



21AIE431

Applied Cryptography

End Semester Project Report

Cipher Connect

Cryptographic Encryption of Video and Audio Data

Faculty Incharge:

Dr. Sunil Kumar

Batch - B Group - 17

Team Members:

Parthvi Manoj – CB.EN.U4AIE21143

Sakthi Swaroopan S – CB.EN.U4AIE21159

Sanjay Chidambaram – CB.EN.U4AIE21160

Taruneshwaran T – CB.EN.U4AIE21170

## ACKNOWLEDGEMENT

We would like to express our sincere appreciation to Dr. K.P. Soman, the Head of the Department, and Dr. Sunil Kumar, our teacher, for their invaluable support and guidance during the development of the Secure Communication project. Dr. Soman's visionary leadership and commitment to our department has been a constant source of inspiration. His guidance and mentorship has played a pivotal role in shaping my academic journey. Likewise, Dr. Sunil Kumar's expertise, insightful feedback, and encouragement has been instrumental in our understanding of the subject matter. We are truly grateful for their contributions to our project and our educational growth. Their dedication to our academic success has been a driving force in our pursuit of excellence.

## PROBLEM STATEMENT:

In the contemporary digital landscape, where information is exchanged ubiquitously, the security and privacy of communication have emerged as critical concerns. Conventional communication channels are susceptible to various forms of cyber threats, ranging from eavesdropping to unauthorized access, thereby compromising the confidentiality of exchanged data. This project addresses the pressing need for secure digital communication by developing an advanced chat application that employs sophisticated cryptographic techniques. The overarching goal is to provide users with a secure and private platform for real-time communication, file sharing, video streaming, and audio communication.

## INTRODUCTION:

Communication in today's digital age demands robust security measures to protect sensitive information from malicious actors. The project introduces a secure chat application that leverages cryptographic principles to guarantee the privacy of communication. By integrating encryption and decryption techniques, the application establishes a secure communication channel between users.

## OBJECTIVES:

1. **Secure Communication:** Develop a communication platform where messages are encrypted and decrypted using advanced cryptographic algorithms, ensuring the confidentiality of exchanged information.
2. **Multimedia Integration:** Extend the capabilities of the chat application beyond text messaging to include secure file transfer, real-time video streaming, and audio communication.
3. **User-Friendly Interface:** Design an intuitive and user-friendly graphical interface that caters to both novice and experienced users, making secure communication accessible to a broad audience.
4. **Advanced Cryptography:** Implement cryptographic techniques, with a focus on the Advanced Encryption Standard (AES), to fortify the security of the communication channel against various threats.
5. **Real-Time Interaction:** Facilitate real-time communication, allowing users to engage in secure conversations, share files, and collaborate seamlessly.

## JAVA IMPLEMENTATION

### TOOLS USED:

- **Programming Language:** Java
- **Encryption:** Advanced Encryption Standard (AES)
- **Networking:** Socket programming in Java
- **User Authentication:** Hashing
- **User Interface:** Swing or JavaFX
- **Webcam Integration:** Webcam-Capture library (sarxos)
- **Dependency Management:** Apache Maven
- **JSON Processing:** GSON library
- **Security Measures:** Secure key management, input validation, and secure data transmission protocols like HTTPS.

### DETAILED METHODOLOGY:

The development of the secure chat application involves a meticulous and multifaceted methodology. The process is divided into several key stages, encompassing the establishment of server-client communication, integration of cryptographic mechanisms, and the inclusion of multimedia features. The following provides a detailed methodology for each phase of the project.

#### 1. Project Initialization

The project commences with the initialization of the development environment and the creation of the core components—chat server and chat client. The choice of Java as the programming language offers portability and compatibility, crucial for deploying the application across diverse systems.

#### 2. Server Initialization and Connection Handling

- **Server Socket Setup:** The server initializes three distinct **ServerSocket** instances for managing chat, video, and audio connections. Each socket listens on a designated port to facilitate different communication channels.
- **Client Connection Handling:** The server employs a perpetual loop to accept incoming connections from chat clients. Upon connection, the server allocates a dedicated thread (**ClientListenThread**) for the client to handle real-time messaging.

#### 3. Shared Secret Text Exchange

- **Group Initialization:** A group chat environment is established with a predefined group name ("Cipher Connect") to create a sense of community among connected users.

- **Shared Secret Text:** The server manages the exchange of a shared secret text string among clients during the initial connection. This shared secret enhances security by serving as a basis for encrypting subsequent communications.

#### 4. Real-Time Chat Messaging

- **Message Encryption and Decryption:** The **ClientListenThread** on the server side ensures real-time chat functionality. Messages are encrypted using the Advanced Encryption Standard (AES) algorithm during transmission and decrypted upon reception. The **Encryption** and **Decryption** classes handle these cryptographic operations.

#### 5. File Transfer

- **File Transfer Protocol:** The application supports secure file transfer between clients. When a client initiates a file transfer request, the server broadcasts the file to all other clients using the **DataOutputStream** of their respective sockets.

#### 6. Video Streaming

- **Video Server Initialization:** The video server (**VideoServer**) accepts incoming video connections, and a dedicated thread (**VideoStreamThread**) is spawned for each client. This thread manages the real-time streaming of video using Java's **ObjectInputStream** and **ObjectOutputStream**.

#### 7. Audio Communication

- **Audio Server Initialization:** Similar to video, the audio server (**AudioServer**) accepts incoming audio connections, and a dedicated thread (**AudioStreamThread**) is created for each client. The audio threads manage the streaming of audio data using Java's **ObjectInputStream** and **ObjectOutputStream**.

#### 8. Integration of Multimedia Features

- **Webcam Support:** The **Sarxos Webcam Library** is integrated to provide webcam support for video streaming.
- **Audio Streaming with Microphone Support:** The application facilitates audio communication with microphone support, enabling real-time voice communication among users.

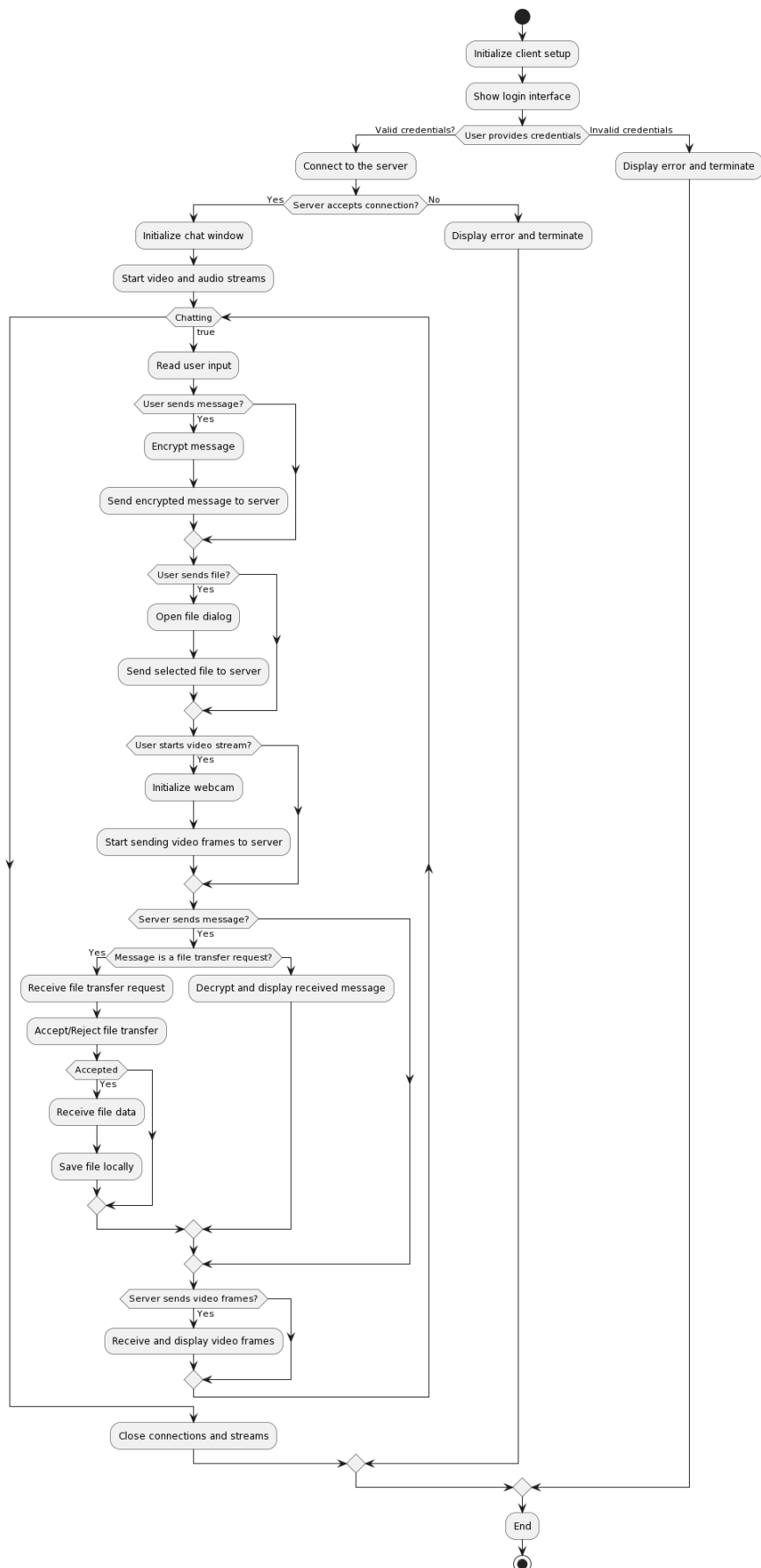
#### 9. User Interface Design

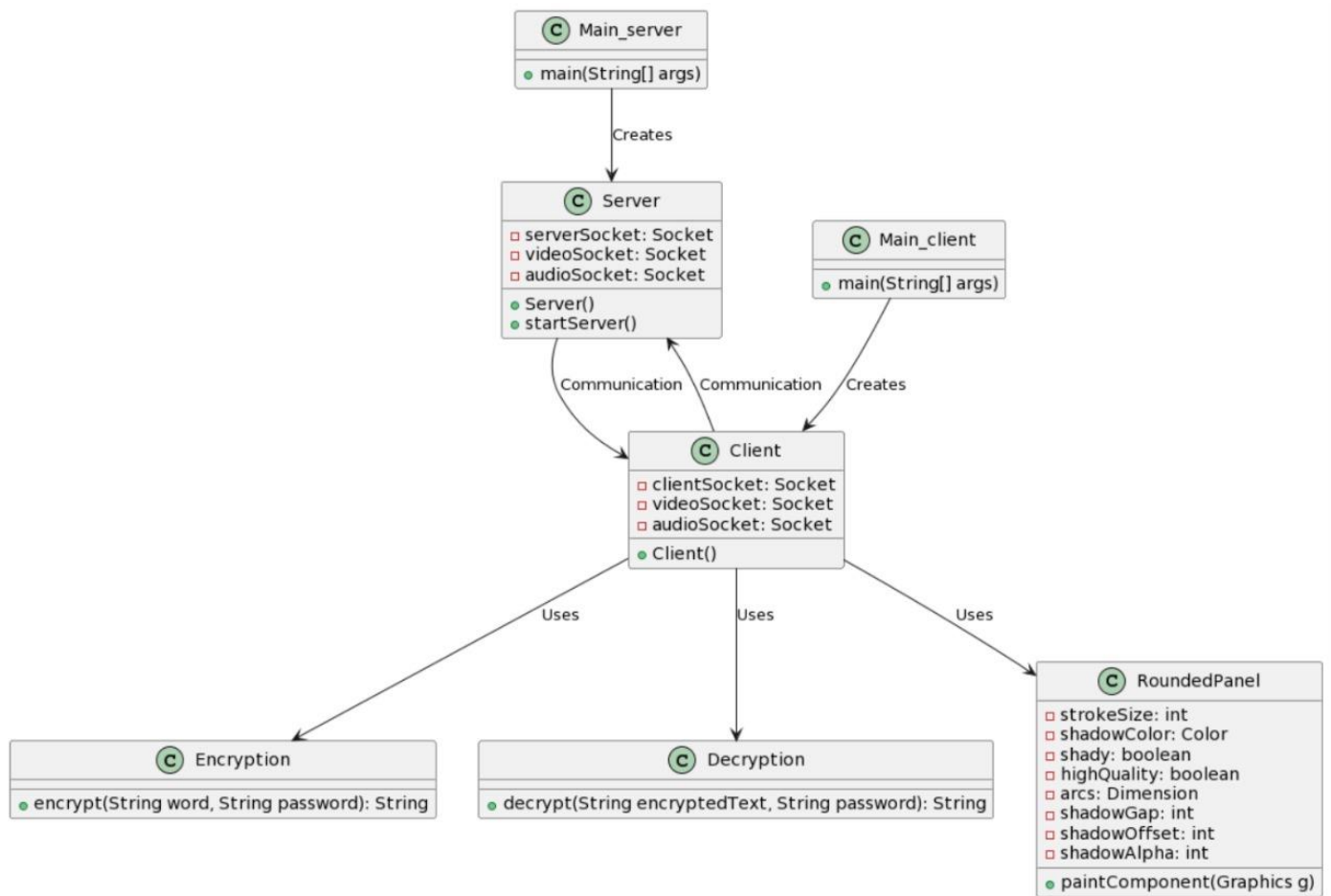
- **Graphical User Interface (GUI):** The chat client features an intuitive GUI designed using Java Swing. Users can configure connections, exchange messages, and engage in multimedia interactions seamlessly.

#### 10. Testing and Debugging

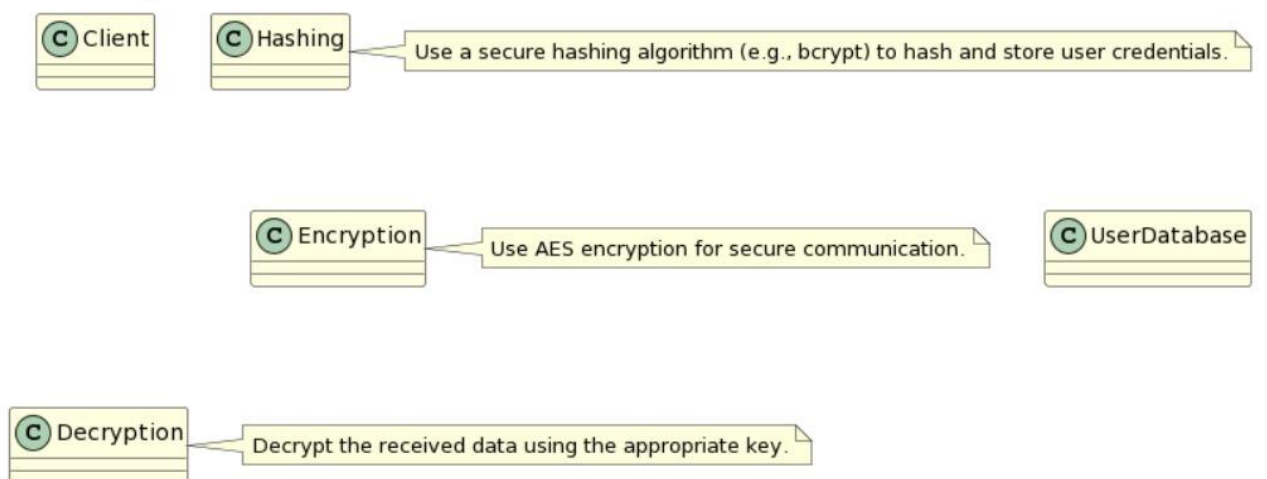
- **Unit Testing:** Each component is rigorously tested in isolation to ensure its functionality meets the specified requirements.
- **Integration Testing:** The complete application undergoes extensive integration testing to identify and address any issues that may arise from the interaction of different modules.

## FLOW CHART:



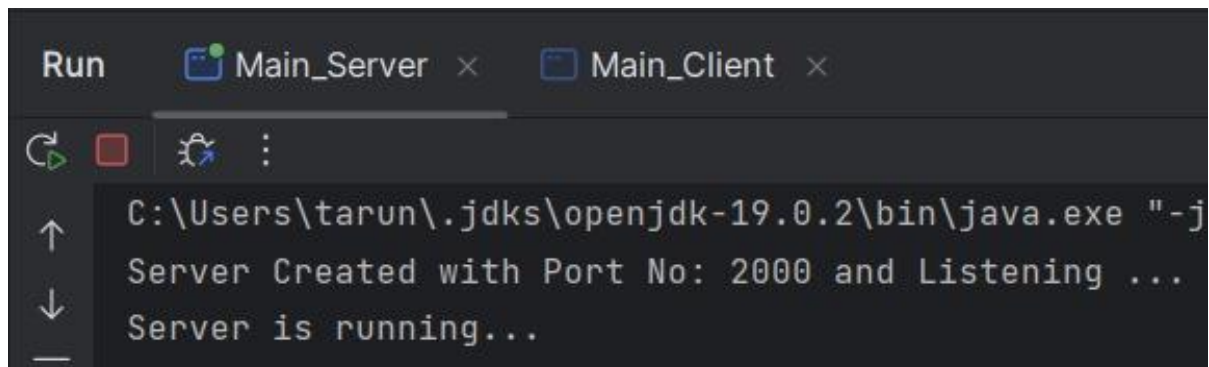


### Main Methodology



**OUTPUT:**

Running Server:



The screenshot shows an IDE with two tabs: 'Main\_Server' and 'Main\_Client'. The 'Main\_Server' tab is active, displaying the following output in the console:

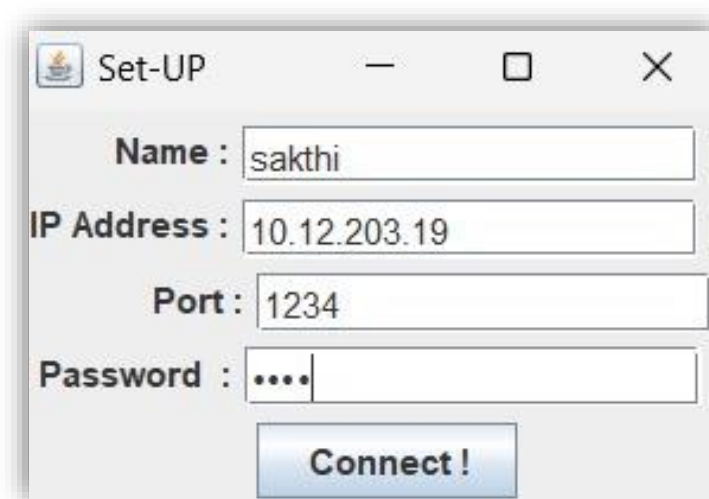
```
C:\Users\tarun\.jdk\openjdk-19.0.2\bin\java.exe "-j
Server Created with Port No: 2000 and Listening ...
Server is running...
```

Client GUI:



The screenshot shows a window titled 'Client GUI'. It contains two input fields: 'Username:' with the text 'Tarun' and 'Password:' with masked characters '.....'. Below these fields are two buttons: 'Sign Up' and 'Sign In'.

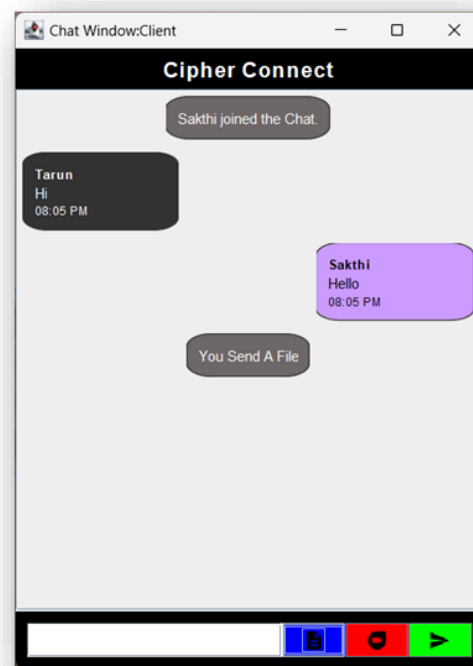
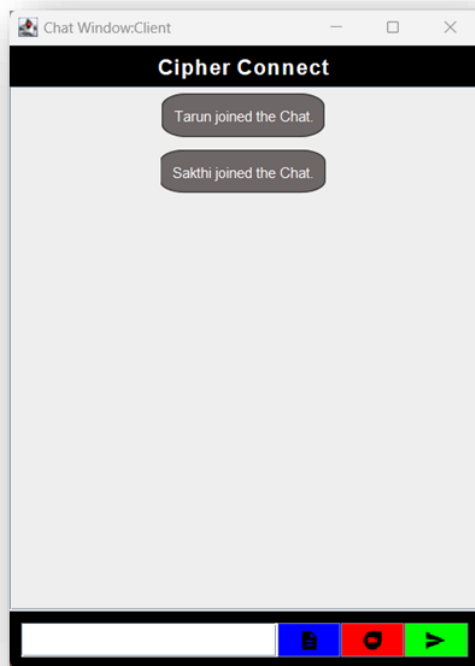
Set- up:



The screenshot shows a window titled 'Set-UP'. It contains four input fields: 'Name : sakthi', 'IP Address : 10.12.203.19', 'Port : 1234', and 'Password : ....'. Below these fields is a button labeled 'Connect!'.



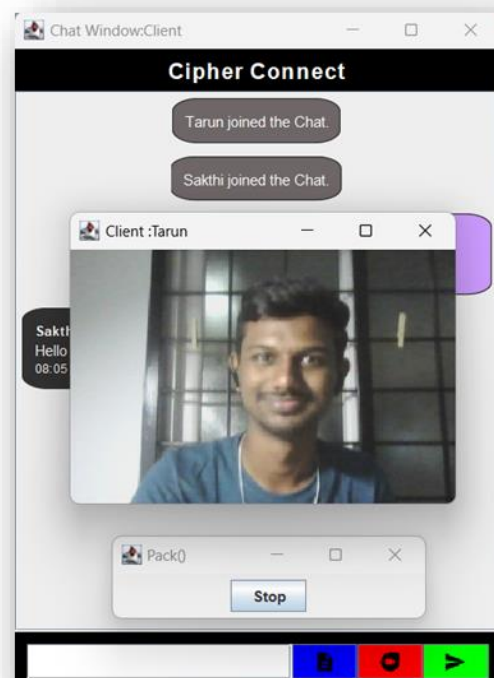
## Chat Window Client:



## Chat Window File Transfer



## Chat Window Video



## **PYTHON IMPLEMENTATION**

### **TOOLS USED:**

- **Python:** Primary programming language.
- **AES Encryption:** Utilized for encrypting video stream.
- **Python Object-Oriented Programming (OOP):** Structuring code using classes and methods.
- **Python Tkinter:** For GUI

### **DETAILED METHODOLOGY:**

The development of the secure chat application involves a meticulous and multifaceted methodology. The process is divided into several key stages, encompassing the establishment of server-client communication, integration of cryptographic mechanisms, and the inclusion of multimedia features. The following provides a detailed methodology for each phase of the project.

#### **1. Objective Definition**

Define the primary objective: Implement a secure video transmission system using AES encryption and socket programming.

#### **2. Environment Setup**

Set up the development environment:

- Choose Python as the programming language.
- Install necessary libraries: OpenCV for video processing, and any other dependencies.

#### **3. Architecture Design**

Design the system architecture:

- Define classes for the transmitter and receiver components.
- Plan the multithreading strategy for concurrent execution.

#### 4. Encryption and Decryption Functions

Implement AES encryption and decryption functions:

- Create functions for padding and unpadding video data.
- Handle exceptions and errors during encryption and decryption processes.

#### 5. Transmitter Implementation

Implement the Transmitter class:

- Initialize a socket for communication.
- Establish a connection with the receiver using socket programming.
- Use OpenCV for video capture.
- Implement multithreading for video capture, encryption, and transmission processes.

#### 6. Receiver Implementation

Implement the Receiver class:

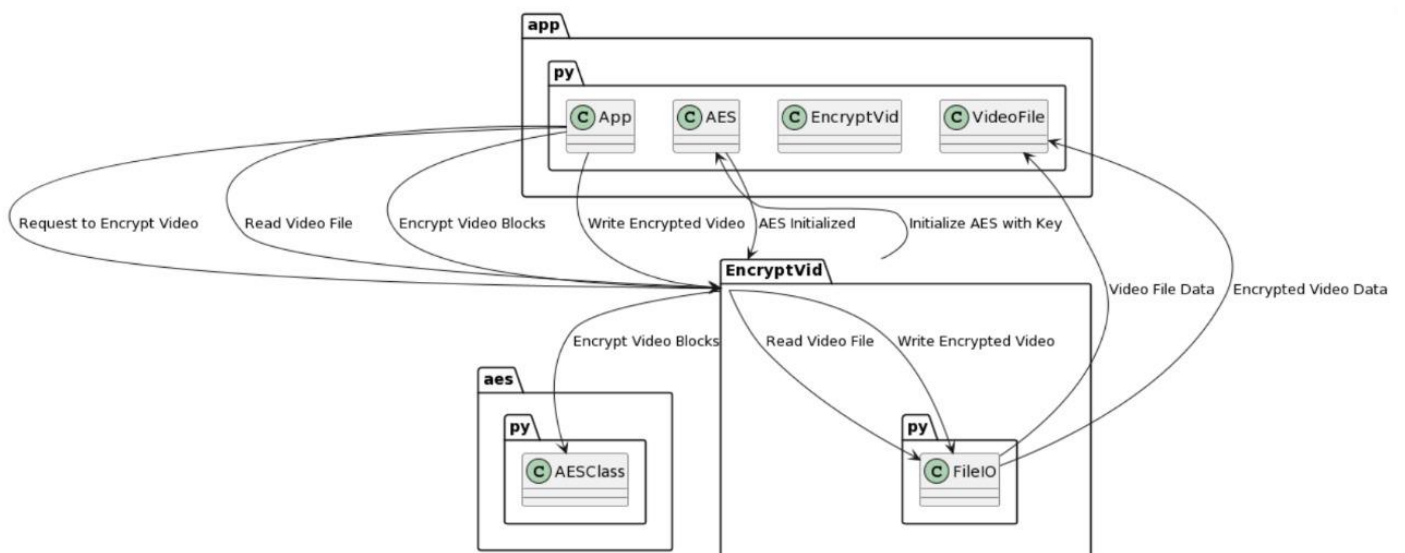
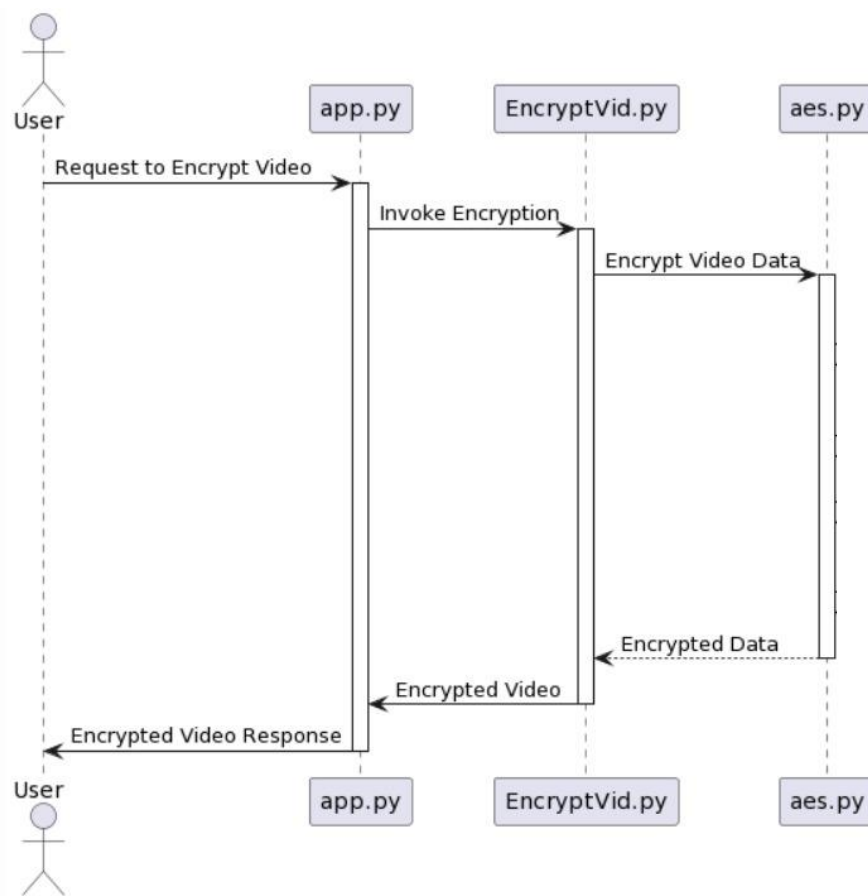
- Initialize a socket, bind it to an IP address and port.
- Implement multithreading for video reception, decryption, and display processes.
- Use OpenCV to display the decrypted video frames in real-time.

#### 7. Testing

Conduct comprehensive testing:

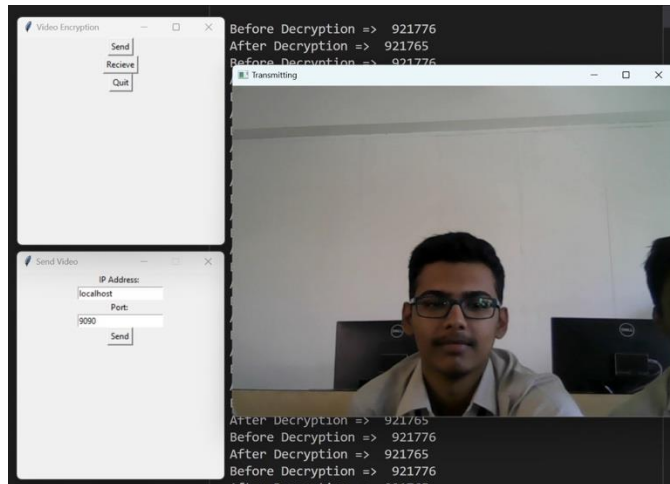
- Perform unit testing for encryption and decryption functions.
- Conduct integration testing to validate end-to-end functionality.

## FLOW CHART:

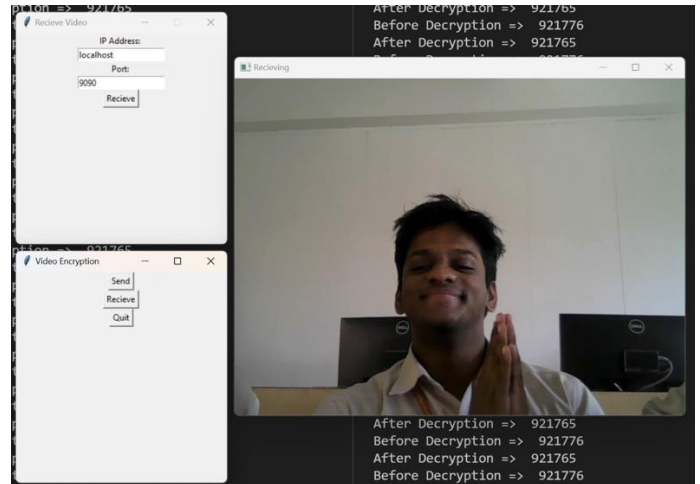


## OUTPUT:

### Transmission



### Receiver



## CONCLUSION:

The output of the project is a fully functional secure chat application with the following features:

- Secure end-to-end chat messaging.
- File transfer functionality.
- Real-time video streaming with webcam support.
- Audio streaming with microphone support.

The secure video streaming application, utilizing AES encryption, delivers encrypted real-time communication across chat, video channels, and audio channels. With a robust server-client setup, it ensures data confidentiality and integrity, offering users a protected environment for seamless interactions and file sharing. The integration of stringent security measures and extensive testing guarantees a highly secure platform for reliable video streams and chats.

## REFERENCES:

- <https://www.vplayed.com/blog/aes-video-encryption-for-streaming/>
- <https://www.vdocipher.com/blog/2020/11/aes-128-encryption-video-drm-secure/>
- <https://www.backlight.co/blog/streaming/how-to-encrypt-video-files-with-aes-encryption>
- <https://www.haivision.com/blog/all/video-security-aes-encryption/>
- <https://blog.video.ibm.com/streaming-video-tips/aes-video-encryption-256-vs-128-bit/>