## SUBJECT: MJ-CS-B10 July Batch

Problem Statement:

Implement a robust image encryption program, utilizing technologies such as Python with specialized cryptographic libraries for AES, DES, and RSA algorithms. Ensure secure remote user connections, implement a login option, and establish a transfer system with advanced encryption methods.

The project is that I have developed has the following file structure:

image_encryption_project/

|-- aes.py # logic for AES encryption and decryption algorithm

|-- des.py # logic for DES encryption and decryption algorithm

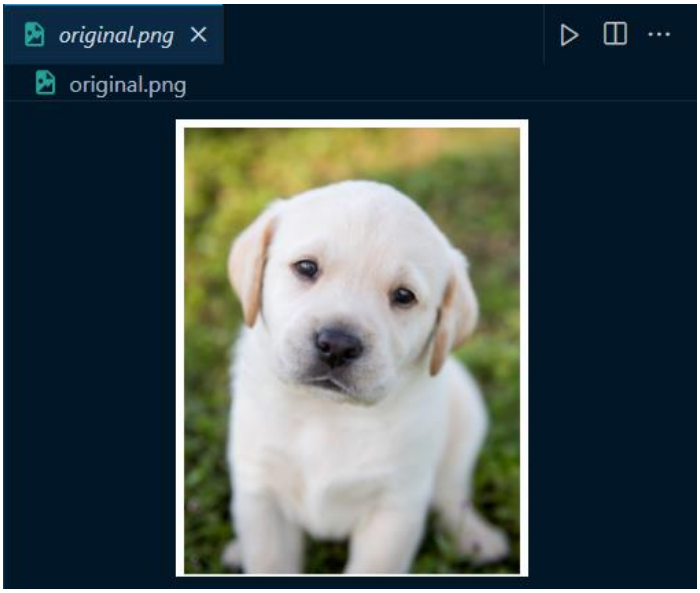
|-- rsa.py # logic for RSA encryption and decryption algorithm

|-- rsakey.py # logic for key generation for client and server (public/private key)

|-- client.py # logic for client-side for encrypting and receiving image

|-- server.py # logic for server-side for receiving and decrypting image

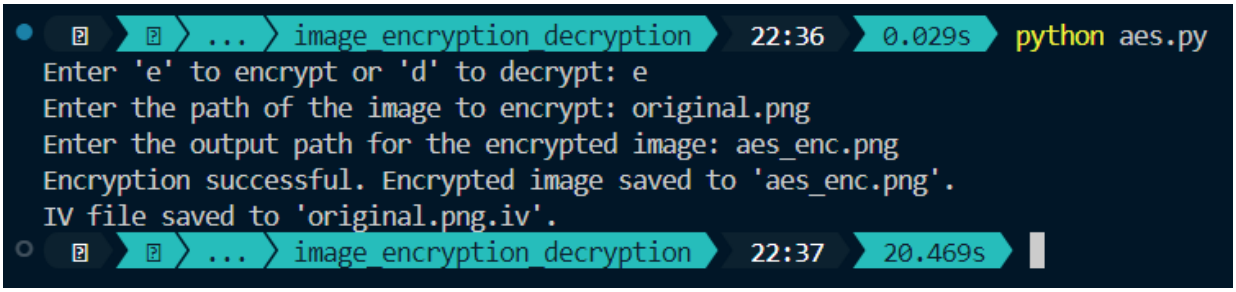
|-- original.(image extension) # a sample image for encryption and decryption testing



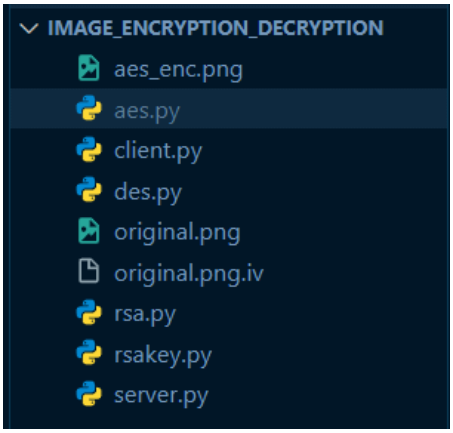
The following steps show the encryption and decryption process on a sample image:

Using AES (aes.py)

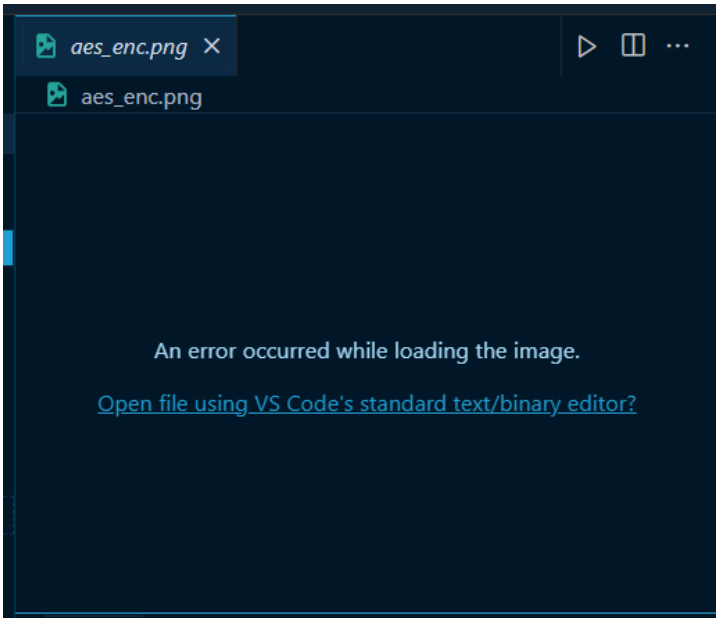
Encryption:




Files in folder:



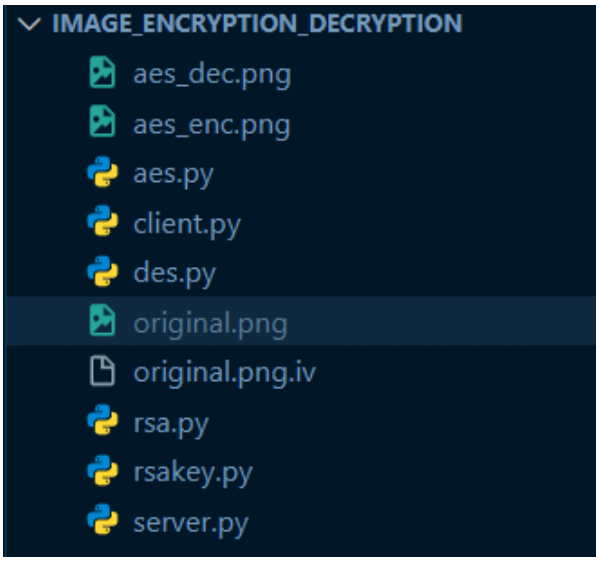

Encrypted image:



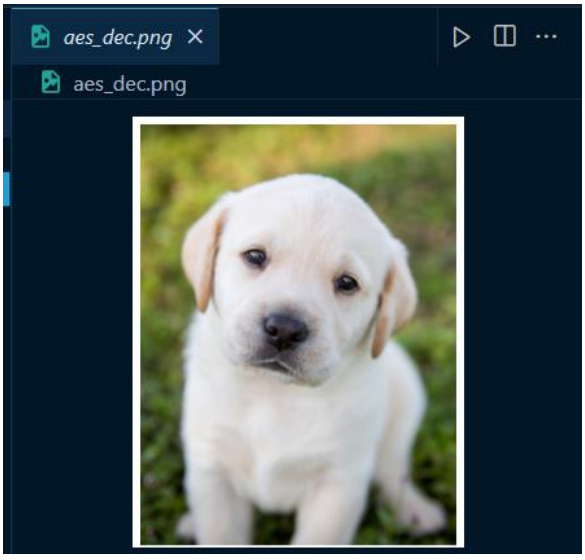
Decryption:

```
... image_encryption_decryption 22:37 20.469s python aes.py
Enter 'e' to encrypt or 'd' to decrypt: d
Enter the path of the encrypted image: aes_enc.png
Enter the output path for the decrypted image: aes_dec.png
Enter the path of the IV file: original.png.iv
Decryption successful. Decrypted image saved to 'aes_dec.png'.
... image_encryption_decryption 22:40 1:12.606s
```

Files in folder:

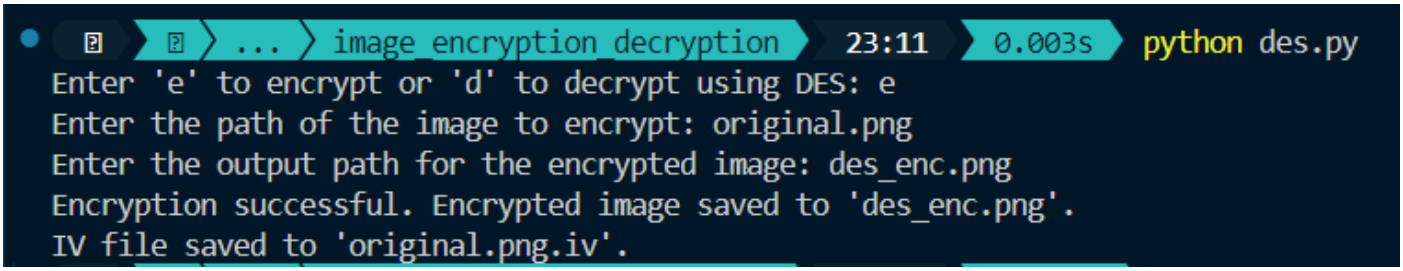



Decrypted image:

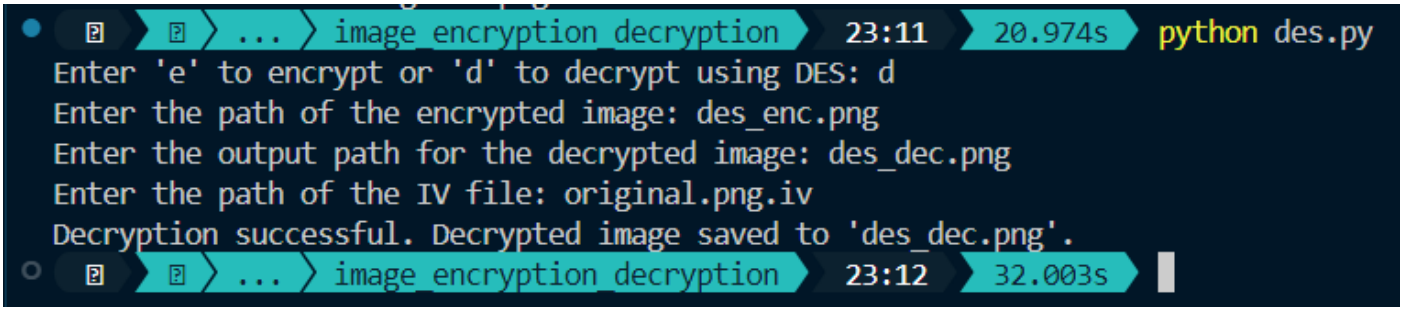


Using DES(des.py):

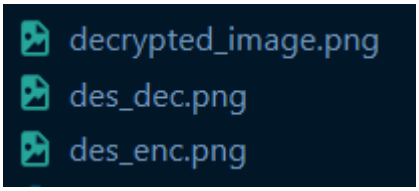
Encryption:

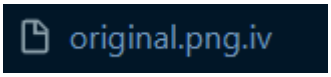



Decryption:



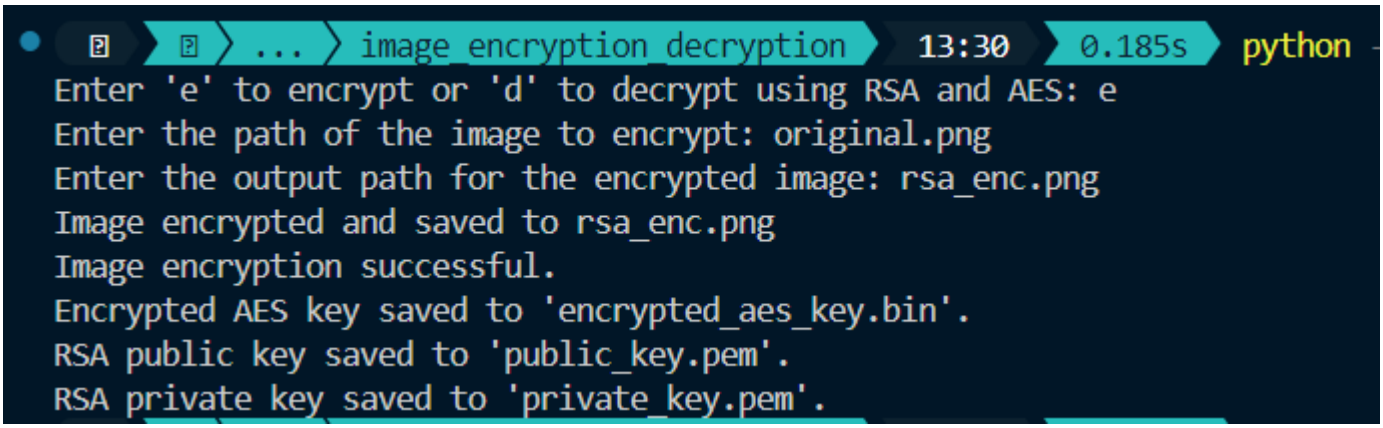

Files formed in the process:



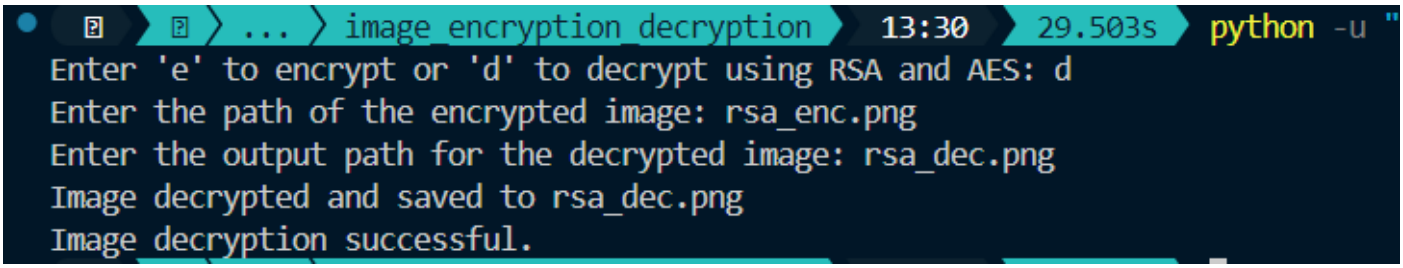



Using RSA (rsa.py):

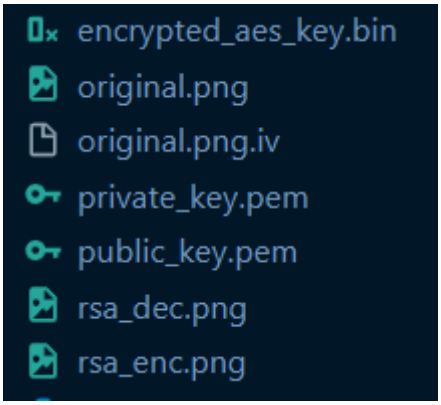
Encryption:




Decryption:



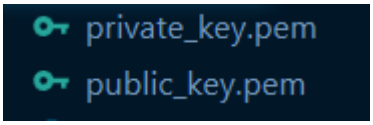

Files formed in the process:



## Client Server Transmission:

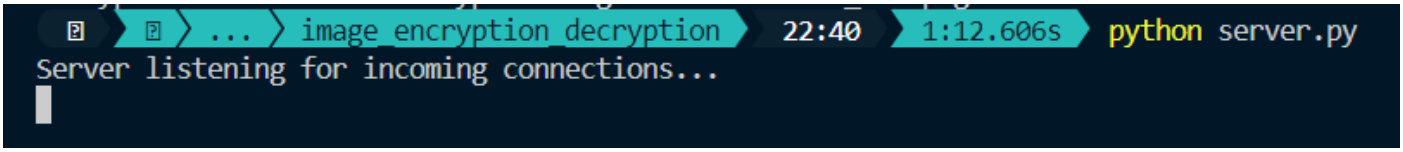
For image transformation and encryption, we run client and server on two different machines. We'll need the IP address for the server machine and both machines need to have their firewalls shut off. Else any socket connection will not be allowed.

We generate common RSA private and public keys using rsakey.py script. The keys are saved as .pem files:

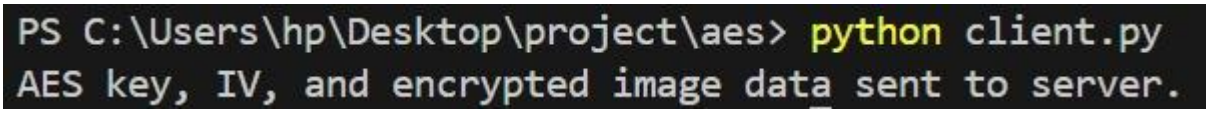



Only the public key is shared to  the server for decryption.

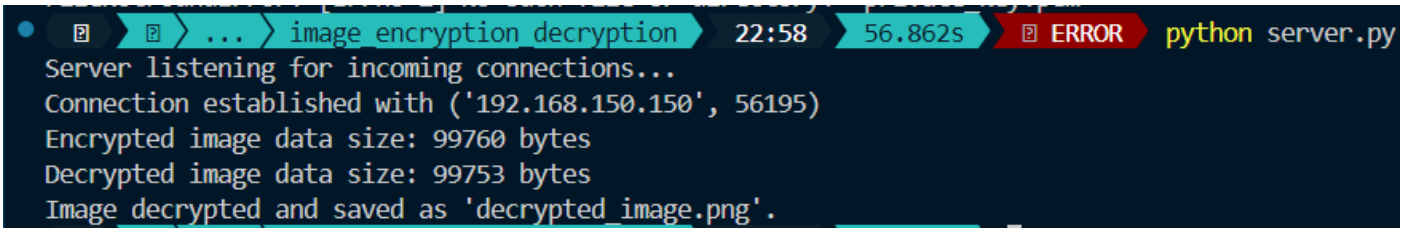
On running server.py on the Machine1:




Running client.py on Machine2:




On successful transfer, the server returns:




The image from client machine is successfully encrypted and transferred to the server, where it is decrypted and stored.

The application is a **client-server image encryption and decryption system** built using **Python**. It integrates **AES (Advanced Encryption Standard)** and **RSA (Rivest-Shamir-Adleman)** encryption algorithms to provide secure image transmission over a network. Here's a detailed breakdown:

**Key Features:**

1. **Hybrid Encryption Approach**:
    - **AES** is used for encrypting and decrypting the image. AES provides fast and efficient encryption for large data like images.
    - **RSA** is used to encrypt the AES key itself, ensuring the key exchange between the client and server remains secure. RSA uses public and private key pairs for encryption and decryption of the AES key.

2. **Client-Side Operations**:
    - The client reads an image and converts it into a byte array using **PIL** (Python Imaging Library).
    - The image is encrypted using **AES** in **CBC (Cipher Block Chaining)** mode with padding. A random **IV (Initialization Vector)** is generated for AES encryption.
    - The AES encryption key is then encrypted using RSA with the server's **public RSA key**. This ensures that only the server, with its private key, can decrypt the AES key.
    - The client transmits the following to the server over a **socket** connection:
        1. **RSA-encrypted AES key**
        2. **AES IV**
        3. **AES-encrypted image data**

3. **Server-Side Operations**:
    - The server listens for incoming connections and receives the data from the client: the **AES key**, **IV**, and **encrypted image**.
    - The **AES key** is decrypted using the server's **private RSA key**.
    - Using the decrypted AES key and IV, the server decrypts the image using AES and restores the original image.
    - The decrypted image is then saved to a specified location.

4. **File Structure**:
    - **AES & DES Encryption**: Separate modules (aes.py and des.py) are provided for encrypting and decrypting images using AES and DES. Each module includes padding, key-specific IV generation, and file management for storing encrypted images and IVs.
    - **RSA Key Generation and Key Exchange**: The rsa.py module handles RSA key generation and the encryption and decryption of AES keys.
    - **Client and Server**: The client.py handles the encryption of images and sending of encrypted data, while the server.py handles decryption and saving of the image.

5. **Security Considerations**:
    - **HMAC** (Hash-based Message Authentication Code) is used in AES and DES encryption for key-specific IV generation to ensure the integrity and uniqueness of encryption keys for each image.
    - **RSA with PKCS1_OAEP padding** provides a secure way to encrypt the AES key, protecting it during transmission.
    - **PIL** is used to handle images, making it versatile for encrypting and decrypting various image formats.

**Workflow:**

- **Client**: Encrypts an image -> Encrypts the AES key -> Sends data to the server.
- **Server**: Decrypts the AES key -> Decrypts the image -> Saves the decrypted image.

This design ensures **confidentiality, integrity, and security** of image data during transmission in a networked environment.

The Python scripts, all used and generated files are provided in the following GitHub repo:

QuantumRitik/image_encryption_decryption (github.com)