Comprensión y Seguridad

Tercera Fase

Integrantes del grupo:

Nieves Almodovar Alegria

Elizabeth Itati Balbín Medina

Andrés Fernández Espliguero

Alejandro Gómez López

Jaime Cremades Gomariz

Grupo de prácticas: Martes 11.00 - 13.00

ÍNDICE

1. Contenido del archivo comprimido

Para esta entrega hemos preparado un archivo comprimido, donde encontraremos todo lo necesario para probar nuestra aplicación y observar detalladamente su funcionamiento. Es importante recalcar los distintos elementos:

Un ejecutable, que es el programa en sí mismo, al abrir esta aplicación podremos acceder a la aplicación y todas sus funcionalidades (explicadas en esta memoria).

Una carpeta con el proyecto Java del programa donde se encuentra todo el código de la aplicación. Diferenciamos entre archivos **.java** (código que rige el funcionamiento correcto de la aplicación) y **.fxml** (parte del código que maneja la parte estética y lo que se visualiza por pantalla.

Una carpeta donde se encuentran los documentos de texto "Claves" y "usuarios". Esta carpeta debe incluirse en el directorio "C:" del equipo, ya que será en ella donde se localicen los archivos, tanto encriptados como en claro. Es de vital importancia que la carpeta se encuentre en el directorio "C:" y que los archivos **claves.txt** y **usuarios.txt** existan. Si se borran por error se deben volver a crear con el mismo el nombre y extensión.

Por último, este mismo documento, la memoria del trabajo detallando el el modo de uso y funcionamiento de la aplicación que hemos desarrollado.

2. Manual de usuario

Primero, para poder ejecutar el programa, se requiere del siguiente ejecutable, "CS.jar". Este ejecutable lo podremos encontrar en la entrega del trabajo. Se puede colocar en el Escritorio o en cualquier directorio de nuestro equipo, no afectará al funcionamiento de la aplicación.

Imagen 1

Al darle doble click al icono, se nos abrirá la siguiente ventana de la interfaz. En esta ventana, el usuario podrá iniciar sesión en el sistema, incluyendo las credenciales de nombre de usuario y contraseña.

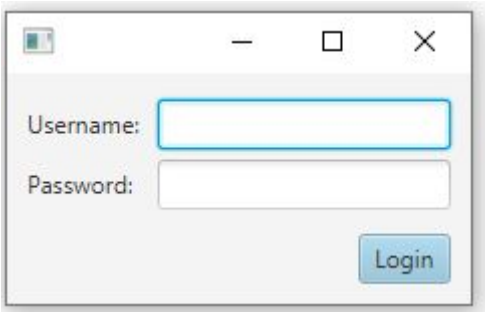

Imagen 2

En el caso de que el usuario no exista en el sistema, se abrirá una ventana emergente de ¡Error! avisando que no se ha podido iniciar sesión (al no existir ese usuario). Se creará un nuevo usuario (si se le da a "Aceptar") **(Imagen 3)** con los parámetros que haya indicado y se accederá a la aplicación.

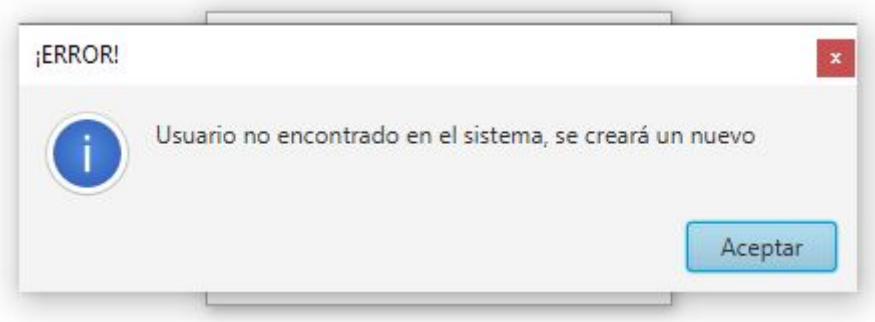

Imagen 3

Al iniciar sesión correctamente o al crear un nuevo usuario, se mostrará la ventana principal de nuestra interfaz. Esta ventana contiene 2 botones.

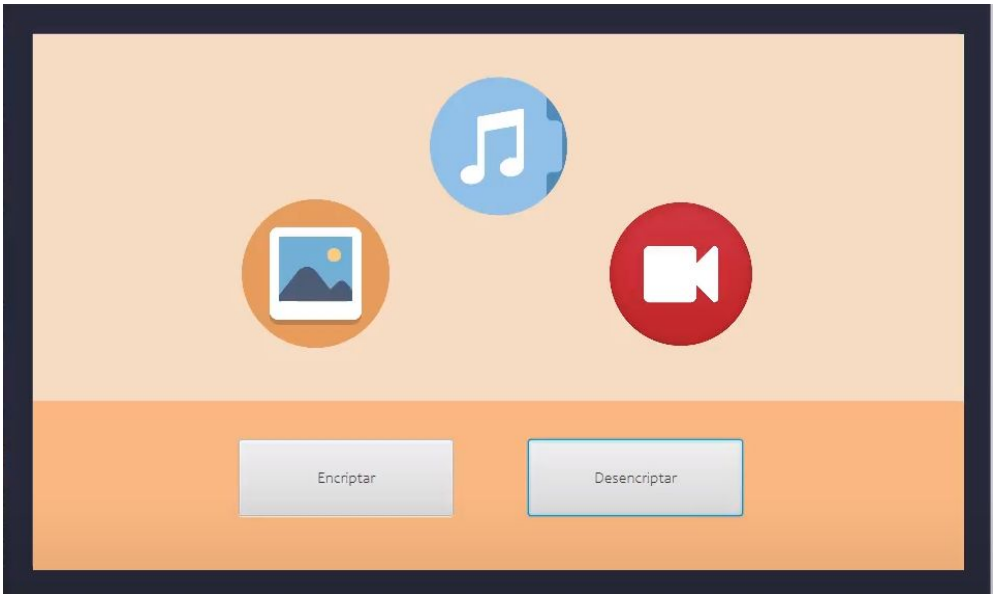

Imagen 4

Si pulsamos el botón "Encriptar", se abrirá una ventana y en él podremos seleccionar cualquier archivo multimedia (imagen, sonido o vídeo) que deseamos encriptar.

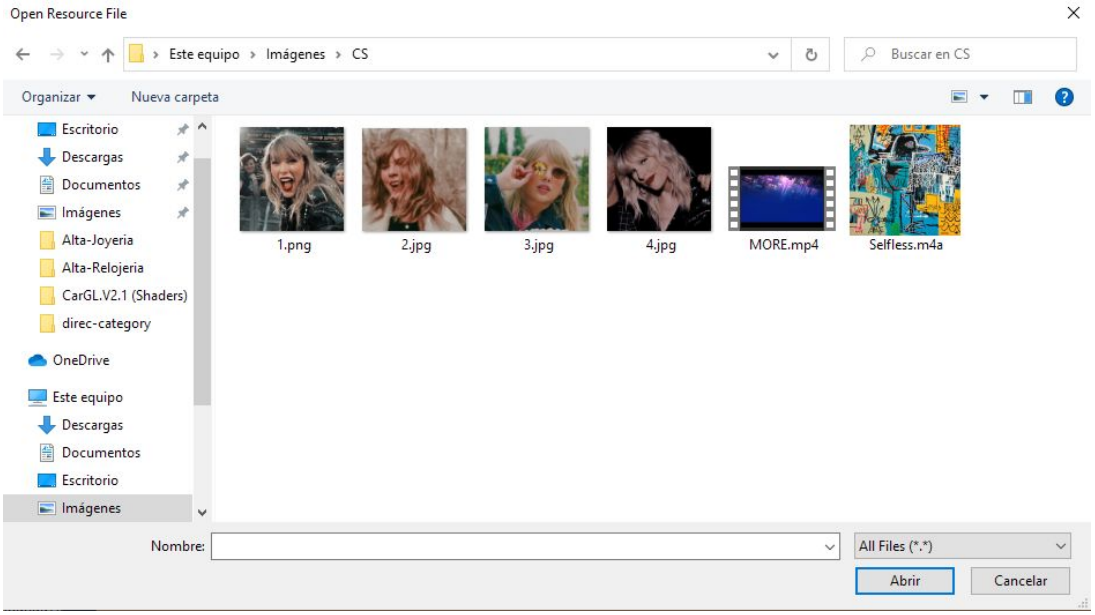

Imagen 5

En este ejemplo se ha seleccionado la imagen "1.png". Como se puede ver a continuación, se ha generado un archivo de la imagen encriptada y un archivo de texto que contiene información que nos servirá para luego poder desencriptarla.

Imagen 6

El archivo de texto se compone de: nombre del fichero, clave privada del usuario encriptada con AES y el usuario del sistema que ha encriptado el archivo.

```
1.png
CtcHzApVZtNOSiXNLL57jQ==
admin
```

Imagen 7

Podemos encriptar otros archivos sin problema y se irán guardando sus claves AES correspondientes en el fichero de Claves, junto con su nombre.

A la hora de desencriptar, simplemente pulsamos el botón "Desencriptar". Se mostrará la siguiente ventana con un menú desplegable con el nombre del archivo encriptado que queramos desencriptar.

Imagen 8

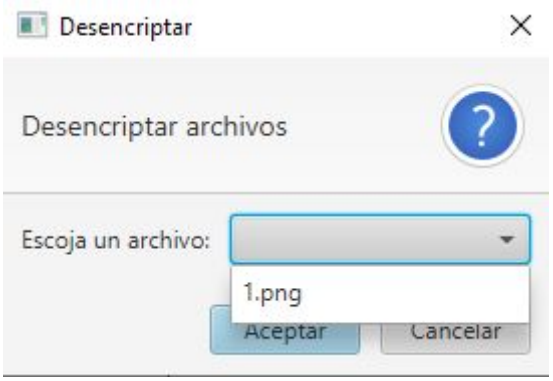

Imagen 9

Después de seleccionar el archivo que queremos desencriptar pulsaremos "Aceptar" y mediante el uso de las claves comentadas anteriormente, obtendremos el archivo desencriptado con un nuevo nombre formado por el nombre del archivo original más la palabra "-nuevo".

Imagen 10

3. Documentación sobre la implementación

3.1 Software y librerías utilizadas

Al igual que en la entrega anterior, hemos decidido continuar con Java como lenguaje de programación usando librerías y funciones correspondientes para la encriptación y desencriptación. Para el desarrollo de la interfaz de la aplicación también continuamos usando JavaFX.

En nuestro código hemos usado una amplia cantidad de librerías, entre las más destacadas estarían algunas como **java.security.Key y java.io.File** que son las más importantes para llevar a cabo la práctica, usamos también librerías derivadas de estas (gestión de ficheros y seguridad), **java.io. FileInputStream, java.io.FileNotFoundException, java.io.FileOutputStream**

Para implementar AES:

javax.crypto.KeyGenerator, javax.crypto.spec.SecretKeySpec, javax.crypto.Cipher y **java.security.NoSuchAlgorithmException**.

Para implementar los algoritmos de encriptación (RSA):

java.security.KeyPair;

java.security.KeyPairGenerator;

java.security.PrivateKey;

java.security.PublicKey;

java.security.SecureRandom;

java.security.KeyFactory;

java.security.SecureRandom;

java.security.interfaces.RSAPrivateKey;

java.security.spec.PKCS8EncodedKeySpec;

java.security.MessageDigest;

Por el nombre de cada librería podemos intuir su funcionamiento pero, en esencia, nos permitirán crear las claves pública y privada de un usuario y realizar todos los pasos de la encriptación RSA.

Como ya se ha comentado, a la hora de desarrollar la interfaz, hemos utilizado **javafx**, y por ende las siguientes librerías:

javafx.application.Application;

javafx.fxml.FXMLLoader;

javafx.scene.Scene;

javafx.scene.control.*;

javafx.scene.control.Button;

javafx.scene.layout.BorderPane;

javafx.event.ActionEvent;

javafx.fxml.FXML;

javafx.scene.text.Text;

javafx.stage.FileChooser;

javafx.stage.FileChooser.ExtensionFilter;

javafx.stage.Stage;

javafx.scene.control.ChoiceDialog;

javafx.scene.control.PasswordField;

javafx.scene.layout.VBox;

javafx.stage.Modality;

Nos han sido de gran utilidad a la hora de desarrollar funcionalidades como: el escoger un archivo (imagen, música, archivos de texto, etc..), mostrar el login de usuario y visualizar los archivos a seleccionar para su desencriptado.

Por último, seguimos usando **java.util.Base64** para pasar la clave AES a base64 a la hora de guardarla en el fichero de claves.

3.2 Descripción del programa

Vamos a comenzar explicando a grandes rasgos el funcionamiento de nuestra aplicación. Con nuestro servicio seguro de contenidos multimedia, un usuario podrá registrarse o iniciar sesión. En caso de que ya haya un usuario con el mismo nombre, se comprobará que la contraseña introducida sea similar a la que tenemos almacenada, en ese caso el inicio de sesión será correcto. Si el nombre de usuario no está registrado, se creará una Clave Privada y Pública únicas para ese usuario y serán almacenadas junto a su nombre y contraseña. Es importante recalcar que todos los datos personales del usuario serán almacenados tras ser protegidos. La contraseña la encriptamos empleando *Salt* para posteriormente aplicarle un hashing y almacenarla. Tanto la clave privada y pública son encriptadas con el algoritmo AES y la contraseña del usuario (MessageDigest con SHA-512).

Al haber accedido correctamente a la aplicación, inicializamos las variables globales pertinentes al usuario que ha iniciado sesión. Las leemos del .txt donde las tenemos almacenadas. En el caso de la clave Privada y Pública tendremos que desencriptarlas al leerlas (mismo proceso que al encriptar pero con el cipher en DECRYPT_MODE). De esta forma para todos los procesos posteriores que hagamos, tendremos ya todas las variables que necesitamos inicializadas.

Ahora bien, ¿qué pasa si un usuario quiere subir un archivo multimedia? En este caso se pasará el contenido del archivo a bytes, estos bytes generarán la clave del archivo mediante una encriptación AES. La clave de este archivo será encriptada a su vez con el algoritmo RSA, empleando la clave Pública del usuario que se encuentra logueado (es decir, quien ha subido el archivo). Escribiremos la clave, junto con el nombre usuario y el nombre del archivo en nuestro .txt de archivos encriptados (claves.txt). También guardaremos en el mismo directorio un "**nombrearchivo-encripted.formato**", este será el archivo multimedia que utilizaremos para la desencriptación (si se intenta abrir no mostrará nada). El hacer esto nos reporta una ventaja primordial, que a parte de desencriptar el archivo para que un nuevo usuario lo obtenga, si el usuario que ha encriptado el archivo lo perdiera, podrá recuperarlo gracias a nuestro programa. Una vez hemos encriptado el archivo, estará siempre disponible en el selector de desencriptar.

Cuando queramos desencriptar un archivo pueden darse dos situaciones: que el usuario que lo quiera desencriptar sea el mismo que lo ha subido o que sea uno distinto. En el primer caso, recuperaremos la clave encriptada del archivo y con la Clave Privada que tenemos inicializada en variable global. Se hará el proceso inverso que cuando encriptamos. Desencriptamos la clave del archivo con RSA y la Clave Privada, y posteriormente con AES, después generamos el archivo. En el segundo caso, leeremos del fichero claves.txt el nombre de usuario que subió el archivo. Buscaríamos su Clave Privada y contraseña para poder hacer la desencriptación.

Como parte adicional, hemos incluido en nuestro código una segunda encriptación para simular de la manera más fidedigna posible el intercambio de archivos entre dos usuarios. El protocolo normal, el usuario que desea desencriptar el archivo manda una petición junto con su clave Pública al usuario propietario del archivo. Este si desea que el otro usuario reciba el archivo, lo desencripta con su clave Privada y lo encripta con la clave Pública

del otro usuario. Envía el archivo encriptado y el usuario que ha hecho la petición podrá desencriptar con su clave privada y visualizarlo. En nuestra aplicación cuando un usuario que no ha encriptado el archivo quiere desencriptar, una vez obtenga la clave del archivo, volverá a hacer una encriptación con sus claves para intentar asemejar el protocolo convencional.

Una vez ya sabemos qué pasos sigue el programa que hemos desarrollado, vamos a analizar en profundidad sus funciones y archivos.

El código está dividido en varias clases: **DecryptController.java SamplerController.java, LoginController.java, LoginDemoApplication.java, LoginManager.java, MainViewController.java** y los archivos .fxml. En los archivos .fxml están definidas todas las estructuras, en lenguaje fxml, de nuestra interfaz gráfica. El lenguaje fxml está basado en XML, pero desarrollado especialmente por Oracle para trabajar con JavaFX. En Main.java nos aseguramos de que nuestra interfaz de usuario, definida en Sample.fxml, se cargue correctamente y se muestre en la pantalla.

En **LoginController.java**, nos aseguramos de que el inicio de sesión y registro de usuarios se haga de manera correcta. Para ello hemos creado una serie de funciones que caben destacar:

-initManager(final LoginManager loginManager)

```
public void initManager(final LoginManager loginManager) {
  loginButton.setOnAction(new EventHandler<ActionEvent>() {

    @Override public void handle(ActionEvent event) {
      String sessionID = authorize();
      if (sessionID != null) {
        loginManager.authenticated(sessionID);

      //Para cerrar la ventana actual
        Stage stage = (Stage) loginButton.getScene().getWindow();
        SampleController.setScene(loginButton.getScene());
        stage.close();
      }
    }
  });
}
```

Función principal del LoginController. Se encarga de autenticar el usuario, es decir, si existe el usuario, se autoriza su acceso y se crea una sesión. Una vez iniciada la sesión, se cierra la ventana de Login que estaba abierta.

-crearUsuario()

```
private void crearUsuario() {
    String userpass = encriptasalt(password.getText());
    String textToAppend = user.getText() + "\n" + userpass + "\n";
    SampleController.user = user.getText();

    try {
        existeu = existep = true;
        SecureRandom sos = new SecureRandom();
        KeyPairGenerator gen;
        try {
            gen = KeyPairGenerator.getInstance(RSA);
             gen.initialize(512, sos);
             KeyPair keyPair = gen.generateKeyPair();
             PublicKey uk = keyPair.getPublic();
             PrivateKey rk = keyPair.getPrivate();
             System.out.println(sos);
             System.out.println("UK " + uk);
             System.out.println("RK: " +rk);
             String claveencr = encryptPrivateKeyAES(rk);
             System.out.println("Clave privada ejemplo:" + claveencr);
             String devuelveukg = encryptPublicKeyAES(uk);
             textToAppend += claveencr + "\n"+devuelveukg+ "\n";
             Files.write(Paths.get("C:\\ApplicationCS\\usuarios.txt"), textToAppend.getBytes(), StandardOpenOption.APPEND);

        } catch (NoSuchAlgorithmException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }catch (IOException e) {
        //exception handling left as an exercise for the reader
    }
}
```

Si el usuario no existe previamente, generará sus claves y se encargará de guardar sus datos en "usuarios.txt".

-encryptPublicKeyAES() y encryptPrivateKeyAES()

```
public String encryptPublicKeyAES(PublicKey keyrk) {
    Cipher cipher;
    String myKey = password.getText(); //CONTRASEÑA
    byte[] encrypted = null;

    MessageDigest sha = null;
    try {
        key = myKey.getBytes("UTF-8");
        sha = MessageDigest.getInstance("SHA-512");
        key = sha.digest(key);
        key = Arrays.copyOf(key, 16);
        secretKey = new SecretKeySpec(key, "AES");

        try {
            cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            encrypted = cipher.doFinal(keyrk.getEncoded());

            return Base64.getEncoder().encodeToString(encrypted);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
  public String encryptPrivateKeyAES(PrivateKey keyrk) {
}     Cipher cipher;
      String myKey = password.getText(); //CONTRASEÑA
      byte[] encrypted = null;

      MessageDigest sha = null;
      try {
          key = myKey.getBytes("UTF-8");
          sha = MessageDigest.getInstance("SHA-1");
          key = sha.digest(key);
          key = Arrays.copyOf(key, 16);
          secretKey = new SecretKeySpec(key, "AES");

          try {
              cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
              cipher.init(Cipher.ENCRYPT_MODE, secretKey);
              encrypted = cipher.doFinal(keyrk.getEncoded());

              return Base64.getEncoder().encodeToString(encrypted);
          } catch (Exception e) {
              e.printStackTrace();
          }
      }
      catch (NoSuchAlgorithmException e) {
          e.printStackTrace();
      }
      catch (UnsupportedEncodingException e) {
          e.printStackTrace();
      }
      return Base64.getEncoder().encodeToString(encrypted);
  }
```

Se encargan de encriptar las claves del usuario con AES y devolver el string correspondiente de cada una de ellas a la función crearUsuario().

En el archivo **SamplerController.java** se controla las interacciones del usuario con la interfaz. En esta versión tenemos dos botones, cuando pulsemos el de "Encriptar" se disparará un evento que nos permitirá seleccionar el archivo multimedia que queremos encriptar, gracias a la clase ***FileChooser*** que ofrece JavaFX.

El botón de "Desencriptar" abre una ventana desde la cual elegimos un archivo a desencriptar. En **SampleController.java** nos encargamos de ir leyendo los archivos txt necesarios según el proceso que se esté llevando a cabo. Esto nos permite recuperar los datos necesarios para los procesos de encriptado y desencriptado. El SampleController.java irá llamando a las funciones implementadas en **ImagenEncDec.java**, que son las encargadas de encriptar y desencriptar los datos que les pasemos en los parámetros.

-getFile(): recibe el fichero que el usuario ha seleccionado y lee su contenido en bytes independientemente del tipo que sea (imagen, audio, video, etc.) y devuelve esos bytes o contenido para su uso posterior.

```
public static byte[] getFile(File ruta) {
    File f = ruta;
    InputStream is = null;
    try {
        is = new FileInputStream(f);
    } catch (FileNotFoundException e2) {
        // TODO Auto-generated catch block
        e2.printStackTrace();
    }

    byte[] content = null;

    try {
        content = new byte[is.available()];
    } catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    try {
        is.read(content);
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return content;
}
```

-**encryptFile()**: recibe una clave generada anteriormente y los bytes de contenido del fichero del usuario. En esencia, esta función crea un cipher para encriptación AES y lo inicializa con la clave pasada para después encriptar el contenido del fichero pasado y devolver ese contenido encriptado.

```
public static byte[] encryptFile(Key key, byte[] content) {
    Cipher cipher;
    byte[] encrypted = null;
    try {
        cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, key);
        encrypted = cipher.doFinal(content);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return encrypted;

}
```

-**decryptFile()**: de manera similar a la anterior función, esta recibe la clave y el contenido encriptado para crear un cipher que desencripte dicho contenido y devolverlo.

```
public static byte[] decryptFile(Key key, byte[] textCryp) {
    Cipher cipher;
    byte[] decrypted = null;
    try {
        cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.DECRYPT_MODE, key);
        decrypted = cipher.doFinal(textCryp);
    } catch (Exception e) {
        e.printStackTrace();
    }

    return decrypted;
}
```

-**saveFile()**: en esta última función únicamente se coge el contenido sin encriptar y se crea un archivo copia del original dado por el usuario en el mismo directorio donde se encuentre este. La copia que se cree siempre se llamará igual que el original pero añadiendo "-nuevo" al final.

```
public static void saveFile(byte[] bytes, File ruta) throws IOException {

    String ruta2 = ruta.getAbsolutePath();
    ruta2.substring(0,ruta2.length()-10);
    String [] partes = ruta2.split("\\.");
    partes[0] = partes[0].substring(0,partes[0].length()-10);
    partes[0] = partes[0] + "-nuevo";
    String fin = partes[0] + "." + partes[1];
    FileOutputStream fos = new FileOutputStream(fin); //ruta
    fos.write(bytes);
    fos.close();
}
```

-saveEncrypted(): hace prácticamente lo mismo que saveFile pero guarda un archivo encriptado, añadiendo al final "-encrypted". El archivo se guarda también en el mismo directorio que el original. Lo necesitaremos posteriormente para la desencriptación.

```
public static void saveEncrypted(byte[] bytes, File ruta) throws IOException {

    String ruta2 = ruta.getAbsolutePath();
    String [] partes = ruta2.split("\\.");
    partes[0] = partes[0] + "-encrypted";
    String fin = partes[0] + "." + partes[1];

    FileOutputStream fos = new FileOutputStream(fin); //ruta
    fos.write(bytes);
    fos.close();
}
```

-generateKey(): se encarga de crear e inicializar las variables globales del usuario que ha iniciado sesión, leyéndolas desde el archivo usuarios.txt. Para identificar cual es el usuario indicado se le pasa por parámetro el nombre de usuario.

```
public static void generateKey(String ruta, String usuario) throws Exception{
    String ejemplo = ruta.substring(0, ruta.lastIndexOf("\\"));

    UserPublic = ejemplo + "\\usuarios.txt";

    //FASE3 leer el txt de usuarios para no sobreescribir la informacion al guardar uno nuevo
    String texto = "";
    boolean a = false;

    try(BufferedReader br = new BufferedReader(new FileReader(UserPublic))) {
            String line = br.readLine();

            while (line != null ) {
                if(line.equalsIgnoreCase(usuario)) {
                    usuariologged = usuario;
                    line = br.readLine();//contraseña
                    contraseñauser = line;
                    line = br.readLine();//privada
                    rk = decryptAESprivatekey(line);
                    line = br.readLine();//publica
                    uk = decryptAESpublickey(line);
                    a = true;
                }else {
                    line = br.readLine(); //USUARIO
                    line = br.readLine();
                    line = br.readLine();
                    line = br.readLine();
                }
             }

            br.close();
    }
}
```

-**decryptAESprivatekey()** y **decryptAESpublickey()**: estos dos métodos leerán las claves almacenadas en usuarios.txt y se encargarán de crear las estructuras de Private Key y Public Key del usuario logueado. Las devolverán a la función generateKey() para que puedan inicializar las variables globales.

```
public static PrivateKey decryptAESprivatekey(String claveS) throws Exception
{
    System.out.println("************************* EMPIEZA ************************************");
    System.out.println(claveS);
    byte[] Clavebyte = Base64.getMimeDecoder().decode(claveS);
     Cipher cipher;
     byte[] decrypted = null;

     String myKey = contraseñauser; //CONTRASEÑA
     MessageDigest sha = null;

     try {
         key = myKey.getBytes("UTF-8");
         sha = MessageDigest.getInstance("SHA-512");
         key = sha.digest(key);
         key = Arrays.copyOf(key, 16);
         secretKey = new SecretKeySpec(key, "AES");
         System.out.println(secretKey);
```

```
            try {
                cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
                cipher.init(Cipher.DECRYPT_MODE, secretKey);
                System.out.println(cipher);
                decrypted = cipher.doFinal(Clavebyte);

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }

        PKCS8EncodedKeySpec keySpec = new PKCS8EncodedKeySpec(decrypted);
        KeyFactory kf = KeyFactory.getInstance("RSA");
        PrivateKey privKey = kf.generatePrivate(keySpec);

        return privKey;
    }
public static PublicKey decryptAESpublickey(String claveS) throws Exception
{
    byte[] Clavebyte = Base64.getMimeDecoder().decode(claveS);
     Cipher cipher;
     byte[] decrypted = null;

     String myKey = contraseñauser; //CONTRASEÑA
     MessageDigest sha = null;

     try {
         key = myKey.getBytes("UTF-8");
         sha = MessageDigest.getInstance("SHA-2");
         key = sha.digest(key);
         key = Arrays.copyOf(key, 16);
         secretKey = new SecretKeySpec(key, "AES");

         try {
             cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
             cipher.init(Cipher.DECRYPT_MODE, secretKey);
             decrypted = cipher.doFinal(Clavebyte);

         } catch (Exception e) {
             e.printStackTrace();
         }
     }
     catch (NoSuchAlgorithmException e) {
         e.printStackTrace();
     }
     catch (UnsupportedEncodingException e) {
         e.printStackTrace();
     }

    PKCS8EncodedKeySpec keySpec = new PKCS8EncodedKeySpec(decrypted);
    KeyFactory kf = KeyFactory.getInstance("RSA");
    PublicKey pubkey = kf.generatePublic(keySpec);

    return pubkey;
}
```

-**BuscarClavePrivUsuarioExt()**, devuelve a la función decrypt RSA la clave privada del usuario que ha subido el archivo que queremos desencriptar

```
public static String BuscarClavePrivUsuarioExt(String usuarioext) throws IOException {
    String claveencontrada = null;
    boolean stop = false;

    try(BufferedReader reader = new BufferedReader(new FileReader(UserPublic))) {
            String line = reader.readLine();

        while (line != null && stop == false) {
            if(line.equalsIgnoreCase(usuarioext)) {
                line = reader.readLine();//linea con la contraseña
                line = reader.readLine();//linea con la clave privada
                claveencontrada = line;
                stop = true;//A deja de leer
            }else {
                line = reader.readLine();//linea con la contraseña
                line = reader.readLine();//linea con la clave privada
                line = reader.readLine();//linea con la clave publica
                line = reader.readLine();//linea con el siguiente usuario a comprobar
            }
        }
        reader.close();
    }

    return claveencontrada;
}
```

-**encryptRSA()** y **decryptRSA()**: se le pasa una variable por parámetro de tipo byte[ ] que será el contenido a encriptar o desencriptar. ***encryptRSA*** es un método muy simple que encripta el contenido pasado por parámetro en RSA de forma directa, en cambio en ***decryptRSA*** observamos cómo leemos la clave privada del fichero KeyPrivate y luego tenemos que desencriptar el AES de esa clave leída del fichero. Una vez hemos hecho esto ya podemos desencriptar en RSA y devolver el resultado.

```
public static byte[] encryptRSA(byte[] text) throws Exception
{
    Cipher cipher = Cipher.getInstance(RSA);
    cipher.init(Cipher.ENCRYPT_MODE, uk);

    return cipher.doFinal(text);
}

public static byte[] decryptRSA(byte[] text, String usuarioext) throws Exception{
    String leeprivada = BuscarClavePrivUsuarioExt(usuarioext);
    System.out.println(leeprivada);

    PrivateKey devuelta = decryptAESprivatekey(leeprivada);
    Cipher cipher = Cipher.getInstance(RSA);
    cipher.init(Cipher.DECRYPT_MODE, devuelta);

    System.out.println("ESTO COMPROBAR RK" + devuelta);

    return cipher.doFinal(text);
}
```

4. Protocolo de seguridad

El protocolo de seguridad constituye la transferencia de mensajes cifrados para cumplir como objetivo el intercambio de claves.

Los protocolos criptográficos se utilizan ampliamente para el transporte seguro de datos a nivel de aplicación, además, permiten a los agentes (en este caso usuario y administrador) autenticarse entre sí, establecer claves de sesión nuevas para la comunicación confidencial y garantizar la autenticidad de los datos y los servicios.

Es por ello que en este apartado se describirán aquellos algoritmos criptográficos con su respectiva notación empleados en el proyecto.

Protocolo

Se dispone de una interfaz en la que se habrá de registrar mediante un usuario y una contraseña para luego poder acceder mediante el apartado de Login.

Una vez realizado el registro,se guardará ese nuevo usuario en el fichero **usuarios.txt**, donde almacenamos los usuarios con su contraseña, la clave privada (cifrada con AES con contraseña) y la clave pública encriptada.

Cada vez que el sistema registra un nuevo usuario, se generará una clave tanto pública como privada asociada a este usuario que estará almacenada en un fichero .txt.

El usuario tendrá que iniciar sesión en el sistema y, cuando lo haga, tendrá la opción de subir un archivo multimedia que posteriormente podrá Encriptar o Desencriptar, en esta última fase, para dar más seguridad al programa, encriptamos nuestra contraseña mediante SALT, en concreto con un SHA-512.

Cuando se sube el archivo multimedia. al darle a "Encriptar" en nuestra interfaz, se genera una clave AES por cada archivo subido. Esta clave AES se almacena en el fichero **claves.txt** con el siguiente formato:

Nombre del archivo
Clave AES
Nombre de usuario

Con la clave pública generada para el usuario, lo que se hará es encriptar aquellas claves encriptadas creadas por los archivos multimedia en RSA para una mayor seguridad. Esta clave es la que almacenaremos finalmente en el archivo **claves.txt**

Las claves públicas y privadas se almacenarán a su modo en el fichero **usuarios.txt**. Por lo tanto, cuando se desee desencriptar un archivo, primero se cogerá la clave privada (se leerá y desencriptará) asociada al usuario y con ella podremos desencriptar la clave RSA del archivo multimedia en cuestión y posteriormente su clave AES.

Algoritmos empleados

Algoritmo simétrico: El algoritmo elegido para este proyecto es AES-128, que significa que se cifra por bloques de 128 bits. Es uno de los algoritmos de cifrado más utilizados y seguros que existe actualmente.

Algoritmo asimétrico: El algoritmo elegido es RSA, con una longitud de clave de 1024 bits dado que es la longitud idónea para nuestros archivos multimedia. A diferencia de AES, RSA trabaja con dos claves diferentes: una pública y una privada. Empleamos este algoritmo ya que es mucho más seguro que el algoritmo simétrico AES.

Notación de protocolo

En esta sección se presenta el siguiente protocolo de distribución de claves. El protocolo tiene dos componentes independientes. Se compone de Usuario A y Fichero File.

Usuario A

- Clave privada del usuario A: KA_{priv}^{RSA}
- Clave pública del usuario A: KA_{pub}^{RSA}

Fichero File

- Clave del fichero: K_{File}^{AES}
- Contraseña del usuario A: A_{pwd}

Encriptaciones, desencriptaciones y descifrado

- Encriptación del archivo multimedia con AES: $E_{K_{File}^{AES}}^{AES}(File)$
- Encriptación de la clave del archivo con clave pública: $E_{KA_{pub}^{RSA}}^{RSA}(K_{File}^{AES})$
- Encriptación de la contraseña del usuario mediante SALT: $E_{Apwd}^{SHA-512}(Password)$
- Desencriptación del archivo multimedia utilizando clave privada del usuario: $D_{KA_{priv}^{RSA}}^{RSA}[E_{KA_{pUB}^{RSA}}^{RSA}(K_{File}^{AES})] = K_{File}^{AES}$
- Descifrado del archivo multimedia utilizando contraseña del usuario previamente encriptada : $D_{Upwd}^{AES}[E_{A_{pwd}}^{AES}(KA_{priv}^{RSA})] = KA_{priv}^{RSA}$

5. Bibliografía

https://www.eclipse.org/efxclipse/install.html

https://medium.com/@osmandi/qu%C3%A9-es-javafx-41cfc327bdf2

https://howtodoinjava.com/java/java-security/java-aes-encryption-example/

http://www.java2s.com/Tutorial/Java/0490__Security/UsingCipherOutputStream.htm

https://www.jackrutorial.com/2018/07/how-to-encrypt-and-decrypt-files-in-java10.html

https://www.oracle.com/java/technologies/javafxscenebuilder-1x-archive-downloads.html