



Al-Imam Muhammad Ibn Saud Islamic University

College of Computer and Information Sciences

Department of Computer Science

CS432 Final Project: XSecurity System Project

By:

Saleh Khalid alsaeed (439016667)

Rayan Khalid almengash (439016742)

Salem Hadi alqahtani (439015967)

Abdulelah Mohammed alanzi (439015929)

Riyad Khalid alkhudhayr (439013717)

Supervisor: Dr. Milad tezeghdanti

April 30, 2022 / 29th Ramadan, 1443h

Table of Contents

1	Overview of the project design	1
2	Approach and steps to implementation	1
3	Code	2
3.1	Java imports	2
3.2	Generating AES keys	2
3.3	Loading the RSA Private Key	3
3.4	Save the AES Key	3
3.5	Write the Initialization Vector	3
3.6	Encrypting the File Contents using the AES Key	3
3.7	Decrypting the File using the RSA Private Key	4
3.8	Load the RSA Public Key from File	4
3.9	Load the AES Secret Key	4
3.10	Read the Initialization Vector	5
3.11	Decrypt the File Contents	5
4	Design and testing	7
5	Challenges	19
6	Team Members	19
7	References	19

1 Overview of the project design

The solution to this problem is to encrypt with a symmetric algorithm like AES while encrypting the AES secret key itself with RSA. The initialization vector (IV) must also be communicated to the receiver in order for the message to be decrypted when using AES encryption.

We'll generate an AES key, encrypt the file with the AES key, and encrypt the AES key with RSA in this case. The encrypted AES key is placed at the top of the output file, followed by the initialization vector (IV), and finally the AES-encrypted file data. The output file, rather than the three separate components, can be delivered to the receiver this way.

2 Approach and steps to implementation

We generate an AES key, then encrypt the message, then encrypt the symmetric key produced, which will produce a ciphertext stored in a file, and the decryption method is the same but in reverse, we used the Netbeans IDE's GUI drag and drop feature for design and java cryptography libraries for encryption and decryption.

3 Code

3.1 Java imports

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.nio.file.Files;
import java.nio.file.Paths;
import javax.crypto.SecretKey;
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileSystemView;
import java.security.KeyFactory;
import java.security.PrivateKey;
import java.security.SecureRandom;
import java.security.spec.PKCS8EncodedKeySpec;
import java.util.Base64;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.spec.IvParameterSpec;
import javax.swing.JOptionPane;
```

3.2 Generating AES keys

First step is to generate the AES key which will be used for the encryption. We generate a key of size 128 bits.

```
KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
kpg.initialize(2048);
KeyPair kp = kpg.generateKeyPair();
```

We also need the initialization vector of the same size as the key, which we generate as follows (along with the required *IvParameterSpec*):

```
byte[] iv = new byte[128 / 8];
srandom.nextBytes(iv);
IvParameterSpec ivspec = new IvParameterSpec(iv);
```

3.3 Loading the RSA Private Key

Load the RSA private key from file using the appropriate class.

```
byte[] bytes = Files.readAllBytes(Paths.get(pvtKeyFile));
PKCS8EncodedKeySpec ks = new PKCS8EncodedKeySpec(bytes);
KeyFactory kf = KeyFactory.getInstance("RSA");
PrivateKey pvt = kf.generatePrivate(ks);
```

3.4 Save the AES Key

As mentioned above, the first part of the output file contains the encrypted AES key. For encryption, we use the *Cipher* class with the RSA public key and save it as follows:

```
try (FileOutputStream out = new FileOutputStream(Keys_Folder_Path + "\\ " + "CiperFile.enc")) {
    {
        Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        cipher.init(Cipher.ENCRYPT_MODE, pvt);
        byte[] b = cipher.doFinal(skey.getEncoded());
        out.write(b);
        System.err.println("AES Key Length: " + b.length);
    }
}
```

3.5 Write the Initialization Vector

The initialization vector is next written to the output file. This is required for decryption using the AES key.

```
out.write(iv);
```

3.6 Encrypting the File Contents using the AES Key

The final step is to encrypt the contents of the file using the AES and write it to the output file.

```

Cipher ci = Cipher.getInstance("AES/CBC/PKCS5Padding");
ci.init(Cipher.ENCRYPT_MODE, key, ivspec);
try (FileInputStream in = new FileInputStream(inputFile)) {
    processFile(ci, in, out);
}

```

Then Close the output file and send it to the receiver.

3.7 Decrypting the File using the RSA Private Key

Since the public key has been used for encryption, the private key can be used for decrypting the file contents. Remember that the file includes the AES key encrypted with the RSA public key at the beginning followed by the initialization vector, and the encrypted file data itself. So the decryption process has to handle all these steps in order to get at the file data.

3.8 Load the RSA Public Key from File

We use the following code to load the RSA private key from a file where it has been saved in binary format.

```

String privKeyFile = Key_Path;
String inputFile = Encrypted_File_Path;
byte[] bytes = Files.readAllBytes(Paths.get(privKeyFile));
X509EncodedKeySpec ks = new X509EncodedKeySpec(bytes);
KeyFactory kf = KeyFactory.getInstance("RSA");
PrivateKey priv = kf.generatePrivate(ks);

```

3.9 Load the AES Secret Key

Open the encrypted file and load the AES secret key. The AES secret key can be obtained from this data by decrypting using the RSA private key.

```

try (FileInputStream in = new FileInputStream(inputFile)) {
    SecretKeySpec skey = null;
    {
        Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        cipher.init(Cipher.DECRYPT_MODE, priv);
        byte[] b = new byte[256];
        in.read(b);
        byte[] keyb = cipher.doFinal(b);
        skey = new SecretKeySpec(keyb, "AES");
    }
}

```

3.10 Read the Initialization Vector

Next in the file is the initialization vector. Load it as follows:

```

byte[] iv = new byte[128 / 8];
in.read(iv);
IvParameterSpec ivspec = new IvParameterSpec(iv);

```

3.11 Decrypt the File Contents

Now comes the actual decryption of the file contents using the AES secret key. The output is written into a file with the extension ".enc" for verification purposes.

```

byte[] iv = new byte[128 / 8];
in.read(iv);
IvParameterSpec ivspec = new IvParameterSpec(iv);

Cipher ci = Cipher.getInstance("AES/CBC/PKCS5Padding");
ci.init(Cipher.DECRYPT_MODE, skey, ivspec);

try (FileOutputStream out = new FileOutputStream(Keys_Folder_Path + "\\\" + "DecryptedFile.txt")) {
    processFile(ci, in, out);
}

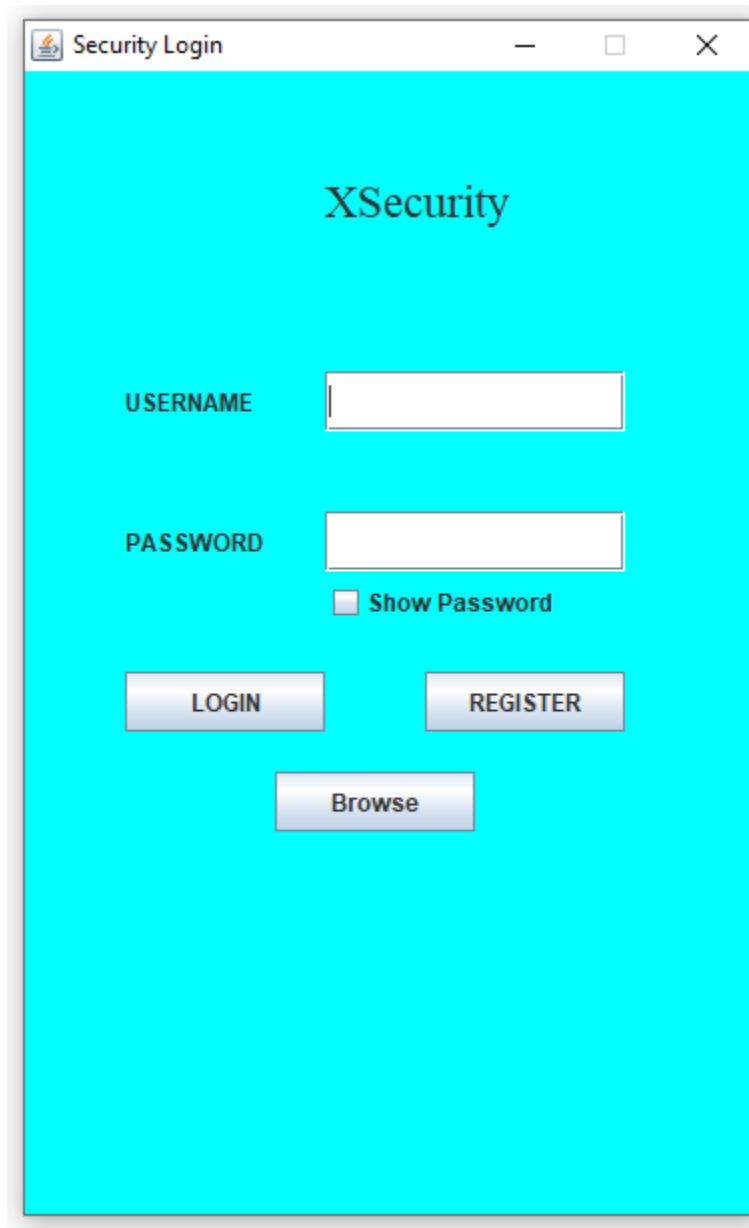
```

The static method *processFile()* is defined as before.

```
static private void processFile(Cipher ci, InputStream in, OutputStream out)
    throws javax.crypto.IllegalBlockSizeException,
           javax.crypto.BadPaddingException,
           java.io.IOException {
    byte[] ibuf = new byte[1024];
    int len;
    while ((len = in.read(ibuf)) != -1) {
        byte[] obuf = ci.update(ibuf, 0, len);
        if (obuf != null) {
            out.write(obuf);
        }
    }
    byte[] obuf = ci.doFinal();
    if (obuf != null) {
        out.write(obuf);
    }
}
```


4 Design and testing

Front page



The image shows a screenshot of a web application window titled "Security Login". The window has a blue header bar with the title and standard window controls (minimize, maximize, close). The main content area is white and features the "XSecurity" logo at the top. Below the logo, there are two input fields: "USERNAME" and "PASSWORD". The "PASSWORD" field has a "Show Password" checkbox next to it. At the bottom, there are three buttons: "LOGIN", "REGISTER", and "Browse".

Security Login

XSecurity

USERNAME

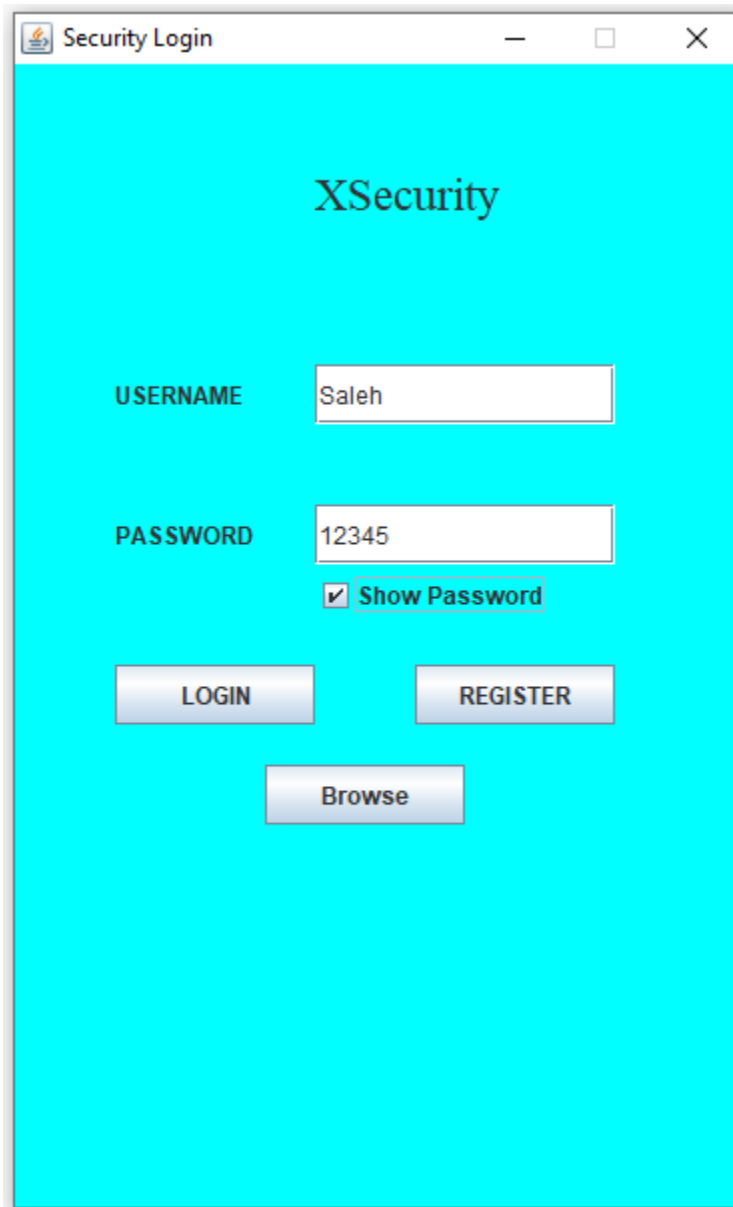
PASSWORD

☐ Show Password

LOGIN REGISTER

Browse

1. Login



A screenshot of a web application window titled "Security Login". The window has a blue header bar with the title and standard window controls (minimize, maximize, close). The main content area is white and features the "XSecurity" logo in a large, bold, black serif font. Below the logo, there are two input fields: "USERNAME" with the value "Saleh" and "PASSWORD" with the value "12345". A checkbox labeled "Show Password" is checked. At the bottom, there are three buttons: "LOGIN", "REGISTER", and "Browse".

Security Login

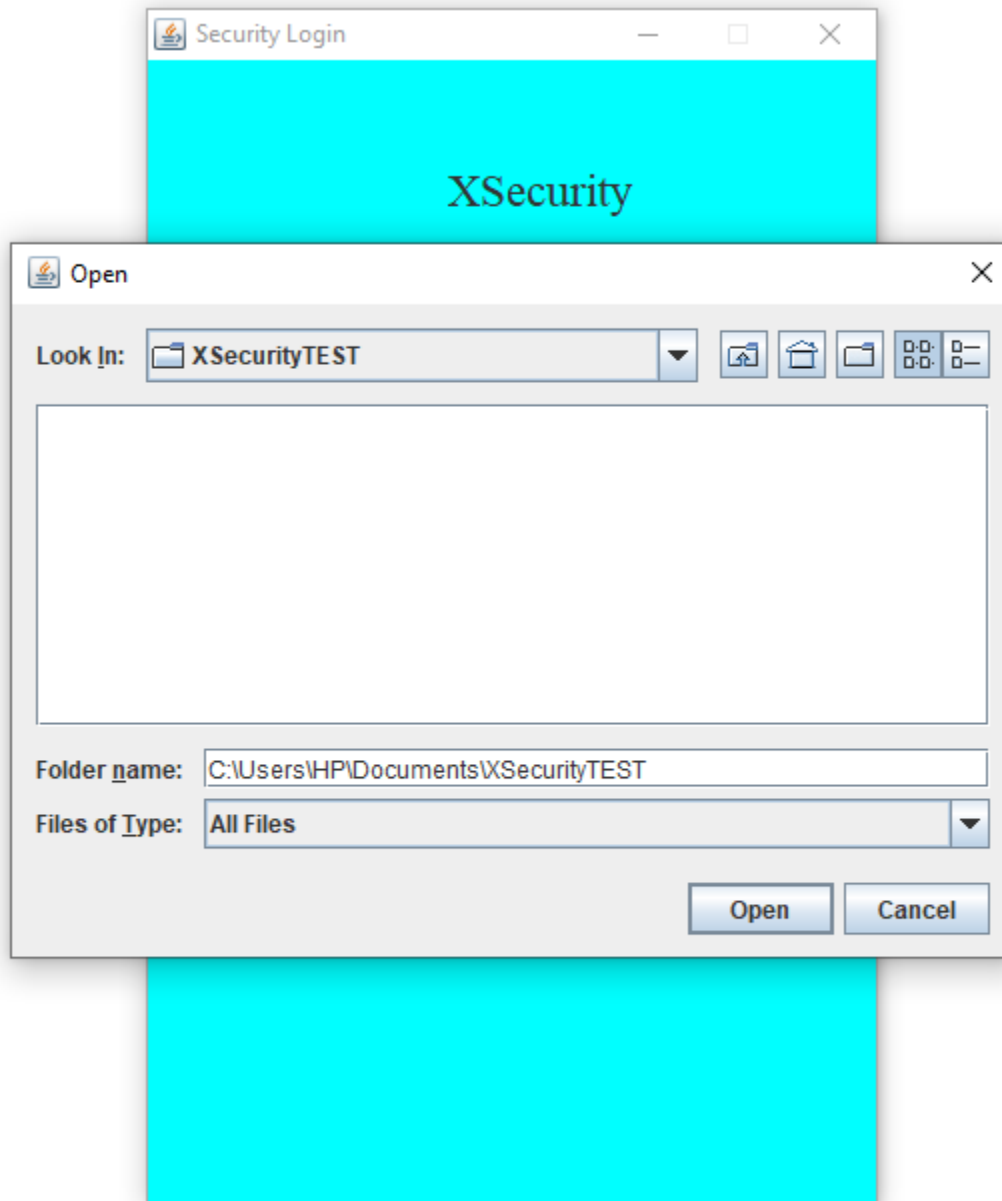
XSecurity

USERNAME

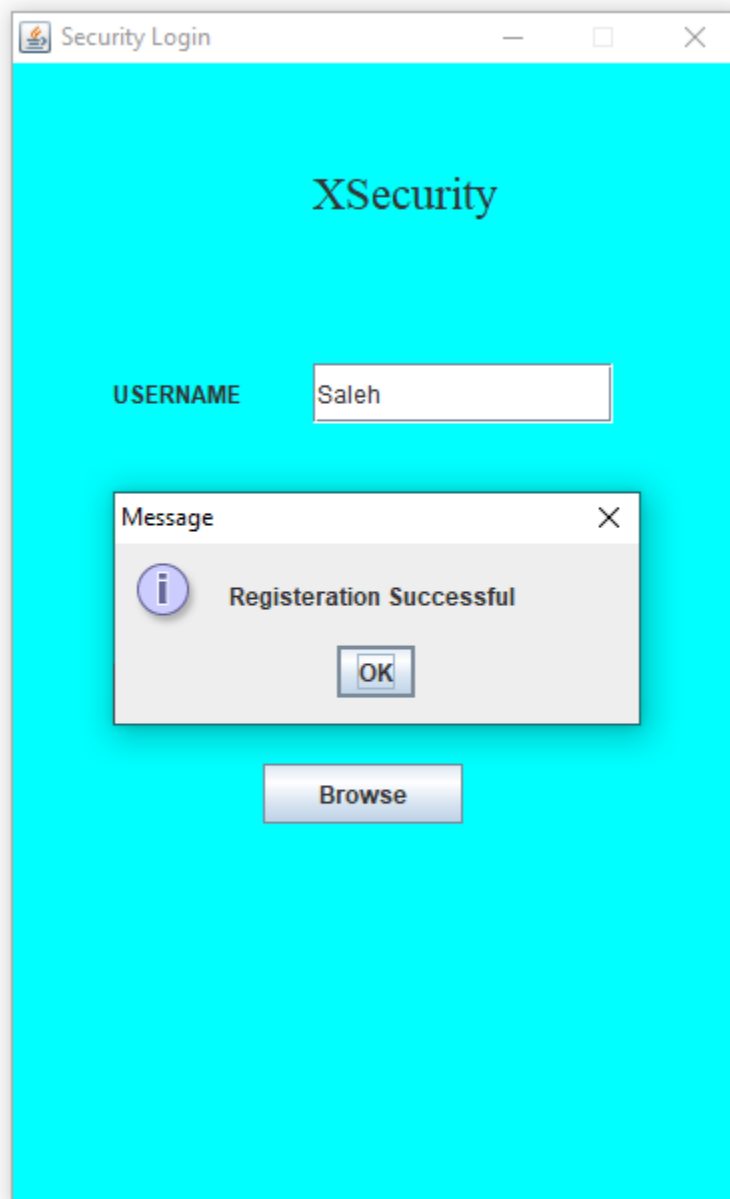
PASSWORD

☒ Show Password

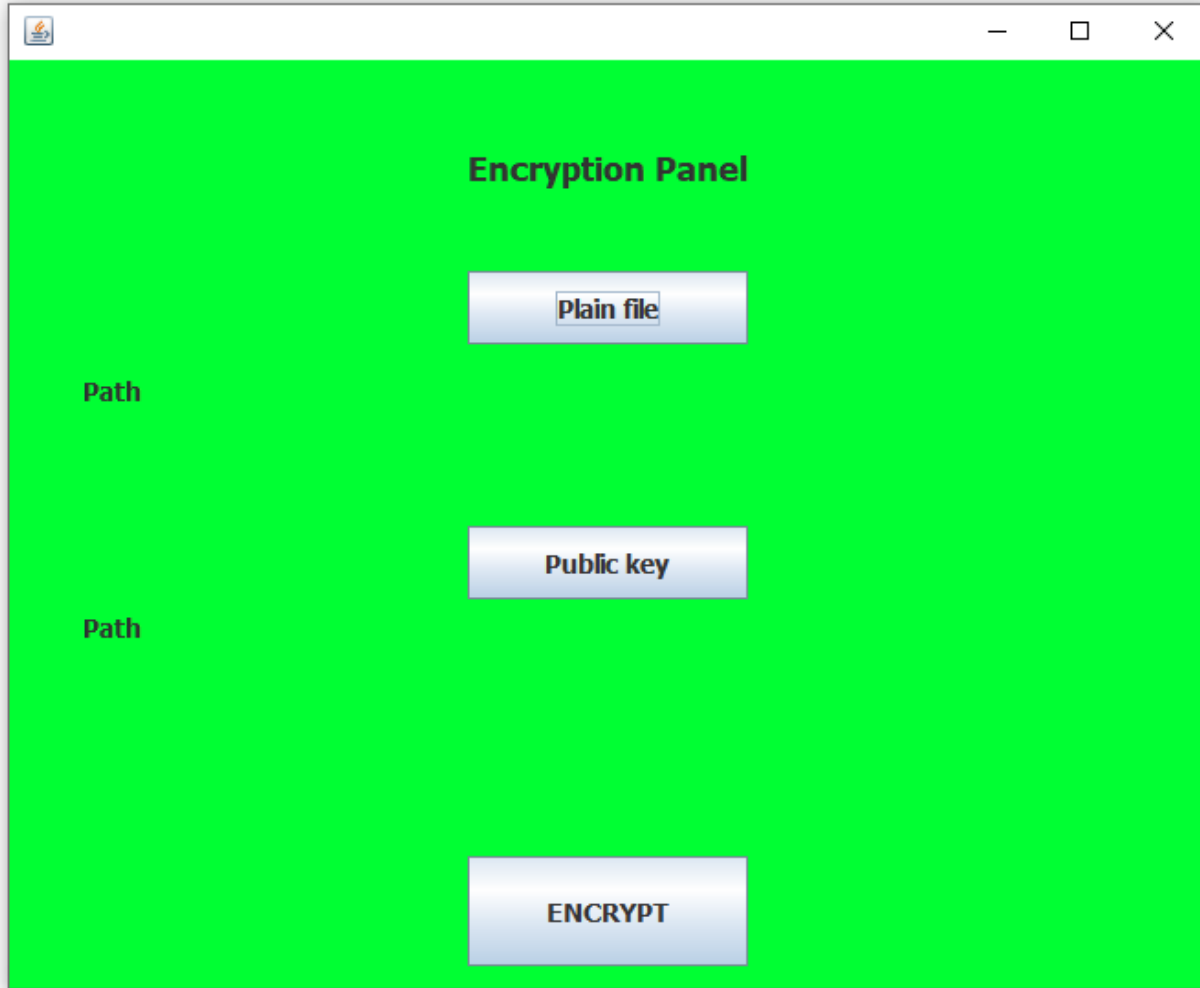
2. Browse and choose folder to store files



3. Register

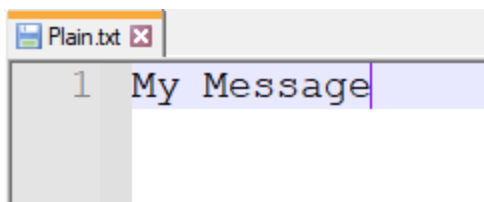


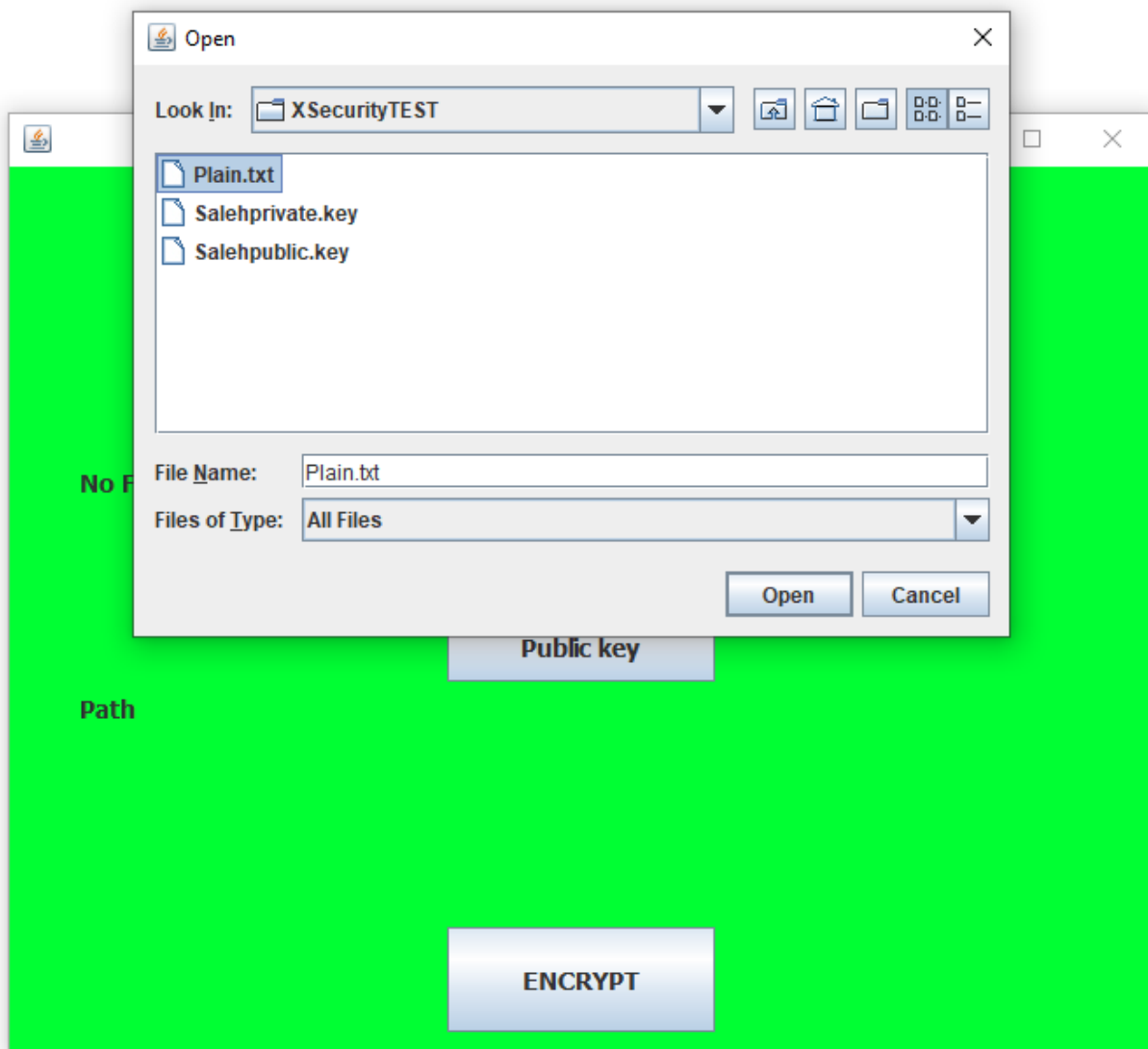
4. Encryption Panel

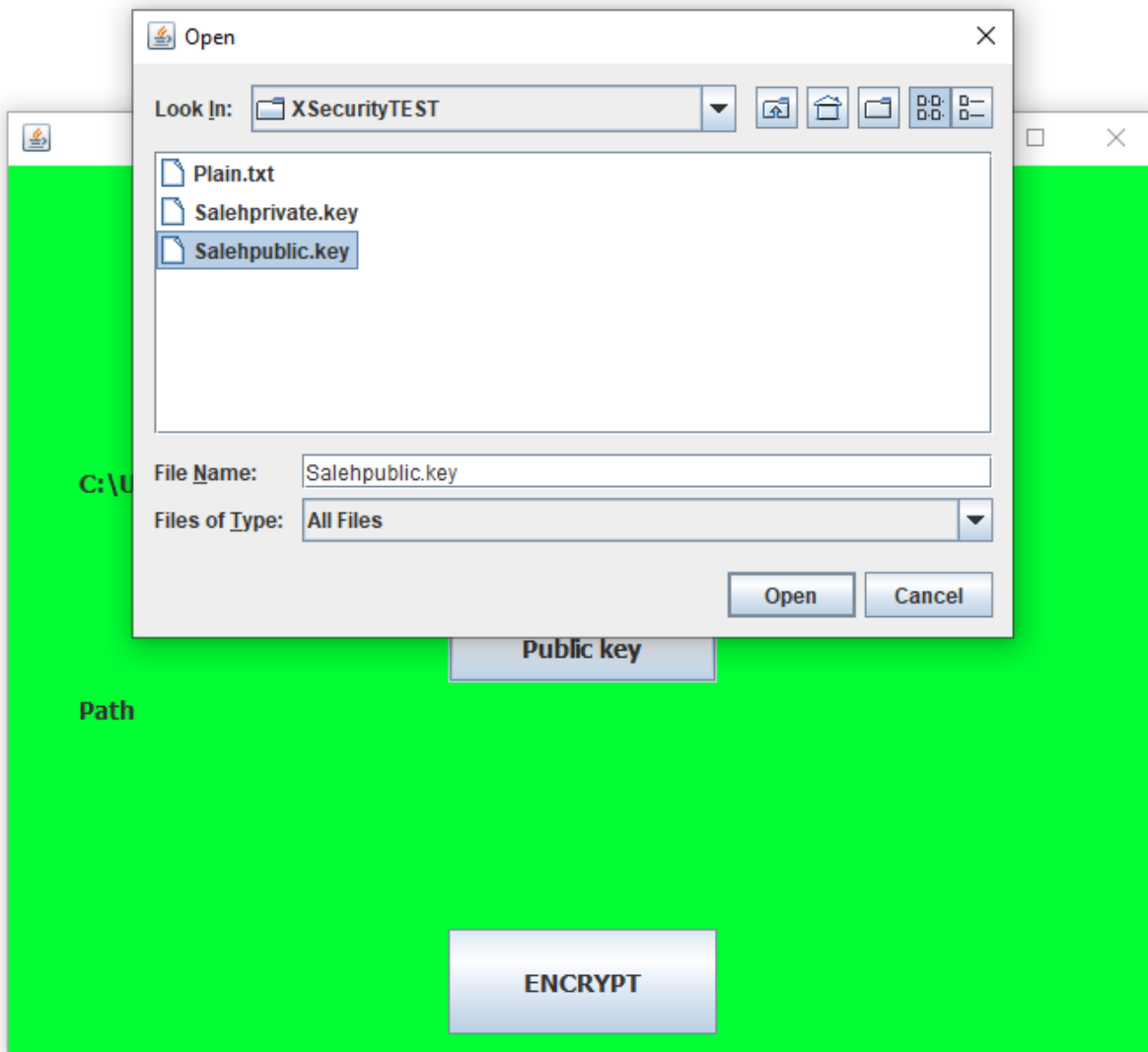


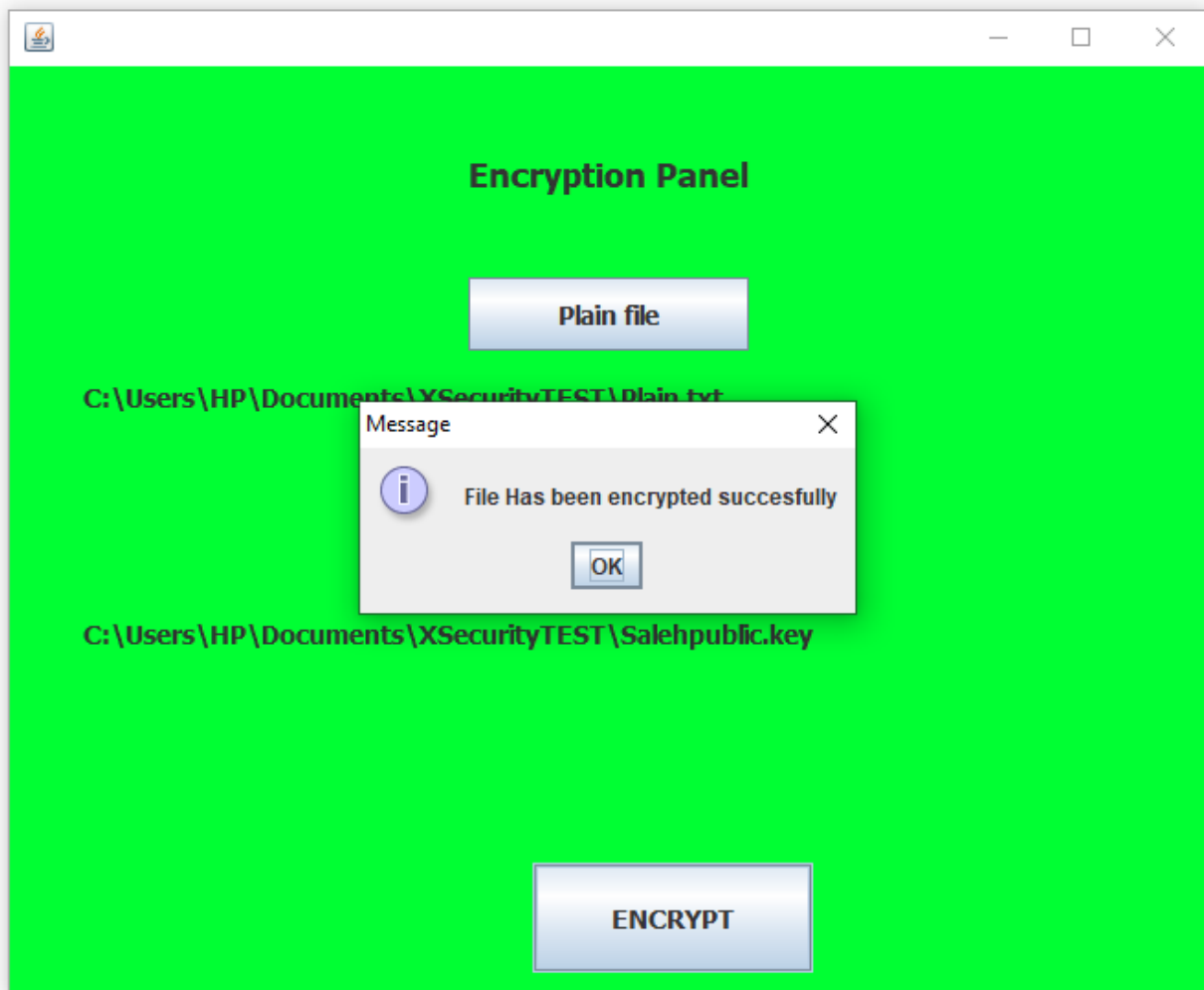
5. Choose the message to encrypt and public key

Plain file content





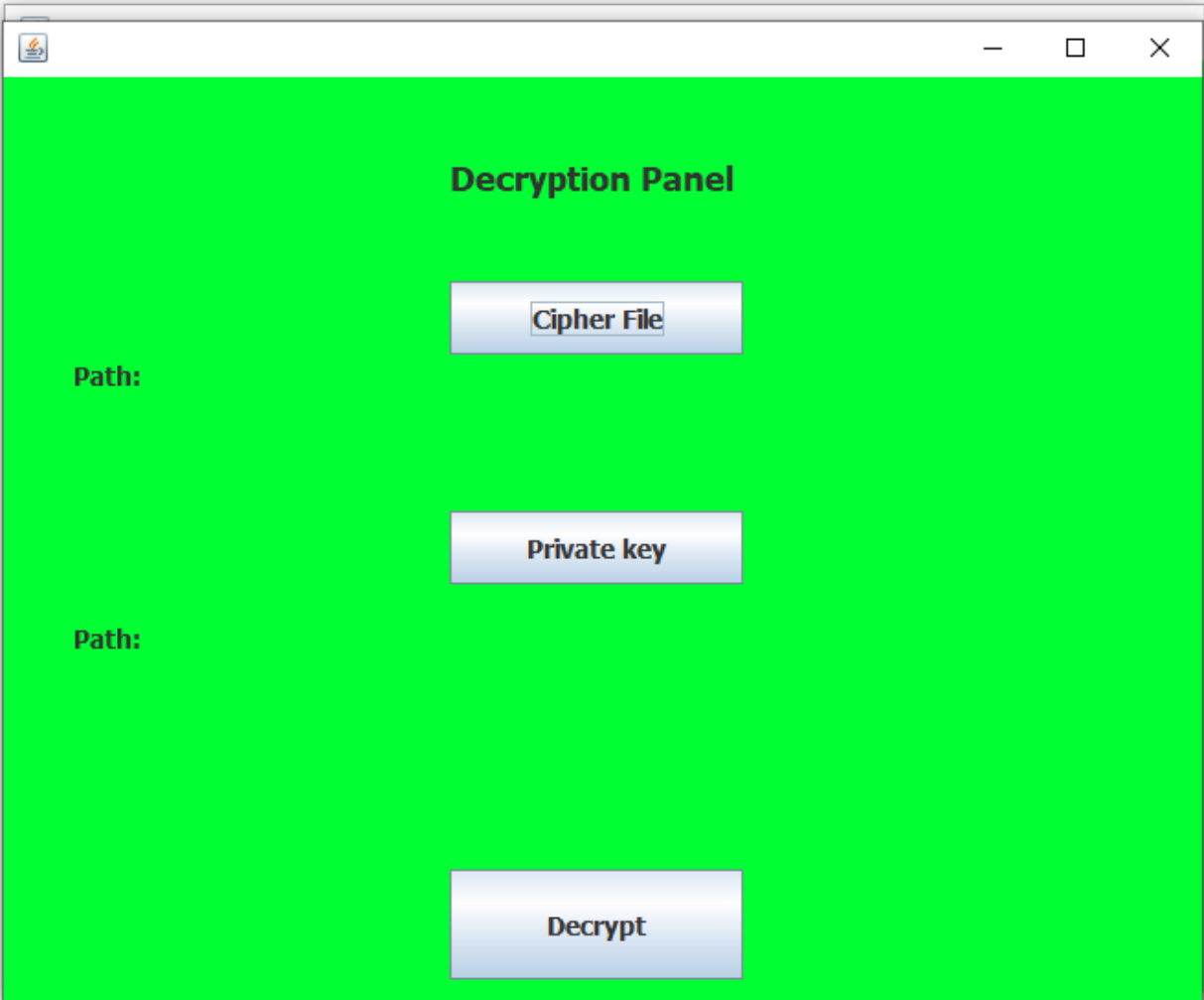




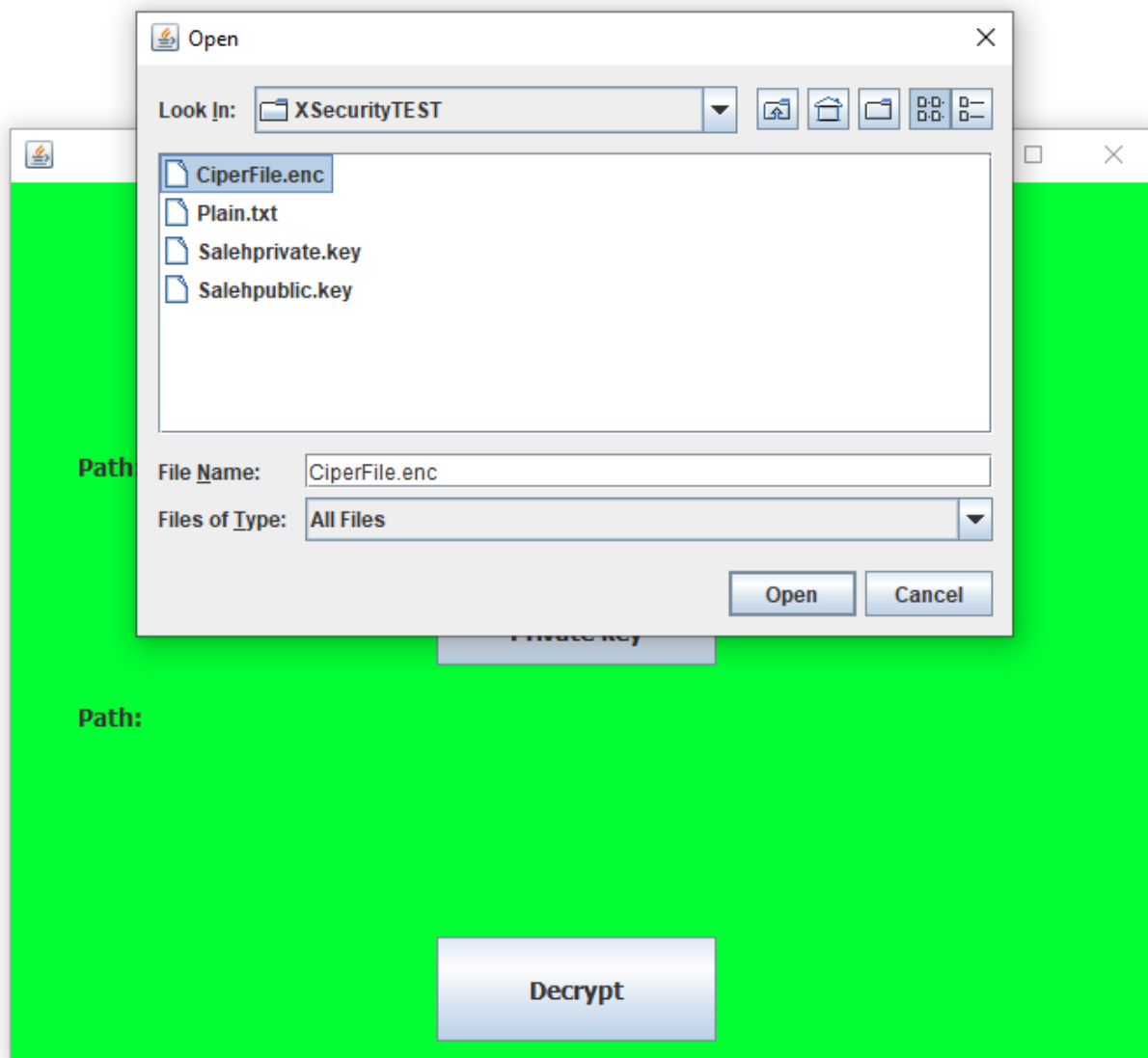
The Encrypted Cipher file

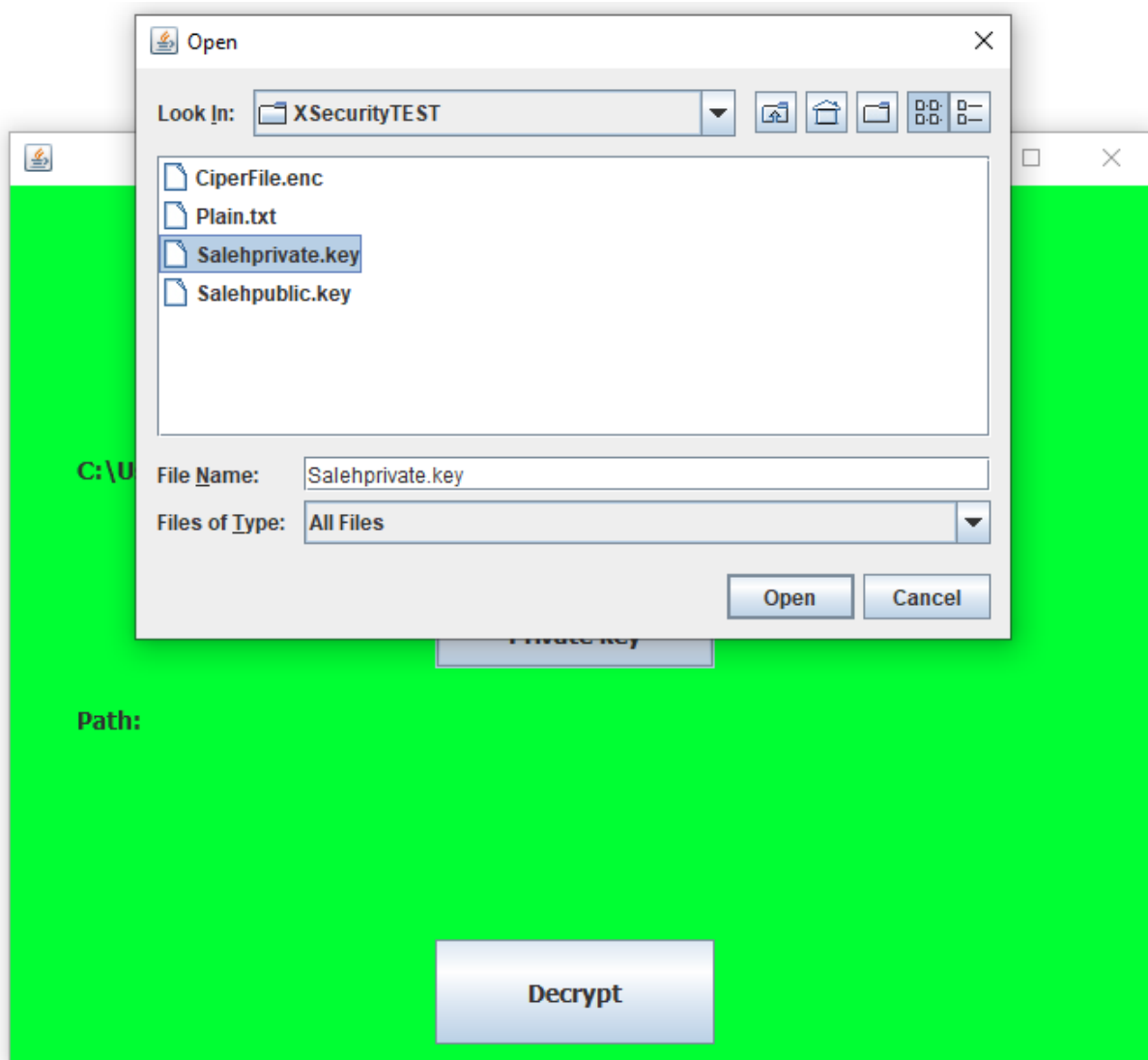
۱ کیلو بایت	Wireshark capture...	۰۲:۳۰ ص ۴۳/۰۹/۲۹	CiperFile
۱ کیلو بایت	ملف TXT	۱۲:۵۲ ص ۴۳/۰۹/۲۸	Plain
۱ کیلو بایت	ملف KEY	۰۲:۲۷ ص ۴۳/۰۹/۲۹	Salehprivate.key
۲ کیلو بایت	ملف KEY	۰۲:۲۷ ص ۴۳/۰۹/۲۹	Salehpublic.key

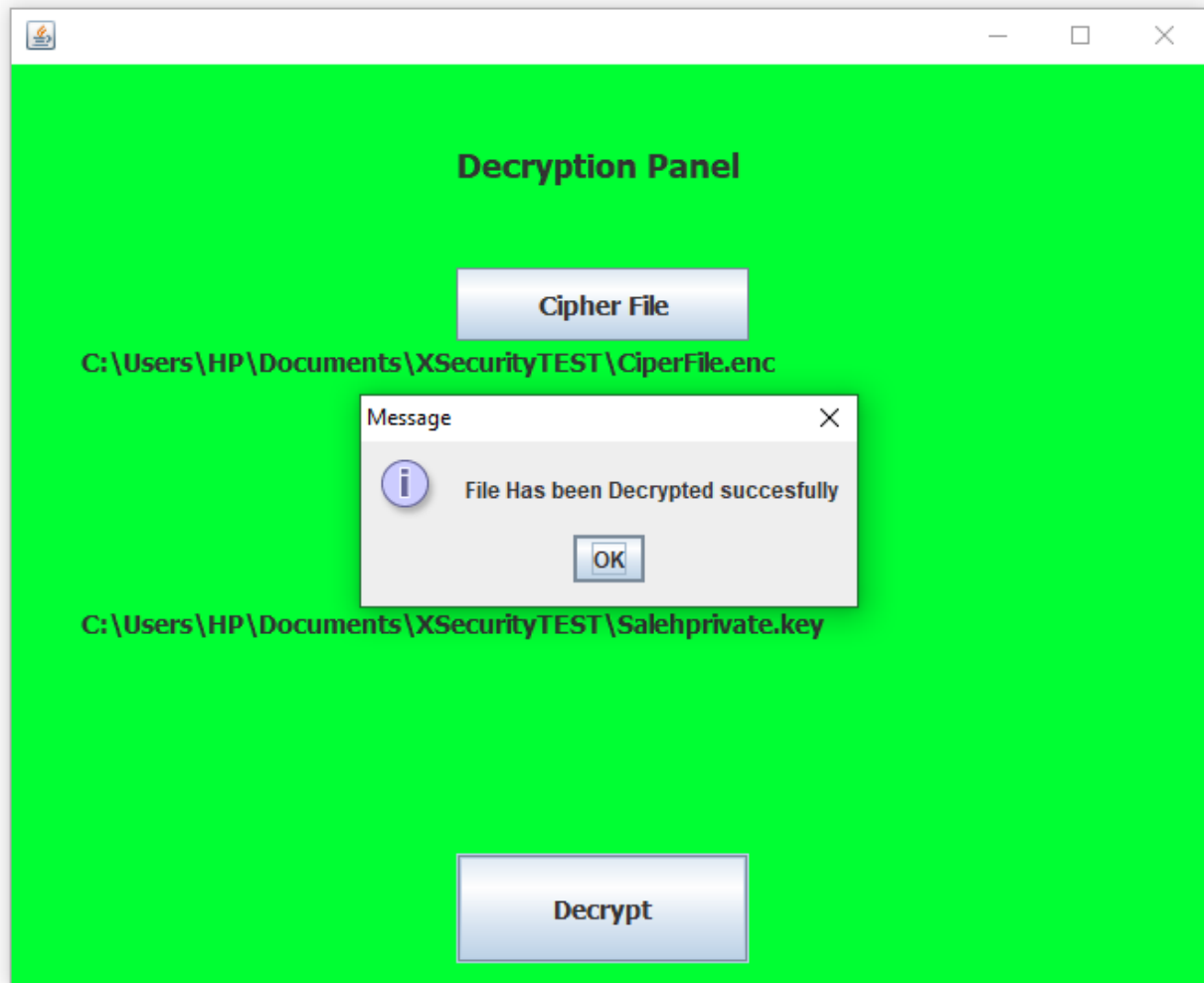
6. Choose the cipher file and decrypting it



The image shows a software window titled "Decryption Panel". The window has a white title bar with a small icon on the left and standard minimize, maximize, and close buttons on the right. The main content area has a solid blue background. At the top center, the title "Decryption Panel" is displayed in bold black text. Below this, there are two input sections. The first section is labeled "Path:" in bold black text on the left, followed by a text input field containing the text "Cipher File". The second section is also labeled "Path:" in bold black text on the left, followed by a text input field containing the text "Private key". At the bottom center of the panel is a large, light blue button with the text "Decrypt" in bold black font.



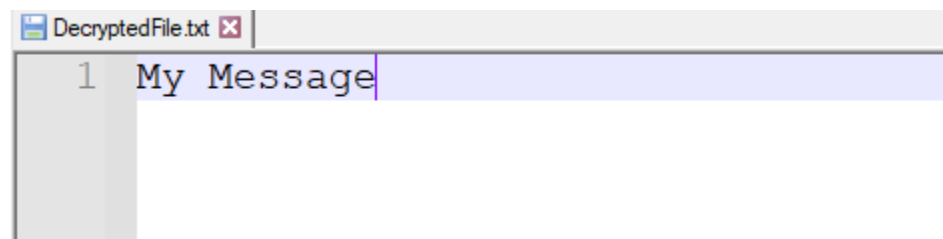




The Decrypted file produced

1 كيلوبايت	Wireshark capture...	٤٣/٠٩/٢٩ ص ٠٢:٣٠	CiperFile
1 كيلوبايت	ملف TXT	٤٣/٠٩/٢٩ ص ٠٢:٣٤	DecryptedFile
1 كيلوبايت	ملف TXT	٤٣/٠٩/٢٨ ص ١٢:٥٢	Plain
1 كيلوبايت	ملف KEY	٤٣/٠٩/٢٩ ص ٠٢:٢٧	Salehprivate.key
٢ كيلوبايت	ملف KEY	٤٣/٠٩/٢٩ ص ٠٢:٢٧	Salehpublic.key

Content



5 Challenges

The main difficulties we encountered were integrating the backend encryption code with the GUI, which necessitated a thorough understanding of the javaFX and java swing libraries, as well as dealing with the large number of errors and exceptions that arose from working with cryptography, which is sensitive and must be built correctly from the start.

6 Team Members

- 1- Saleh Alsaeed
- 2- Rayan Almengash
- 3- Abdulelah alanzi
- 4- Salem alqahtani
- 5- Riyadh alkhudir

7 References

[1] Novixys Software Dev Blog, Using AES With RSA for File Encryption and Decryption in Java available at: https://www.novixys.com/blog/using-aes-rsa-file-encryption-decryption-java/#2_Java_Imports