TUNISIAN REPUBLIC
Ministry of Higher Education and Scientific Research
University of Carthage
National Institute of Applied Science and Technology

Personal Professional Project

Secure Image Transfer System

Prepared by
Idris SADDI

Software Engineering
Department of Computer Science and Mathematics Engineering
INSAT, University of Carthage, Tunisia

Academic Year: 2023-2024

Table of Content

Table of Figures

Abstract

In today's digital age, the use of devices such as computers and mobile phones for communication, data storage, and transmission has significantly increased. Consequently, the number of users has surged, leading to increased unauthorized attempts to access data. This poses a critical challenge to data security. Images, often containing sensitive or confidential information, are frequently transmitted over insecure channels, necessitating robust security measures to protect them from unauthorized access.

To address this issue, the Advanced Encryption Standard (AES) algorithm is employed for encrypting and decrypting images. The encrypted data becomes unreadable to unauthorized users, ensuring secure network transmission. At the receiving end, the data can be decrypted using the same AES algorithm, thereby guaranteeing the integrity and confidentiality of the transmitted images.

The implementation code for this project can be found on the GitHub repository: idris-saddi/Secure-Image-Transfer-System-AES

Aim and Objectives

Aim of the project

This project aims to develop a secure method for transferring images between a sender and a receiver. Images must be encrypted before transmission over the network and correctly decrypted upon reception.

Objectives of the project

- Encrypt images into an unreadable format using AES.
- Decrypt encrypted images back to their original format.
- Ensure secure transfer of images over networks like the internet.
- Prevent unauthorized modifications during transmission.

Scope of the project

Product Scope

The project involves encrypting images using the AES algorithm to facilitate secure transmission over the network. The receiver can decrypt the image using the provided decryption code to retrieve the original image. This method is beneficial for transmitting sensitive information in medicine and the military.

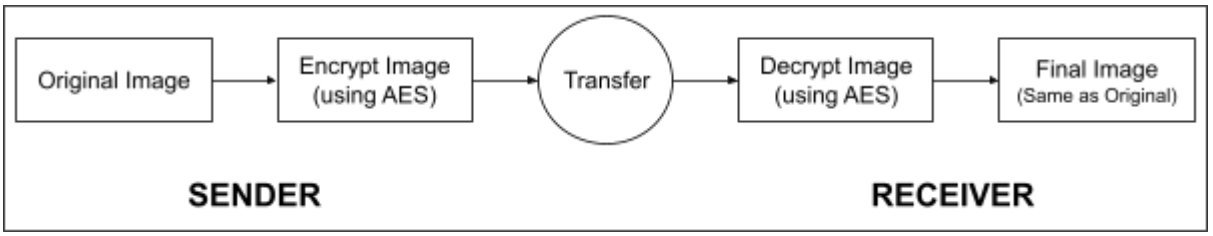

Fig01. Project Scope

Design and Implementation Constraints

- Encryption and decryption must be performed using the AES algorithm.
- The original image must be in .jpeg or .png format.

Assumptions and Dependencies

- The sender and receiver are connected to a network.

Functional Requirements

Main Features

- Image Encryption: The system shall encrypt the image to an unreadable format using the AES encryption function.
- Image Decryption: The system shall decrypt the received encrypted image to its original, readable format using the AES decryption function, ensuring the output matches the original image.
- Secure NetworkTransmission: The system ensures safe image transmission, preventing unauthorized third-party modifications.

Additional Features

- Image Compression: The system can include a feature to compress images before encryption to reduce file size and speed up the transfer.
- Transfer Notification: The system can send notifications to users to inform them of the success or failure of an image transfer.
- Error Handling: The system should be capable of managing errors that may occur during transfer, notifying users, and taking appropriate measures to resolve the issues.

Non-Functional Requirements

Performance Requirements

- For efficient encryption, the image size must be less than 5MB.
- Decryption should take at most 10 seconds.

Safety and Security Requirements

- If decryption exceeds 10 seconds, the message should be discarded to prevent processing potentially corrupted data, and the sender should be asked to resend the message.
- Encryption is done using an encryption key. Decryption will happen only when the same encryption key is used on the receiver.

Software Quality Attributes

Reliability

External factors do not affect the system. The AES algorithm is universally accepted and produces consistent results, minimizing error likelihood. Errors are rare and usually due to transmission glitches, so the system is reliable.

Usability

The system uses a Python-based GUI, providing a user-friendly interface with easy navigation buttons. Users with basic computer knowledge can encrypt/decrypt images using a key.

Testability

The system's modular design facilitates easy testing and defect identification. Each module performs specific functions that can be tested individually.

External Interface Requirements

User Interfaces

The interface window allows users to input the image location and provides buttons for encoding and decoding. After selecting an option, a subsequent window prompts for a password and displays the filename of the generated image file upon submission.

Hardware Interfaces

- Sender Computer: Displays the original and encrypted image.
- Receiver Computer: Displays the decrypted image.

Software Interfaces

The desktop app, built using the Tkinter module in Python, offers an interactive window where users input the image location. Users can encode or decode the image by providing an encryption key (password). The Python functions in the code handle the AES encryption/decryption, and the user receives the processed image as output.

Related Work / Basic Concepts

Related Work

Several studies have explored the use of encryption techniques to secure image transmission. The Advanced Encryption Standard (AES) is frequently highlighted for its robustness and speed, making it ideal for encrypting large image files. Research has emphasized the importance of secure image transmission in medical imaging and military communication, where data integrity and confidentiality are paramount. Integrating image compression with encryption has also been examined to optimize transmission efficiency, reducing file size without compromising security. Additionally, secure transmission protocols with robust error-handling mechanisms are essential for detecting and addressing any issues during data transfer, ensuring reliable and secure communication.

Basic Concepts

Advanced Encryption Standard (AES)

AES is a symmetric encryption algorithm widely used for securing data. It encrypts data in fixed-size blocks utilizing a series of transformations based on substitution, permutation, and key expansion. AES is known for its high security and efficiency, making it suitable for encrypting large files like images.

Image Encryption and Decryption:

Image encryption involves converting the image data into an unreadable format using an encryption key. The decryption process reverses this transformation, restoring the image to its original state using the same key. This ensures that only authorized users with the correct key can access the original image.

Secure Transmission

Secure transmission refers to the practice of sending data over a network in a manner that prevents unauthorized access and tampering. This often involves using encryption to protect the data during transfer and ensuring that both the sender and receiver are authenticated.

Image Compression

Image compression reduces the size of image files by removing redundant or unnecessary data. This process can be lossless (preserving the original image quality) or lossy (reducing quality for greater size reduction). Compression is often used to optimize storage and transmission efficiency.

Error Handling

Error handling in data transmission involves detecting and managing errors that occur during the transfer process. Effective error handling ensures data integrity and reliability, notifying users of issues and attempting to recover or retransmit corrupted data as needed.

FastAPI Framework

FastAPI is a modern web framework for building APIs with Python. It is known for its high performance, ease of use, and support for asynchronous programming. FastAPI is particularly suitable for developing APIs that require fast response times and efficient handling of concurrent requests.

Uvicorn

Uvicorn is an ASGI web server implementation for Python. Until recently Python has lacked a minimal low-level server/application interface for async frameworks. The ASGI specification fills this gap, and means we're now able to start building a common set of tooling usable across all async frameworks. Uvicorn currently supports HTTP/1.1 and WebSockets.

Tkinter

Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit and is Python's de facto standard GUI. Tkinter is included with standard Linux, Microsoft Windows, and macOS installs of Python. License. Python license. The name Tkinter comes from the Tk interface.

Pillow

The Python Imaging Library adds image processing capabilities to your Python interpreter. This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities. The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.

Project Plan and Timeline

Definition of Project Objectives and Scope: (1 week)

Objective:
Clearly define the project objectives and scope to guide the subsequent phases.

Main Activities:

- Meeting with the client to discuss objectives and requirements.
- Analyzing needs and constraints.
- Documenting project objectives and scope.

Deliverable:
Document describing project objectives, functional scope, constraints, and initial requirements.

Needs Analysis (2 weeks)

Objective:
Exhaustively identify the functional and non-functional requirements of the system.

Main Activities:

- Identification of functional and non-functional requirements.
- Study of technical constraints and performance and security requirements.
- Documentation of analyzed needs.

Deliverable:
Detailed document of functional and non-functional requirements, including constraints and requirements.

System Design (2 weeks)

Objective:
Design a robust software architecture and a user-friendly interface.

Main Activities:

- Design of the software architecture with different modules.
- Development of flow diagrams to visualize the system's operation.
- Documentation of the design.

Deliverable:
Software architecture document, flow diagrams, and user interface specifications.

Implementation (3 weeks)

Objective:
Implement the functionalities according to the established design.

Main Activities:

- Development of the user interface with Tkinter.
- Implementation of AES functions for encryption/decryption.
- Integration of modules and exception handling.

Deliverable:
Functional application with user interface, implemented and integrated AES functions.

Testing (2 weeks)

Objective:
Validate the proper functioning of each feature and interaction between modules.

Main Activities:

- Unit tests for each feature (encryption, decryption, error handling, user interface).
- Integration tests to validate the complete system.
- Integration of client feedback for adjustments.

Deliverable:
Test reports, corrections, and adjustments based on client feedback.

Validation (1 week)

Objective:
Validate that the system meets the requirements and functions correctly in real scenarios.

Main Activities:

- Real transfer tests to verify encryption/decryption and error handling.
- Validation of performance, security, and usability.
- Final integration of client feedback.

Deliverable:
Validated system ready for deployment.

Documentation (1 week)

Objective:
Document the code, architecture, and processes to facilitate maintenance and use.

Main Activities:

- Adding comments in the code.
- Writing the user manual and technical documentation.
- Documentation of development processes.

Deliverable:
A commented code, user manual, and technical documentation.

Deployment (1 week)

Objective:
Prepare and deploy the system in the production environment.

Main Activities:

- Packaging files for deployment.
- Creating clear installation instructions.
- Testing on target environments.

Deliverable:
The application is deployed and functional in the production environment.

Maintenance and Support (Ongoing)

Objective:
Ensure continuous maintenance and support of the system after deployment.

Main Activities:

- Establishing a maintenance process.
- Client support to answer questions and resolve issues.

Deliverable:
Established maintenance process, and system under ongoing support and maintenance.

System Design

Use Case Diagram

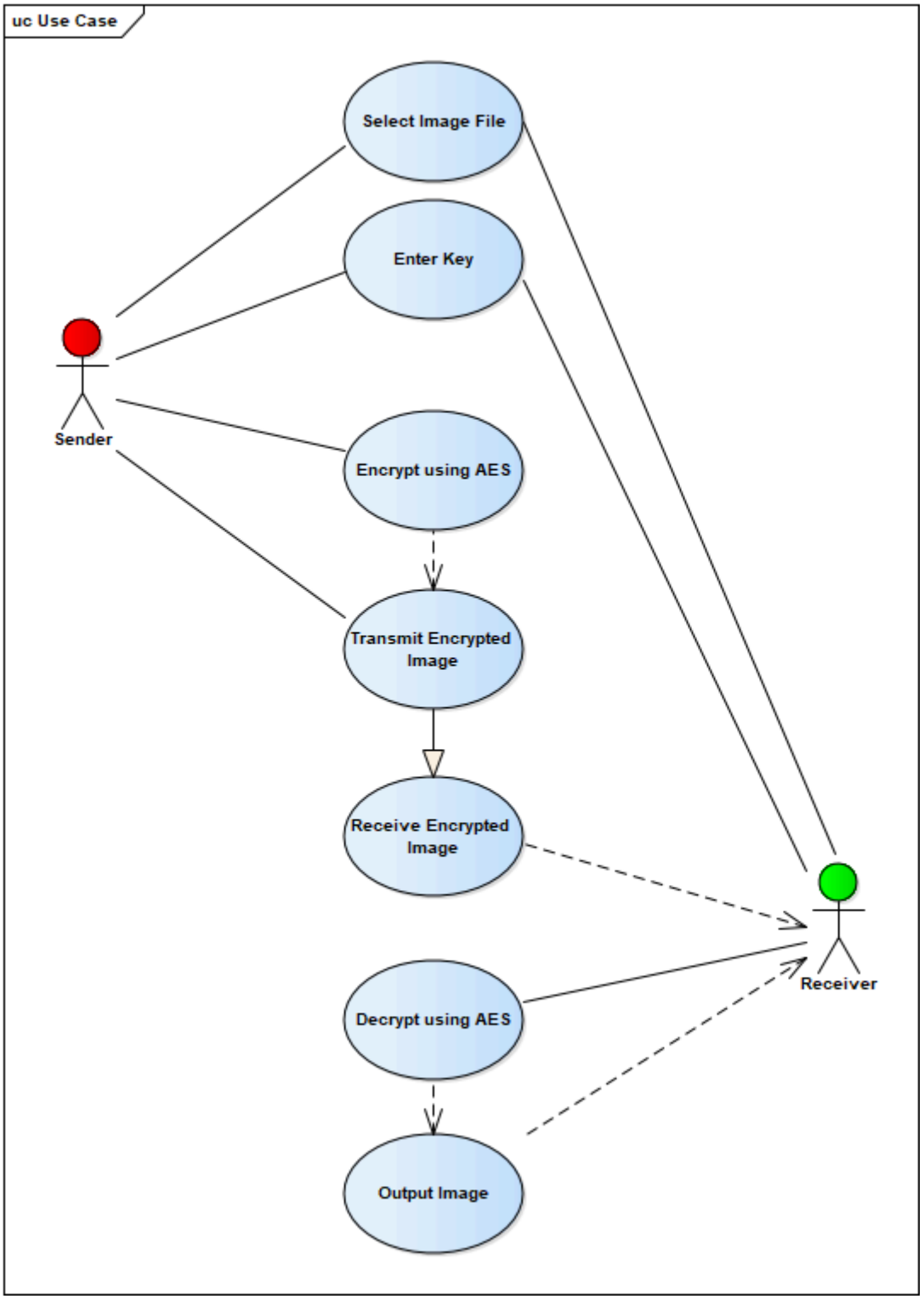

Fig02. Use Case Diagram

Class Diagram

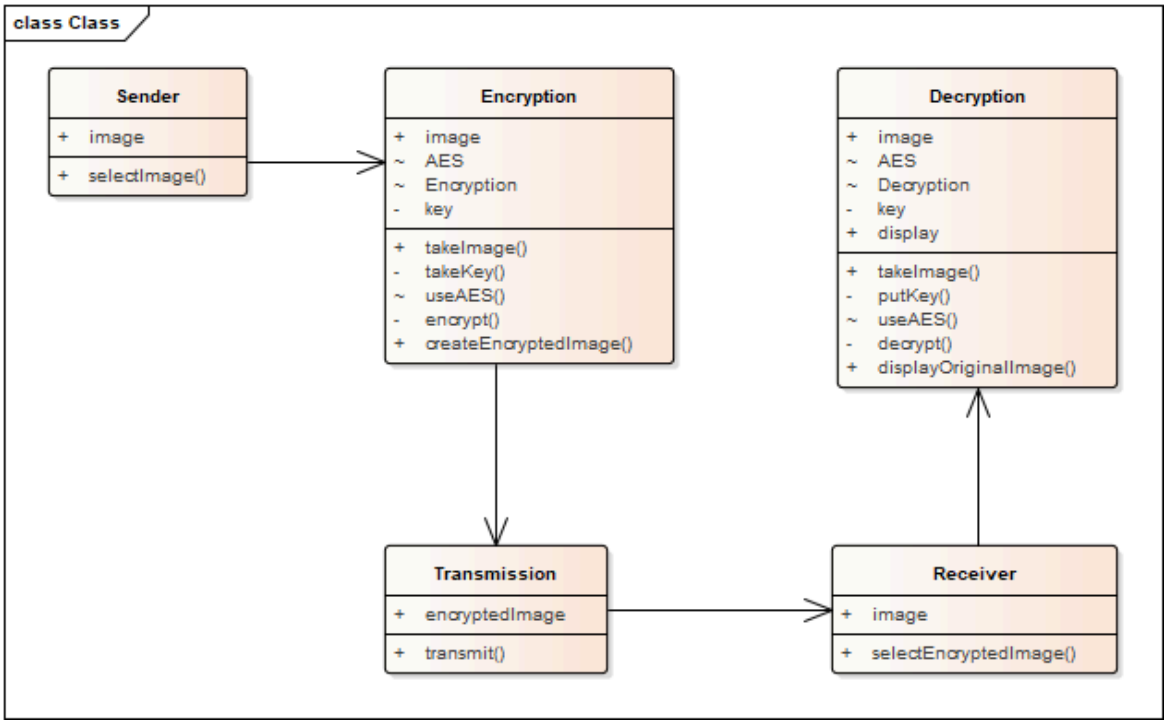

Fig03. Class Diagram

Sequence Diagram

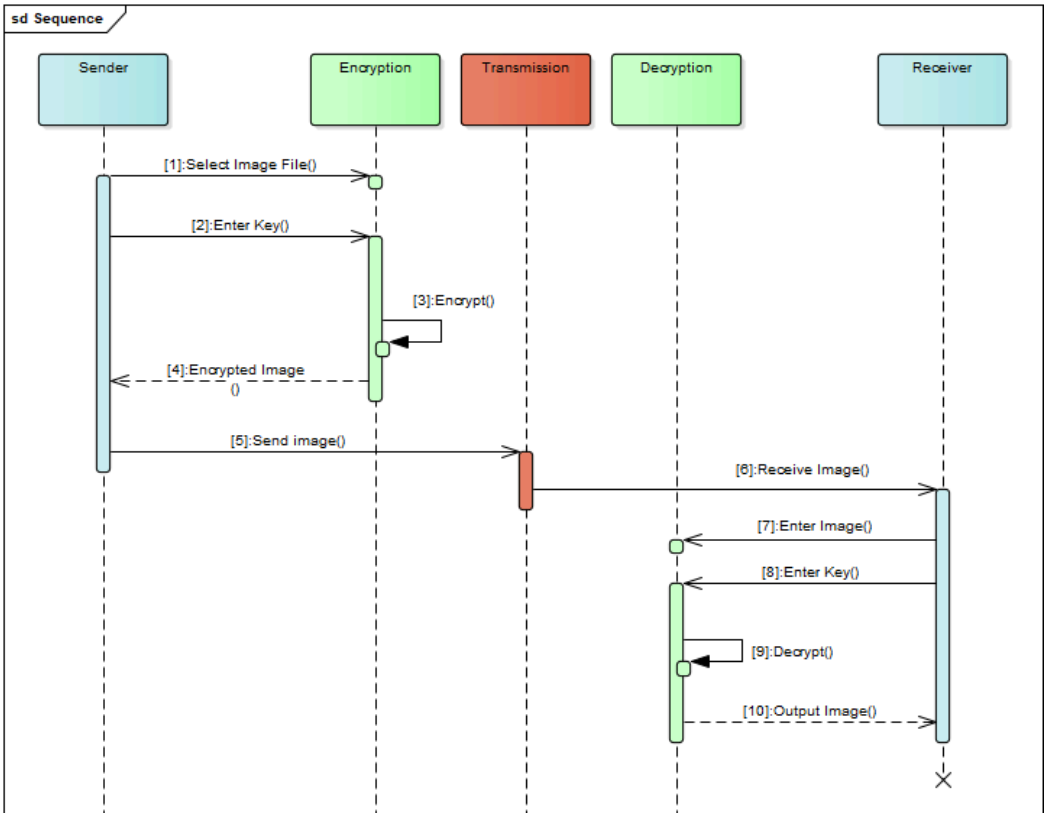

Fig04. Sequence Diagram

State Chart Diagram

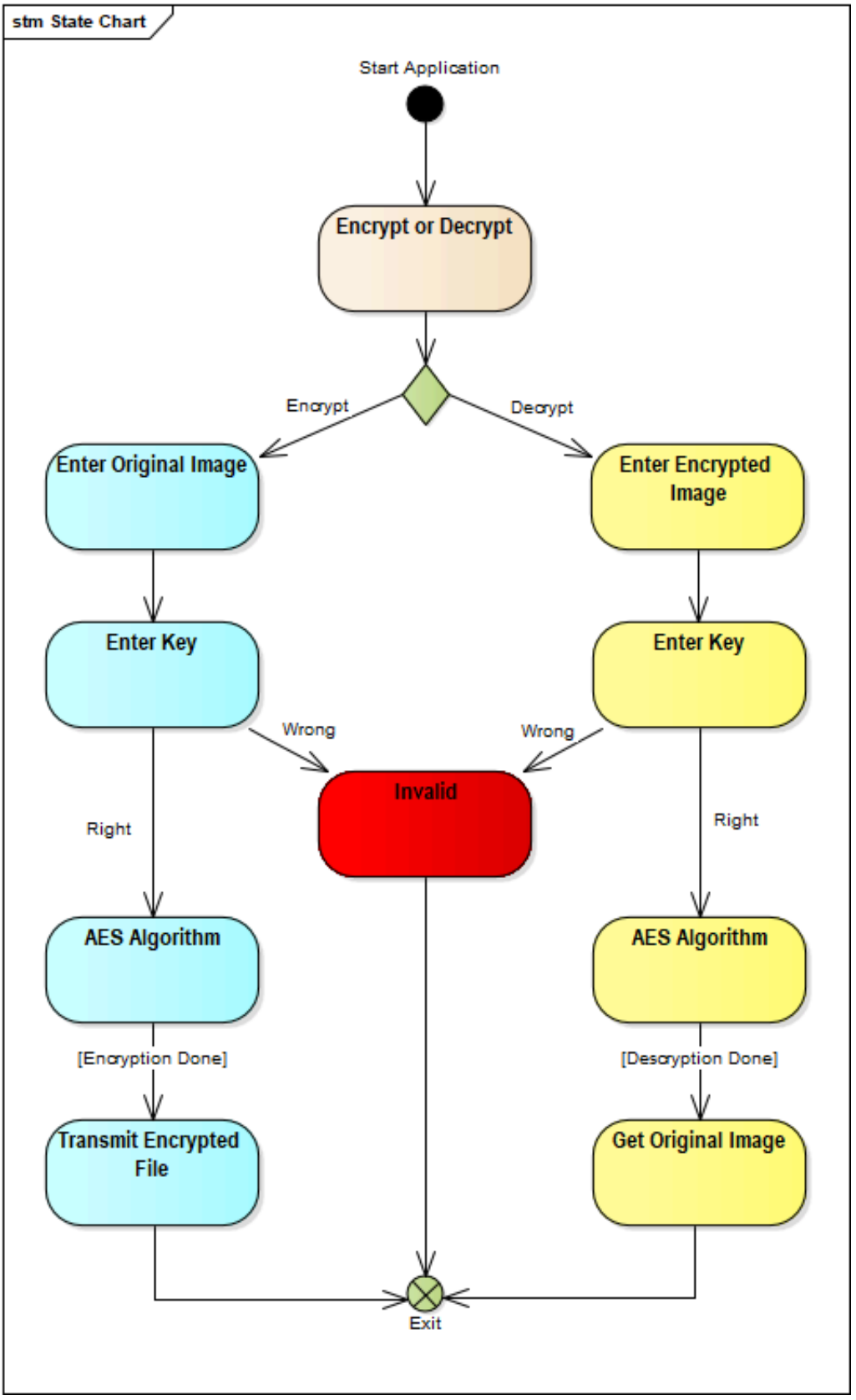

Fig05. State Chart Diagram

Implementation

Logic functionalities

Utility Functions

```
def load_image(name):
    return Image.open(name)
```

This function takes the name of an image file as input and returns the loaded image using the Python Imaging Library (PIL). This function is a utility to simplify the process of loading images from a file.

Functions for encryption

```
def prepare_message_image(image, size):
    if size != image.size:
        image = image.resize(size, Image.ANTIALIAS)
    return image
```

This function ensures that the input image is resized to match the specified size. If the image's current size is different from the specified size, it resizes the image using antialiasing to maintain quality. This is crucial for ensuring that the message image and secret image match in size during encryption.

```
def generate_secret(size, secret_image = None):
    width, height = size
    new_secret_image = Image.new(mode = "RGB", size = (width * 2, height * 2))

    for x in range(0, 2 * width, 2):
        for y in range(0, 2 * height, 2):
            color1 = np.random.randint(255)
            color2 = np.random.randint(255)
            color3 = np.random.randint(255)
            new_secret_image.putpixel((x,  y), (color1,color2,color3))
            new_secret_image.putpixel((x+1,y), (255-color1,255-color2,255-color3))
            new_secret_image.putpixel((x,  y+1), (255-color1,255-color2,255-color3))
            new_secret_image.putpixel((x+1,y+1), (color1,color2,color3))

    return new_secret_image
```

This function generates a secret image of the given size. It creates a new image with twice the width and height of the specified size. For each 2x2 block of pixels, it generates random colors and their complementary colors. This secret image is used in the visual encryption process to hide the message image.

```
def generate_ciphered_image(secret_image, prepared_image):
    width, height = prepared_image.size
    ciphered_image = Image.new(mode = "RGB", size = (width * 2, height * 2))
    for x in range(0, width*2, 2):
        for y in range(0, height*2, 2):
            sec = secret_image.getpixel((x,y))
            msssg = prepared_image.getpixel((int(x/2),int(y/2)))
            color1 = (msssg[0]+sec[0])%256
            color2 = (msssg[1]+sec[1])%256
            color3 = (msssg[2]+sec[2])%256
            ciphered_image.putpixel((x,  y),   (color1,color2,color3))
            ciphered_image.putpixel((x+1,y),   (255-color1,255-color2,255-color3))
            ciphered_image.putpixel((x,  y+1), (255-color1,255-color2,255-color3))
            ciphered_image.putpixel((x+1,y+1), (color1,color2,color3))

    return ciphered_image
```

This function generates a ciphered image by combining the secret image and the prepared message image. For each pixel in the prepared image, it adds the corresponding pixel values from the secret image and the prepared image, ensuring the resulting values are within the valid color range. The result is saved as a new image with twice the width and height of the original, containing both the ciphered message and its complementary colors.

```
def generate_image_back(secret_image, ciphered_image):
    width, height = secret_image.size
    new_image = Image.new(mode = "RGB", size = (int(width / 2), int(height / 2)))
    for x in range(0, width, 2):
        for y in range(0, height, 2):
            sec = secret_image.getpixel((x,y))
            cip = ciphered_image.getpixel((x,y))
            color1 = (cip[0]-sec[0])%256
            color2 = (cip[1]-sec[1])%256
            color3 = (cip[2]-sec[2])%256
            new_image.putpixel((int(x/2),  int(y/2)),   (color1,color2,color3))

    return new_image
```

This function reconstructs the original message image from the secret image and the ciphered image. It extracts the pixel values of the message image by subtracting the secret image's pixel values from the ciphered image's pixel values, ensuring the resulting values are within the valid color range. The result is saved as a new image with half the width and height of the secret and ciphered images.

```
def level_one_encrypt(Imagename):
    message_image = load_image(Imagename)
    size = message_image.size
    secret_image = generate_secret(size)
    secret_image.save(path_join(files_folder_path, "secret.jpeg"))

    prepared_image = prepare_message_image(message_image, size)
    ciphered_image = generate_ciphered_image(secret_image, prepared_image)
    ciphered_image.save(path_join(files_folder_path, "2-share_encrypt.jpeg"))
```

This function performs the first level of encryption. It loads the specified image, generates a secret image, and then creates a ciphered image by combining the secret and message images. The secret and ciphered images are saved to disk for later use.

```
def construct_enc_image(ciphertext, relength, width, height):
    asciicipher = binascii.hexlify(ciphertext).decode()
    def replace_all(text, dic):
        res = text
        for i, j in dic.items():
            res = res.replace(i, j)
        return res

    # use replace function to replace ascii cipher characters with numbers
    reps = {letter: str(index + 1) for index, letter in numerate(string.scii_lowercase)}
    asciiciphertxt = replace_all(asciicipher, reps)

    # construct encrypted image
    step = 3
    encimageone=[asciiciphertxt[i:i+step] for i in range(0, len(asciiciphertxt), step)]
    # if the last pixel RGB value is less than 3-digits, add a digit a 1
    if int(encimageone[len(encimageone)-1]) < 100:
        encimageone[len(encimageone)-1] += "1"
    # check to see if we can divide the string into partitions of 3 digits.
    # if not, fill in with some garbage RGB values
    if len(encimageone) % 3 != 0:
        while (len(encimageone) % 3 != 0):
            encimageone.append("101")

    encimagetwo=[(
       int(encimageone[int(i)]),
       int(encimageone[int(i+1)]),
       int(encimageone[int(i+2)])
       ) for i in range(0, len(encimageone), step)]

    while (int(relength) != len(encimagetwo)):
        encimagetwo.pop()

    encim = Image.new("RGB", (int(width),int(height)))
    encim.putdata(encimagetwo)

    encim.save(path_join(files_folder_path, "visual_encrypt.jpeg"))
```

This function constructs an encrypted image from the ciphertext. It converts the ciphertext to a hexadecimal string, replaces certain characters with numbers, and then divides the string into chunks representing RGB values. These values are used to create a new image with the specified width and height, which is saved to disk.

Main Functions

```
def encrypt(imagename, password):
    plaintext = list()
    plaintextstr = ""

    im = Image.open(imagename)
    pix = im.load()

    width , height = im.size

    # break up the image into a list, each with pixel values and then append to a string
    for y in range(0,height):
        for x in range(0,width):
            plaintext.append(pix[x,y])

    # add 100 to each tuple value to make sure each are 3 digits long.
    for i in range(0,len(plaintext)):
        for j in range(0,3):
            aa = int(plaintext[i][j])+100
            plaintextstr = plaintextstr + str(aa)

    # length save for encrypted image reconstruction
    relength = len(plaintext)

    # append dimensions of image for reconstruction after decryption
    plaintextstr += "h" + str(height) + "h" + "w" + str(width) + "w"

    # make sure that plantextstr length is a multiple of 16 for AES. if not, append "n".
    while (len(plaintextstr) % 16 != 0):
        plaintextstr = plaintextstr + "n"

    # encrypt plaintext
    obj = AES.new(bytes(password), AES.MODE_CBC, b'This is an IV456')
    ciphertext = obj.encrypt(plaintextstr.encode())

    # write ciphertext to file for analysis
    cipher_name = path_join(cipher_folder_path, imagename.split("/")[-1] + ".crypt")
    g = open(cipher_name, 'wb')
    g.write(ciphertext)
    construct_enc_image(ciphertext,relength,width,height)
    print("Visual Encryption done.......")
    level_one_encrypt(path_join(files_folder_path,"visual_encrypt.jpeg"))
    print("2-Share Encryption done.......")
    return cipher_name
```

This function performs visual encryption on the specified image using the provided password. It converts the image's pixel values to a string, appends metadata for reconstruction, and pads the string as a multiple of 16 bytes. The string is then encrypted using AES with the provided password. The resulting ciphertext is used to construct an encrypted image, and a second level of encryption is performed using level_one_encrypt.

```
def decrypt(ciphername,password):
    secret_image = Image.open(path_join(files_folder_path, "secret.jpeg"))
    ima = Image.open(path_join(files_folder_path, "2-share_encrypt.jpeg"))
    new_image = generate_image_back(secret_image, ima)
    new_image.save(path_join(files_folder_path, "2-share_decrypt.jpeg"))
    print("2-share Decryption done....")
    cipher = open(ciphername,'rb')
    ciphertext = cipher.read()

    # decrypt ciphertext with password
    obj2 = AES.new(bytes(password), AES.MODE_CBC, b'This is an IV456')
    try:
        decrypted_bytes = obj2.decrypt(ciphertext)
    except Exception as e:
        print("Decryption failed:", e)

    # remove padding, convert bytes to string and parse it back into integer string
    decrypted_bytes = decrypted_bytes.rstrip(b'n')
    decrypted = decrypted_bytes.decode()
    decrypted = decrypted.replace("n","")

    # extract dimensions of images
    newwidth = decrypted.split("w")[1]
    newheight = decrypted.split("h")[1]

    # replace height and width with emptyspace in decrypted plaintext
    heightr = "h" + str(newheight) + "h"
    widthr = "w" + str(newwidth) + "w"
    decrypted = decrypted.replace(heightr,"")
    decrypted = decrypted.replace(widthr,"")

    # reconstruct the list of RGB tuples from the decrypted plaintext
    step = 3
    finaltextone=[decrypted[i:i+step] for i in range(0, len(decrypted), step)]

finaltexttwo=[(int(finaltextone[int(i)])-100,int(finaltextone[int(i+1)])-100,int(finaltexto
ne[int(i+2)])-100) for i in range(0, len(finaltextone), step)]

    # reconstruct image from list of pixel RGB tuples
    newim = Image.new("RGB", (int(newwidth), int(newheight)))
    newim.putdata(finaltexttwo)
    outputfilename = path_join(files_folder_path, "visual_decrypt.jpeg")
    newim.save(outputfilename)
    print("Visual Decryption done......")
    return outputfilename
```

This function decrypts an encrypted image file using the provided password. It first reconstructed the original image using the secret and ciphered images saved during encryption. It then decrypts the ciphertext using AES, removes padding, and extracts the image dimensions. Finally, it reconstructs the image from the decrypted pixel values and saves it to disk.

Graphic Interface

The image encryption application features a graphical user interface (GUI) developed using the Tkinter library, designed to facilitate the encryption and decryption of image files. Key components include text messages for the title and author, a password entry field with masking for security, and two main buttons for encryption and decryption processes. Upon user interaction, the application prompts for file selection and performs the desired action while providing success or error notifications through message boxes.

To enhance security, the application uses the **SHA-256** hashing algorithm to hash user-entered passwords before performing encryption or decryption. The main functionalities are managed by event-driven functions: **image_open()** for encryption and **cipher_open()** for decryption. These functions handle file selection, password hashing, and calling the respective encryption or decryption logic, ensuring a smooth and secure user experience.

Overall, the application provides an intuitive and straightforward method for users to secure their images. The user-friendly interface, combined with robust password hashing, ensures that users can easily encrypt and decrypt image files with confidence. The design prioritizes ease of use and security, making it a practical tool for image protection.

FastAPI application

The FastAPI application is designed to provide a secure and efficient way to handle image encryption and decryption through **HTTP** endpoints. The application consists of two main routes: **/encrypt** and **/decrypt**, which facilitate the encryption and decryption processes, respectively. Users upload an image file along with a password, and the application processes the request to either encrypt or decrypt the image using the provided password.

In the **/encrypt** endpoint, the application extracts the password and file from the request, pads the password to ensure it meets the required length, and then calls the encrypt function to secure the image. The encrypted data is then returned to the user in **JSON** format. Error handling is implemented to manage any exceptions that may occur during the encryption process, providing a clear error message to the user.

The **/decrypt** endpoint operates similarly by extracting the password and file from the request and constructing the path to the encrypted file stored in a predefined directory. The application then decrypts the file and streams the decrypted image back to the user using **StreamingResponse**. This approach ensures that the decrypted image is delivered efficiently and in the correct format, enhancing the application's usability and security for users needing to encrypt and decrypt images.

Illustration

Fig06. First Interface

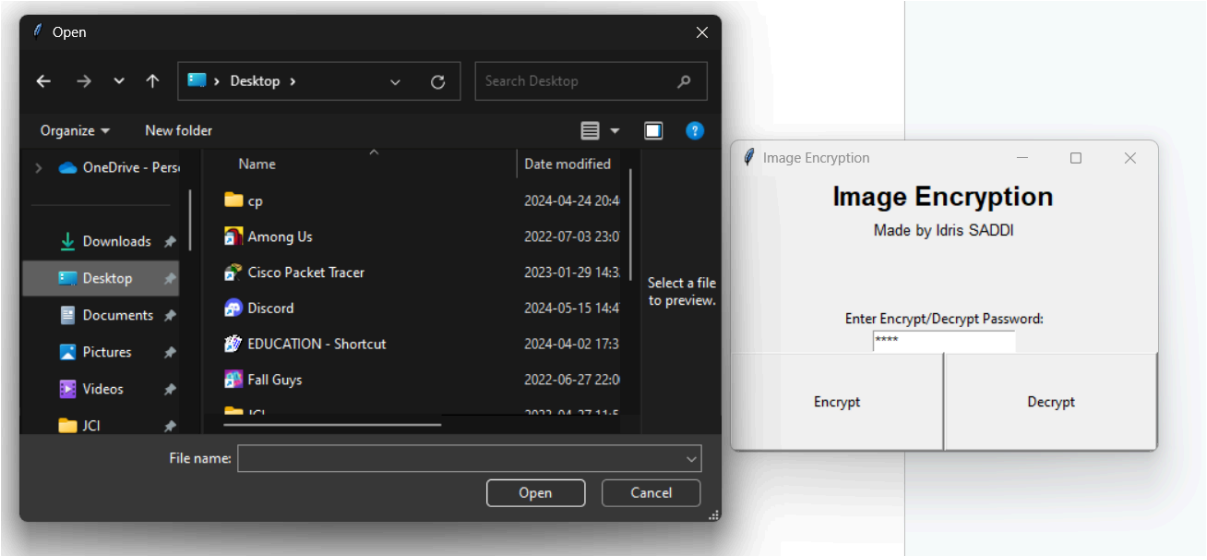

Fig07. Selecting an image or the cipher file

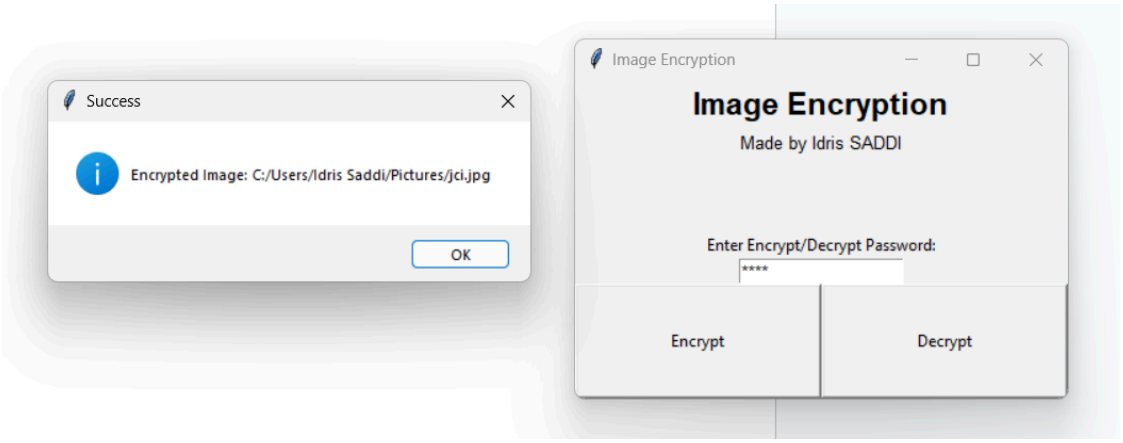

Fig08. Successful Encryption

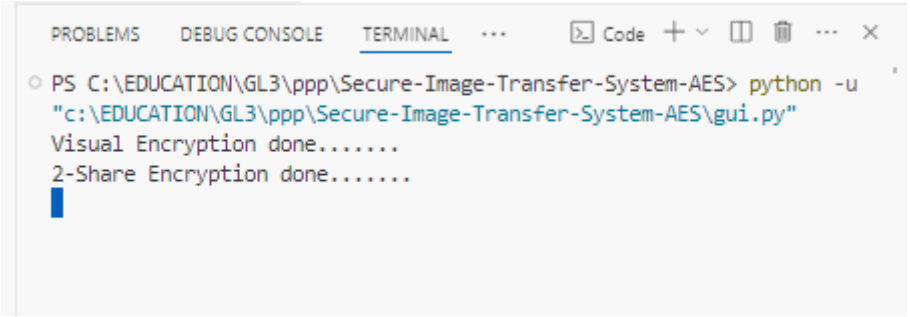

Fig09. Console log Encryption

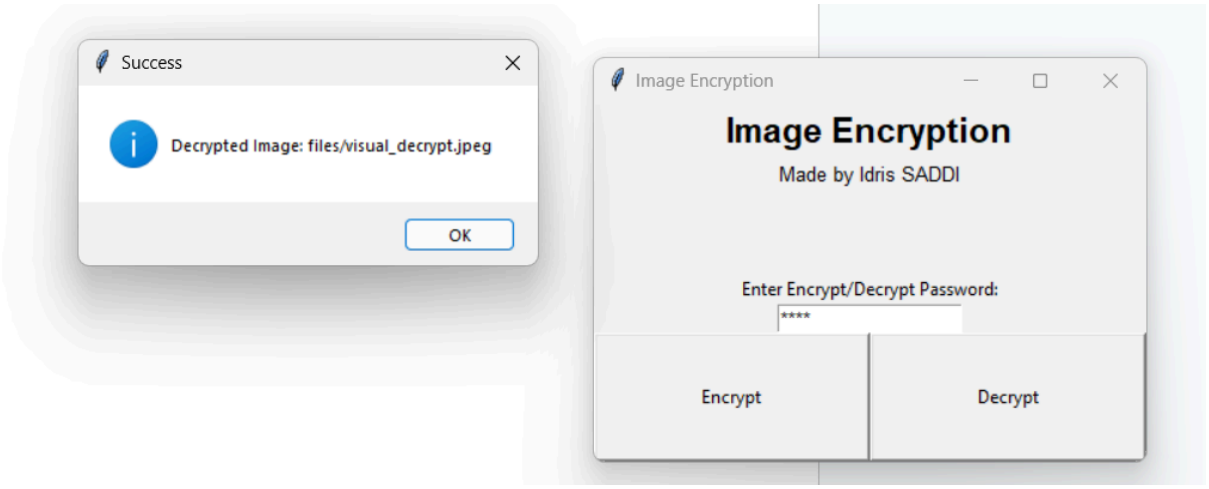

Fig10. Successful Decryption

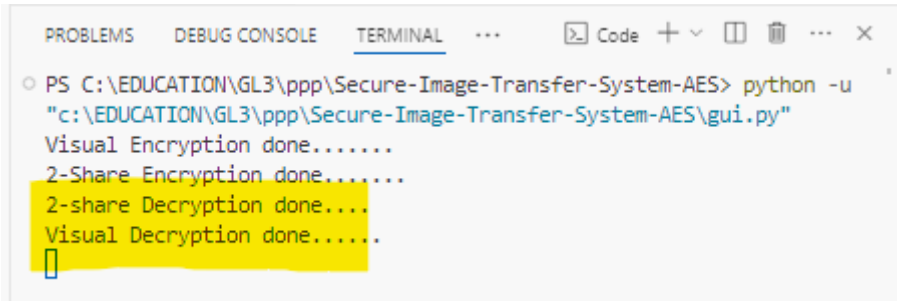

Fig11. Console log Decryption

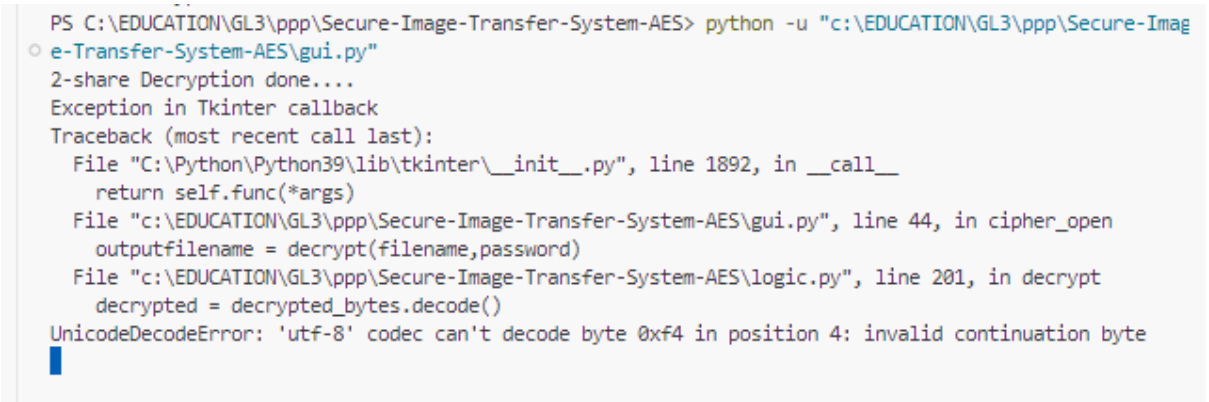

Fig12. Exception on Wrong Password

Conclusion

In this project, I developed a comprehensive solution for secure image encryption and decryption using FastAPI, a modern web framework for building APIs with Python. Our implementation focuses on leveraging AES encryption to ensure the confidentiality and integrity of image data during transfer. By incorporating the FastAPI framework, I have created a robust and scalable web service that allows users to upload images for encryption and later decrypt them with the correct password. This solution demonstrates the practical application of cryptographic techniques in real-world scenarios, providing a foundation for further enhancements and integrations.

Retrospective

Reflecting on this project, several key learnings and improvements stand out. First, the use of FastAPI proved highly effective due to its simplicity, asynchronous capabilities, and automatic documentation generation. These features facilitated rapid development and testing. However, we encountered challenges related to managing file uploads and ensuring compatibility across different environments. Future iterations could focus on optimizing the file-handling process and enhancing the user experience through a more intuitive interface.

Moreover, the integration of robust error-handling mechanisms was crucial in providing clear feedback to users and maintaining the application's stability. Going forward, implementing additional security measures such as input validation, rate limiting, and user authentication would further strengthen the system. Overall, this project has been a valuable exercise in applying encryption techniques within a web service context, highlighting areas for continuous improvement and innovation.

References

1. J. Daemen , V. Rijmen, “The Design of Rijndael, AES - The Advanced Encryption Standard”, Springer Berlin Heidelberg, 2002.
2. W. Stallings, "Cryptography and Network Security: Principles and Practice", Pearson Education Limited 2017.
3. NIST Special Publication 800-38A, "Recommendation for Block Cipher Modes of Operation: Methods and Techniques", Nat. Inst. of Standards and Technology, 2001
4. tkinter — Python interface to Tcl/Tk — Python 3.12.3 documentation
5. FastAPI Documentation: https://fastapi.tiangolo.com/
6. Python Cryptography Toolkit: PyCryptodome 3.210b0 documentation
7. Web Security Fundamentals: OWASP Top Ten | OWASP Foundation
8. Real Python: Asynchronous Programming in Python: https://realpython.com/async-io-python/
9. Pillow (PIL Fork) 10.3.0 documentation
10. Uvicorn : https://www.uvicorn.org/
11. ChatGPT: https://chat.openai.com