كلية علوم الحاسب والمعلومات
College of Computer and Information Sciences

Al-Imam Muhammad Bin Saud Islamic University

College of Computer and Information Sciences,

Department of Computer Science

CS 334 Information Security

Course project

May 8th, 2022

Section: 371

Name	Email	ID
Wea'am Alghaith	wksalghait@sm.imamu.edu.sa	440023306
Khloud Alnufaie	khaalnufaie@sm.imamu.edu.sa	440020617
Raghad Albosais	rkaalbosais@sm.imamu.edu.sa	440020209

Index of content:

1. DETAILED DESCRIPTION OF YOUR DESIGN AND IMPLEMENTATION 4

1.1 OVERVIEW OF THE PROJECT DESIGN 4

1.1.1 Description: *4*

1.2 APPROACH AND STEPS TO IMPLEMENTATION 5

1.2.1 Graphical user interface (GUI) *5*

1.2.2 Encryption and Decryption *10*

1. Generate and load keys 10

1.1 Generate public and private keys 10

1.2 Load public and private keys 10

2. Encryption part: encrypt_file_and_key() method 11

2.1 Initial setting before encryption 11

2.2 aes_file_encrypt() helper method 12

2.3 rsa_key_encrypt() helper method 14

3. Decryption part: decrypt_file_and_key() method 16

3.1 Initial setting before decryption 16

3.2 rsa_key_decrypt() helper method 16

3.3 aes_file_decrypt() helper method 18

1.3 SOURCE CODE 21

1.4 CHALLENGES 29

2. REFERENCES 30

List of figures:

Figure 1 Workflow of the project ____ *4*

Figure 4 up normal case of sign up entered username does not exist ____ *5*

Figure 3 Navigation design ____ *5*

Figure 2 Main window ____ *5*

Figure 5 Up normal case for signup password to short ____ *6*

Figure 6 Signup case ____ *6*

Figure 7 Login case ____ *6*

Figure 8 Successfully signup window ____ *6*

Figure 9 Send - receive file window ____ *7*

Figure 10 Sender window ____ *7*

Figure 12 Second normal case of send a file ____ *7*

Figure 11 First normal case to send a file ____ *7*

Figure 13 First up normal case of send file ____ *7*

Figure 14 Third up normal case of send file ____ *8*

Figure 15 Second up normal case of send file 8
Figure 16 Successful Send window 8
Figure 17 Received files window 9
Figure 18 Send - receive file window 9
Figure 19 Login and chose receive file 9
Figure 20 The content of selected file 9
Figure 21 Generate keys method 10
Figure 22 Load/Get keys method 11
Figure 23 Parameters of encrypt_file_and_key() method 11
Figure 24 Initial setting1 in encrypt_file_and_key() method 11
Figure 25 Initial setting2 in encrypt_file_and_key() method 12
Figure 26 Initial setting3 in encrypt_file_and_key() method 12
Figure 27 Calling the aes_file_encrypt() helper method 12
Figure 28 Parameters of aes_file_encrypt() helper method 12
Figure 29 Step1 in aes_file_encrypt() helper method 13
Figure 30 Step2 in aes_file_encrypt() helper method 13
Figure 31 Step3 in aes_file_encrypt() helper method 13
Figure 32 Step4 in aes_file_encrypt() helper method 13
Figure 33 Step5 in aes_file_encrypt() helper method 14
Figure 34 Step6 in aes_file_encrypt() helper method 14
Figure 35 Step7 in aes_file_encrypt() helper method 14
Figure 36 Calling the rsa_key_encrypt() helper method 14
Figure 37 Parameters of rsa_key_encrypt() helper method 15
Figure 38 Step1 in rsa_key_encrypt() helper method 15
Figure 39 Step2 in rsa_key_encrypt() helper method 15
Figure 40 Step3 in rsa_key_encrypt() helper method 15
Figure 41 Parameters of decrypt_file_and_key() method 16
Figure 42 Initial setting1 in decrypt_file_and_key() method 16
Figure 43 Calling the rsa_key_decrypt() helper method 16
Figure 44 Parameters of rsa_key_decrypt() helper method 17
Figure 45 Step1 in rsa_key_decrypt() helper method 17
Figure 46 Step2 in rsa_key_decrypt() helper method 17
Figure 47 Step3 in rsa_key_decrypt() helper method 17
Figure 48 Calling the rsa_file_decrypt() helper method 18
Figure 49 Step1 in rsa_file_decrypt() helper method 18
Figure 50 Step2 in rsa_file_decrypt() helper method 18
Figure 51 Step3 in rsa_file_decrypt() helper method 19
Figure 52 Step4 in rsa_file_decrypt() helper method 19
Figure 53 Step5 in rsa_file_decrypt() helper method 19
Figure 54 Step6 in rsa_file_decrypt() helper method 19
Figure 55 Step7 in rsa_file_decrypt() helper method 20

1. Detailed description of your design and implementation

1.1 Overview of the project design

1.1.1 Description:

We design a secure file sharing application. That contains GUI application where a user has to log in with a username and password for sharing (to send a file or receive a file). Each user has a pair of a private and public keys. The sender use a symmetric key to encrypt the file using the AES algorithm, and the symmetric key was encrypted by using the RSA algorithm and the receiver's public key. For decryption, the receiver uses its own private key to decrypt the symmetric key using the RSA and then uses that symmetric key to decrypt the file using the AES. Figure 1 below is an illustration of the workflow.

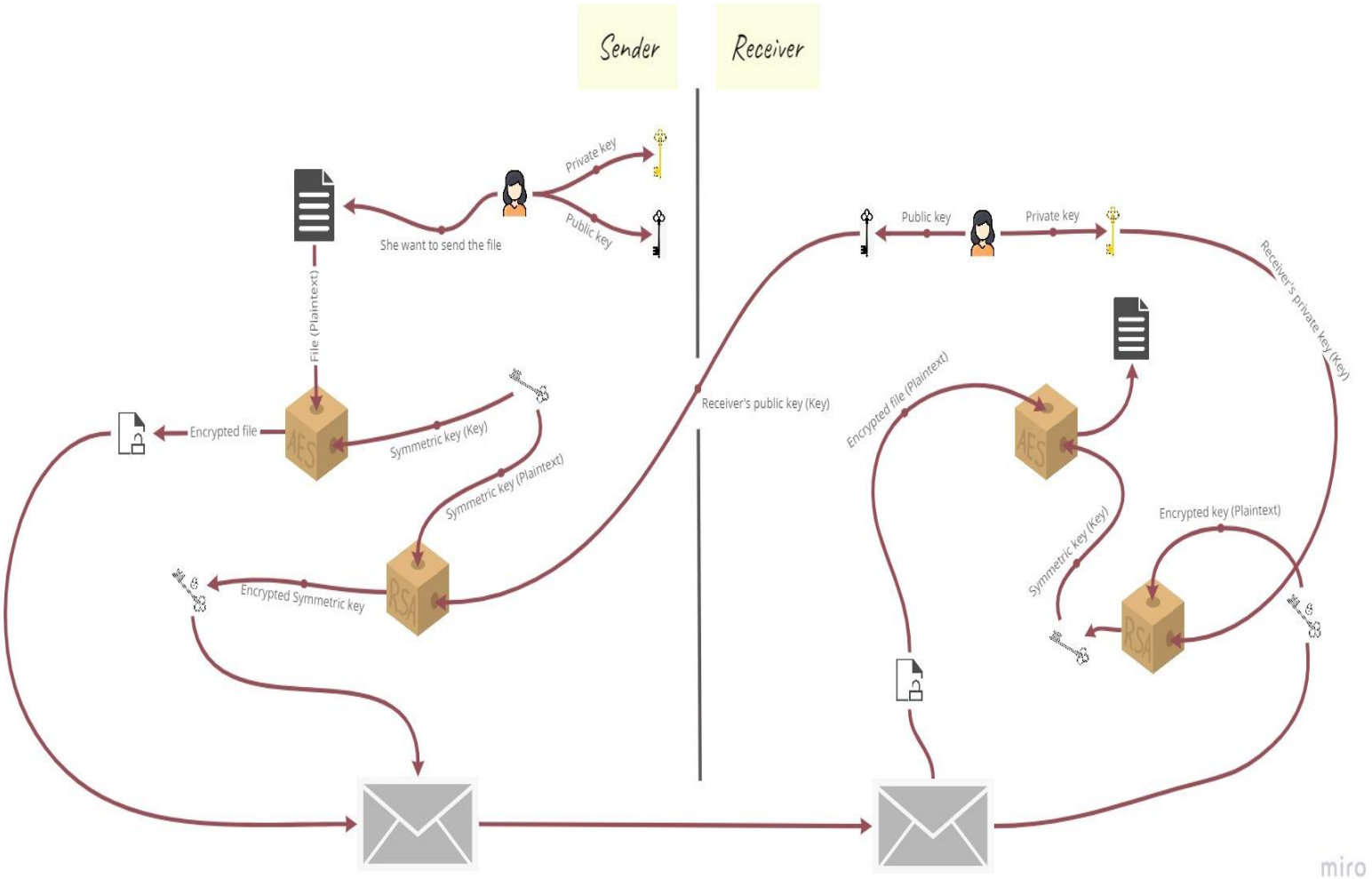


Figure 1 Workflow of the project

1.2 Approach and steps to implementation

1.2.1 Graphical user interface (GUI)

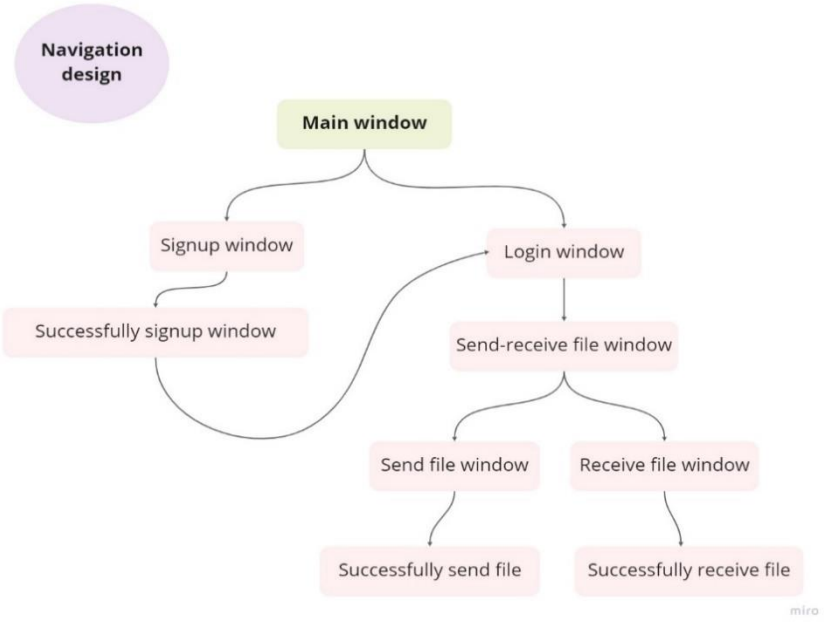


Figure3 Navigation design

Main window in Figure2 is the main window in our application. From main window the user either sign up or log in. If a new user trying to enter our application the application will give the user an error massage like shown in Figure4. After, a new user press sign up button as in Figure6 the app will show successful massage as in Figure8 in this moment **we store the password securely** since we encrypt the password before we store it in database. In addition, **public and private key will be generated** for a new user as shown in Figure6. If the password is to short error massage will output on the window as shown in Figure5. After sign up successfully in our app the user will press back button to login in our app as shown in Figure8. The user will enter his information and press log in button as shown in Figure7.

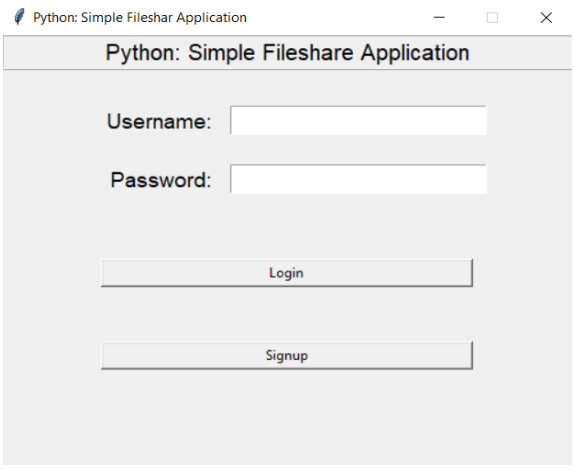


Figure4 Main window

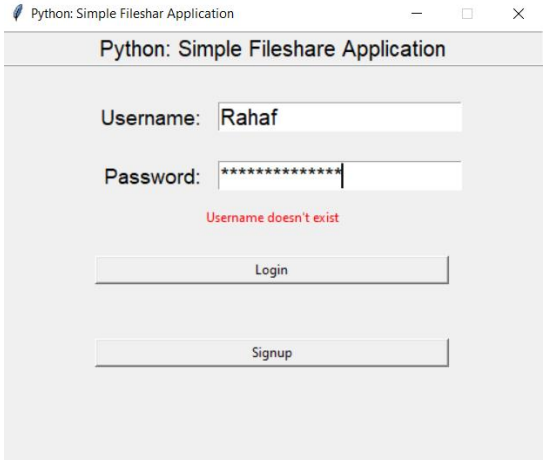


Figure2 up normal case of sign up entered username does not exist

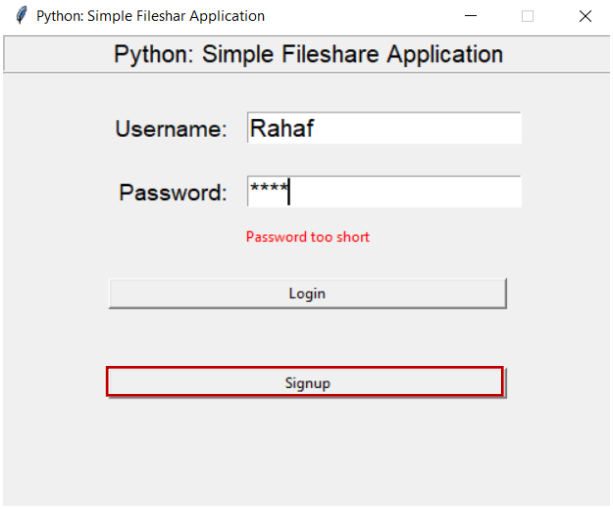
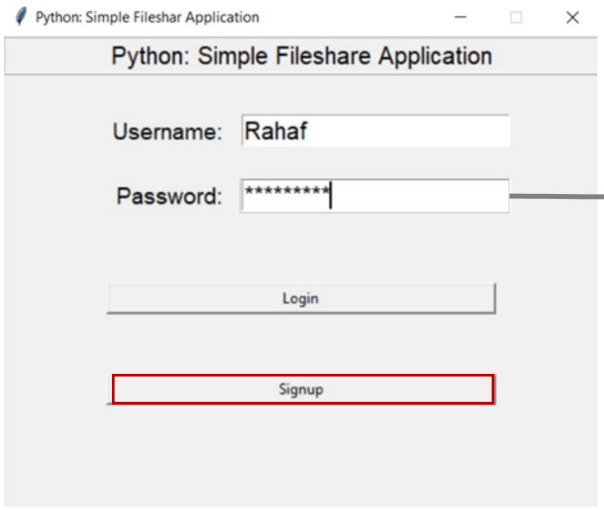


Figure5 Up normal case for signup password to short

Figure9 will show after the user log in. The user has menu contains two chooses if he wants to send a file or chick his received files. **In case the user wants to send a file,** he will press send file button. The new window will appear as shown in Figure10. The first normal case is if the user write username and select right file that he wants to send as shown in Figure11. The second normal case is when user select a file and add some words and save all changes that he made and then send the file as shown in Figure12.



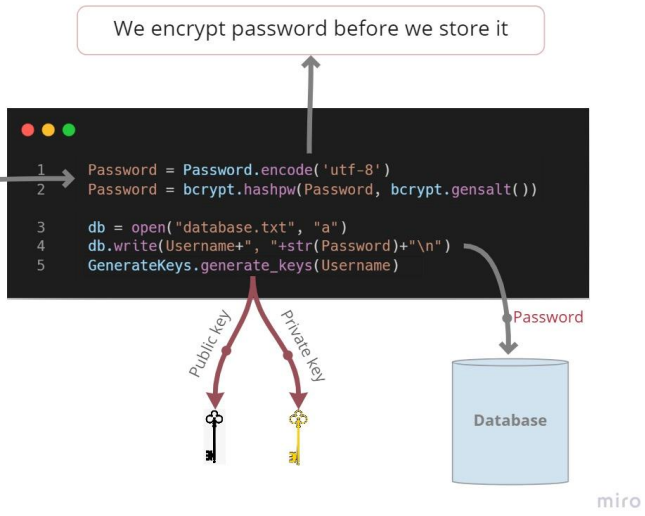


Figure7 Signup case

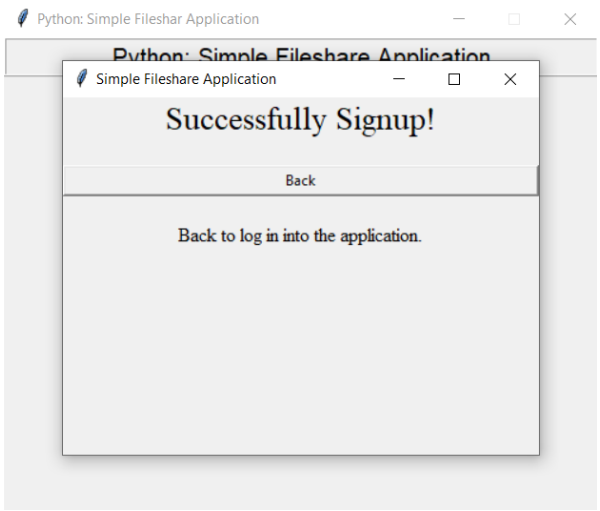


Figure6 Successfully signup window

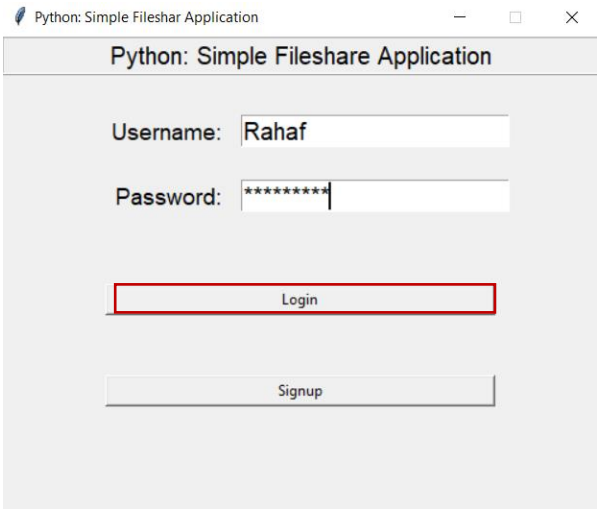


Figure8 Login case

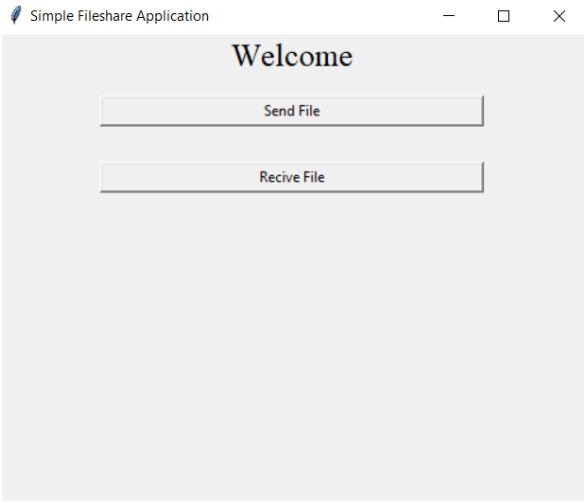


Figure10 Send - receive file window

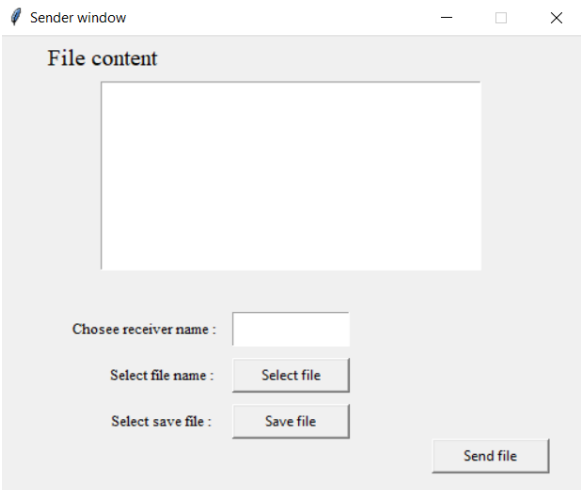


Figure9 Sender window

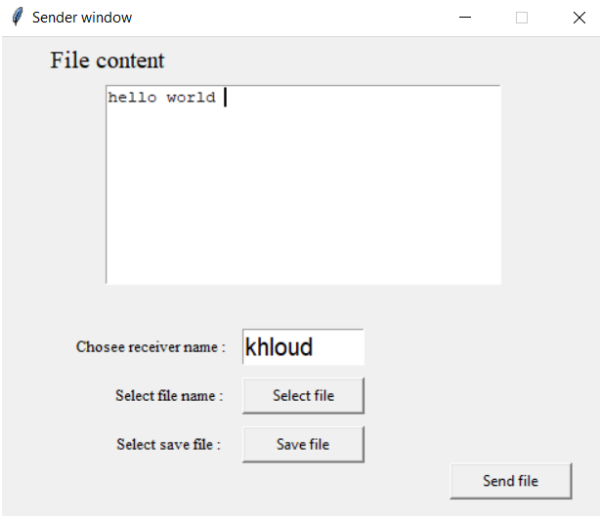


Figure11 First normal case to send a file

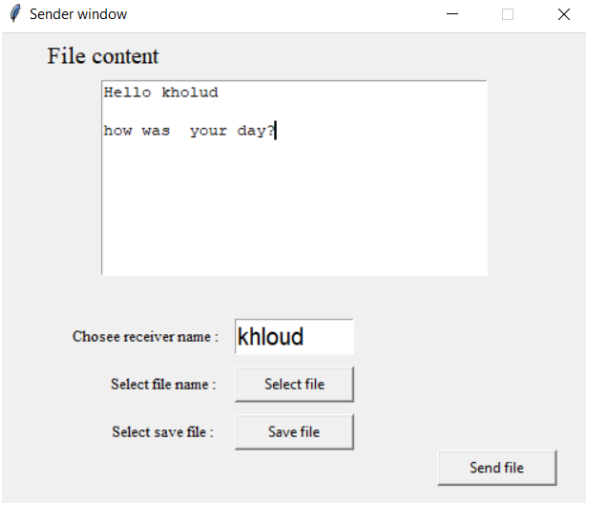


Figure12 Second normal case of send a file

Up normal cases:

- The user did not write receiver name and not select file as shown in Figure13.
- The user did not write receiver name as shown in Figure15.
- The user did not select file as shown in Figure14.

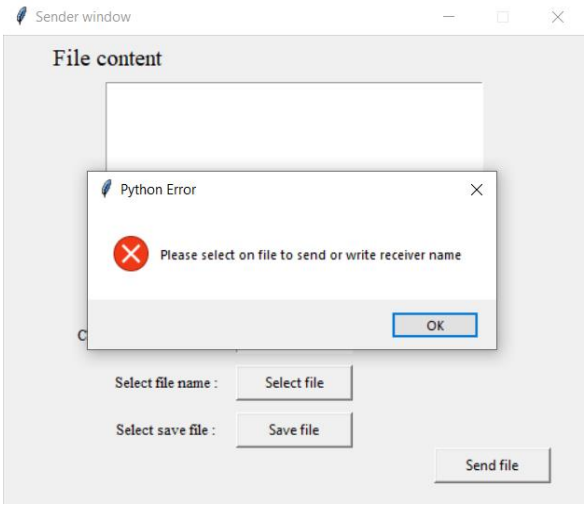


Figure13 First up normal case of send file

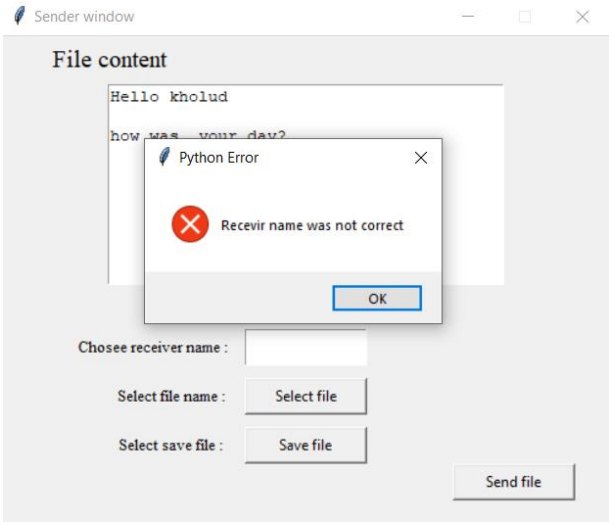


Figure 15 Second up normal case of send file

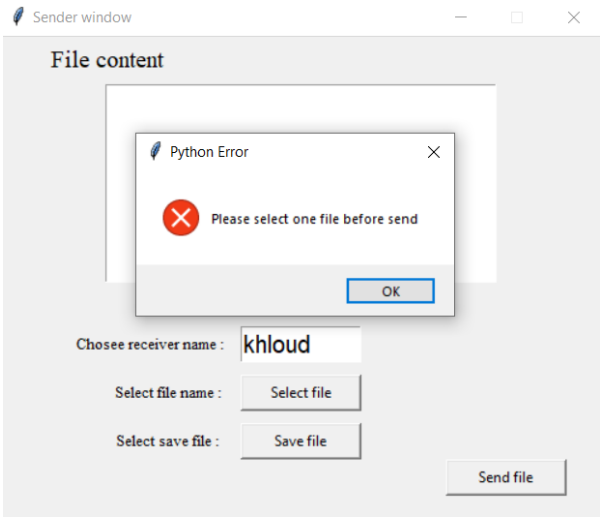


Figure 14 Third up normal case of send file

Either user do first, or second normal case and press send button new window will appear to show the success of sending file and output ciphertext as shown in Figure16.

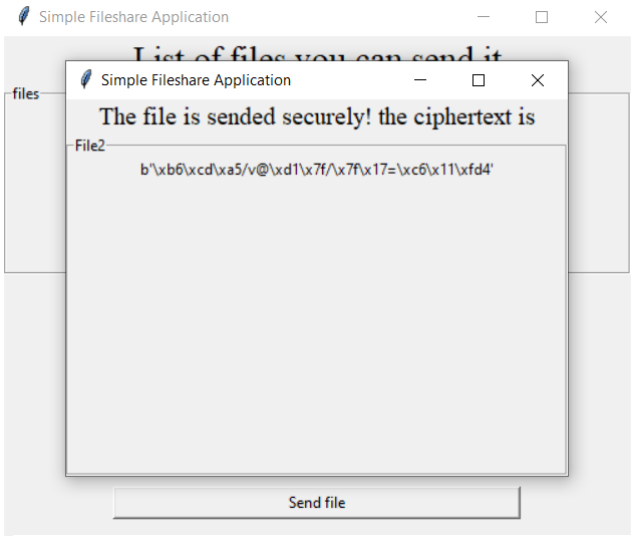


Figure 16 Successful Send window

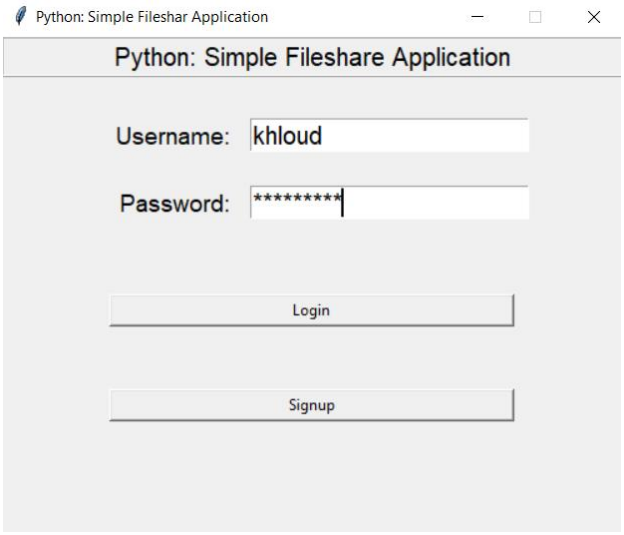


Figure 19 Login and chose receive file

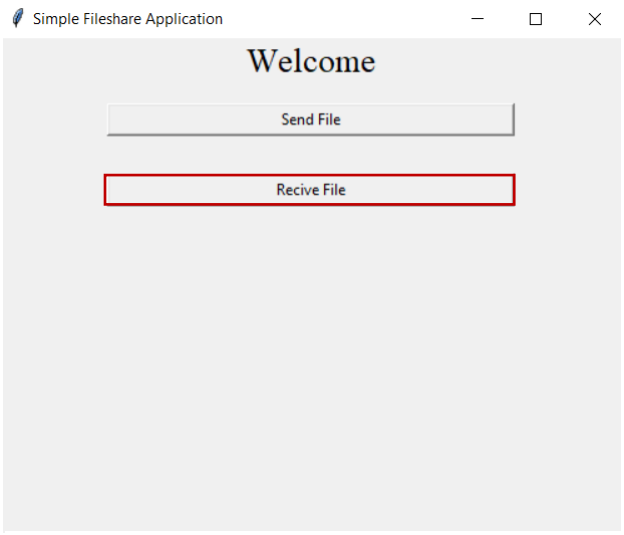


Figure 18 Send - receive file window

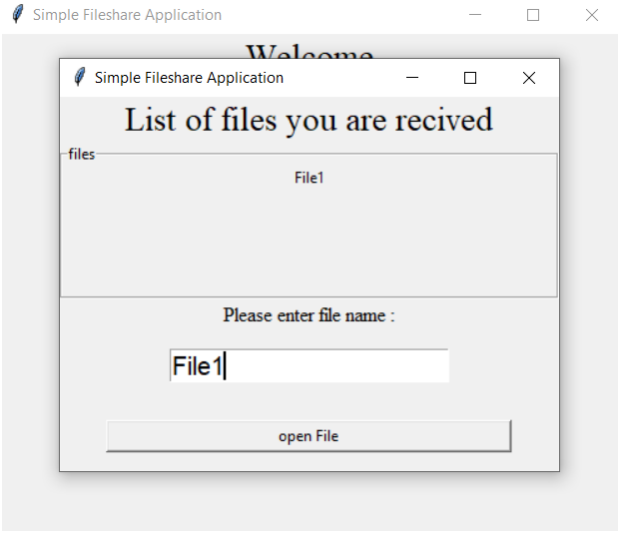


Figure 17 Received files window

Figure 20 The content of selected file

After the user send his file the receiver can see all received files on his account after he log in as shown in Figure19 and then chose receive file button as shown in Figure 18. If the recipient wants to read the file, he must first type in the name of the file he wants to read and then press open file button as shown in Figure17 and then the user will see his file content as shown in Figure20.

1.2.2 Encryption and Decryption

1. Generate and load keys

1.1 Generate public and private keys

After the user signs up for the application, the public and private keys will be generated and saved in the 'Users Keys' folder. The 'Users Keys' folder contains two files for each user, one for the public key and the other for the private key. **Note, the format of the files that contain the public and the private key is 'PEM' file format. The 'PEM' file format saved the keys in Base64 encoded. It is usually used to store cryptographic keys.**

The `generate_key()` method is shown in Figure 21. It takes the username as a parameter and generates the keys using `newkeys()` method. `The newkeys()` method takes one parameter that represents the size of generated keys, here we choose 1024 keys size. Then create and write the keys in two files (named as 'username' +'privkey' or 'pubkey').

```
5   def generate_keys(username):
6       # generate keys, the size of key in bit is 1024
7       (pubKey, privKey) = rsa.newkeys(1024)
8
9       # the path and name of file that will contain the public key
10      directoryname1 = 'Users Keys/'+username + 'pubkey.pem'
11      # create and open the file, write the public key on it
12      # the 'PEM' file format often used to store cryptographic keys
13      with open(directoryname1, 'wb') as f:
14          f.write(pubKey.save_pkcs1('PEM'))
15
16      # the path and name of file that will contain the private key
17      directoryname2 = 'Users Keys/'+username + 'privkey.pem'
18      # create and open the file, write the public key on it
19      # the 'PEM' file format often used to store cryptographic keys
20      with open(directoryname2, 'wb') as f:
21          f.write(privKey.save_pkcs1('PEM'))
```

Figure 21 Generate keys method

1.2 Load public and private keys

The `load_keys()` method load and return the keys(public and private) and used them as a parameter to the RSA cipher either in the encryption part or decryption part.

The `load_key()` method is shown in Figure 22. It takes the username as a parameter, opens the files that contain the public key and private key of the user, then returns it.

```
4   def load_keys(username):
5
6       # the path and name of file that contain the public key
7       directoryname1 = 'Users Keys/'+username + 'pubkey.pem'
8       # open file and read the public key
9       with open(directoryname1, 'rb') as f:
10          pubKey = rsa.PublicKey.load_pkcs1(f.read())
11
12      # the path and name of file that contain the private key
13      directoryname2 = 'Users Keys/'+username + 'privkey.pem'
14      # open file and read the private key
15      with open(directoryname2, 'rb') as f:
16          privKey = rsa.PrivateKey.load_pkcs1(f.read())
17
18      return pubKey, privKey
```

Figure 22 Load/Get keys method

2. Encryption part: encrypt_file_and_key() method

After the user chooses the 'Send' operation from the menu in the window and selects the file name and enters the receiver's name. Then, the user clicks on the 'Send File' button, The file name and receiver name will be sent to the `encrypt_file_and_key()` method. Then the file will be encrypted and show the content of the file as ciphertext. Also, the symmetric key will be encrypted. The `encrypt_file_and_key()` perform the encryption process of file and key with two helper methods, `aes_file_encrypt()` and `rsa_key_encrypt()`.

2.1 Initial setting before encryption

In `encrypt_file_and_key()` method in `SenderEnc` class, it takes two parameters: File name and Receiver name.

```
13  def encrypt_file_and_key(receiver_name, filename):
```

Figure 23 Parameters of encrypt_file_and_key() method

The initial setting of `encypt_file_and_key()` method is:

- Load public key of the receiver to use it in RSA encryption.

```
15      # load public key of receiver to use it in RSA encryption
16      pubKeyReceiver, privKeyReceiver = LoadKeys.load_keys(receiver_name)
17      receiver_public_key = pubKeyReceiver
```

Figure 24 Initial setting1 in encrypt_file_and_key() method

- Write in CSV file, the receiver's name and the file name that will be sent. The goal of doing so is when the receiver logs in to the application, the received file names will appear in the window, then it can choose which file he/she want to opened/decrypted.

```
19      # write in cvs the receiver name and the file name will be send to it
20      file = open('ReceiversMSG.csv', 'a+', newline='')
21      csvwriter = csv.writer(file)
22
23      new_receiver = [receiver_name, filename]
24      csvwriter.writerow(new_receiver)
25      file.close()
```

Figure 25 Initial setting2 in encrypt_file_and_key() method

- The16-byte symmetric key will be generated randomly, to be used in the AES as key and RSA as data.

```
27      # Generate random 16-Byte symmetric key
28      Key = token_bytes(16)
```

Figure 26 Initial setting3 in encrypt_file_and_key() method

2.2 aes_file_encrypt() helper method

```
30      # encrypt file and return the cipher text in the file
31      cipherMSG = aes_file_encrypt(Key, filename)
```

Figure 27 Calling the aes_file_encrypt() helper method

The `aes_file_encrypt()` method takes two parameters and one default parameter. (key: the symmetric key that is randomly generated by `token_bytes(16)`. Filename: the name of the file that contains the message to be encrypted. The chuck size: the message(plaintext) must be written as a chunk(block) before it is encrypted by the AES. The AES required that the size of the plaintext is the block size, or multiple of block size if the plaintext is larger than the block size. Here we have chosen the block size to be 2048, it could be any number that is multiple of 16. This parameter is used during reading the message from the file as a block).

```
40  def aes_file_encrypt(key, filename, chunk_size=2048):
41      """
42      :param key: the symmetric key
43      :param filename: the file that will be encrypted
44      :param chunk_size: to read message as chunk(block)
45      :return: encrypted content(message) in the file
46      """
```

Figure 28 Parameters of aes_file_encrypt() helper method

The details of the method are illustrated below:

1. The new file name has the same name as the original file. But with adding the ‘encrypted’ word.

```
48      # store the name for the new file, will contain a message after encrypted
49      output_filename = filename + 'encrypted.txt'
```

Figure 29 Step1 in aes_file_encrypt() helper method

2. Get the file size to write it in the new file. The benefit of doing so is to store the original size of the message before it is encrypted. This will help the receiver side to know the size of the original message and see if the message has been padded before encryption or not. If has been padded, the receiver will remove the padding.

```
51      # store the file size (original message size before benign encrypted)
52      # it used in the decryption to see if the message have been padded or not in the encryption.
53      filesize = os.path.getsize(filename+'.txt')
```

Figure 30 Step2 in aes_file_encrypt() helper method

3. Initialize the IV and create the AES cipher.

```
55      # generate the iv
56      iv = token_bytes(16)
57      # create the AES cipher
58      aes = AES.new(key, AES.MODE_CBC, iv)
```

Figure 31 Step3 in aes_file_encrypt() helper method

4. Open the file that contains the plaintext and open the new file that will contain the ciphertext.

```
60      # open the file that contain message, name the object file as inputfile
61      with open(filename+'.txt', 'rb') as inputfile:
62          # create and open new file to save the encrypted message into it, name the object file as outputfile
63          with open(output_filename, 'wb') as outputfile:
```

Figure 32 Step4 in aes_file_encrypt() helper method

5. Store the file size and the IV to be used on the receiver side. The reason behind storing the file size (plaintext size before it has been encrypted) is to know the amount of padded that is done before encryption. Thus, we ensure to take the plaintext without any padding after decryption. The IV must be shared between the sender and the receiver, so the receiver can used it in decryption.

```
65          # write file size into the output file, to used it at receiver side and check the size of original message the see if it has been padded or not
66          outputfile.write(struct.pack('<Q', filesize))
67          # write the iv into the outputfile, to used it at receiver side while decryption
68          outputfile.write(iv)
```

Figure 33 Step5 in aes_file_encrypt() helper method

6. Read the message in the original file as a block in Line 73. If the plaintext is less than the block size, do padding in the last block. If the plaintext is bigger than the block size, here is where the mode of operation (CBC) of AES comes into play. The mode of operation (e.g. CBC) allows the plaintext to be multiple of block size to encrypted. If the last block is less than the block size, do padding.

```
70          while True:
71              # the AES required the data to be in chunks, each chunk is multiple of 16 bytes
72              # read the message as chunks
73              data = inputfile.read(chunk_size)
74              n = len(data)
75              if n == 0:
76                  break
77              # do padding if we need
78              # (if the last chunk of the message is less than multiple of 16 byte, it will be padded (add space to complete the chunk size)
79              elif n % 16 != 0:
80                  data += ' ' * (16 - n % 16)  # <- padded with spaces
```

Figure 34 Step6 in aes_file_encrypt() helper method

7. Encrypt the plaintext using AES cipher, then write the ciphertext in the new file.

```
82                  # encrypt the message using AES
83                  encryptedMSG = aes.encrypt(data)
84                  # write the encrypted message to the new file
85                  outputfile.write(encryptedMSG)
```

Figure 35 Step7 in aes_file_encrypt() helper method

2.3 rsa_key_encrypt() helper method

The sender and receiver must share two things to do encryption and decryption in AES, the IV(it is publicly shared) and the symmetric key(it must be secure between the sender and the receiver). So, this is why we need to encrypt the symmetric key to keep it secure. Then move it to the receiver. Here we will encrypt the symmetric key using RSA.

```
33      # encrypt key
34      rsa_key_encrypt(receiver_public_key, Key, filename)
```

Figure 36 Calling the rsa_key_encrypt() helper method

The `enrypt_key()` method take three parameters. (receiver_public_key: it is used in RSA encryption. Key: the symmetric key(data) that will be encrypted using RSA. Filename: the name of the file that is encrypted by AES using this symmetric key, we used its name just to create a new file with a consistent name. the new file has the same name but adds the word 'encryptedkey').

```
91  def rsa_key_encrypt(receiver_public_key, key, filename):
92      """
93      :param receiver_public_key: the public key of the receiver to used it in the RSA encryption
94      :param key: the data(symmetric key) that will encrypted
95      :param filename: the file name of the message that are encrypted by the AES.
96      :return:
97      """
```

Figure 37 Parameters of rsa_key_encrypt() helper method

The details of the method are illustrated below:

1. Store the name of the new file.

```
99      # store the name for the new file, will contain a key after encrypted
100     output_filename = filename + 'encryptedKey.txt'
```

Figure 38 Step1 in rsa_key_encrypt() helper method

2. Encrypt the key by RSA cipher using the receiver public key.

```
102     # encrypt the key using RSA
103     encrypted_key= rsa.encrypt(key, receiver_public_key)
```

Figure 39 Step2 in rsa_key_encrypt() helper method

3. Write the encrypted key in the new file.

```
105     # write the encrypted key to the new file
106     with open(output_filename, 'wb') as outputfile:
107         outputfile.write(encrypted_key)
```

Figure 40 Step3 in rsa_key_encrypt() helper method

3. Decryption part: decrypt_file_and_key() method

After the user chooses the 'Receive' operation from the menu and enters the file name. Then, the user clicks on the 'open File' button, The file name and user name will be sent to the `decrypt_file_and_key()` method. Then the key will be decrypted to use in file decryption operation. After that, the file will be decrypted and show the content of the file as plain text. The `decrypt_file_and_key()` perform decryption process of file and key with two helper methods, `rsa_key_decrypt()` and `aes_file_decrypt()`.

3.1 Initial setting before decryption

The `decrypt_file_and_key()` method in `ReceiverDec` class, it takes two parameters: File name and Username.

```
9    def decrypt_file_and_key(username, filename):
```

Figure 41 Parameters of decrypt_file_and_key() method

The initial setting of `decrypt_file_and_key()` method is:

- Load private key of the user to use it in RSA decryption.

```
11       # load keys then take the private key of the user to use it in RSA decryption
12       pubKey, privKey = LoadKeys.load_keys(username)
13       reciver_private_key = privKey
```

Figure 42 Initial setting1 in decrypt_file_and_key() method

3.2 rsa_key_decrypt() helper method

The sender and receiver must share the symmetric key to do the encryption and decryption using AES. The symmetric key has been received at the receiver side. So, we need to decrypt it to use it in the AES decryption of the ciphertext.

```
15       # decrypt key
16       rsa_key_decrypt(user_private_key, filename)
```

Figure 43 Calling the rsa_key_decrypt() helper method

The `rsa_key_decrypt()` method takes two parameters. (user_private_key: it is used in RSA decryption. Filename: Name of the file that contains original text(plaintext), it is used to get the file name that contains the encrypted key, because the encrypted key file has the same name as the original text file with add 'encryptedKey' word to it.)

```
25  def rsa_key_decrypt(user_private_key, filename):
26      """
27      :param user_private_key: it is used in RSA decryption
28      :param filename: filename of the original text
29      :return:
30      """
```

Figure 44 Parameters of rsa_key_decrypt() helper method

The details of the method are illustrated below:

1- Open the file that contains the encrypted key, read, and store the encrypted key to be decrypted.

```
32      # get the name of file that contain the encrypted key
33      # it has same name of original text file, with add 'encryptedKey'
34      encryptedkey_filename = filename + 'encryptedKey.txt'
35
36      # open the file and read the encrypted key
37      with open(encryptedkey_filename, 'rb') as inputfile:
38          encryptedkey = inputfile.read()
39
```

Figure 45 Step1 in rsa_key_decrypt() helper method

2- Decrypt the key by RSA cipher using the user's private key.

```
40      # decrypt the key using RSA
41      decryptedkey = rsa.decrypt(encryptedkey, user_private_key)
```

Figure 46 Step2 in rsa_key_decrypt() helper method

3- Write the decrypted key on the new file to be used in AES decryption.

```
43      # name for new file that will contain a key after decrypted
44      output_filename = filename + 'decryptedKey.txt'
45      # write decrypted key into new file
46      with open(output_filename, 'wb') as outputfile:
47          outputfile.write(decryptedkey)
```

Figure 47 Step3 in rsa_key_decrypt() helper method

3.3 aes_file_decrypt() helper method

```
18      # decrypt file and return the plaintext in the file
19      decryptedMSG = aes_file_decrypt(filename)
20
```

Figure 48 Calling the rsa_file_decrypt() helper method

The aes_file_decrypt() method takes one parameter and one default parameter. (filename: Name of the file that contains original text(plaintext), it is used to get the file name that contains encrypted text(ciphertext), because the encrypted text(ciphertext) file has the same name as the original text file with add 'encrypted' word to it. The chuck size: the size of a chunk (block) of the ciphertext, it is used during reading the ciphertext from the file. The AES required that the size of the ciphertext be the same as the block size, or multiple of block size if the ciphertext is larger than the block size. Here we have chosen the block size to be 2048, it could be any number that is multiple of 16.)

The details of the method are illustrated below:

1- Read the ciphertext from the encrypted file.

```
57      # open file and read decrypted key
58      decryptedkey_filename = filename + 'decryptedKey.txt'
59      with open(decryptedkey_filename, 'rb') as inputfile:
60          Key = inputfile.read(16)
```

Figure 49 Step1 in rsa_file_decrypt() helper method

2- Prepare the name of the file that contains decrypted key, and the name of the new file that will contain the plaintext after being decrypted by the AES.

```
62      # store the name for the new file, will contain a message after decrypted
63      output_filename = filename + 'decrypted.txt'
64      # name of the file that contain the encrypted message
65      encryptedMSG_filename = filename + 'encrypted.txt'
```

Figure 50 Step2 in rsa_file_decrypt() helper method

3- Open the file that contains the ciphertext and open the file that will contain the text after decrypted.

```
    # open the file that contain the decrypted message, name the object file as inputfile
    with open(encryptedMSG_filename, 'rb') as inputfile:
        # create and open new file to save the decrypted message into it, name the object file as outputfile
        with open(output_filename, 'wb') as outputfile:
```

Figure 51 Step3 in rsa_file_decrypt() helper method

4- Read the file size and IV. The file size is used to know the original size of the plaintext before encryption, so we can remove any padding if any. The IV is used in AES decryption. It must be shared between the sender and receiver.

```
            # read file size(original plaintext size), it used to check if it has been padded or not
            filesize = struct.unpack('<Q', inputfile.read(struct.calcsize('<Q')))[0]
            # read the iv to used it in ASA cipher decryption
            iv = inputfile.read(16)
```

Figure 52 Step4 in rsa_file_decrypt() helper method

5- Create the AES cipher. With the shared IV and symmetric key.

```
            # create the AES cipher
            aes = AES.new(Key, AES.MODE_CBC, iv)

```

Figure 53 Step5 in rsa_file_decrypt() helper method

6- Read ciphertext from the file.

```
            while True:
                # the AES required the data to be in chunks, each chunk is multiple of 16 bytes
                # read the message as chunks
                data = inputfile.read(chunk_size)
                n = len(data)
                if n == 0:
                    break
```

Figure 54 Step6 in rsa_file_decrypt() helper method

7- Decrypt the ciphertext using AES cipher, then write the plaintext into the new file with remove any padding if any.

```
87              # decrypt the message using AES
88              decryptedMSG = aes.decrypt(data)
89
90          # write in file the decrypted message
91          outputfile.write(decryptedMSG)
92
93          # remove padding if any
94          outputfile.truncate(filesize)
```

Figure 55 Step7 in rsa_file_decrypt() helper method

1.3 Source code

Main

```
import tkinter
from tkinter import *
import sign
import login

#Setting up the Main Frame
root = Tk()
root.title("Python: Simple Fileshar Application")
width = 500
height = 400
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
x = (screen_width/2) - (width/2)
y = (screen_height/2) - (height/2)
root.geometry("%dx%d+%d+%d" % (width, height, x, y))
root.resizable(0, 0)

#Designing the Layout

#VARIABLES
USERNAME = StringVar()
PASSWORD = StringVar()
FILENAME = StringVar()
#FRAMES
Top = Frame(root, bd=2,  relief=RIDGE)
Top.pack(side=TOP, fill=X)
Form = Frame(root, height=400)
Form.pack(side=TOP, pady=20)

#LABELS
lbl_title = Label(Top, text = "Python: Simple Fileshare
Application", font=('arial', 15))
lbl_title.pack(fill=X)
lbl_username = Label(Form, text = "Username:", font=('arial', 14),
bd=15)
lbl_username.grid(row=0, sticky="e")
lbl_password = Label(Form, text = "Password:", font=('arial', 14),
bd=15)
lbl_password.grid(row=1, sticky="e")
lbl_text = Label(Form)
lbl_text.grid(row=2, columnspan=2)

#ENTRY WIDGETS
username = Entry(Form, textvariable=USERNAME, font=(14))
username.grid(row=0, column=1)
password = Entry(Form, textvariable=PASSWORD, show="*", font=(14))
password.grid(row=1, column=1)

def Login(event=None):
        login.gainAccess(USERNAME.get(),PASSWORD.get())

def signup(event=None):
        sign.register(USERNAME.get(),PASSWORD.get())

#BUTTON WIDGETS
btn_login = Button(Form, text="Login", width=45, command = Login)
btn_login.grid(pady=25, row=3, columnspan=2)
btn_login.bind('<Return>', Login)

btn_signup = Button(Form, text="Signup", width=45, command = signup)
btn_signup.grid(pady=25, row=4, columnspan=2)
btn_signup.bind('<Return>', signup)

#Initializing the Application
if __name__ == '__main__':
    root.mainloop()
```

Signup

```
from tkinter import *
import bcrypt
import main
import GenerateKeys

#signup_window
def HomeWindow1():
    global Home
    main.root.withdraw()
    Home = Toplevel()
    Home.title("Simple Fileshare Application")
    width = 400
    height = 300
    screen_width =  main.root.winfo_screenwidth()
    screen_height =  main.root.winfo_screenheight()
    x = (screen_width/2) - (width/2)
    y = (screen_height/2) - (height/2)
    main.root.resizable(0, 0)
    Home.geometry("%dx%d+%d+%d" % (width, height, x, y))
    lbl_home = Label(Home, text="Successfully Signup!", font=('times new roman', 20)).pack()
    btn_back = Button(Home, text='Back', command=Back).pack(pady=20, fill=X)

    lb2_home = Label(Home, text="Back to log in into the application.", font=('times new roman', 12)).pack()

# to get register in application
def register(Username,Password):
    Username = Username
    Password = Password
    db = open("database.txt", "r")
    d = []
    for i in db:
        a,b = i.split(",")
        b = b.strip()
        c = a,b
        d.append(a)
    if not len(Password)<=8:
        db = open("database.txt", "r")
        if Username in d:
            main.lbl_text.config(text="username already exist", fg="red")
        elif not Username ==None:
                    Password = Password.encode('utf-8')
                    Password = bcrypt.hashpw(Password, bcrypt.gensalt())
                    db = open("database.txt", "a")
                    db.write(Username+", "+str(Password)+"\n")
                    GenerateKeys.generate_keys(Username)
                    HomeWindow1()
    else:
          main.lbl_text.config(text="Password too short", fg="red")

#define event
def Back():
    Home.destroy()
    main.root.deiconify()

```

Login

```
import tkinter
from tkinter import *
import bcrypt
import main
import reciverGUI
import senderGUI

# login_window
def HomeWindow():
    global Home
    main.root.withdraw()
    Home = Toplevel()
    Home.title("Simple Fileshare Application")
    width = 500
    height = 400
    screen_width = main.root.winfo_screenwidth()
    screen_height = main.root.winfo_screenheight()
    x = (screen_width/2) - (width/2)
    y = (screen_height/2) - (height/2)
    main.root.resizable(0, 0)
    Home.geometry("%dx%d+%d+%d" % (width, height, x, y))
    lbl_home = Label(Home, text="Welcome", font=('times new roman', 20)).pack()
    btn_send = Button(Home, text='Send File', width=45, command=send).pack(pady=15)
    btn_recive = Button(Home, text='Recive File', width=45, command=recive).pack(pady=15)

# to get access to application
def gainAccess(Username, Password):
    Username1 = Username
    Password1 = Password

    if not len(Username1 or Password1) < 1:
        if True:
            db = open("database.txt", "r")
            d = []
            f = []
            for i in db:
                a,b = i.split(",")
                b = b.strip()
                c = a,b
                d.append(a)
                f.append(b)
                data = dict(zip(d, f))
            try:
                if Username1 in data:
                    hashed = data[Username1].strip('b')
                    hashed = hashed.replace("'", "")
                    hashed = hashed.encode('utf-8')

                    try:
                        if bcrypt.checkpw(Password1.encode(), hashed):
                            HomeWindow()
                        else:
                            main.lbl_text.config(text="Invalid username or password", fg="red")

                    except:
                        main.lbl_text.config(text="Incorrect passwords or username", fg="red")
                else:
                    main.lbl_text.config(text="Username doesn't exist", fg="red")
            except:
                main.lbl_text.config(text="Password or username doesn't exist", fg="red")
        else:
            main.lbl_text.config(text="Error logging into the system", fg="red")
    else:
        main.lbl_text.config(text="Please attempt login again", fg="red")


def send():
    senderGUI.Window3()


def recive():
    reciverGUI.Window1()
```

SenderEnc

```python
import LoadKeys  # to return the receiver public key
import csv  # to open and write in csv file

from secrets import token_bytes  # to random 16 byte initialization vector(iv)
from Crypto.Cipher import AES  # to create AES cipher
import os  # to get file size
import struct  # to write file size in the file, we need this package
import rsa  # to create RSA cipher


def encrypt_file_and_key(receiver_name, filename):

    # load public key of receiver to use it in RSA encryption
    pubKeyReceiver, privKeyReceiver = LoadKeys.load_keys(receiver_name)
    receiver_public_key = pubKeyReceiver

    # write in cvs the receiver name and the file name will be send to it
    file = open('ReceiversMSG.csv', 'a+', newline='')
    csvwriter = csv.writer(file)

    new_receiver = [receiver_name, filename]
    csvwriter.writerow(new_receiver)
    file.close()

    # Generate random 16-Byte symmetric key
    Key = token_bytes(16)

    # encrypt file and return the cipher text in the file
    cipherMSG = aes_file_encrypt(Key, filename)

    # encrypt key
    rsa_key_encrypt(receiver_public_key, Key, filename)

    # return the cipher text to be appeared in the window
    return str(cipherMSG)


def aes_file_encrypt(key, filename, chunk_size=2048):
    """
    :param key: the symmetric key
    :param filename: the file that will be encrypted
    :param chunk_size: to read message as chunk(block)
    :return: encrypted content(message) in the file
    """

    # store the name for the new file, will contain a message after encrypted
    output_filename = filename + 'encrypted.txt'

    # store the file size (original message size before benign encrypted)
    # it used in the decryption to see if the message have been padded or not in the encryption.
    filesize = os.path.getsize(filename+'.txt')

    # generate the iv
    iv = token_bytes(16)
    # create the AES cipher
    aes = AES.new(key, AES.MODE_CBC, iv)

    # open the file that contain message, name the object file as inputfile
    with open(filename+'.txt', 'rb') as inputfile:
        # create and open new file to save the encrypted message into it, name the object file as outputfile
        with open(output_filename, 'wb') as outputfile:

            # write file size into the output file, to used it at receiver side and check the size of original message then see if it has been padded or not
            outputfile.write(struct.pack('<Q', filesize))
            # write the iv into the outputfile, to used it at receiver side while decryption
            outputfile.write(iv)

            while True:
                # the AES required the data to be in chunks, each chunk is multiple of 16 bytes
                # read the message as chunks
                data = inputfile.read(chunk_size)
                n = len(data)
                if n == 0:
                    break
                # do padding if we need
                # (if the last chunk of the message is less than multiple of 16 byte, it will be padded (add space to complete the chunk size)
                elif n % 16 != 0:
                    data += ' ' * (16 - n % 16)  # <- padded with spaces

                # encrypt the message using AES
                encryptedMSG = aes.encrypt(data)
                # write the encrypted message to the new file
                outputfile.write(encryptedMSG)

    # return the encrypted message to be written in the window
    return encryptedMSG


def rsa_key_encrypt(receiver_public_key, key, filename):
    """
    :param receiver_public_key: the public key of the receiver to used it in the RSA encryption
    :param key: the data(symmetric key) that will encrypted
    :param filename: the file name of the message that are encrypted by the AES.
    :return:
    """

    # store the name for the new file, will contain a key after encrypted
    output_filename = filename + 'encryptedKey.txt'

    # encrypt the key using RSA
    encrypted_key= rsa.encrypt(key, receiver_public_key)

    # write the encrypted key to the new file
    with open(output_filename, 'wb') as outputfile:
        outputfile.write(encrypted_key)
```

RecevierDec

```python
import LoadKeys  # to return the receiver public key

from Crypto.Cipher import AES
import struct
import rsa


def decrypt_file_and_key(username, filename):

    # load keys then take the private key of the user to use it in RSA decryption
    pubKey, privKey = LoadKeys.load_keys(username)
    user_private_key = privKey

    # decrypt key
    rsa_key_decrypt(user_private_key, filename)

    # decrypt file and return the plaintext in the file
    decryptedMSG = aes_file_decrypt(filename)

    # return the decrypted plaintext to appear in the window.
    # decode() method to remove prefix 'b' from the plaintext
    return decryptedMSG.decode()


def rsa_key_decrypt(user_private_key, filename):
    """
    :param user_private_key: it is used in RSA decryption
    :param filename: filename of the original text
    :return:
    """

    # get the name of file that contain the encrypted key
    # it has same name of original text file, with add 'encryptedKey'
    encryptedkey_filename = filename + 'encryptedKey.txt'

    # open the file and read the encrypted key
    with open(encryptedkey_filename, 'rb') as inputfile:
        encryptedkey = inputfile.read()

    # decrypt the key using RSA
    decryptedkey = rsa.decrypt(encryptedkey, user_private_key)

    # name for new file that will contain a key after decrypted
    output_filename = filename + 'decryptedKey.txt'
    # write decrypted key into new file
    with open(output_filename, 'wb') as outputfile:
        outputfile.write(decryptedkey)


def aes_file_decrypt(filename, chunk_size=2048):
    """
    :param filename: the file that will be decrypted
    :param chunk_size: to read message as chunk(block)
    :return:
    """

    # open file and read decrypted key
    decryptedkey_filename = filename + 'decryptedKey.txt'
    with open(decryptedkey_filename, 'rb') as inputfile:
        Key = inputfile.read(16)

    # store the name for the new file, will contain a message after decrypted
    output_filename = filename + 'decrypted.txt'
    # name of the file that contain the encrypted message
    encryptedMSG_filename = filename + 'encrypted.txt'

    # open the file that contain the decrypted message, name the object file as inputfile
    with open(encryptedMSG_filename, 'rb') as inputfile:
        # create and open new file to save the decrypted message into it, name the object file as outputfile
        with open(output_filename, 'wb') as outputfile:

            # read file size(original plaintext size), it used to check if it has been padded or not
            filesize = struct.unpack('<Q', inputfile.read(struct.calcsize('<Q')))[0]
            # read the iv to used it in ASA cipher decryption
            iv = inputfile.read(16)

            # create the AES cipher
            aes = AES.new(Key, AES.MODE_CBC, iv)

            while True:
                # the AES required the data to be in chunks, each chunk is multiple of 16 bytes
                # read the message as chunks
                data = inputfile.read(chunk_size)
                n = len(data)
                if n == 0:
                    break
                # decrypt the message using AES
                decryptedMSG = aes.decrypt(data)

            # write in file the decrypted message
            outputfile.write(decryptedMSG)

            # remove padding if any
            outputfile.truncate(filesize)

    # return the decrypted message to be written in the window
    return decryptedMSG
```

GenerateKeys

```python
import rsa


def generate_keys(username):
    # generate keys, the size of key in bit is 1024
    (pubKey, privKey) = rsa.newkeys(1024)

    # the path and name of file that will contain the public key
    directoryname1 = 'Users Keys/'+username + 'pubkey.pem'
    # create and open the file, write the public key on it
    # the 'PEM' file format often used to store cryptographic keys
    with open(directoryname1, 'wb') as f:
        f.write(pubKey.save_pkcs1('PEM'))

    # the path and name of file that will contain the private key
    directoryname2 = 'Users Keys/'+username + 'privkey.pem'
    # create and open the file, write the public key on it
    # the 'PEM' file format often used to store cryptographic keys
    with open(directoryname2, 'wb') as f:
        f.write(privKey.save_pkcs1('PEM'))
```

LoadKey

```python
import rsa


def load_keys(username):

    # the path and name of file that contain the public key
    directoryname1 = 'Users Keys/'+username + 'pubkey.pem'
    # open file and read the public key
    with open(directoryname1, 'rb') as f:
        pubKey = rsa.PublicKey.load_pkcs1(f.read())

    # the path and name of file that contain the private key
    directoryname2 = 'Users Keys/'+username + 'privkey.pem'
    # open file and read the private key
    with open(directoryname2, 'rb') as f:
        privKey = rsa.PrivateKey.load_pkcs1(f.read())

    return pubKey, privKey
```

SenderGUI

```python
import SenderEnc
import main
import tkinter
from tkinter import *
from tkinter import filedialog
from pathlib import Path
from tkinter import messagebox

file_name = ''
ReciverName = ''
def on_click(error_message):
   messagebox.showerror('Python Error', error_message)

def open_text_file(text_filed):
    try:
            global file_name
            # Choose file
            text_file = filedialog.askopenfilename(filetypes=[("Text files", "*.txt")])
            # store path
            file_name = str(Path(text_file)).split('\\')[-1].split('.')[0]
            # Read file content
            text_file = open(text_file, 'r')
            # Read any additional text
            new_words = text_file.read()
            # Add any additional text to text_filed
            text_filed.insert(tkinter.END, new_words)
            # close file
            text_file.close()
            # return file name
            #return file_name
    except:
        on_click('Please select one file to send')

def save_text_file(text_filed):
    if len(file_name) > 0:
        # Choose file
        text_file_1 = filedialog.askopenfilename(filetypes=[("Text files", "*.txt")])
        # open text_file
        text_file_1 = open(text_file_1, 'w')
        # get all text in text_filed from the first line to the end and write it on the main file
        text_file_1.write(text_filed.get(1.0, END))
    else:
        # pop up an error massege
        on_click("Please select file first")

def extraxt_names():
    options = []
    lines = []
    # Take all registered people names
    with open('database.txt') as f:
        lines = f.readlines()
    for line in lines:
        options.append(line.split(',')[0])
    return options

def send_text(recivername):

   global ReciverName
   names = extraxt_names()
   ReciverName = recivername
   if len(recivername) > 0 or len(file_name) > 0:
       if recivername in names:
           if len(file_name) > 0:
               Window4()
           else:
               # pop up an error massege
               on_click("Please select one file before send")
       else:
           # pop up an error massege
           on_click("Recevir name was not correct")
   else:
       on_click("Please select on file to send or write receiver name")


def Window3():
    # Create window
    sender_win = Tk()
    # put window at the center
    sender_win.eval('tk::PlaceWindow . center')
    # Determine title of created window
    sender_win.title("Sender window")
    # Determine size of the window and replace at the center
    width = 500
    height = 400
    screen_width = sender_win.winfo_screenwidth()
    screen_height = sender_win.winfo_screenheight()
    x = (screen_width / 2) - (width / 2)
    y = (screen_height / 2) - (height / 2)
    sender_win.resizable(0, 0)
    sender_win.geometry("%dx%d+%d+%d" % (width, height, x, y))
    # Create all labels in the window
    choose_file_name = Label(sender_win, text="Select file name :", font=('times new roman', 10))
    choose_save_file = Label(sender_win, text="Select save file :", font=('times new roman', 10))
    Describtion = Label(sender_win, text="File content", font=('times new roman', 15))
    choose_receiver_name = Label(sender_win, text="Enter receiver name :", font=('times new roman', 10))
    # Create text filed
    text_filed = Text(sender_win, width=40, height=10)
    reciver_name_text_filed = Text(sender_win, font=(8))
    # Create all buttons
    open_file_button = Button(sender_win, text="Select file", command= lambda: open_text_file(text_filed))
    save_button = Button(sender_win, text="Save file", command= lambda:save_text_file(text_filed))
    # Show all created labels
    text_filed.pack(pady=40)
    open_file_button.place(height=30, width=100, x=200, y=280)
    save_button.place(height=30, width=100, x=200, y=320)
    reciver_name_text_filed.place(height=30, width=100, x=200, y=240)
    choose_file_name.place(height=30, width=200, x=40, y=280)
    choose_save_file.place(height=30, width=200, x=40, y=320)
    choose_receiver_name.place(height=30, width=170, x=40, y=240)
    Describtion.place(height=30, width=170, x=5, y=5)



    send_file = Button(sender_win, text="Send file", command= lambda:send_text(reciver_name_text_filed.get("1.0","end-1c")))
    send_file.place(height=30, width=100, x=370, y=350)

    # Show sender window
    sender_win.mainloop()


def Window4():
    global Home
    main.root.withdraw()
    Home = Toplevel()
    Home.title("Simple Fileshare Application")
    width = 400
    height = 300
    screen_width = main.root.winfo_screenwidth()
    screen_height = main.root.winfo_screenheight()
    x = (screen_width / 2) - (width / 2)
    y = (screen_height / 2) - (height / 2)
    main.root.resizable(0, 0)
    Home.geometry("%dx%d+%d+%d" % (width, height, x, y))
    lbl_fileist = Label(Home, text="The file is sended securely! the ciphertext is", font=('times new roman', 15)).pack()
    labelframe1 = LabelFrame(Home, text=main.FILENAME.get())
    labelframe1.pack(fill="both", expand="yes")

    #import
    recivername11 = ReciverName
    FileName = file_name
    ciphertext = SenderEnc.encrypt_file_and_key(recivername11, FileName)

    bottomlabel = Label(labelframe1, text=ciphertext)
    bottomlabel.pack()
```

ReceiverGUI

```
from tkinter import *
import main
import RecevierDec
import csv

# Designing the Layout and Setting up the receiver Frames
def Window1():
    global Home
    main.root.withdraw()
    # Determine size of the window
    Home = Toplevel()
    Home.title("Simple Fileshare Application")
    width = 400
    height = 300
    screen_width =  main.root.winfo_screenwidth()
    screen_height =  main.root.winfo_screenheight()
    x = (screen_width/2) - (width/2)
    y = (screen_height/2) - (height/2)
    main.root.resizable(0, 0)
    Home.geometry("%dx%d+%d+%d" % (width, height, x, y))
    # Create labels in the window
    lbl_fileist = Label(Home, text="List of files you are recived",
font=('times new roman', 20)).pack()
    labelframe1 = LabelFrame(Home, text="files")
    labelframe1.pack(fill="both", expand="yes")
    # import
    file = open('ReceiversMSG.csv', 'r')
    csvreader = csv.reader(file)

    MSGList = ''
    for row in csvreader:
        if row[0] == main.USERNAME.get():
            MSGList = MSGList + row[1] + '\n'
    file.close()
   # Create labels and button in the window
    toplabel = Label(labelframe1, text=str(MSGList))
    toplabel.pack()

    lbl_fileist = Label(Home, text="Please enter file name :", font=
('times new roman', 12)).pack()
    filename = Entry(Home, textvariable=main.FILENAME, font=(10))
    filename.pack(pady=15)
    btn_open = Button(Home, text='open File', width=45,
command=open1).pack(pady=15)


def Window2():
    global Home
    main.root.withdraw()
    Home = Toplevel()
    # Determine size of the window
    Home.title("Simple Fileshare Application")
    width = 400
    height = 300
    screen_width =  main.root.winfo_screenwidth()
    screen_height =  main.root.winfo_screenheight()
    x = (screen_width/2) - (width/2)
    y = (screen_height/2) - (height/2)
    main.root.resizable(0, 0)
    Home.geometry("%dx%d+%d+%d" % (width, height, x, y))
    # Create labels in the window
    lbl_fileist = Label(Home, text="The file is opened! the plaintext
is", font=('times new roman', 20)).pack()
    labelframe1 = LabelFrame(Home, text=main.FILENAME.get())
    labelframe1.pack(fill="both", expand="yes")
    #import
    UserName = main.USERNAME.get()
    FileName = main.FILENAME.get()
    plaintext = RecevierDec.decrypt_file_and_key(UserName,FileName)
    # Create button in the window
    bottomlabel = Label(labelframe1,text = plaintext)
    bottomlabel.pack()
    # define event
def open1():
    Window2()
```

1.4 Challenges

1- Programming the GUI using Python language. It was the first time we use Tkinter framework to build the GUI.
2- Choose the mode of operation in the AES algorithm. We try different modes of operation that has a different parameter.
3- Dealing with ‘txt’ and ‘csv’ files in Python. Either write or read.
4- Merge the Encryption and Decryption code with the GUI code.
5- Link the GUI windows together.
6- Generalized the generate and load keys method to be used for any new user entering the system. And create each folder with two files.
7- Send the IV along with the ciphertext to the receiver, so it needs it to decrypt the ciphertext.
8- Show the received files on the receiver side.

2. References

[1] img A. “Using AES for Encryption and Decryption in Python Pycrypto,” *Novixys Software Dev Blog*, Feb. 08, 2018. https://www.novixys.com/blog/using-aes-encryption-decryption-python-pycrypto/

[2] J. LaBelle, “Code snippets,” *jonlabelle.com*, 2019. https://jonlabelle.com/snippets/view/python/aes-file-encryption-in-python

[3] D. Masika, “Implementing RSA Encryption and Decryption in Python,” *Engineering Education (EngEd) Program | Section*, Jan. 28, 2022. https://www.section.io/engineering-education/rsa-encryption-and-decryption-in-python/

[4] razormist, “Simple Login Application in Python Tutorial with Source Code | Free Source Code,” *www.sourcecodester.com*, Jan. 04, 2021. https://www.sourcecodester.com/tutorials/python/11351/python-simple-login-application.html (accessed Apr. 22, 2022).

[5] O. I. Paul, “Python-authentication-system,” *GitHub*, Apr. 19, 2022. https://github.com/omisolaidowu/Python-authentication-system/blob/master/auth.py (accessed Apr. 22, 2022).

[6] Bansal, Rishabh. “Python GUI - Tkinter - GeeksforGeeks.” GeeksforGeeks, 17 June 2017, https://www.geeksforgeeks.org/python-gui-tkinter/.

[7] codemy. “Read And Write To Text Files - Python Tkinter GUI Tutorial #100.” *You Tube*, YouTube, 27 July 2020, https://www.youtube.com/watch?v=Z_0ISFfT_eM.