57 66 98 106 103 84 80 117 70 78 116 116 82 101 71 116 47 90 83 66 121
121 105 120 83 110 49 67 79 85 101 86 56 43 108 67 90 112 106 108 106 66 52 87 52 74 106 83 67 5
3 113 108 80 56 78 114 55 48 68 107 71 105 76 108 48 100 78 78 76 69 118 108 55 116 50 52 108 90
110 85 120 72 65 86 51 97 66 72 99 51 120 108 90 83 79 90 67 82 113 77 75 79 80 106 77 82 68 50 1
04 117 112 106 79 112 90 50 117 67 113 57 81 118 90 101 47 122 102 82 52 74 115 48 65 98 106 107
71 103 84 90 122 55 87 116 86 118 65 121 48 103 84 69 111 70 101 83 71 101 121 48 97 73 99 61

Input descifrado en byte[]:
35 81 117 101 100 97 116 101 69 110 84 117 67 97 115 97

Input descifrado convertido a texto plano: #QuedateEnTuCasa
```

- Escriba el método **main()**.

9. Responda las siguientes preguntas:

- ¿Cuál es el propósito del llamado al método **initialize()** con el parámetro **1024**?

- Describa brevemente el funcionamiento del algoritmo RSA.

- ¿Si desea confidencialidad, con cuál llave debe cifrar y con cuál llave debe descifrar?

- ¿Si desea autenticación del emisor, con cuál llave debe cifrar y con cuál llave debe descifrar?

- ¿Cuánto tiempo tarda cifrar y descifrar el mensaje de entrada? Utilice el siguiente fragmento de código:

```
long tiempoInicial = System.nanoTime();  
long tiempoFinal = System.nanoTime();  
  
long tiempo = tiempoFinal - tiempoInicial;
```

10. Cree una clase llamada **Main2** y en ella cree el método **main()** de acuerdo con las siguientes modificaciones al método **main()** inicial y responda:

- Utilice el método **cifrar()** con la llave privada y el método **descifrar()** con la llave pública. Explique el resultado obtenido.

- Escriba el método **main()**.

11. Cree una clase llamada **Main3** en ella cree el método **main()** de acuerdo con las siguientes modificaciones al método **main()** inicial y responda

- Genere en un archivo la llave pública y en otro la llave privada. Utilice el siguiente fragmento de código:

```
archivo = new FileOutputStream(nombreArchivo);  
oos = new ObjectOutputStream(archivo);  
  
oos.writeObject(llave);  
oos.close();
```

- Cifre un mensaje de entrada. Almacene el texto cifrado en un archivo. Utilice el siguiente fragmento de código:

```
archivo = new FileOutputStream(nombreArchivo);  
oos = new ObjectOutputStream(archivo);  
  
oos.writeObject(textoCifrado);  
oos.close();
```


12. Cree una clase llamada **Main4** en ella cree el método **main()** de acuerdo con las siguientes modificaciones al método **main()** inicial y responda

- Recupere la llave pública que está en un archivo. Recupere la llave privada que está en otro archivo. Utilice el siguiente fragmento de código:

```
archivo = new FileInputStream(nombreArchivo);
ObjectInputStream ois = new ObjectInputStream(archivo);
textoCifrado = (byte[]) ois.readObject();
ois.close();
```

- Recupere el texto cifrado que está en el archivo. Utilice el siguiente fragmento de código:

```
archivo = new FileInputStream(nombreArchivo);
ObjectInputStream ois = new ObjectInputStream(archivo);
llave = (SecretKey) ois.readObject();
ois.close();
```

Parte C: Control de Integridad

1. Escriba una clase llamada **Digest** en la cual vamos a incluir el método **getDigest()**.

- Cree el método **getDigest()** usando el siguiente código.

```
public static byte[] getDigest(String algorithm, byte[] buffer) {
    try {
        MessageDigest digest = MessageDigest.getInstance(algorithm);
        digest.update(buffer);
        return digest.digest();
    } catch (Exception e) {
        return null;
    }
}
```

- Imprima el digest obtenido. Utilice el método **imprimirHexa()**.

```
public static void imprimirHexa(byte[] byteArray) {
    String out = "";
    for (int i = 0; i < byteArray.length; i++) {
        if ((byteArray[i] & 0xff) <= 0xf) {
            out += "0";
        }
        out += Integer.toHexString(byteArray[i] & 0xff).toLowerCase();
    }
    System.out.println(out);
}
```

2. Escriba una clase **Main** para crear una aplicación que use el método **getDigest()**.

En el método **main()**:

- Reciba por teclado la entrada de un texto.
- Imprima el texto recibido por el teclado.
- Convierta el **String** a **byte []** usando el método **getBytes()** de la clase **String**.
- Utilizando el método **getDigest()** calcule el digest MD5 del mensaje e imprima el valor. Guarde una imagen de la captura de pantalla.
- Utilizando el método **getDigest()** calcule el digest SHA-1 del mensaje e imprima el valor. Guarde una imagen de la captura de pantalla. Guarde una imagen de la captura de pantalla.
- Utilizando el método **getDigest()** calcule el digest SHA-256 del mensaje e imprima el valor. Guarde una imagen de la captura de pantalla. Guarde una imagen de la captura de pantalla.
- Utilizando la aplicación disponible en la página <http://onlinemd5.com/> calcule y compare los resultados obtenidos. Guarde una imagen de la captura de pantalla.

The screenshot shows a web application titled "MD5 & SHA1 Hash Generator For Text". It has a blue header bar with the title. Below the header, there is a text input field containing the text "Semestre virtual es semestre en pijama". Below the input field, there are three radio buttons for "Checksum type": "MD5" (selected), "SHA1", and "SHA-256". Below the radio buttons, there is a text field labeled "String hash:" containing the value "2BA9D9C1BC6B6392B43A2B50D43CCDCB". At the bottom right, there is a "Calculate" button.

Un ejemplo de la salida en consola de la ejecución del programa con digest MD5 se puede ver a continuación:

```
Escriba un mensaje de texto:
Semestre virtual es semestre en pijama
Mensaje de entrada: Semestre virtual es semestre en pijama
Digest MD5 obtenido: 2ba9d9c1bc6b6392b43a2b50d43ccdc
```

- Desarrolle un método **verificar()** que reciba dos parámetros de tipo **byte []** y determine si su contenido es idéntico o no.
- Escriba el método **main()**.

3. Agregue el método **getDigestFile()** a la clase **Digest** para obtener el digest de un archivo, y cree una nueva clase **Main2** con un método **main ()** en la que se pueda probar.

- Cree el método **getDigestFile()** usando el siguiente código.

```
public static byte[] getDigestFile(String algorithm, String fileName) {
    MessageDigest md = null;
    try {
        md = MessageDigest.getInstance(algorithm);

        FileInputStream in = new FileInputStream(fileName);
        byte [] buffer = new byte [1024];

        int length;
        while ((length = in.read(buffer)) != -1) {
            md.update(buffer, 0, length);
        }
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

    return md.digest();
}
```

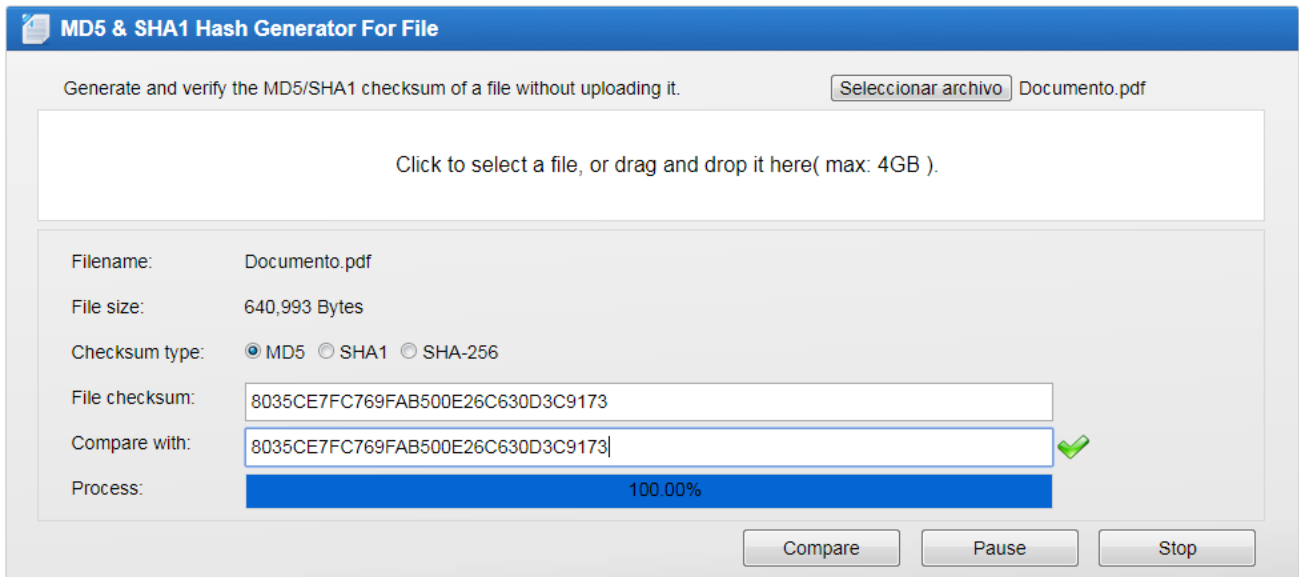
En el método **main()**:

- Reciba por teclado el nombre de un archivo que esté en la carpeta raíz del proyecto.
- Calcule el digest MD5 del archivo e imprima el valor obtenido.

Calcule el digest SHA-1 del archivo e imprima el valor obtenido.

Calcule el digest SHA-256 del archivo e imprima el valor obtenido.


- Utilizando la aplicación disponible en la página <http://onlinemd5.com/> calcule y compare los resultados obtenidos.



MD5 & SHA1 Hash Generator For File

Generate and verify the MD5/SHA1 checksum of a file without uploading it. Seleccionar archivo Documento.pdf

Click to select a file, or drag and drop it here(max: 4GB).

Filename: Documento.pdf
File size: 640,993 Bytes
Checksum type: ☒ MD5 ☐ SHA1 ☐ SHA-256
File checksum: 8035CE7FC769FAB500E26C630D3C9173
Compare with: 8035CE7FC769FAB500E26C630D3C9173 
Process:

100.00%

Compare Pause Stop

Un ejemplo de la salida en consola de la ejecución del programa con digest MD5 se puede ver a continuación:

```
Escriba el nombre del archivo:  
Documento.pdf  
Nombre del archivo: Documento.pdf  
Digest MD5 obtenido: 8035ce7fc769fab500e26c630d3c9173
```

- Escriba el método **main()**.