

## **ABSTRACT**

The rapid increase of internet-based data transfers has resulted in escalating concerns regarding data security, particularly for sensitive data types such as images. To address this, we propose a novel approach to Image Encryption for Secure Internet Transfer, leveraging both symmetric and asymmetric cryptographic techniques for enhancing the security of image transfers. By utilizing Advanced Encryption Standard (AES) for encrypting the image data and Rivest-Shamir-Adleman (RSA) for securing the AES key, this approach ensures robust security measures. The implementation is presented via a user-friendly GUI, providing straightforward interaction with the encryption and decryption processes. The findings demonstrate that this dual-encryption approach significantly enhances the security of image transfers, deterring unauthorized decryption even in interception scenarios.

## **INTRODUCTION**

As we continue to initiate in the digital age, data transfers over the internet have become a fundamental part of our lives, encompassing various formats such as texts, audio, video, and images. This widespread reliance on internet-based transfers, while facilitating convenience, also opens the gateway for potential unauthorized access and manipulation of data. In particular, image data, which often carries sensitive information, needs to be protected from potential threats. This concern has inspired the project "Image Encryption for Secure Internet Transfer," aimed at providing robust security for images during their transit on the internet.

The fundamental objective of this project is to design and implement an advanced encryption mechanism for securing image data. By using cryptographic techniques, we can encrypt the image data before transmission, thereby rendering it useless to any potential interceptor. Only with the correct decryption key can the original image be retrieved, ensuring that the data remains secure during its journey on the internet.

The project employs a combination of symmetric and asymmetric encryption, namely the Advanced Encryption Standard (AES) and Rivest-Shamir-Adleman (RSA),

respectively. Symmetric encryption, represented here by AES, is a method of encryption where the same key is used for both encryption and decryption. Despite its robustness and efficiency, a major challenge with symmetric encryption is the safe transfer of the key itself; if it were to be intercepted, the security of the data would be compromised. To circumvent this issue, we incorporate an asymmetric encryption layer using RSA. In RSA, two different but mathematically linked keys are used – a public key for encryption and a private key for decryption. We use RSA to encrypt the AES key itself, providing an additional layer of security.

To make the process of encryption and decryption more accessible, the project is implemented as a Python-based application with a graphical user interface (GUI) using the PyQt5 library. This interface allows users to easily select images, generate keys, and encrypt or decrypt the image data, thereby demonstrating the viability and effectiveness of the proposed encryption mechanism.

The following sections provide a comprehensive overview of the tools and techniques used in the project, followed by a detailed walk-through of the implementation process.

## **TOOLS AND TECHNIQUES**

The implementation of this project makes use of several tools and techniques, including programming languages, cryptographic libraries, and GUI development libraries.

### **Programming Language: Python**

Python was chosen as the programming language for this project due to its ease of use, readability, and the robustness of its libraries, particularly for cryptographic and GUI development.

### **Cryptography Libraries: PyCryptoDome and Cryptography**

The PyCryptoDome library is a self-contained Python package for cryptographic operations which serves as a direct replacement for the now unmaintained PyCrypto library. It supports a range of cryptographic operations including AES encryption, which is a key part of this project.

Cryptography is another Python library used in this project, providing cryptographic recipes and primitives to Python developers. It includes a flexible and easy-to-use implementation of RSA encryption, which is used in this project to encrypt the AES key.

### **GUI Development: PyQt5**

PyQt5 is a set of Python bindings for Qt libraries which can be used to create cross-platform applications with attractive graphical user interfaces. It provides a wide range of functionalities including file dialogs, message boxes, and complex layouts which were all utilized in this project.

### **Techniques: Symmetric and Asymmetric Encryption**

Symmetric encryption was used for the image encryption, through the AES encryption algorithm. This method was chosen for its speed and efficiency when dealing with larger data sets, such as images.

Asymmetric encryption was used for encrypting the AES key, through the RSA encryption algorithm. This technique was employed to safely transport the AES key to the receiver without the risk of it being intercepted and used to decrypt the image.

## **IMPLEMENTATION**

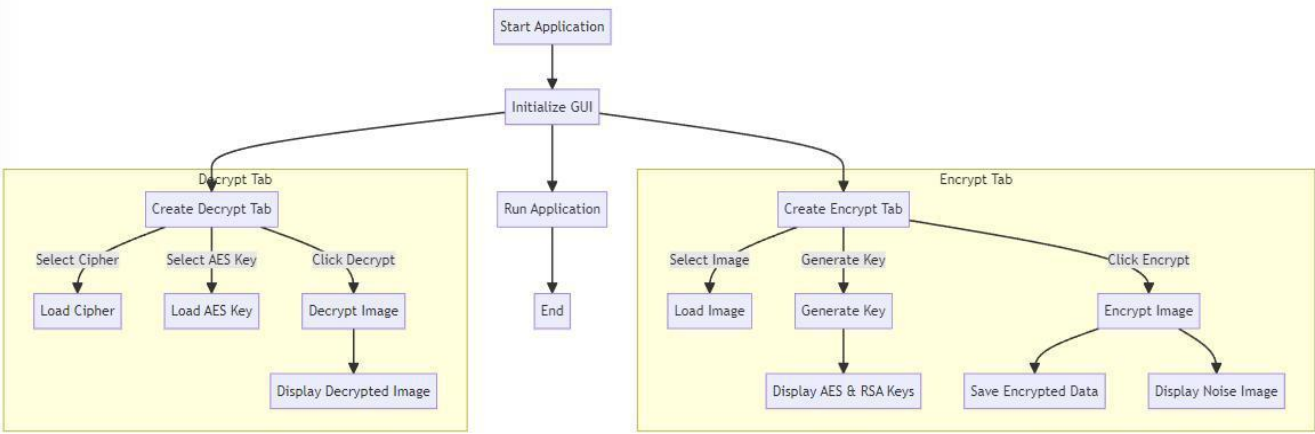
The image encryption and decryption process can be split into three main parts - the GUI implementation, the key generation process, and the encryption/decryption process.

### **GUI Implementation**

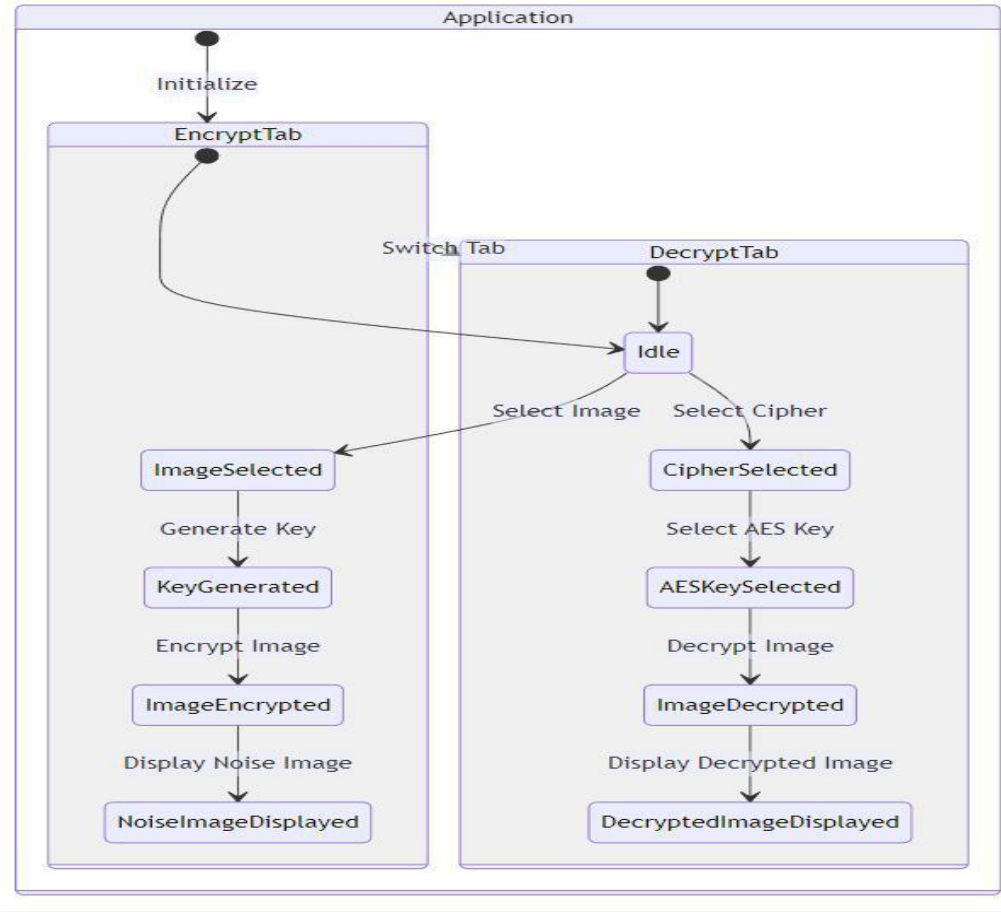
The user interface was implemented using PyQt5. The application window contains two main tabs, "Encrypt" and "Decrypt". Each of these tabs contains the respective functionalities for image encryption and decryption.

For the "Encrypt" tab, it includes a QLabel to display the image, QLineEdit widgets to display the AES key and RSA public key, QPushButton widgets for loading an image, generating the keys, and encrypting the image. For the "Decrypt" tab, it includes a QLabel to display the encrypted or decrypted image, QLineEdit widgets to enter the RSA private key, QPushButton widgets to load the cipher file, the encrypted AES key file, and decrypt the image.

FLOWCHART DIAGRAM



STATE DIAGRAM



## Encrpyt Tab:

Encrypt

Decrypt

AES Key:  RSA Public Key: 

Key Generate

Encrypt

## Decrypt Tab:

Encrypt

Decrypt

RSA Private Key: 

Select Cipher File

Select AES Key File

Decrypt

## Key Generation

A key generation function was implemented to generate a random 16-byte AES key using the `get_random_bytes` function from the `Crypto.Random` library. The AES key is then encoded into a base64 string to make it easier to handle.

An RSA key pair was also generated using the `rsa.generate_private_key` function from the `cryptography.hazmat.primitives.asymmetric` library. The private key is saved into the `rsa_password_input` QLineEdit, and the public key is saved into the `rsa_public_key_input` QLineEdit.

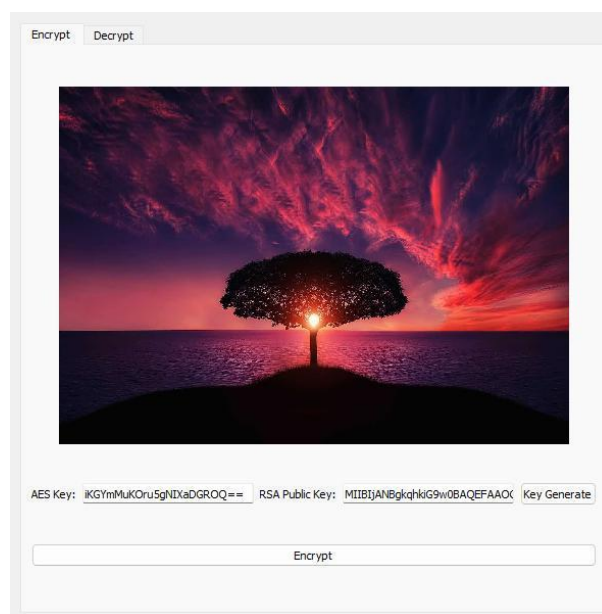
## Encryption Process

The image encryption function starts by checking if an image has been selected and if keys have been generated. If not, an error message is displayed.

The AES key is encrypted with the RSA public key using the RSA encryption functions provided by the cryptography library. This encrypted key is then saved to a text file.

The selected image is saved into a `QByteArray` object, which is then encrypted using the AES encryption functions from the `PyCryptoDome` library. The encrypted image data is then saved into a text file, and a noise image is displayed in the `QLabel`.

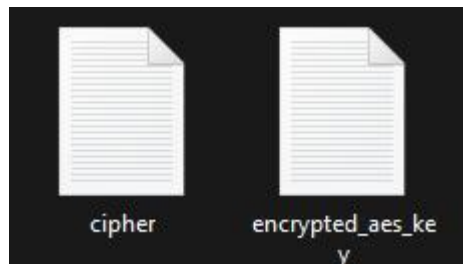
## Before Encryption:



## After Encryption:



## Creates two text file for keys



## Decryption Process

The decryption process involves several steps. Firstly, the user is asked to select the cipher file and the file containing the encrypted AES key.

The private key is retrieved from the QLineEdit and the AES key is decrypted. An error message is displayed if the AES key could not be decrypted with the provided private key.

Then the image cipher is decrypted using the decrypted AES key, and the decrypted image data is used to generate a QImage object, which is then displayed in the QLabel.

## Before Decryption:

Encrypt

Decrypt




RSA Private Key:

## After Decryption:

Encrypt

Decrypt



RSA Private Key:



## **RESULTS**

The project successfully demonstrated the use of AES and RSA encryption for secure image transfer. The implementation of the encryption and decryption processes was straightforward, with clear error messages for potential issues such as missing keys or files.

The use of a graphical user interface greatly enhanced the usability of the application, allowing for easy selection of images and keys, and clear visual feedback of the encryption and decryption processes.