Implementation:

The program was implemented using Netbeans. The UI was built in Netbeans' GUI builder (Swing). Below is the list of the main libraries used:

- Java IO
- Javax swing (GUI)
- Java security
- Java crypto

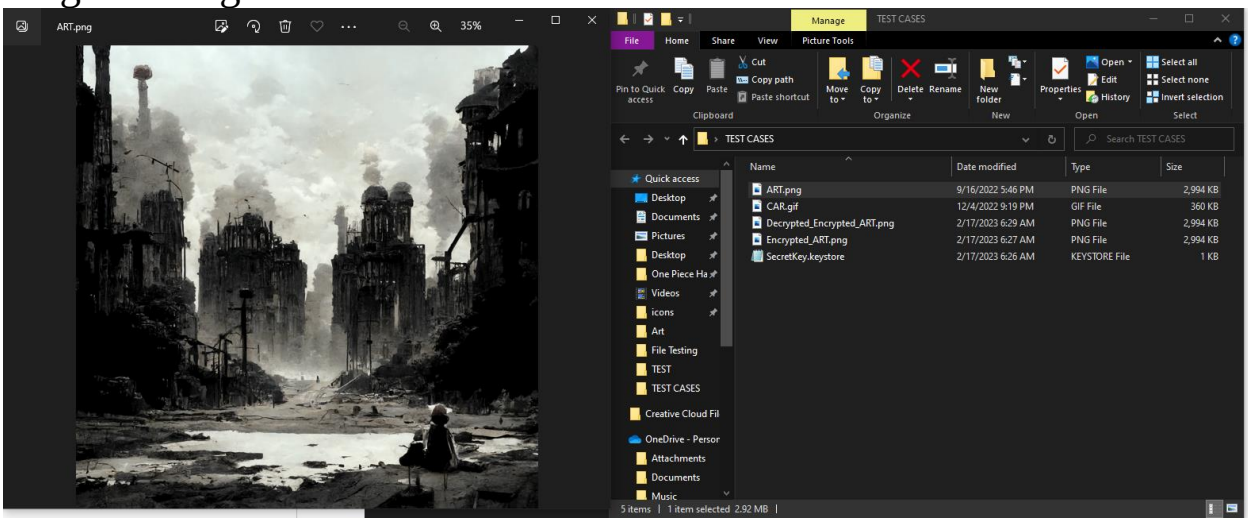
AES was implemented with a key size of 256(43 chars) and an ECB mode of encryption. The 256 bit key was generated using the KeyGenerator method available through the Java.Crypto library.

RSA was implemented with a key size of 2048(540 chars). Both 2048 bit keys (Private and public) were generated using the KeyGenerator method available through the Java.Crypto library.

Test cases:

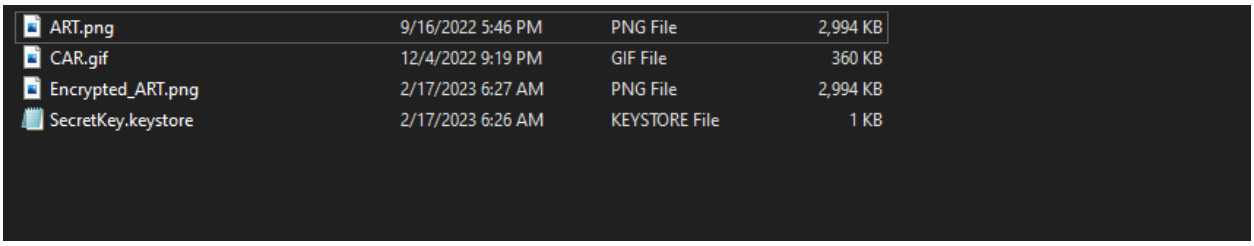
AES (Image):

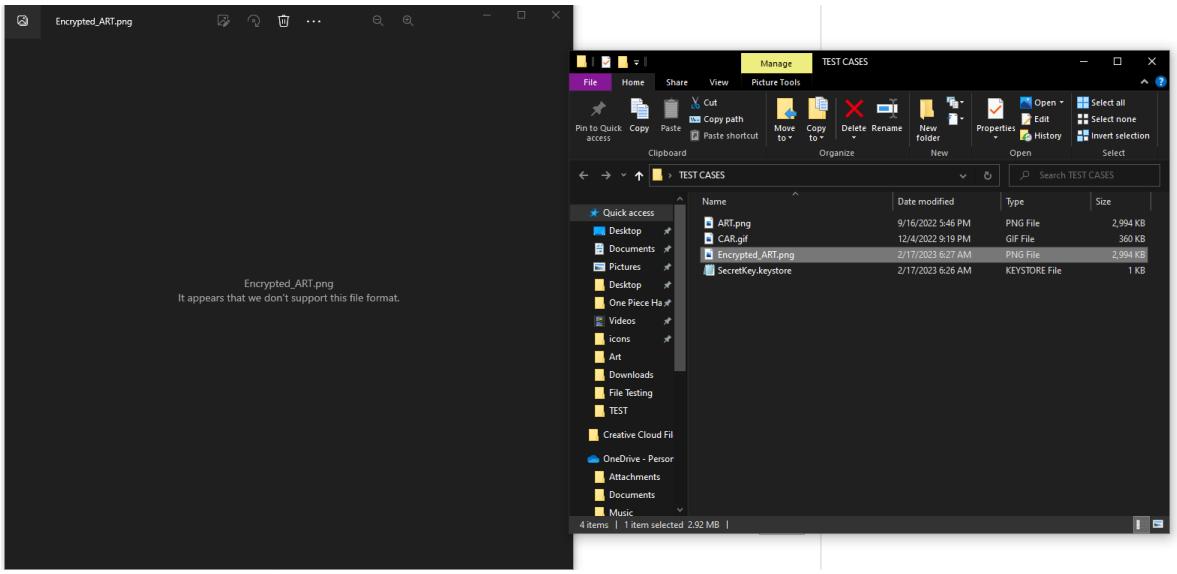
Original image:



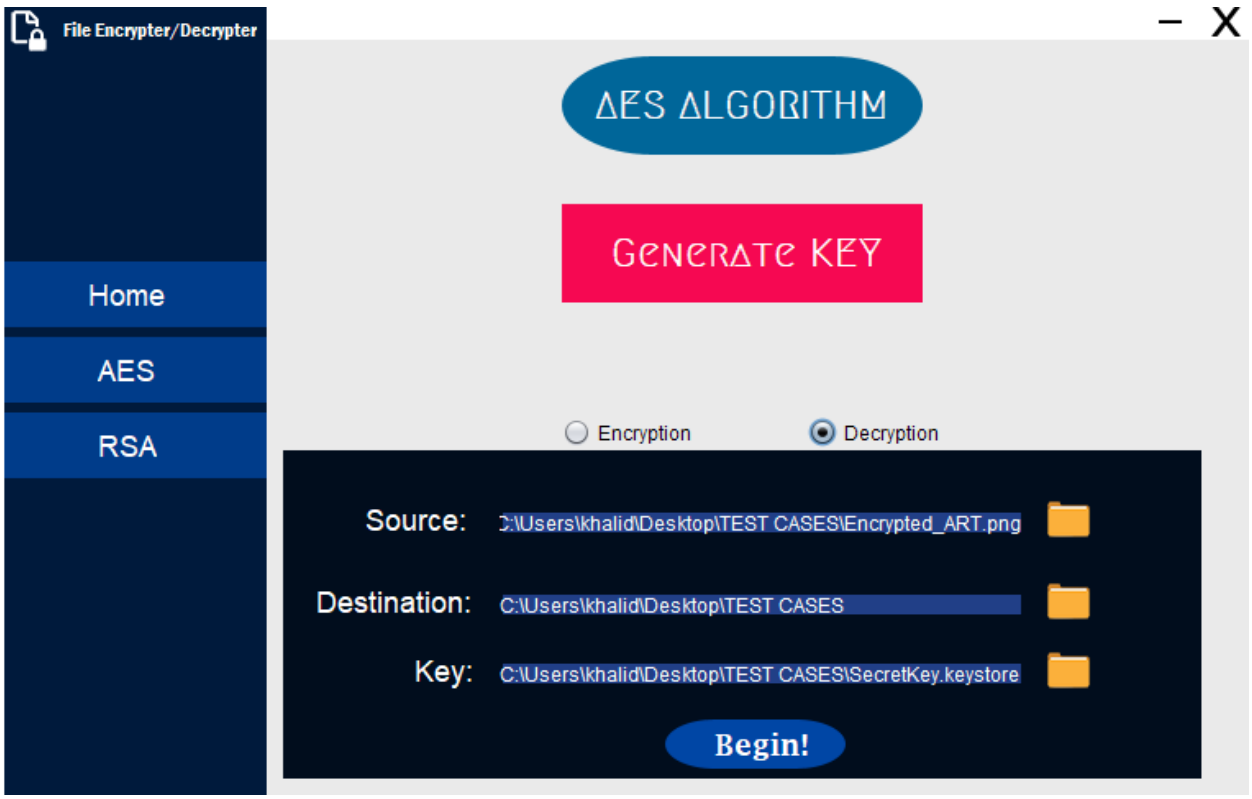
Encryption

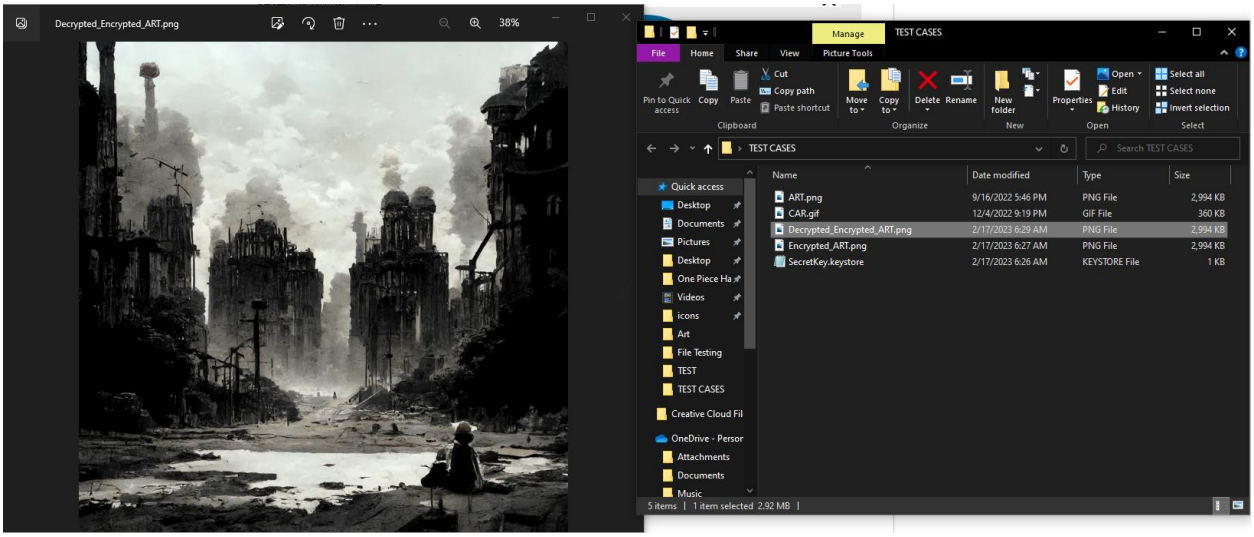

File Encrypter/Decrypter
AES ALGORITHM
Generate KEY
Home
AES
RSA
Encryption
Decryption
Source:
C:\Users\khalid\Desktop\TEST CASES\ART.png
Destination:
C:\Users\khalid\Desktop\TEST CASES
Key:
C:\Users\khalid\Desktop\TEST CASES\SecretKey.keystore
Begin!


ART.png 9/16/2022 5:46 PM PNG File 2,994 KB
CAR.gif 12/4/2022 9:19 PM GIF File 360 KB
Encrypted_ART.png 2/17/2023 6:27 AM PNG File 2,994 KB
SecretKey.keystore 2/17/2023 6:26 AM KEYSTORE File 1 KB


Encrypted_ART.png
Encrypted_ART.png
It appears that we don't support this file format.
Manage
TEST CASES
File
Home
Share
View
Picture Tools
Pin to Quick access
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New folder
New
Properties
Open
Edit
History
Open
Select all
Select none
Invert selection
Select
TEST CASES
Search TEST CASES
Quick access
Desktop
Documents
Pictures
Desktop
One Piece Ha
Videos
icons
Art
Downloads
File Testing
TEST
Creative Cloud Fil
OneDrive - Persor
Attachments
Documents
Music
Name
Date modified
Type
Size
ART.png
9/16/2022 5:46 PM
PNG File
2,994 KB
CAR.gif
12/4/2022 9:19 PM
GIF File
360 KB
Encrypted_ART.png
2/17/2023 6:27 AM
PNG File
2,994 KB
SecretKey.keystore
2/17/2023 6:26 AM
KEYSTORE File
1 KB
4 items | 1 item selected 2.92 MB

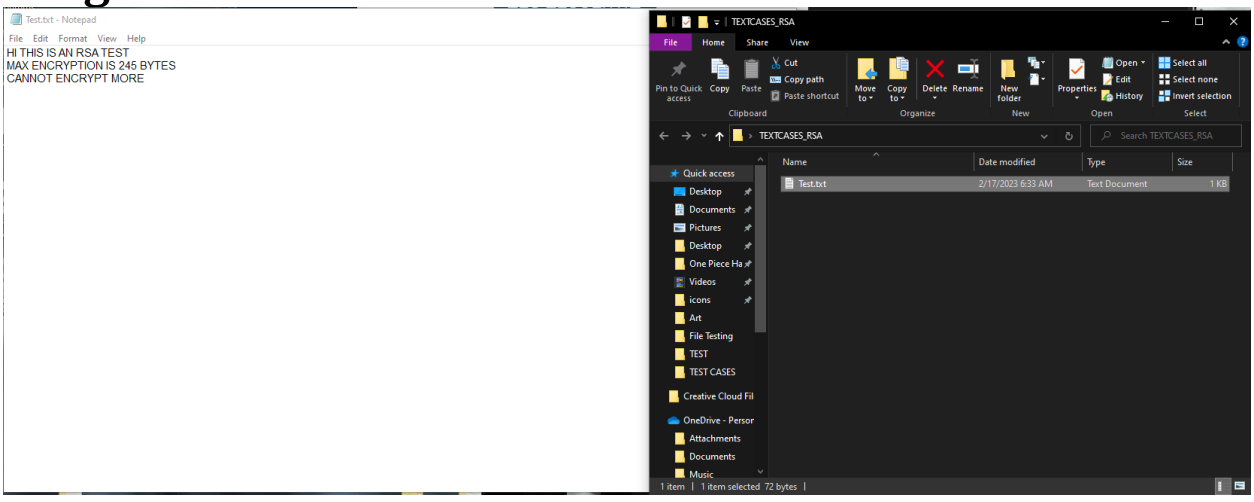
Decryption:


File Encrypter/Decrypter
AES ALGORITHM
Generate KEY
Home
AES
RSA
Encryption
Decryption
Source:
C:\Users\khalid\Desktop\TEST CASES\Encrypted_ART.png
Destination:
C:\Users\khalid\Desktop\TEST CASES
Key:
C:\Users\khalid\Desktop\TEST CASES\SecretKey.keystore
Begin!

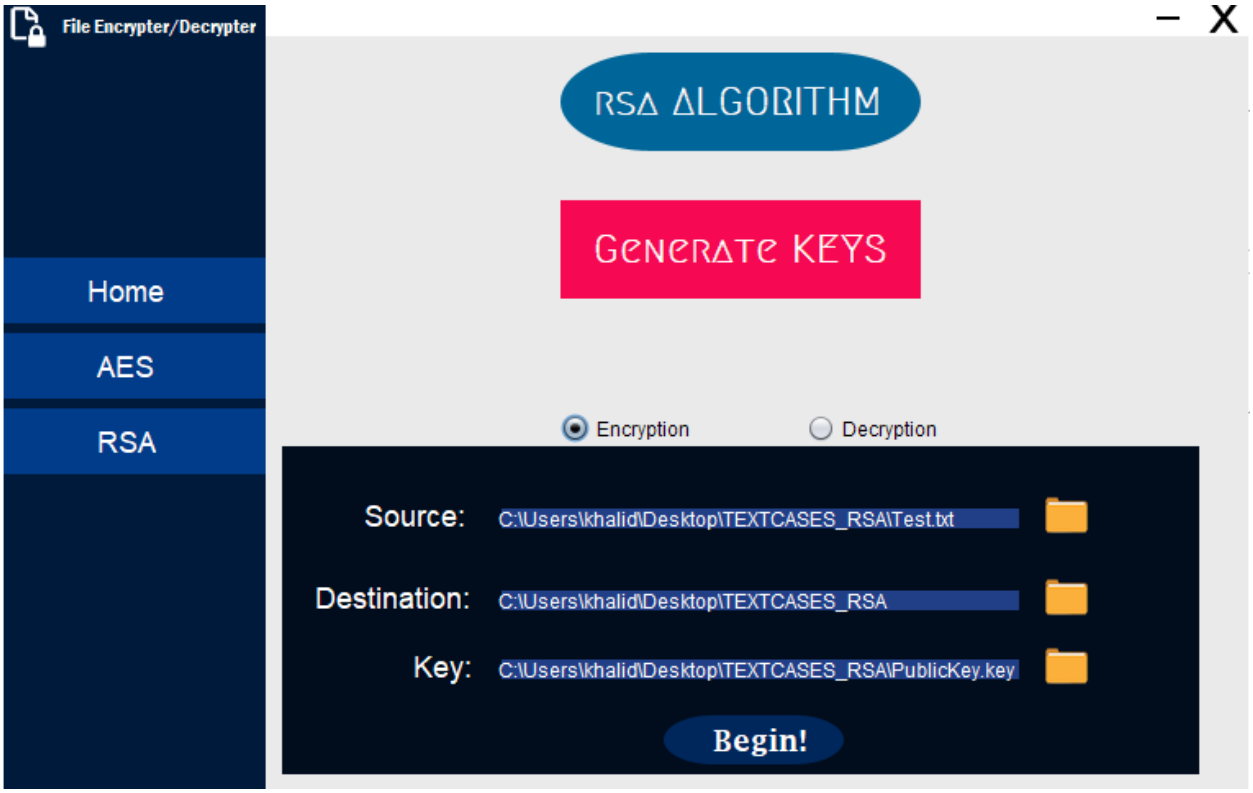


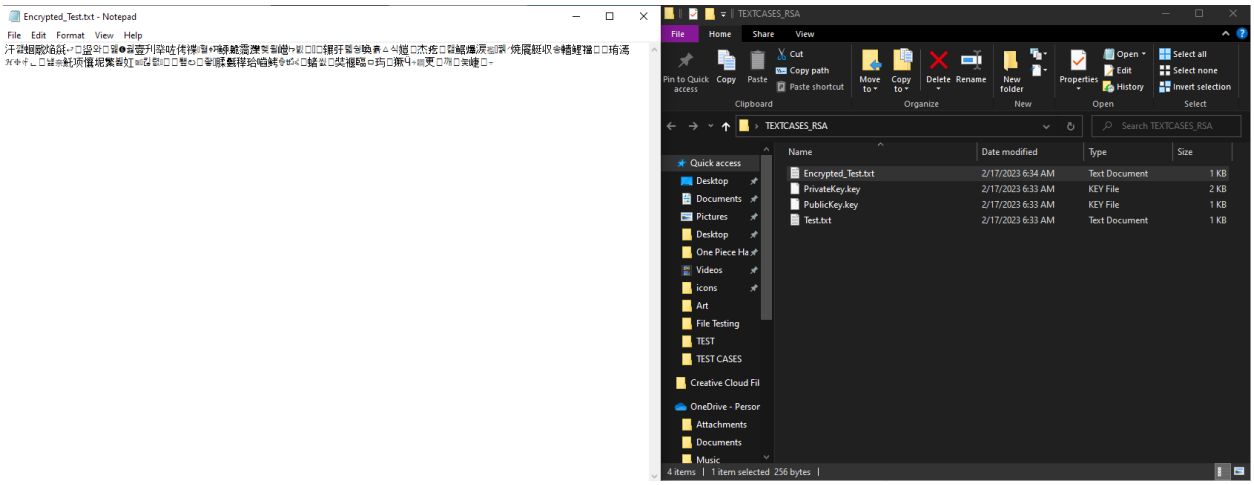
RSA(TEXT):

Orignal text:

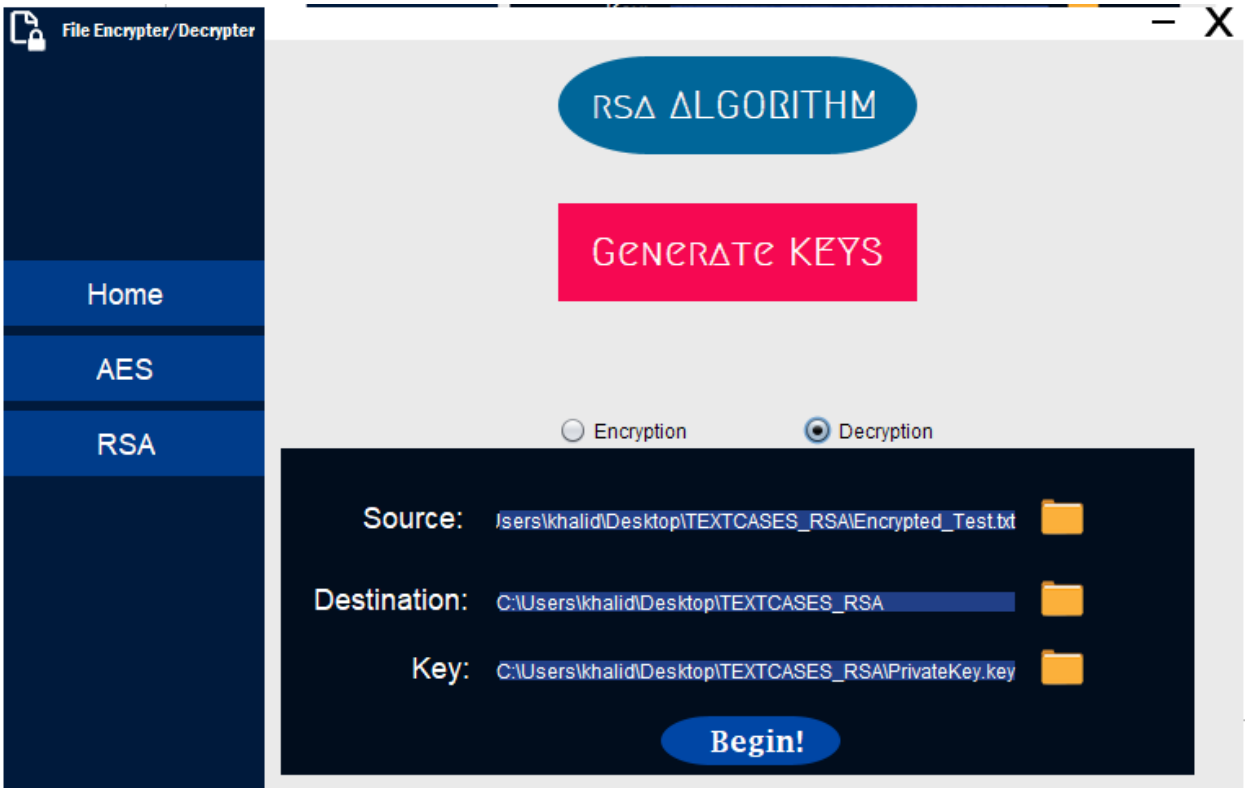


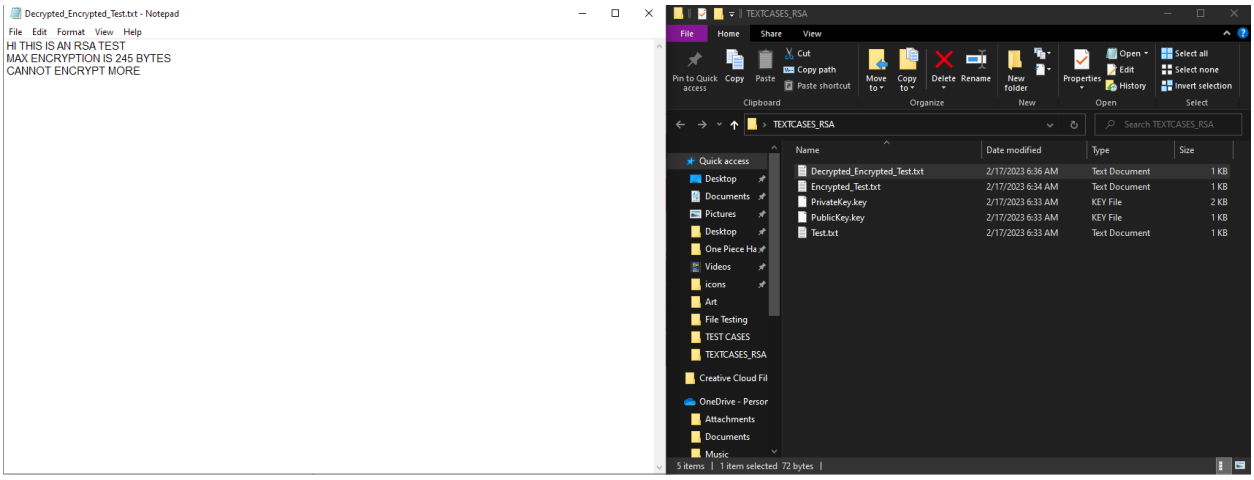
Encryption:


File Encrypter/Decrypter
Home
AES
RSA
RSA ALGORITHM
GENERATE KEYS
Encryption
Decryption
Source:
C:\Users\khalid\Desktop\TEXTCASES_RSA\Test.txt
Destination:
C:\Users\khalid\Desktop\TEXTCASES_RSA
Key:
C:\Users\khalid\Desktop\TEXTCASES_RSA\PublicKey.key
Begin!


Encrypted_Test.txt - Notepad
File Edit Format View Help
TEXTCASES_RSA
Name
Date modified
Type
Size
Encrypted_Test.txt 2/17/2023 6:34 AM Text Document 1 KB
PrivateKey.key 2/17/2023 6:33 AM KEY File 2 KB
PublicKey.key 2/17/2023 6:33 AM KEY File 1 KB
Test.txt 2/17/2023 6:33 AM Text Document 1 KB
4 items | 1 item selected 256 bytes

Decryption:

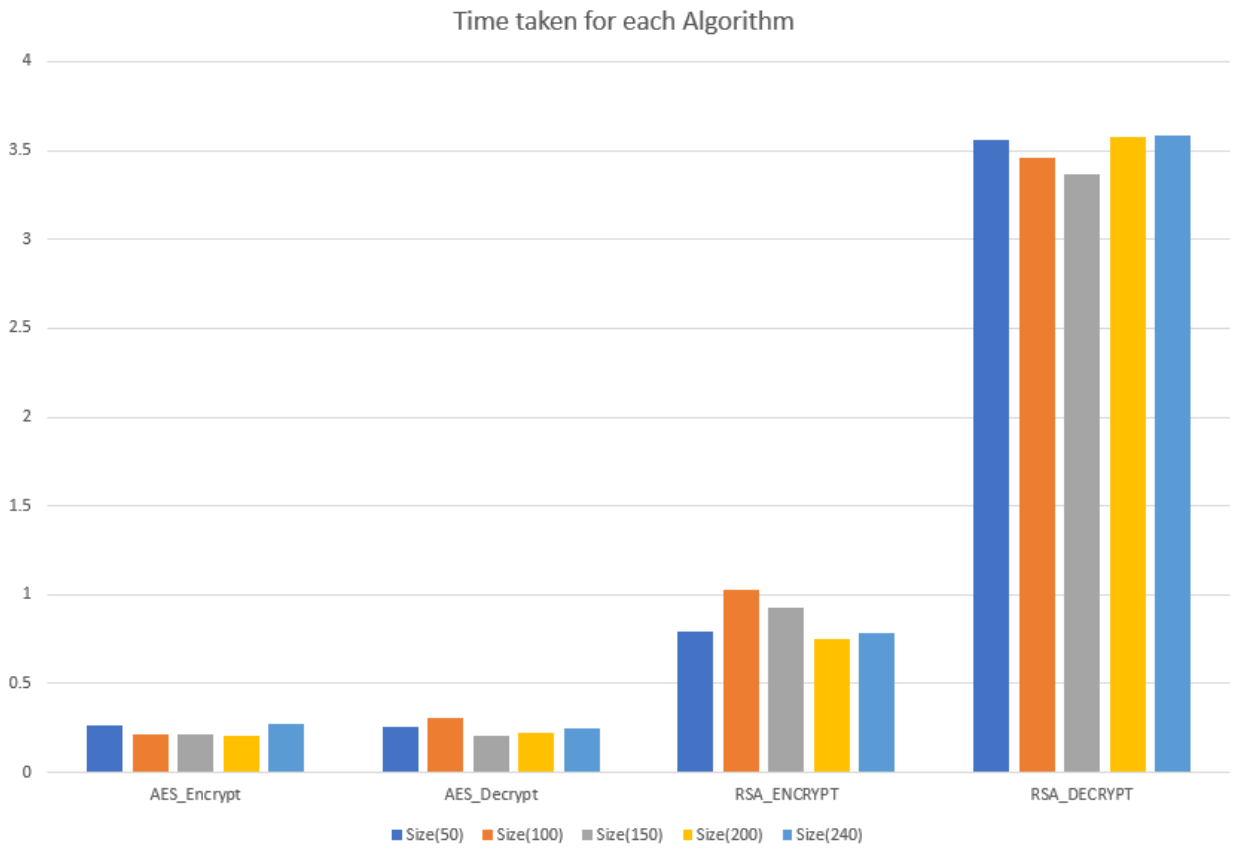

File Encrypter/Decrypter
RSA ALGORITHM
Generate KEYS
Home
AES
RSA
Encryption
Decryption
Source:
/sers\khalid\Desktop\TEXTCASES_RSA\Encrypted_Test.txt
Destination:
C:\Users\khalid\Desktop\TEXTCASES_RSA
Key:
C:\Users\khalid\Desktop\TEXTCASES_RSA\PrivateKey.key
Begin!


Decrypted_Encrypted_Test.txt - Notepad
File Edit Format View Help
HI THIS IS AN RSA TEST
MAX ENCRYPTION IS 245 BYTES
CANNOT ENCRYPT MORE
TEXTCASES_RSA
Decrypted_Encrypted_Test.txt 2/17/2023 6:36 AM Text Document 1 KB
Encrypted_Test.txt 2/17/2023 6:34 AM Text Document 1 KB
PrivateKey.key 2/17/2023 6:33 AM KEY File 2 KB
PublicKey.key 2/17/2023 6:33 AM KEY File 1 KB
Test.txt 2/17/2023 6:33 AM Text Document 1 KB

Algorithm comparison:

Initially the tests were run on different types of data, i.e., text, images, videos and audio files. However, we soon learned that the RSA algorithm has a maximum possible encryption size of 245 bytes. Seeing as how it wasn't designed to encrypt large files but instead the keys themselves, we decided to encrypt test text files. That way, we can simulate the act of encrypting keys while also providing accurate test results.

Here are the results of both algorithms running on the same data set. We have 5 files with sizes starting at 50 bytes and capping at 240. The y-axis is the time in ms.



Since the size disparity is very small, each algorithm runs in relatively the same time for each of the files in the dataset. And that is to be expected. However when we compare the algorithms with each other and not with the different input sizes we can clearly see a difference. RSA is clearly the algorithm with the slower run time, and that is due to the complex prime number computations it has to do. Compare that with the simple conversion of bits in the AES algorithm, and the difference in running time is more than understandable.

Appendix

Since the code is mixed in with the GUI code it will be counterintuitive to link the entire code file here. Instead, we will only link the important parts of the code. (Algorithm initialization, Algorithm runtime, etc.)

Generate AES KEY:

```
    KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
    keyGenerator.init(256);
    SecretKey mykey = keyGenerator.generateKey();


JFileChooser filechooser = new JFileChooser();
filechooser.setFileSelectionMode(filechooser.DIRECTORIES_ONLY);
int response =filechooser.showSaveDialog(null);

if(response == JFileChooser.APPROVE_OPTION){
    Key_File_AES= new File(filechooser.getSelectedFile().toString() +
"\\SecretKey.keystore");
    StoreKey(mykey,"Khalid",Key_File_AES);
    Key_location_AES.setText(filechooser.getSelectedFile().toString() +
"\\SecretKey.keystore");
    JOptionPane.showMessageDialog(this,"Generation Complete");
}
```

Generate RSA KEY:

```
KeyPairGenerator generator = KeyPairGenerator.getInstance("RSA");
generator.initialize(2048);
KeyPair pair = generator.generateKeyPair();
PrivateKey privateKey = pair.getPrivate();
PublicKey publicKey = pair.getPublic();

JFileChooser filechooser = new JFileChooser();
filechooser.setFileSelectionMode(filechooser.DIRECTORIES_ONLY);
int response =filechooser.showSaveDialog(null);

if(response == JFileChooser.APPROVE_OPTION){

Key_File_RSA= new File(filechooser.getSelectedFile().toString() +
"\\PublicKey.key");
FileOutputStream fos = new FileOutputStream(Key_File_RSA);
fos.write(publicKey.getEncoded());
fos.close();

Key_File_RSA= new File(filechooser.getSelectedFile().toString() +
"\\PrivateKey.key");
fos = new FileOutputStream(Key_File_RSA);
fos.write(privateKey.getEncoded());
fos.close();


JOptionPane.showMessageDialog(this,"Generation Complete");
```

RUN AES:

```
FileInputStream fis=null;
FileOutputStream fos=null;
 try {

     String CipherMode = Ciphermode_AES.getSelection().getActionCommand();

     Source_File_AES = new File(Source_location_AES.getText());

     if(CipherMode.equals("Encryption")){
```

```
    Destination_File_AES= new File(Destination_location_AES.getText() +
"\\Encrypted_" + Source_File_AES.getName());
    }
    else if(CipherMode.equals("Decryption"))
    Destination_File_AES= new File(Destination_location_AES.getText() +
"\\Decrypted_" + Source_File_AES.getName());


    Key_File_AES = new File(Key_location_AES.getText());



    fis = new FileInputStream(Source_File_AES);
    fos=new FileOutputStream(Destination_File_AES);


    SecretKey  mykey = (SecretKey) LoadKey(Key_File_AES,"Khalid");




    Cipher cipher =null;
    if(CipherMode.equals("Encryption")){

        long start = System.nanoTime();
        cipher = javax.crypto.Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(javax.crypto.Cipher.ENCRYPT_MODE,mykey,SecureRandom.getInsta
nce("SHA1PRNG"));
        CipherInputStream cis = new CipherInputStream(fis, cipher);
        long end = System.nanoTime();

        System.out.println("TIME SPENT: AES_ENCRYPTION:" + (end -
start)/1000000.0);
        write(cis, fos);
        JOptionPane.showMessageDialog(this,"Encryption Complete");
    }
    else if(CipherMode.equals("Decryption")){

        long start = System.nanoTime();
        cipher =
javax.crypto.Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(javax.crypto.Cipher.DECRYPT_MODE,mykey,SecureRandom.getInsta
nce("SHA1PRNG"));
        CipherOutputStream cos = new CipherOutputStream(fos, cipher);
        long end = System.nanoTime();
```

```
        System.out.println("TIME SPENT: AES_DECRYPTION:" + (end -
start)/1000000.0);

        write(fis, cos);
        JOptionPane.showMessageDialog(this,"Decryption Complete");
```

RUN RSA:

```
try {
    String CipherMode = Ciphermode_RSA.getSelection().getActionCommand();

    Source_File_RSA = new File(Source_location_RSA.getText());

    if(CipherMode.equals("Encryption")){
        Destination_File_RSA= new File(Destination_location_RSA.getText() +
"\\Encrypted_" + Source_File_RSA.getName());


    }
    else if(CipherMode.equals("Decryption"))
        Destination_File_RSA= new File(Destination_location_RSA.getText() +
"\\Decrypted_" + Source_File_RSA.getName());



        Key_File_RSA = new File(Key_location_RSA.getText());
        byte[] KeyBytes = Files.readAllBytes(Key_File_RSA.toPath());
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");


        if(CipherMode.equals("Encryption"))

            publickey = keyFactory.generatePublic(new
X509EncodedKeySpec(KeyBytes));
        else
            privatekey = keyFactory.generatePrivate(new
PKCS8EncodedKeySpec(KeyBytes));
```

```
    byte[] fileBytes =
Files.readAllBytes(Paths.get(Source_File_RSA.getAbsolutePath()));

    if(CipherMode.equals("Encryption")){

    long start = System.nanoTime();
    Cipher encryptCipher = Cipher.getInstance("RSA");
    encryptCipher.init(Cipher.ENCRYPT_MODE, publickey);
    byte[] encryptedFileBytes = encryptCipher.doFinal(fileBytes);
    FileOutputStream stream = new FileOutputStream(Destination_File_RSA);
    long end = System.nanoTime();
    System.out.println("TIME SPENT: RSA_ENCRYPTION:" + (end - start)/1000000.0);

    stream.write(encryptedFileBytes);
    stream.close();
    JOptionPane.showMessageDialog(this,"Encryption Complete");

    }
    else if(CipherMode.equals("Decryption")){

    long start = System.nanoTime();
    Cipher decryptCipher = Cipher.getInstance("RSA");
    decryptCipher.init(Cipher.DECRYPT_MODE, privatekey);
    byte[] decryptedFileBytes = decryptCipher.doFinal(fileBytes);
    FileOutputStream stream = new
FileOutputStream(Destination_File_RSA);
    long end = System.nanoTime();
    System.out.println("TIME SPENT RSA_DECRYPTION: " + (end - start)/1000000.0);

    stream.write(decryptedFileBytes);
    stream.close();
    JOptionPane.showMessageDialog(this,"Decryption
Complete");

    }
```