Universitat d'Alacant
Universidad de Alicante

Comprensión y Seguridad

Segunda 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. Software y librerías utilizadas

Al igual que en la anterior entrega, 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 nuevos algoritmos de encriptación de esta entrega (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.

2. Manual de usuario

Primero que todo, tenemos el ejecutable, "CS.jar".

Imagen 1

Al darle doble click, se nos abrirá la siguiente ventana de la interfaz (nótese que se ha modificado con respecto a la anterior entrega donde únicamente aparece el botón "Encriptar").

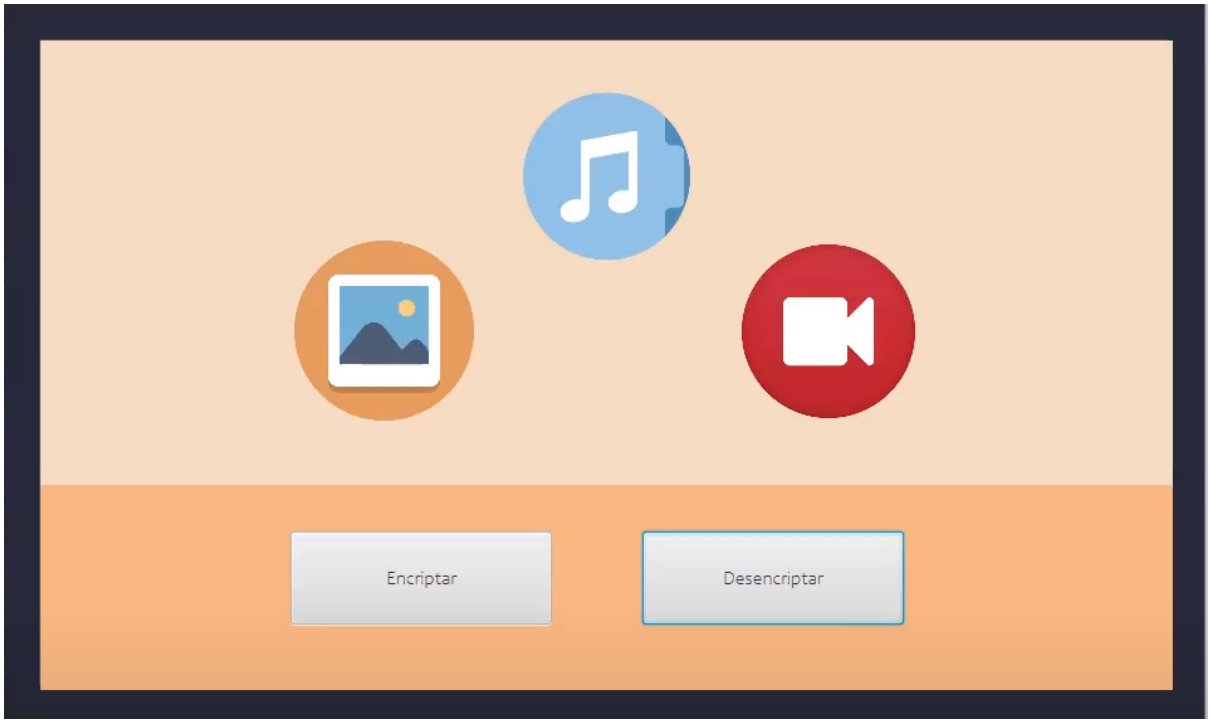

Imagen 2

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

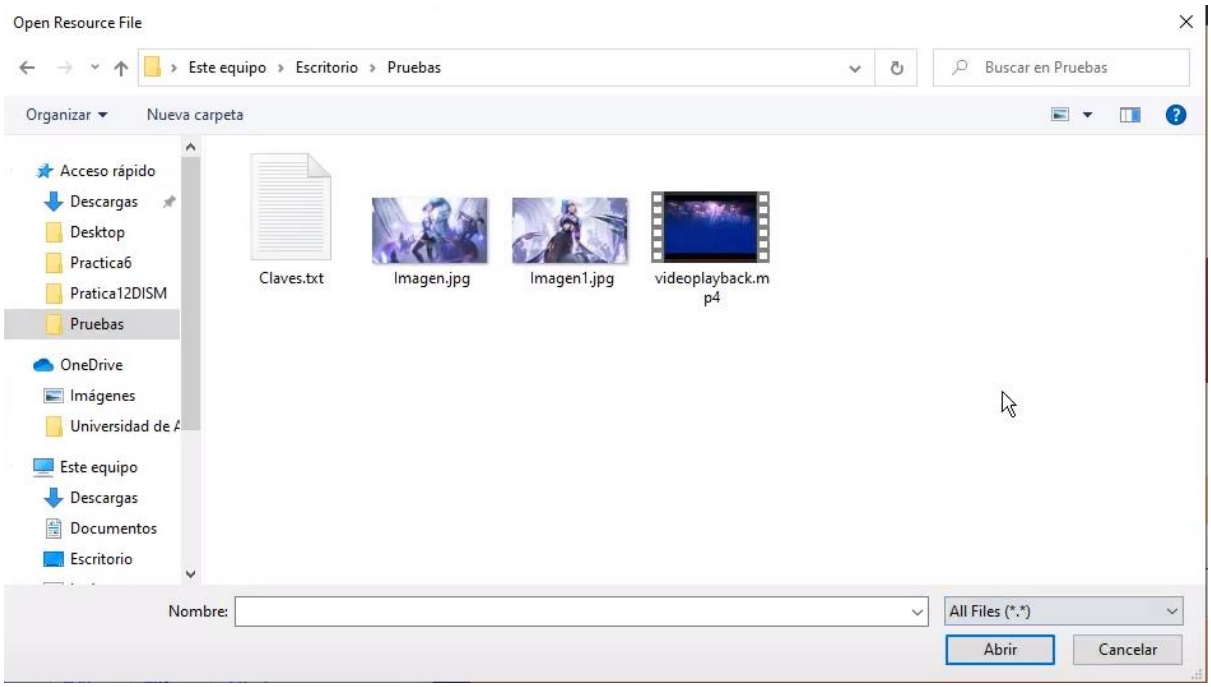

Imagen 3

En este ejemplo hemos pulsado imagen 1. Como se puede ver a continuación, se ha generado un archivo de la imagen encriptada y dos archivos de texto que contienen uno la clave pública y otro la privada del usuario, según sus nombres indican (Es necesario que exista un archivo txt vacío tal y como se muestra en la imagen 3. Además, cabe destacar que la clave privada se guardará en su fichero de texto encriptada con AES).

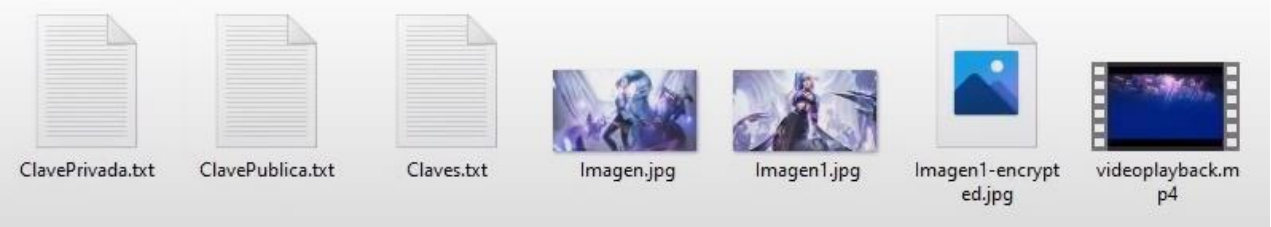

Imagen 4

Podemos encriptar otros archivos sin problema y se irán guardando sus claves AES correspondientes el 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 5

Después de seleccionar el archivo que queremos desencriptar pulsamos "Aceptar" y mediante el uso de las claves comentadas anteriormente, obtendremos el archivo desencriptado.

Imagen 6

3. Detalles sobre la implementación

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 el archivo **SamplerController.java**, controlamos las interacciones del usuario con la interfaz. En esta segunda versión pasamos a tener dos botones, "Encriptar", que cuando pulsemos este botón, se "disparará" un evento que nos permitirá seleccionar el archivo multimedia que queremos encriptar, gracias al elemento **FileChooser**. (ver imagen 3), y "Desencriptar" el cual abre una ventana

desde la cual elegimos un archivo a desencriptar, el sampleController se encarga de buscar en el fichero claves cuál es la clave del archivo para ese nombre dado y finalmente hace uso de las técnicas de desencriptado implementadas.

Pasamos a generar la clave con la que encriptamos el archivo. Usaremos el tipo de encriptación AES junto al RSA. Para ello creamos una clave pública y una privada, con la clave pública cogeremos las claves encriptadas en AES y las encriptamos con RSA para una mayor protección.

Luego guardamos esas claves pública y privada en un txt, con la diferencia de que la pública la guardaremos tal cual pasada a BASE64 para guardarla en el fichero. y la privada la encriptamos con AES para que esté más protegida. De este modo cuando seleccionemos en nuestro programa un archivo a desencriptar, leeremos la clave privada y la desencriptaremos (ya que estaba encriptada con AES) y ya tendremos la clave privada para poder desencriptar el RSA, y de esta forma el AES, como ya hacíamos en la primera entrega.

Como paso final, desencriptamos el archivo multimedia y lo guardamos en la misma carpeta que se encontraba el original.

En concreto en el archivo **ImageEncDec.java** encontramos varias funciones claves:

getFile, que 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.

encryptFile, que 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.

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.

saveFile, en esta última función únicamente se coge el contenido sin ecriptar 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.

saveEncrypted, que hace prácticamente los mismo que saveFile pero guarda un archivo encryptado, añadiendo al final "-encrypted", el archivo se guarda también en el mismo directorio que el original.

El método generateKey, se encarga de crear un par de claves, una pública y otra privada y de escribirlas en sus respectivos ficheros.txt para una ruta determinada que se la pasa por parámetro. El par de claves se genera mediante RSA.

encryptPrivateKeyAES, es el método que necesitamos para encriptar la clave privada a AES y de esta forma aumentar la protección. Este método a partir de la contraseña del usuario y la clave privada que se le pasa por parámetro se encripta mediante AES y se devuelve.

decryptAESprivatekey, se encarga de desencriptar mediante AES, la clave privada encriptada anteriormente, esto se hace gracias a un string pasado por parámetro que será la clave de desencriptado, de esta forma usaremos el método Base64.getMimeDecoder().decode para obtener el resultado que queremos.

Y por último los métodos RSA, encryptRSA y decryptRSA a los que se le pasa una variable 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.

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 el que se habrá que registrar mediante un usuario y una contraseña para luego poder acceder mediante el apartado de Login.

Una vez realizado el registro, 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. Cuando este archivo multimedia se suba en el sistema, se generará una clave tanto pública como privada asociada a este usuario que estará almacenada en un fichero .txt.

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 un fichero .txt con el siguiente formato:

```
NombreDelArchivo
ClaveAES
```

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.

Las claves públicas y privadas se almacenarán a su mismo modo en un fichero .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 y RSA: $E_{K_{File}^{AES}}^{AES}(File)/E_{K_{File}^{RSA}}^{RSA}(File)$
- Encriptación de la clave del archivo con clave pública: $E_{KA_{pub}^{RSA}}^{RSA}(K_{File}^{AES})$
- 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:
  $D_{Upwd}^{AES}[E_{A_{pwd}}^{AES}(KA_{priv}^{RSA})] = KA_{priv}^{RSA}$