BDIC3025J-Security & Privacy-2023/24 Autumn

Beijing-Dublin International College

DES-based Secure Communication Report

Students:
Jiahe Zhang 20372118
UCD 20205722

November 21, 2023

Contents

1 General Overview

1.1 Project Overview

The primary goal of this project is to design and implement a secure communication application based on the Data Encryption Standard (DES) algorithm. The application facilitates encrypted data exchange between multiple users engaged in interactive sessions. The DES algorithm, developed in Project #1, forms the core of the encryption and decryption processes, ensuring the confidentiality and integrity of transmitted data.

1.2 Objectives

- **Secure Communication:** The project emphasizes the secure transmission of data or files between users. The DES algorithm is employed to encrypt and decrypt the exchanged information, ensuring that sensitive data remains confidential during transmission.
- **Independently Performed Data Exchange:** The application supports independent data exchanges between users. Each user can send and receive messages or files separately, contributing to a seamless and secure communication experience.
- **User-Friendly Interface:** To enhance usability, a user-friendly interface is implemented on both the server and client sides. The interface provides a platform for users to interact, exchange messages, and share files securely.

1.3 Components

The project comprises four main components:

- **Configuration (config.py):** Building upon the configuration module from Project #1, this component sets the groundwork for the DES algorithm and encryption key management.
- **DES Implementation (DES.py):** Leveraging the DES implementation developed in Project #1 but add some updates, this component handles the encryption and decryption processes essential for secure communication.
- **Server Component(server.py):** The server component establishes and manages connections between users. It handles incoming messages and files, ensuring their secure distribution to the intended recipients.
- **Client Component(client.py):** The client component facilitates user interaction with the application. Users can log in, send messages, share files, and receive encrypted content, all through a user-friendly interface.

2 Implementation Details

2.1 Configurations

The file **config.py** contains:

- initial permutation (IP) and its inverse (IP-RE)
- expansion permutation (E)
- permutation P
- S-boxes (S)
- key generation constants include PC-1 and PC-2, as well as the key schedule shift values All configurations follow the guide from **Data Encryption Standard (DES) by NIST, FIPS PUB 46-3, October 25, 1999 (Reaffirmed)**.

2.2 DES Algorithm Functions and Difference from Project #1

In Project #2, the implementation heavily relies on the DES (Data Encryption Standard) algorithm, which was initially developed in Project #1. The DES functions are organized in the DES.py module, which is shared between Project #1 and Project #2.

Key Modifications for Project #2

- **UI Handling for DES.py:** To prevent conflicts with the Tkinter UI of Project #1 when running the **client.py** script, the UI handling code at the end of **DES.py** has been encapsulated within the **create_des_gui** function. This function is now invoked under the condition **if__name__ == "__main__":**. As a result, when running the **client.py**, the DES UI is not immediately executed, avoiding interference with the main chat room interface.

- **Global Variables result_text and root:** To address issues where **result_text** and **root** were defined within the **create_des_gui** function and faced referencing problems in other functions, they have been declared as global variables. This adjustment ensures that **result_text** and **root** are initialized in the **create_des_gui** function and can be accessed globally throughout the script.

This modification strategy resolves potential "Unresolved reference" or "NameError" problems encountered when referencing these variables in various functions.

2.3 Server-Client-based Chatroom

The chatroom framework draws inspiration from a prior distributed systems project in Java. The adaptation to Python and integration with the DES algorithm cater to the specific requirements of this project.

This section provides a high-level overview of the server-client-based chatroom, focusing on its structure, key features, and communication flow.

2.3.1 Server

The server is implemented as a separate thread using the **ChatServer** class. It handles incoming connections from clients, manages user authentication, and facilitates message and file exchange. The server consists of the following key components:

Socket Communication: The server establishes a socket connection to listen for incoming client connections.

User Authentication: Clients provide a username upon connection. In cases where a username is not specified, the server uses the client's IP and port as a default. To prevent duplicate usernames, a suffix is appended if needed.

User List Management: The server maintains a list of online users, and updates are broadcasted to all connected clients.

Message Queue: Messages to be sent are stored in a queue (messages), and a separate thread (sendData) continuously sends messages from the queue to clients.

File Transfer: File data is encoded in Base64 and sent along with relevant information like sender, receiver, and filename.

2.3.2 Client

The client side is implemented using the **client.py** script. It involves the following components:

User Authentication: Clients provide a username during login, and the server handles any necessary modifications to avoid duplicate usernames.

Chat Interface: The main chat window is created using Tkinter, providing a user-friendly environment for sending and receiving messages.

Message Encryption: Messages are encrypted using the DES algorithm before being sent to the server.

File Transmission: Files are selected by the user and sent to the server, which then distributes them to the intended recipients.

Message Decryption: Incoming messages are decrypted using the DES algorithm before being displayed in the chat interface.

2.3.3 Communication Flow

The communication flow involves the server managing incoming connections, authenticating users, and facilitating message and file exchange. Clients interact with the server to send and receive messages and files. The server acts as a central hub, ensuring seamless communication among connected clients.

2.3.4 User Interface Interaction

The client's user interface provides a convenient platform for users to send and receive messages, select files for transmission, and interact with the chatroom seamlessly. The design emphasizes user-friendliness and responsiveness.

2.4 Server-Side Implementation

2.4.1 Server Initialization and Configuration

The server-side implementation revolves around the ChatServer class, responsible for managing connections, message handling, and file transmission. Let's delve into the key components and functionalities of the server-side implementation:

- **Server Initialization:** The server socket is created using the **socket** module, configured for IPv4 communication (**socket.AF_INET**) and streaming sockets (**socket.SOCK_STREAM**).
- **Working Directory:** The working directory is set to the current directory using **os.chdir(sys.path[0])**. This ensures that file operations are relative to the location of the server script.

2.4.2 User Connection and Handling

- **User Authentication:** Upon connection, the server receives the username from the client. If the username is not provided (**user name not exist**), it defaults to the combination of the client's IP and port. To handle duplicate usernames, a suffix is appended to the duplicate usernames (e.g., 'user2', 'user3').
- **Message and File Reception:** The server continuously listens for messages from clients. If a message starts with the 'File ' tag, it indicates a file transmission. The server decodes the file content and saves it with a unique filename.
- **User Disconnection Handling:** If a user disconnects, the server removes the user from the list and updates the online user list.

2.4.3 Message Queue Handling and Broadcasting

- **Message Queue Management:** The **Load** function adds data (address and data to be sent) into the messages queue. The **sendData** function continuously checks the messages queue and sends messages to clients.
- **Message Broadcasting:** Messages can be either strings (regular chat messages) or lists (user lists). The server broadcasts messages to all connected clients accordingly.

2.4.4 File Transmission

- **File Encoding and Transmission:** This function reads the file content, encodes it in base64, and constructs a message with the 'File ' tag. The message is then added to the messages queue for transmission.

2.4.5 Server Thread Execution

- **Server Initialization:** The server socket is bound to the specified IP and port, and the **sendData** function is started in a separate thread to handle message broadcasting.
- **Connection Handling:** The server continuously accepts incoming connections and initiates a new thread (**receive**) for each connected client.

2.5 Client-Side Implementation

2.5.1 User Authentication and Connection

The client-side implementation is responsible for user authentication, connection establishment, and the creation of the graphical user interface (GUI). Let's break down the key components and functionalities:

- **Login Window:** The initial part of the client script is dedicated to creating a simple login window using the Tkinter library.
- **Variables:** Tkinter variables are used to store the entered IP address and username.
- **Login Function:** The **Login** function is called when the user clicks the login button. It extracts the IP, port, and username from the input fields and closes the login window.
- **Connection Establishment:** A socket is created, and a connection is established with the server using the provided IP and port. The client sends the chosen username or defaults to the combination of IP and port.

2.5.2 Main Chat Window

- **Main Chat Window:** After successful login, a new Tkinter window is created for the main chat interface.

2.5.3 GUI Components

- **Chat Display Area:** The **ScrolledText** widget is used to display chat messages with scrolling functionality.
- **Input Variables:** Tkinter variable to store the user's input in the text input area.
- **Text Input Area:** A multiline text input area for typing messages.
- **Online Users Listbox:** A listbox to display online users.

2.5.4 Message Encryption and Sending

- **Encryption Key:** A variable to store the encryption key for secure message transmission.
- **Message Encryption and Sending:** The **send** function is responsible for encrypting messages using DES and sending them to the server.

2.5.5 File Transmission

- **File Transmission:** The **sendFile** function allows users to select a file, encode it in base64, and send it to the server with appropriate metadata.

2.5.6 Message Decryption

- **Message Decryption:** The **decryptMessages** function prompts the user to enter a decryption key and decrypts the displayed messages accordingly.

2.5.7 Message and File Reception

- **Message and File Reception:** The **receive** function continuously listens for incoming messages and files. It updates the user list and processes messages or files accordingly.

2.5.8 User Interface Interactions

- **GUI Main Loop:** The main loop of the Tkinter GUI is started, allowing for user interactions and continuous message reception. The socket connection is closed when the user exits the application.

3 User Instructions

In this section, I will provide a step-by-step guide on utilizing the secure communication application. The primary objective is to ensure the encryption of data transferred between users engaged in interactive sessions. The user interface allows for independent data exchanges, demonstrating the secure transmission of messages and files. Each step ensures that data (or files) is encrypted during transmission, maintaining confidentiality and integrity. Let's delve into the instructions to effectively use the secure communication features.

```
Traceback (most recent call last):
  File "D:\studyANDworkFiles\UnderGraduate\Sem4-1\BDIC3025JSec&Pri\Project1\sender.py", line 131, in <module>
    client_socket = connect_to_server()
                    ^^^^^^^^^^^^^^^^^^^
  File "D:\studyANDworkFiles\UnderGraduate\Sem4-1\BDIC3025JSec&Pri\Project1\sender.py", line 14, in connect_to_server
    client_socket.connect(server_address)
ConnectionRefusedError: [WinError 10061] 由于目标计算机积极拒绝，无法连接。
```

Figure 1: Possible Error: ConnectionRefusedError

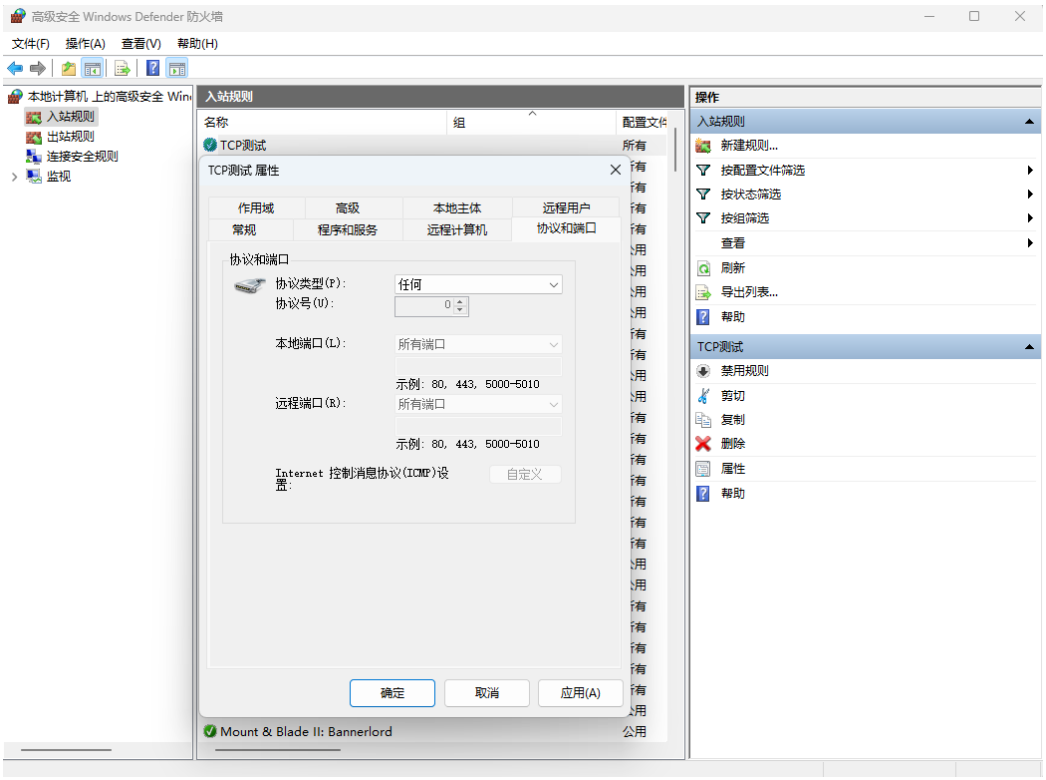

Figure 2: Possible Solution: Configure Port Settings

- Before running the project, ensure to prevent the occurrence of the error shown in Figure **??**, specifically, the 'ConnectionRefusedError: [WinError 10061]' error.
- To address this, configure the port settings in advance as shown in Figure **??** to grant permission for usage.

3.1 Starting the Server and Clients

To begin using the secure communication application, follow these steps to start the server and client components.

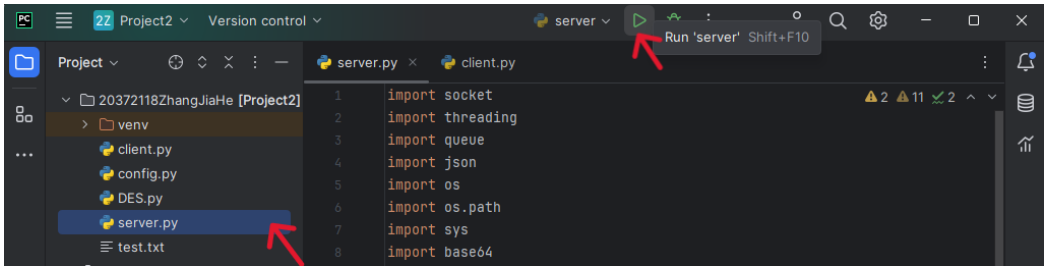

Figure 3

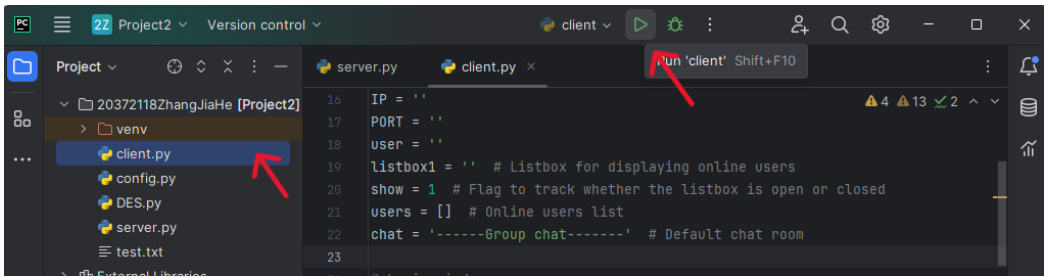

Figure 4

- Ensure that `DES.py` is in the same directory as this document. Before starting the client, initiate the server by running the `server.py` script. Start the server only once.
- Once the server is successfully running, proceed to start the client. Multiple clients can be started simultaneously for testing purposes.

3.2 Client Initialization: Entering Username and Port

Once the client is launched, you will be prompted to enter your chosen username and port number. This step ensures a personalized and secure connection to the chatroom.

Upon launching the client, a login window will appear, prompting you to enter your chosen username and port number.

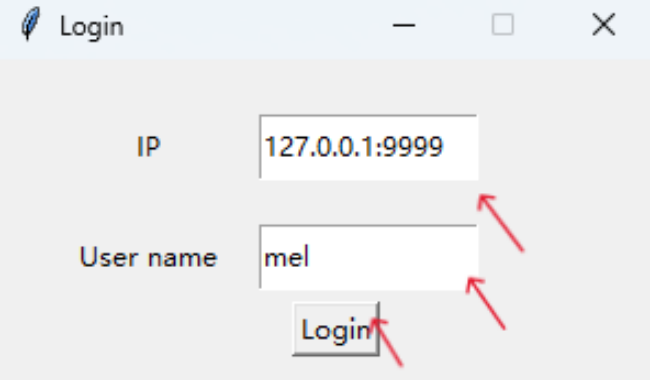

Figure 5

Follow these steps:

- Enter the desired IP address and username in the respective fields.
- Click the "Login" button to join the corresponding chatroom.

3.3 Navigating the Chatroom Interface

Upon entering the chatroom, familiarize yourself with the layout of the user interface. This subsection provides an overview of the different sections and functionalities available to you.

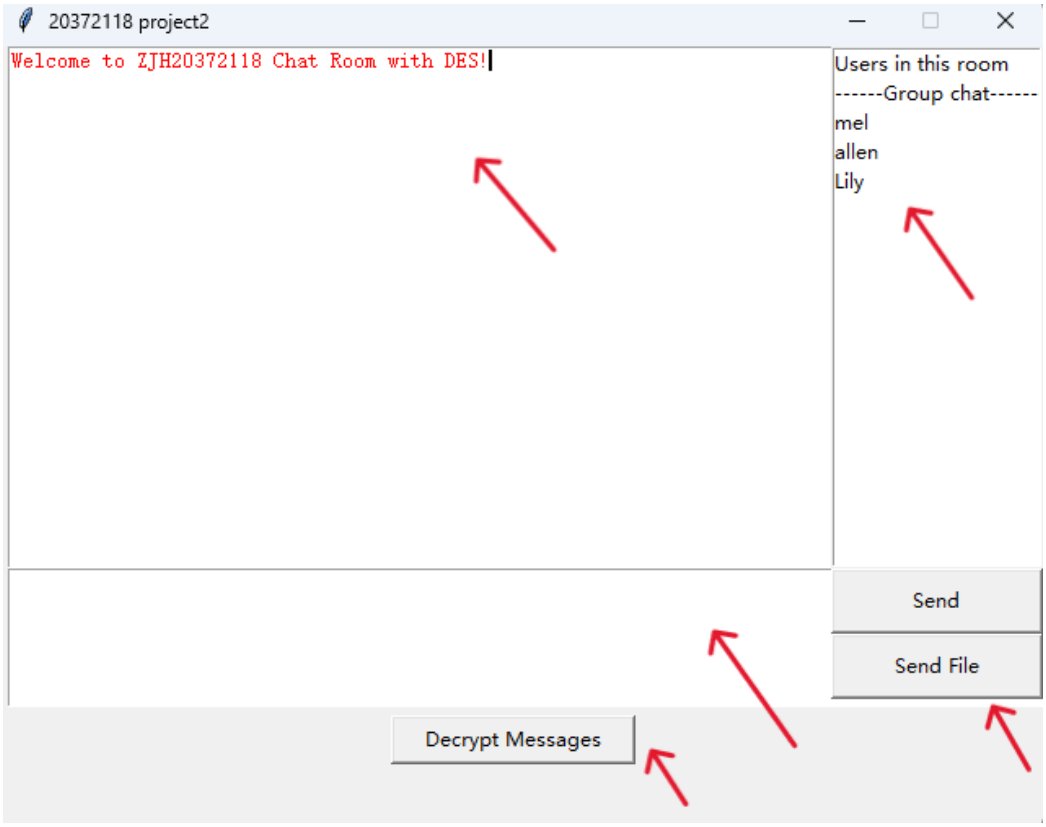

Figure 6

Key features of the interface:

- Message Display Area (Top Left): Displays system prompts and messages from all users in the current room.
- Online Users (Top Right): Real-time list of users currently online in the room.
- Text Input Area (Bottom): Enter your text messages in this area.
- Send Message and Upload File Buttons: Located to the right of the text input area, these buttons allow you to send text messages and upload files, respectively.
- Decrypt Last Message Button: Decrypts the most recent encrypted message in the chat.

3.4 Sending Text Messages and Encryption

Learn how to compose and send text messages securely. This part guides you through the process of encrypting your messages before transmitting them to ensure confidentiality.

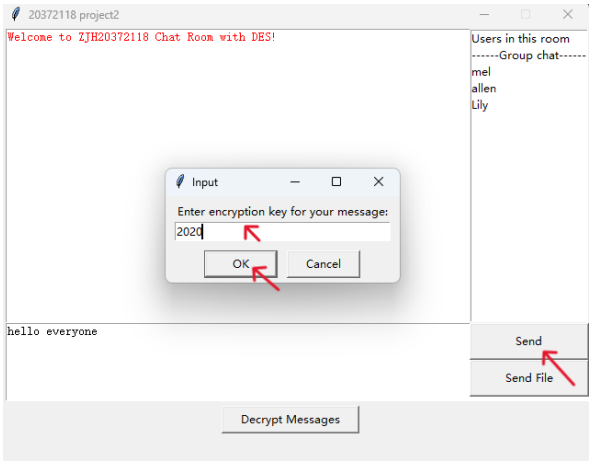

Figure 7: Sending Text Messages

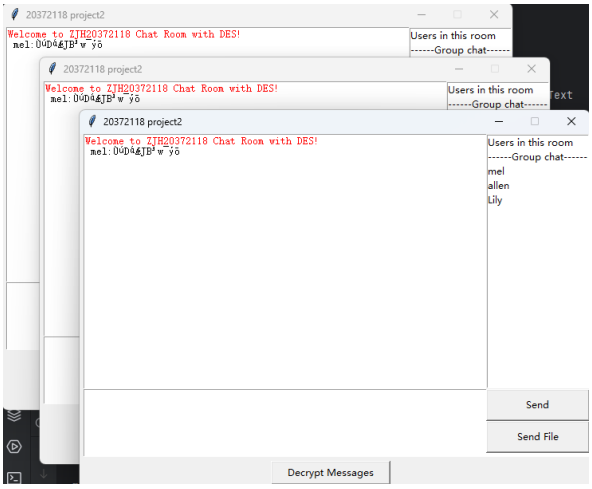

Figure 8: Message Visibility

To send a text message securely:

- Enter your message in the text input area.
- Click the "Send Text" button.
- Upon the first attempt to send a message, you will be prompted to enter your encryption key specific to the current room. This key is used to encrypt your messages locally before transmission.
- Subsequent messages will be encrypted using the entered key.

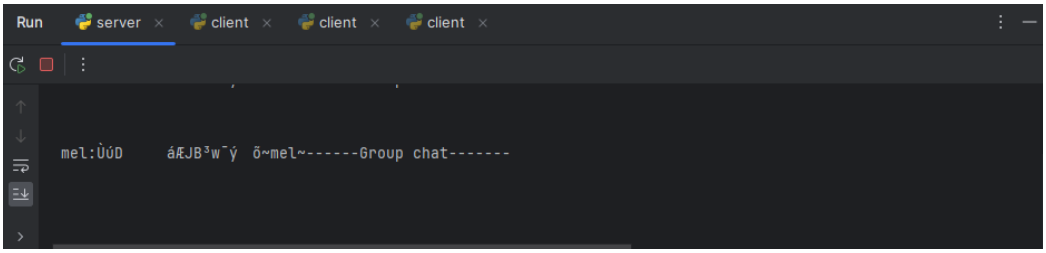

Figure 9: Server Cannot See Original Messages

Figure 9 illustrates that the server only receives and forwards encrypted messages. Your original message content remains confidential, enhancing the security of the communication process.

3.5 Decrypting Received Messages

For users receiving encrypted messages, follow these steps to decrypt and view the content securely.

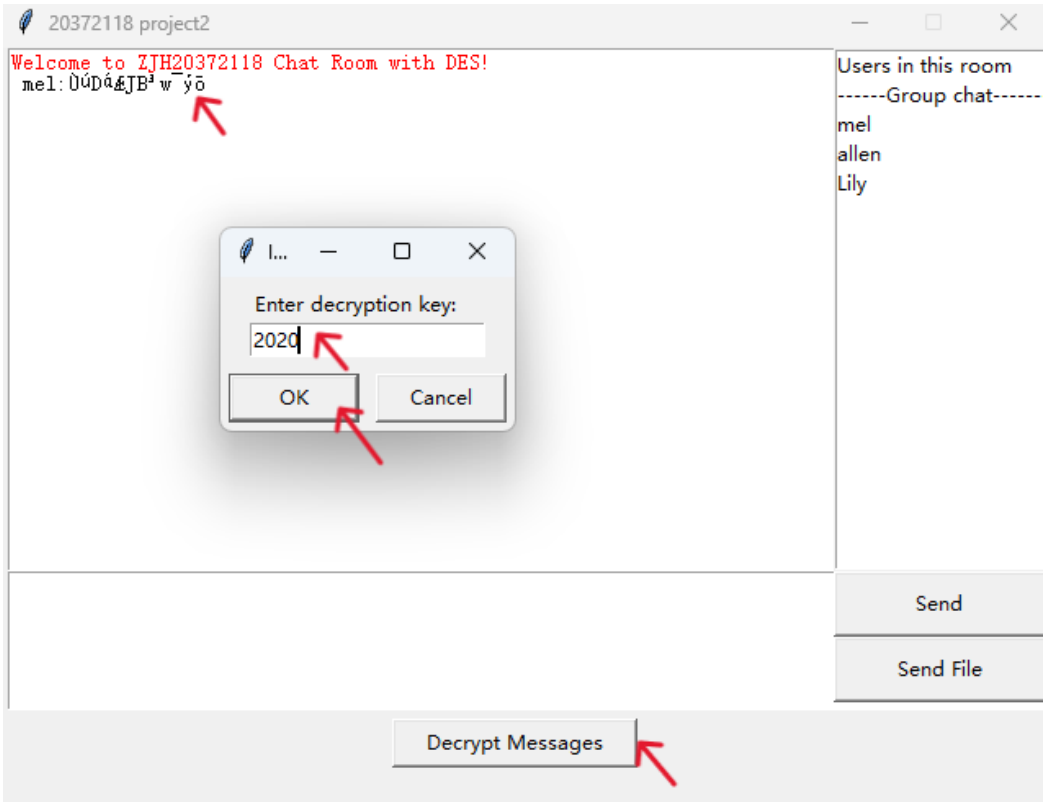

Figure 10: Decrypting Received Messages

To decrypt a received message:

- Click on the "Decrypt" button located at the bottom of the chat interface.
- Enter the corresponding encryption key when prompted.
- The decrypted message will be displayed in the message area.

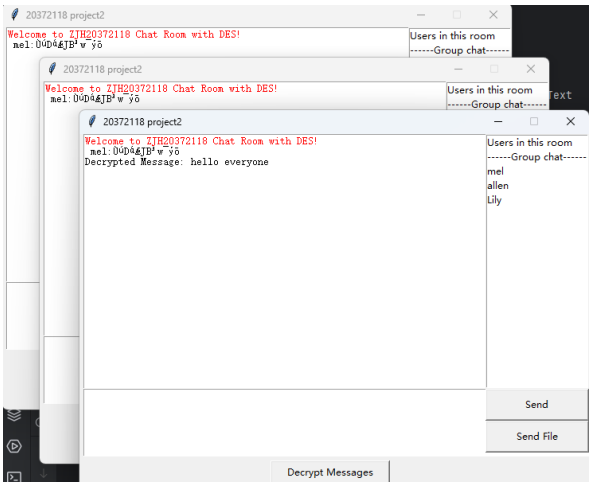

Figure 11: Decryption Result

Figure 12: Decryption Sample

In the left image (11), the decrypted result is displayed in the chat interface after decryption. On the right (12), it illustrates a more complex interaction between three users, demonstrating real-time synchronization of messages and encrypted content across multiple user interfaces.

3.6 File Transmission

Discover how to transmit files securely within the chatroom. Follow these steps to select and send files to other users.

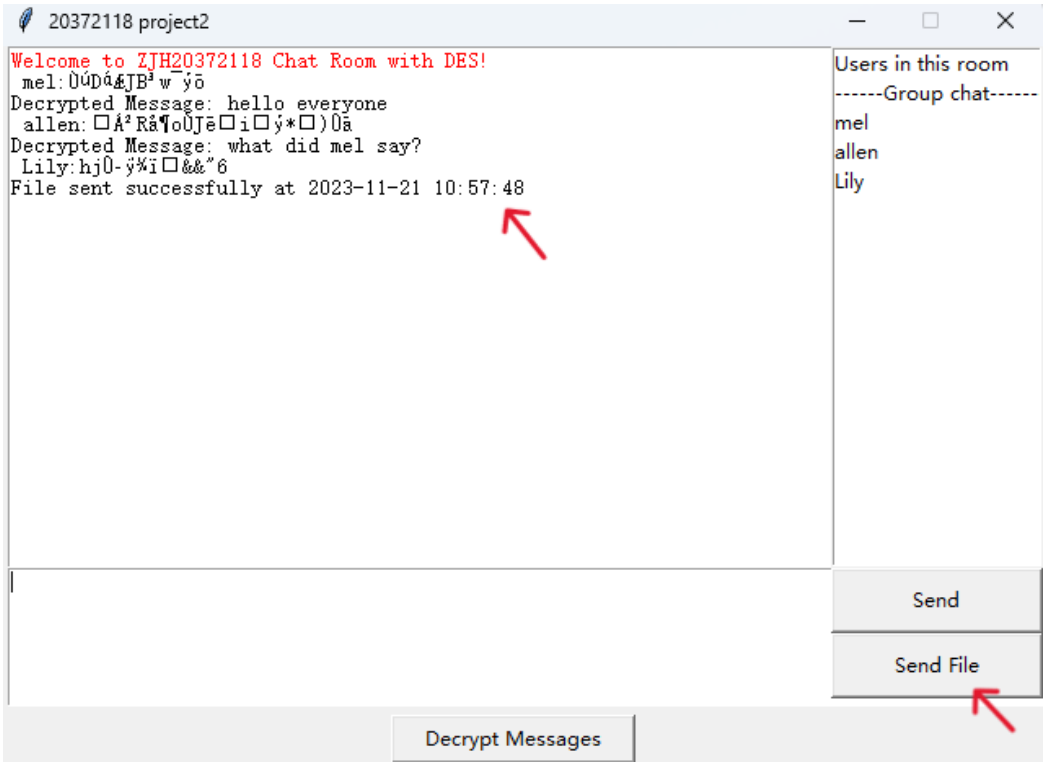

Figure 13: Sending a File

To send a file:

1. Click on the "Send File" button.
2. Choose the file you want to send.
3. The system will automatically encrypt the file using the user's local encryption key.
4. Click "Send" to initiate the file transmission.

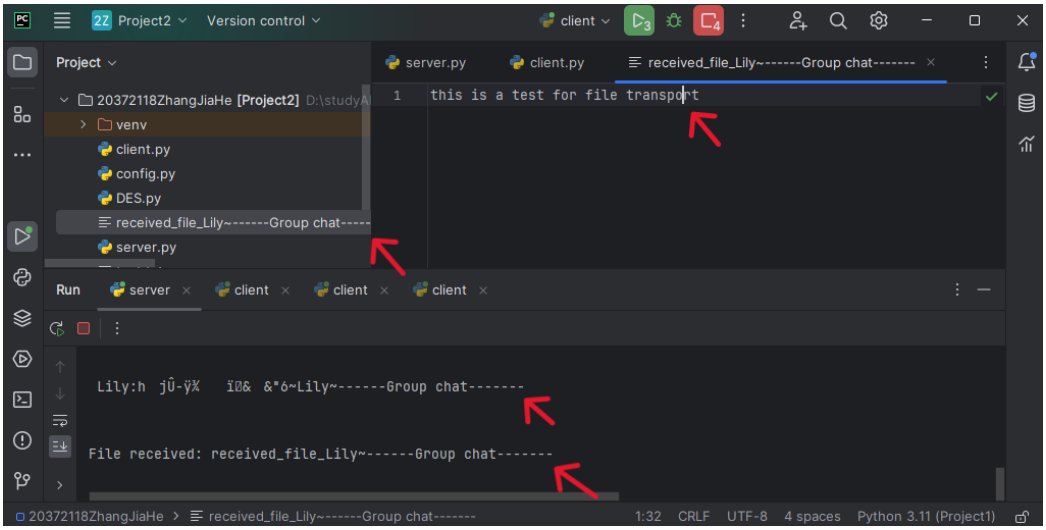

Figure 14: File Transmission Result

In the first image (13), it illustrates the process of initiating file transmission. The second image (14) demonstrates the result of the file transmission, emphasizing the confidentiality of the transmission process where the server cannot access the original content.