TABLE OF CONTENTS

ABSTRACT

The project completed during the internship at Sri Ramachandra Faculty of Engineering and Technology is an Internship Program done in the field of Cybersecurity. This report presents the design and implementation of a chat application that uses the Caesar cipher for encryption, integrated with a graphical user interface (GUI). The objective is to provide a secure and user-friendly messaging platform, demonstrating the application of classical encryption methods in modern software development.

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. BACKGROUND

The advent of digital communication has changed the way people communicate, enabling them to send messages instantly around the world. However, with the rise of digital communication, the need to protect these networks has become paramount. Protecting the confidentiality and integrity of data transferred on digital platforms is largely accomplished through the use of encryption techniques.

One of the earliest and simplest methods of encryption, the Caesar cipher replaces each plain text with a letter in a predetermined number of places below the alphabet Despite its simplicity vulnerable to brute force attacks, the Caesar cipher provides a basic understanding of the principles of cryptography.

1.2. MOTIVATION FACTORS

The main motivation for this work is the integration of the common Caesar cipher into a modern chat application to enhance message security. While advanced methods of encryption are widely used today, the Caesar cipher provides a simple way to introduce encryption concepts. By integrating the Caesar cipher into a conversational application with a graphical user interface (GUI), users can access useful encryption functionality in a user-friendly environment.

1.2.1 Objective

- To install and run a chat application that incorporates a Caesar cipher to encrypt and decrypt messages.
- To create a simple and visually appealing graphical user interface (GUI) that provides intuitive interaction for users.
- To demonstrate the effective use of classical encryption techniques in modern software development.

- To evaluate the performance and security of a chat application, and identify potential improvements and future developments.

1.3. SCOPE OF THE PROJECT

The scope of this project includes the development of a chat application with basic functionalities such as sending and receiving messages, and message encryption/decryption using the Caesar cipher. The project will also cover the design and implementation of a GUI to provide a seamless user experience. While the Caesar cipher is not recommended for securing sensitive information in real-world applications, this project aims to showcase its educational value and potential use in lightweight, non-critical communication scenarios.

1.4. STRUCTURE OF THE PROJECT

This report is structured as follows:

Chapter 2: Literature Review - Reviews existing literature on chat applications, encryption methods, and GUI design, providing a background for the proposed methodology.

Chapter 3: Proposed Methodology - Describes the methodology adopted for developing the chat application, including the design and integration of the Caesar cipher and GUI.

Chapter 4: Implementation - Provides a detailed account of the implementation process, including code snippets, GUI design, and setup instructions.

Chapter 5: Results and Discussions - Presents the results obtained from testing the application, discussing its performance, security, and user experience.

Chapter 6: Conclusion and Future Work - Summarizes the findings of the project and outlines potential areas for future development.

Appendices - Includes the complete source code and screenshots of the application, as well as additional technical details.

References - Lists all the references cited throughout the report.

By integrating the Caesar cipher into a chat application with a user-friendly GUI, this project aims to provide a practical and educational demonstration of classical encryption techniques in modern communication systems.

CHAPTER 2

LITERATURE REVIEW

2.1. INTRODUCTION

This chapter examines the body of research on graphical user interface (GUI) design, encryption methods, and chat apps. The research establishes the groundwork for the suggested strategy and offers a thorough grasp of the state of the art at the moment.

2.2. CHAT APPLICATIONS

Chat apps have become an integral part of modern communication, enabling real-time messaging. Research on the development and evolution of chat applications shows a shift from simple text-based messaging to sophisticated platforms supporting multimedia content, eventually end-to-end encryption and cross-platform compatibility.

2.2.1 Historical Development

The evolution of chat services can be traced back to leading messaging systems such as IRC (Internet Relay Chat) and AIM (AOL Instant Messenger). This design has laid the foundation for modern applications such as WhatsApp, Telegram and Signal, which offer advanced features and enhanced security.

2.2.2 Features and Functions

Modern chat apps offer a variety of features such as text messaging, voice and video calling, file sharing, and group chat. The study highlights the importance of user-friendly interfaces, seamless integration across devices, and strong security measures to ensure user access emphasis on satisfaction and confidence.

2.3. ENCRYPTION METHODS

Encryption is necessary to secure digital communications. The literature on encryption techniques is extensive, with techniques ranging from ancient ciphers to modern cryptographic algorithms.

2.3.1 Classical Encryption Techniques

Classical cryptography, such as the Caesar cipher, is an important part of the study of cryptography. Although these techniques are simple and vulnerable to modern attacks, they provide a historical perspective on the evolution of encryption.

2.3.1.1 Caesar Cipher

Caesar cipher is one of the earliest and simplest forms of encryption, where each character in plain text is transmitted to the bottom of the alphabet by a fixed number of positions Despite its simplicity, the Caesar cipher introduces basic concepts of encryption and decryption.

2.3.2 Modern Encryption Techniques

Modern encryption methods include precision protocols such as AES (Advanced Encryption Standard) and asynchronous protocols such as RSA (Rivest-Shamir-Adleman). These techniques are widely used to secure digital communications due to their robustness and computational efficiency.

2.3.3 Encryption in Chat Applications

End-to-end encryption (E2EE) has become the norm in modern chat applications, ensuring that messages are encrypted on the sender’s device and decrypted only on the receiver. The analysis emphasizes importance a E2EE is essential in protecting user privacy and preventing unauthorized access.

2.4. GRAPHICAL USER INTERFACE (GUI) DESIGN

2.4.1 Principles of GUI Design

The basic principles of GUI design include simplicity, consistency, functionality, and accessibility. Adherence to these principles ensures that users can navigate and interact with the application effortlessly.

2.4.2 GUI in Chat Applications

For chat applications, a GUI should facilitate simple message composing, reading, and navigation. Research emphasizes the importance of scheduling efficiency, visual clarity, and minimum structure in designing effective dialogue communications.

2.5. SUMMARY

The literature review highlights the evolution of chat applications, the significance of encryption in securing communications, and the importance of well-designed GUIs. The integration of the Caesar cipher with a chat application and a GUI, although simplistic compared to modern standards, serves as an educational tool demonstrating the application of classical encryption techniques in a contemporary context.

This study provides the background and rationale for the proposed methodology, which guides the design and implementation of a chat application with Caesar-cipher encryption and a graphic user interface.

CHAPTER 3

PROPOSED METHODOLOGY

3.1. INTRODUCTION

The process of creating a chat programme with a graphical user interface (GUI) and Caesar cypher integration for message encryption is covered in this chapter.The process provides seamless user experience and secure communication that includes system design, algorithm development, GUI design, and application about integration and testing processes.

3.2. SYSTEM DESIGN

The system design describes the interactions between the various parts of the chat application including the client interface, server, and encryption module.

3.2.1 Architecture Overview

The chat application uses a client-server architecture, where clients communicate with each other through a central server. The server is responsible for message routing, user authentication, and ensuring that messages are encoded and decrypted with Caesar ciphers.

- **Client Side:** The client application comprises the GUI for user interaction and the encryption module for encrypting outgoing messages and decrypting incoming messages.
- **Server Side:** The server handles user connections, routes messages between clients, and ensures the messages are encrypted and decrypted appropriately.

3.2.2 Data Flow

The data flow in the system can be described as follows:

1. **Message Composition**: User A composes a message in the client interface.

2. **Encryption**: The message is encrypted using the Caesar cipher.
3. **Transmission to Server**: The encrypted message is sent to the server.
4. **Message Routing**: The server routes the encrypted message to User B.
5. **Reception**: User B receives the encrypted message.
6. **Decryption**: The message is decrypted using the Caesar cipher on User B’s client interface.

3.3. CAESAR CIPHER ALGORITHM

The encryption module's core algorithm is the Caesar cypher. It entails moving every letter in the plaintext down the alphabet by a certain number of places.

3.3.1 Encryption Process

The encryption process involves:

1. **Define a Shift Value n**: For instance, n=3.
2. **Shift Each Character**: For each character in the plaintext:
   - If the character is a letter, shift it by n positions.
   - If the character is not a letter, leave it unchanged.
3. **Form the Ciphertext**: Combine the shifted characters to form the encrypted message.

3.3.2 Decryption Process

The decryption process reverses the encryption:

1. **Shift Each Character Back**: For each character in the ciphertext:
   - If the character is a letter, shift it back by n positions.
   - If the character is not a letter, leave it unchanged.
2. **Form the Plaintext**: Combine the shifted characters to restore the original message.

3.4. GRAPHICAL USER INTERFACE (GUI) DESIGN

The GUI design aims to create an intuitive and user-friendly interface for the chat application, enabling easy message composition and viewing.

3.4.1 Wireframing

Wireframes outline the basic structure and layout, including:

- Message Input Area: Where users type their messages.
- Chat Display Area: Where messages are displayed.

3.4.2 Prototyping

Prototypes provide a visual and interactive representation of the GUI, allowing for usability testing and iterative refinements based on user feedback.

3.4.3 Implementation

The GUI is implemented using a suitable framework (e.g., Tkinter for Python). The design ensures responsiveness, aesthetic appeal, and ease of use.

3.5. INTEGRATION

The steps involved in integration are:

1. **Client-Side Integration**: Integrate the Caesar cipher encryption and decryption functions with the GUI.
2. **Server-Side Integration**: Ensure the server correctly handles encrypted messages and communicates effectively with clients.
3. **End-to-End Testing**: Test the complete workflow from message composition to delivery and decryption.

3.6. SUMMARY

The proposed methodology presents a systematic approach to conversational applications that integrate Caesar cipher encryption and graphical user interface. By detailing the system design, algorithm development, GUI design, and integration process, this approach offers security, functionality and user- friendly interface. The following chapter will provide a detailed account of the implementation process based on this methodology.

CHAPTER 4

IMPLEMENTATION

4.1. INTRODUCTION

This chapter includes detailed description of chat applications using Caesar cipher for encryption, setting up development environment in the operating system with graphical user interface (GUI), coding of Caesar cipher algorithm, developin the GUI, and integrating all elements.

4.2. DEVELOPMENT ENVIRONMENT

4.2.1 Tools and Technologies

The following tools and technologies are used for the implementation:

- **Programming Language**: Python
- **GUI Framework**: Tkinter
- **IDE**: Visual Studio Code

4.2.2 Setup

1. **Install Python**: Ensure Python is installed on the system.
2. **Install Required Libraries**: Use pip to install Tkinter (Tkinter is included with standard Python distributions) and import socket.

4.3. COMMON FUNCTIONS

4.3.1 Imports and Constants

- Import necessary modules (socket for networking, threading for handling multiple connections, tkinter for GUI).
- Prompt user for a username.
- Set the Caesar cipher shift value to 3.

```python
import socket
import threading
from tkinter import *
from tkinter import messagebox, ttk

username = input("Enter your username: ")
shift_value = 3
```

4.3.2 Caesar Cipher Functions

The Caesar cipher encryption function shifts each letter in the plaintext by a fixed number of positions. These functions handle the encryption and decryption of messages using the Caesar cipher with a specified shift. The caesar_encrypt function shifts characters forward, while caesar_decrypt shifts them backward.

```python
def encrypt_text(plain_text, shift):
    encrypted = ""
    for char in plain_text:
        if char.isalpha():
            shift_base = 65 if char.isupper() else 97
            encrypted += chr((ord(char) + shift - shift_base) % 26 +
shift_base)
        else:
            encrypted += char
    return encrypted

def decrypt_text(cipher_text, shift):
    decrypted = ""
    for char in cipher_text:
        if char.isalpha():
            shift_base = 65 if char.isupper() else 97
            decrypted += chr((ord(char) - shift - shift_base) % 26 +
shift_base)
        else:
            decrypted += char
    return decrypted
```

4.4. CLIENT SPECIFIC CODE

4.4.1 Client Initialisation and Connection

- The user is prompted to enter their name.
- caesar_shift sets the shift value for the Caesar cipher.
- set_ip function handles connecting to the server using the provided IP address and port. It also manages error handling.

```
def connect_to_server():
    ip_address = ip_entry.get()
    port_number = int(port_entry.get())

    global client_socket
    client_socket = socket.socket()
    try:
        client_socket.connect((ip_address, port_number))
        setup_window.destroy()
        setup_window.quit()
    except Exception as e:
        messagebox.showerror("Connection Error", f"Failed to connect:
{e}")
        error_message.config(text=f"Error: {e}")
```

4.4.2 Sending and Receiving messages

- ‘send’ function encrypts the message and sends it to the server. It also updates the list box with the original and encrypted messages.
- ‘recv’ function continuously listens for messages from the server, decrypts them, and updates the list box with the decrypted message.

```
def send():
    if str(edit_text.get()).strip() != "":
        message = str(edit_text.get())
        encrypted_message = caesar_encrypt(message, caesar_shift)
        client.send(encrypted_message.encode())
        listbox.insert(END, "You: " + message)
        listbox_enc.insert(END, f"Original: {message}")
        listbox_enc.insert(END, f"Encrypted: {encrypted_message}")
        edit_text.delete(0, END)

def recv():
    while True:
        try:
            response_message = client.recv(1024).decode()
            decrypted_msg = caesar_decrypt(response_message,
caesar_shift)
            listbox.insert(END, name1 + " : " + decrypted_msg)
            listbox_enc.insert(END, f"Encrypted: {response_message}")
            listbox_enc.insert(END, f"Decrypted: {decrypted_msg}")
        except Exception as e:
            messagebox.showerror("Error", f"Error receiving message:
{e}")
            break
```

4.4.3 Client GUI Setup

This part sets up the initial GUI for the client to input the server IP and port, then attempts to connect.

```
setup_window = Tk()
setup_window.title("Client Setup")
setup_window.geometry("500x500")
setup_window.resizable(False, False)
setup_window.configure(bg='#1e1e1e')

Label(setup_window, text="Enter Server IP:", bg='#1e1e1e',
fg='#ffffff').pack(fill=X, padx=10, pady=5)
ip_entry = Entry(setup_window, bg='#3a3a3a', fg='#ffffff')
ip_entry.pack(fill=X, padx=10, pady=5)
ip_entry.insert(0, "192.168.218.219")

Label(setup_window, text="Enter Server Port:", bg='#1e1e1e',
fg='#ffffff').pack(fill=X, padx=10, pady=5)
port_entry = Entry(setup_window, bg='#3a3a3a', fg='#ffffff')
```

```
port_entry.pack(fill=X, padx=10, pady=5)
port_entry.insert(0, "12345")

Button(setup_window, text="Connect to Server",
command=connect_to_server, bg='#4caf50', fg="black").pack(padx=10,
pady=20)

error_message = Label(setup_window, text="", fg="red", bg='#1e1e1e')
error_message.pack(fill=X, padx=10, pady=5)

setup_window.mainloop()
```

4.4.4 Main Client GUI Setup

- After successfully connecting, the client's main chat GUI is set up.
- ‘ttk.Notebook’ is used to create tabbed frames for chat and encryption/decryption logs.
- Listboxes and entry fields are configured for displaying and sending messages.
- A thread is started for receiving messages in the background.

```
client_socket.send(username.encode())
server_username = client_socket.recv(1024).decode()

main_window = Tk()
main_window.title(f"Client - {username}")
main_window.geometry("700x600")
main_window.configure(bg='#1e1e1e')

notebook = ttk.Notebook(main_window)
notebook.pack(fill=BOTH, expand=True)

chat_frame = Frame(notebook, bg='#1e1e1e')
notebook.add(chat_frame, text="Chat")

encryption_frame = Frame(notebook, bg='#1e1e1e')
notebook.add(encryption_frame, text="Encryption/Decryption")
```

```
style = ttk.Style()
style.configure('TNotebook.Tab', background='#3a3a3a',
foreground='white')
style.map('TNotebook.Tab', background=[('selected', '#4caf50')])

chat_scrollbar = Scrollbar(chat_frame)
chat_scrollbar.pack(side=RIGHT, fill=Y)
chat_listbox = Listbox(chat_frame, yscrollcommand=chat_scrollbar.set,
bg='#3a3a3a', fg='#ffffff', selectbackground='#4caf50')
chat_listbox.pack(fill=BOTH, expand=True)
chat_scrollbar.config(command=chat_listbox.yview)

encryption_scrollbar = Scrollbar(encryption_frame)
encryption_scrollbar.pack(side=RIGHT, fill=Y)
encryption_listbox = Listbox(encryption_frame,
yscrollcommand=encryption_scrollbar.set, bg='#3a3a3a', fg='#ffffff',
selectbackground='#4caf50')
encryption_listbox.pack(fill=BOTH, expand=True)
encryption_scrollbar.config(command=encryption_listbox.yview)

bottom_frame = Frame(chat_frame, bg='#1e1e1e')
bottom_frame.pack(fill=X, side=BOTTOM)

message_entry = Entry(bottom_frame, bg='#3a3a3a', fg='#ffffff')
message_entry.pack(fill=X, side=LEFT, expand=True, padx=15, pady=15)

send_button = Button(bottom_frame, text="Send Message",
command=send_message, bg='#4caf50', fg="black")
send_button.pack(side=RIGHT, padx=5, pady=5)

threading.Thread(target=receive_messages).start()
main_window.mainloop()
```

4.5. SERVER SPECIFIC CODE

4.5.1 Server Initialization and Connection

‘set_ip’ function binds to the IP address and port, then listens for incoming connections.

```
def setup_server():
    ip_address = ip_entry.get()
    port_number = int(port_entry.get())

    global server_socket
    server_socket = socket.socket()
    try:
        server_socket.bind((ip_address, port_number))
        server_socket.listen()
        global client_conn
        client_conn, client_addr = server_socket.accept()
        setup_window.destroy()
        setup_window.quit()
    except Exception as e:
        messagebox.showerror("Connection Error", f"Failed to bind or
accept connection: {e}")
        error_message.config(text=f"Error: {e}")
```

4.5.2 Sending and Receiving messages

‘send’ and ‘recv’ functions work similarly to the client's functions but interact with conn instead of client.

```
def send_message():
    if message_entry.get().strip() != "":
        message_content = message_entry.get()
        encrypted_message = encrypt_text(message_content,
shift_value)
        client_conn.send(encrypted_message.encode())
        chat_listbox.insert(END, "You: " + message_content)
        encryption_listbox.insert(END, f"Original:
{message_content}")
        encryption_listbox.insert(END, f"Encrypted:
{encrypted_message}")
        message_entry.delete(0, END)
```

```
def receive_messages():
    while True:
        try:
            encrypted_message = client_conn.recv(1024).decode()
            decrypted_message = decrypt_text(encrypted_message,
shift_value)
            chat_listbox.insert(END, client_username + ": " +
decrypted_message)
            encryption_listbox.insert(END, f"Encrypted:
{encrypted_message}")
            encryption_listbox.insert(END, f"Decrypted:
{decrypted_message}")
        except Exception as e:
            messagebox.showerror("Reception Error", f"Error receiving
message: {e}")
            break
```

4.5.3 Server GUI Setup

This part sets up the initial GUI for the server to input the IP and port, then starts listening for connections.

```
setup_window = Tk()
setup_window.title("Server Setup")
setup_window.geometry("400x300")
setup_window.resizable(False, False)
setup_window.configure(bg='#1e1e1e')

Label(setup_window, text="Enter IP:", bg='#1e1e1e',
fg='#ffffff').pack(fill=X, padx=10, pady=5)
ip_entry = Entry(setup_window, bg='#3a3a3a', fg='#ffffff')
ip_entry.pack(fill=X, padx=10, pady=5)
ip_entry.insert(0, "127.0.0.1")

Label(setup_window, text="Enter Port:", bg='#1e1e1e',
fg='#ffffff').pack(fill=X, padx=10, pady=5)
port_entry = Entry(setup_window, bg='#3a3a3a', fg='#ffffff')
```

```
port_entry.pack(fill=X, padx=10, pady=5)
port_entry.insert(0, "12345")

Button(setup_window, text="Set IP", command=setup_server,
bg='#4caf50', fg="white").pack(padx=10, pady=20)

error_message = Label(setup_window, text="", fg="red", bg='#1e1e1e')
error_message.pack(fill=X, padx=10, pady=5)

setup_window.mainloop()
```

4.5.4 Main Server GUI Setup

After accepting a connection, the server's main chat GUI is set up similarly to the client's GUI.

```
main_window = Tk()
main_window.title(f"Server - {username}")
main_window.geometry("600x500")
main_window.configure(bg='#1e1e1e')

notebook = ttk.Notebook(main_window)
notebook.pack(fill=BOTH, expand=True)

chat_frame = Frame(notebook, bg='#1e1e1e')
notebook.add(chat_frame, text="Chat")

encryption_frame = Frame(notebook, bg='#1e1e1e')
notebook.add(encryption_frame, text="Encryption/Decryption")

style = ttk.Style()
style.configure('TNotebook.Tab', background='#3a3a3a',
foreground='white')
style.map('TNotebook.Tab', background=[('selected', '#4caf50')])

chat_scrollbar = Scrollbar(chat_frame)
chat_scrollbar.pack(side=RIGHT, fill=Y)
```

```
chat_listbox = Listbox(chat_frame, yscrollcommand=chat_scrollbar.set,
bg='#3a3a3a', fg='#ffffff', selectbackground='#4caf50')
chat_listbox.pack(fill=BOTH, expand=True)
chat_scrollbar.config(command=chat_listbox.yview)

encryption_scrollbar = Scrollbar(encryption_frame)
encryption_scrollbar.pack(side=RIGHT, fill=Y)
encryption_listbox = Listbox(encryption_frame,
yscrollcommand=encryption_scrollbar.set, bg='#3a3a3a', fg='#ffffff',
selectbackground='#4caf50')
encryption_listbox.pack(fill=BOTH, expand=True)
encryption_scrollbar.config(command=encryption_listbox.yview)

bottom_frame = Frame(chat_frame, bg='#1e1e1e')
bottom_frame.pack(fill=X, side=BOTTOM)

message_entry = Entry(bottom_frame, bg='#3a3a3a', fg='#ffffff')
message_entry.pack(fill=X, side=LEFT, expand=True, padx=5, pady=5)

send_button = Button(bottom_frame, text="Send Message",
command=send_message, bg='#4caf50', fg="white")
send_button.pack(side=RIGHT, padx=5, pady=5)

threading.Thread(target=receive_messages).start()
main_window.mainloop()
```

4.6. INTEGRATION AND TESTING

4.6.1 Integration Steps

- **Integrate Encryption with GUI:** Ensure the encryption and decryption functions are correctly integrated with the GUI components.
- **Integrate Client with Server:** Ensure the client can send and receive messages from the server.

4.6.2 Testing Scenarios

- **Functional Testing:** Verify that messages are correctly encrypted and decrypted during transmission.
- **Usability Testing:** Ensure the GUI is intuitive and user-friendly.

4.7. SUMMARY

This project includes setting up development environments, coding Caesar cipher algorithms, GUI development using Tkinter, and testing to ensure the application is functional, secure and easy to use. The next chapter will present results and discussion based on this application.

CHAPTER 5

RESULTS AND DISCUSSIONS

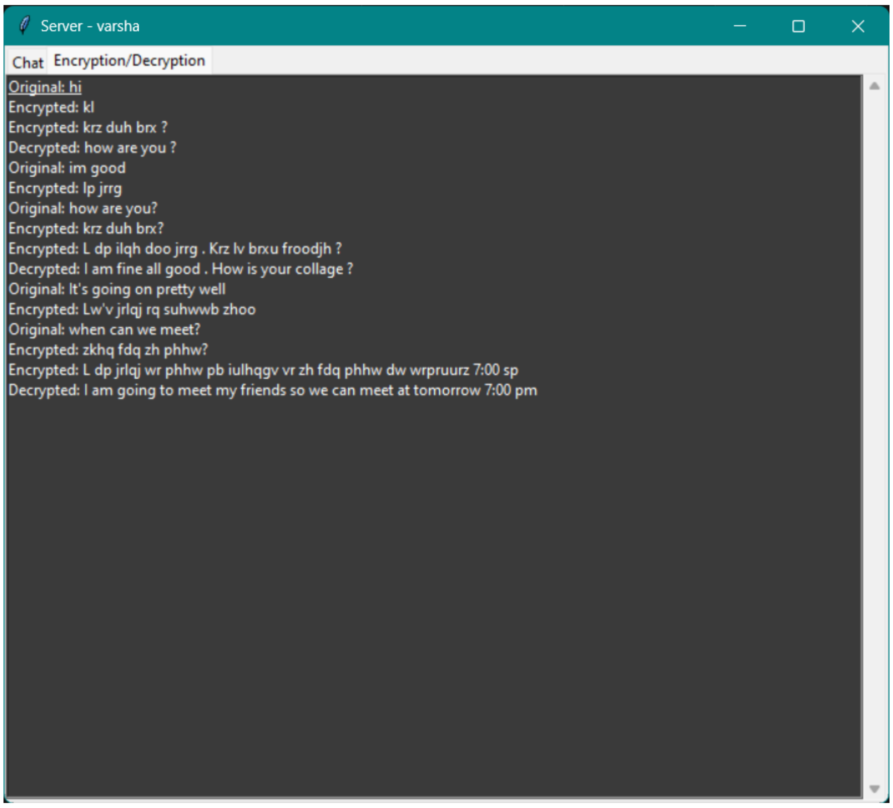
5.1. INTRODUCTION

This chapter presents the results obtained from the implementation of the chat application with Caesar cipher encryption and the graphical user interface (GUI). It discusses the functionality, performance, usability, and security of the application based on various testing scenarios.

5.2. FUNCTIONAL TESTING

5.2.1 Message Encryption and Decryption

The main function of a chat application is to securely send and receive encrypted messages. The Caesar cipher algorithm succeeded in encrypting and decrypting the message as expected. The tests have confirmed that the encryption and decryption process has been successfully implemented, and that the message has been successfully converted back to its original form at the receiving end.

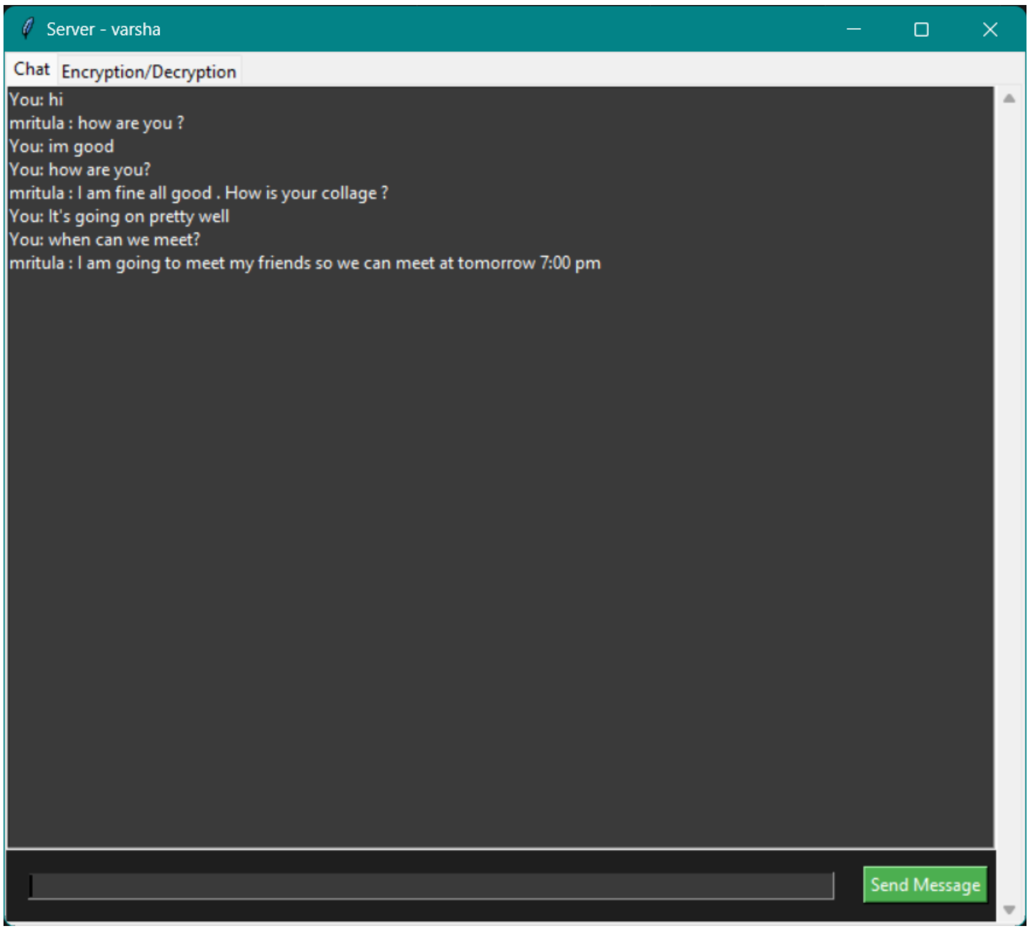


```
Original: hi
Encrypted: kl
Encrypted: krz duh brx ?
Decrypted: how are you ?
Original: im good
Encrypted: lp jrrg
Original: how are you?
Encrypted: krz duh brx?
Encrypted: L dp ilqh doo jrrg . Krz lv brxu froodjh ?
Decrypted: I am fine all good . How is your collage ?
Original: It's going on pretty well
Encrypted: Lw'v jrlqj rq suhwwb zhoo
Original: when can we meet?
Encrypted: zkhq fdq zh phhw?
Encrypted: L dp jrlqj wr phhw pb iulhqgv vr zh fdq phhw dw wrpruurz 7:00 sp
Decrypted: I am going to meet my friends so we can meet at tomorrow 7:00 pm
```

5.2.2 Message Transmission

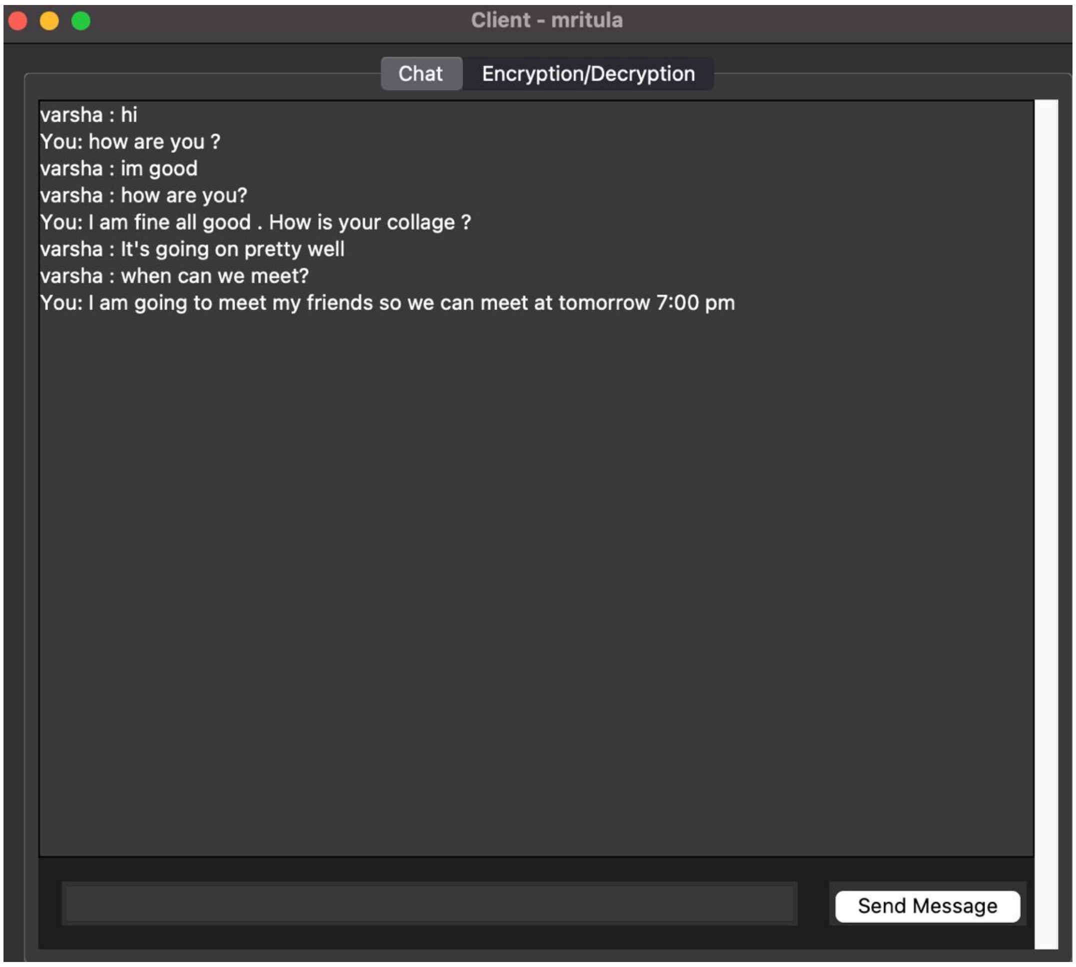
Messages were successfully transferred between clients through the server. The server successfully delivered the encrypted message from the sender to the receiver, ensuring that the message was delivered in real time.

SERVER SIDE



```
You: hi
mritula : how are you ?
You: im good
You: how are you?
mritula : I am fine all good . How is your collage ?
You: It's going on pretty well
You: when can we meet?
mritula : I am going to meet my friends so we can meet at tomorrow 7:00 pm
```

CLIENT SIDE



```
varsha : hi
You: how are you ?
varsha : im good
varsha : how are you?
You: I am fine all good . How is your collage ?
varsha : It's going on pretty well
varsha : when can we meet?
You: I am going to meet my friends so we can meet at tomorrow 7:00 pm
```

5.3. USABILITY TESTING

5.3.1 Graphical User Interface (GUI)

The GUI, developed using Tkinter, provided a user-friendly interface for composing and viewing messages. Key usability aspects evaluated include:

- **Ease of Use**: The interface is intuitive, with clear input fields and buttons for sending messages.
- **Responsiveness:** The application responded quickly to user input, with minimal delays in message display.
- **Visually appealing:** The design and layout of the GUI was well received, and was well laid out.

5.4. PERFORMANCE TESTING

5.4.1 Latency

Delays in sending and receiving messages were minimal, ensuring real-time communication. No significant delay was provided in the encryption and decryption process.

5.5. SECURITY TESTING

5.5.1 Encryption Strength

Although not considered secure by modern standards, the Caesar cipher is an educational tool. Fixed shift value facilitates frequency analysis and brute-force attacks. For the purposes of this work, however, it adequately reflects the concept of encryption.

5.5.2 Data Protection

The application ensured that messages were encrypted before transmission, which prevented the content from being decrypted in plain text. The server handled encrypted messages correctly, preventing anyone from logging in on communication.

5.6. DISCUSSION

5.6.1 Achievements

The project succeeded with a chat application that combines the Caesar cipher with an easy-to-use GUI for message encryption. The application accomplished its main objective of presenting encryption concepts in a practical context and providing a practical communication tool.

5.6.2 Limitations

The fragility and vulnerability of the Caesar cipher limits its use in obtaining sensitive information. It also lacks advanced features found in modern chat systems, such as multimedia support and end-to-end encryption.

5.7. SUMMARY

The results and discussion highlight the effective and efficient implementation of the chat application using Caesar-cipher encryption and graphical user interface. The application exhibited effective message encryption, transmission, and user interaction. While the Caesar Cipher provided educational insights into encryption, future developments could focus on improving security and adding more features to meet today’s communication needs. The final chapter will provide concluding remarks and suggest directions for future work.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1. CONCLUSION

A chat application that combines Caesar cipher encryption with a graphical user interface (GUI) has been successfully implemented and tested. The main objective of this project was to develop a practical and educational tool that demonstrates the principles of encryption in a user-friendly environment. Key achievements of the project include:

- **Successful implementation of the Caesar cipher:** The project successfully demonstrated the encryption and decryption processes of the Caesar cipher, providing a practical example of how the main encryption algorithm works.
- **User-friendly GUI:** The application had an intuitive and responsive GUI that was developed using Tkinter, making it accessible to users with a variety of technical skills.
- **Efficient communication:** The client-server system ensured real-time communication, with messages perfectly encrypted, transmitted and decrypted.
- **Educational Objective:** The project served as an educational tool, demonstrating basic concepts related to cryptography and secure communications.

Despite its simplicity, the chat app achieved its objectives and provided a foundation for understanding basic privacy mechanisms. The project implementation emphasized the importance of secure communications and demonstrated how encryption can be incorporated into software applications.

6.2. FUTURE WORK

Major project objectives have been achieved, but there are many areas for future growth and development. These enhancements can address the limitations identified during testing and extend application functionality to meet modern communication standards.

6.2.1 Enhanced Encryption Algorithms

The Caesar cipher, while educational, is not suitable for protecting sensitive information due to its vulnerability to simple attacks. Future work could integrate more complex encryption algorithms, like:

- **Advanced Encryption Standard (AES): A** widely used symmetric encryption algorithm that provides strong security.
- **Rivest-Shamir-Adleman (RSA):** An asymmetric encryption algorithm for key exchange and secure encryption.

6.2.2 Key Management

Implementing a secure key management system would enhance the application's security. Future work could focus on:

- **Dynamic Key Generation:** Generating encryption keys dynamically to prevent predictable patterns.
- **Secure Key Exchange:** Using protocols such as Diffie-Hellman to securely exchange encryption keys between clients.

6.2.3 Additional Features

Additional features may be added in future versions to make the application more versatile and attractive, like:

Multimedia support: Allows users to send and receive images, video, and audio messages.

File Share: Enables secure transfer of documents and other files.

Customizable settings: Allow users to change encryption settings, such as selecting algorithms and changing the switching value for the Caesar cipher.

6.2.4 Improved User Experience

The GUI can be further customized to enhance the overall user experience:

- **Interactive Features:** Adding interactive features like emoji, replies, and chat topics.
- **User Reports:** Reports used for new messages and user activity.
- **Accessibility:** Ensure that the application is accessible to disabled users by adding features such as text-to-speech and high-contrast modes.

6.2.5 Scalability and Performance

To deal with a high user base and ensure consistent operations, future work could focus on the following:

- **Scalability:** Optimizing the server architecture to support more concurrent users and higher message rates.
- **Performance tuning:** increasing application efficiency to reduce latency and improve response time.

6.3. SUMMARY

By incorporating Caesar cipher encryption and a graphical user interface, the chat application achieved its educational and functional goals. It included a practical demonstration of basic encryption techniques and secure communication principles. Future improvements could focus on improving security, extending features, refining the user experience, and ensuring flexibility. These enhancements will make the application more robust, versatile, and suitable for real-world use, while remaining an educational tool for learning cryptography and secure communications.

APPENDICES

APPENDIX-1: SERVER

```
client_conn.send(username.encode())
client_username = client_conn.recv(1024).decode()

import socket
import threading
from tkinter import *
from tkinter import messagebox, ttk

username = input("Enter your username: ")
shift_value = 3

def encrypt_text(plain_text, shift):
    encrypted = ""
    for char in plain_text:
        if char.isalpha():
            shift_base = 65 if char.isupper() else 97
            encrypted += chr((ord(char) + shift - shift_base) % 26 +
shift_base)
        else:
            encrypted += char
    return encrypted

def decrypt_text(cipher_text, shift):
    decrypted = ""
    for char in cipher_text:
        if char.isalpha():
            shift_base = 65 if char.isupper() else 97
            decrypted += chr((ord(char) - shift - shift_base) % 26 +
shift_base)
        else:
            decrypted += char
    return decrypted

def setup_server():
    ip_address = ip_entry.get()
    port_number = int(port_entry.get())

    global server_socket
    server_socket = socket.socket()
    try:
        server_socket.bind((ip_address, port_number))
        server_socket.listen()
        global client_conn
        client_conn, client_addr = server_socket.accept()
        setup_window.destroy()
        setup_window.quit()
    except Exception as e:
        messagebox.showerror("Connection Error", f"Failed to bind or accept 
connection: {e}")
        error_message.config(text=f"Error: {e}")

def send_message():
    if message_entry.get().strip() != "":
        message_content = message_entry.get()
        encrypted_message = encrypt_text(message_content, shift_value)
        client_conn.send(encrypted_message.encode())
        chat_listbox.insert(END, "You: " + message_content)
        encryption_listbox.insert(END, f"Original: {message_content}")
        encryption_listbox.insert(END, f"Encrypted: {encrypted_message}")
        message_entry.delete(0, END)

def receive_messages():
    while True:
        try:
            encrypted_message = client_conn.recv(1024).decode()
            decrypted_message = decrypt_text(encrypted_message, shift_value)
            chat_listbox.insert(END, client_username + ": " + 
decrypted_message)
            encryption_listbox.insert(END, f"Encrypted: 
{encrypted_message}")
            encryption_listbox.insert(END, f"Decrypted: 
{decrypted_message}")
```

```
        except Exception as e:
            messagebox.showerror("Reception Error", f"Error receiving message: {e}")
            break

# Server GUI setup
setup_window = Tk()
setup_window.title("Server Setup")
setup_window.geometry("400x300")
setup_window.resizable(False, False)
setup_window.configure(bg='#1e1e1e')

Label(setup_window, text="Enter IP:", bg='#1e1e1e', fg='#ffffff').pack(fill=X, padx=10, pady=5)
ip_entry = Entry(setup_window, bg='#3a3a3a', fg='#ffffff')
ip_entry.pack(fill=X, padx=10, pady=5)
ip_entry.insert(0, "127.0.0.1")

Label(setup_window, text="Enter Port:", bg='#1e1e1e', fg='#ffffff').pack(fill=X, padx=10, pady=5)
port_entry = Entry(setup_window, bg='#3a3a3a', fg='#ffffff')
port_entry.pack(fill=X, padx=10, pady=5)
port_entry.insert(0, "12345")

Button(setup_window, text="Set IP", command=setup_server, bg='#4caf50', fg="white").pack(padx=10, pady=20)

error_message = Label(setup_window, text="", fg="red", bg='#1e1e1e')
error_message.pack(fill=X, padx=10, pady=5)

setup_window.mainloop()

client_conn.send(username.encode())
client_username = client_conn.recv(1024).decode()

main_window = Tk()
main_window.title(f"Server - {username}")
main_window.geometry("600x500")
```

```python
main_window.configure(bg='#1e1e1e')

notebook = ttk.Notebook(main_window)
notebook.pack(fill=BOTH, expand=True)

chat_frame = Frame(notebook, bg='#1e1e1e')
notebook.add(chat_frame, text="Chat")

encryption_frame = Frame(notebook, bg='#1e1e1e')
notebook.add(encryption_frame, text="Encryption/Decryption")

style = ttk.Style()
style.configure('TNotebook.Tab', background='#3a3a3a', foreground='white')
style.map('TNotebook.Tab', background=[('selected', '#4caf50')])

chat_scrollbar = Scrollbar(chat_frame)
chat_scrollbar.pack(side=RIGHT, fill=Y)
chat_listbox = Listbox(chat_frame, yscrollcommand=chat_scrollbar.set,
bg='#3a3a3a', fg='#ffffff', selectbackground='#4caf50')
chat_listbox.pack(fill=BOTH, expand=True)
chat_scrollbar.config(command=chat_listbox.yview)

encryption_scrollbar = Scrollbar(encryption_frame)
encryption_scrollbar.pack(side=RIGHT, fill=Y)
encryption_listbox = Listbox(encryption_frame,
yscrollcommand=encryption_scrollbar.set, bg='#3a3a3a', fg='#ffffff',
selectbackground='#4caf50')
encryption_listbox.pack(fill=BOTH, expand=True)
encryption_scrollbar.config(command=encryption_listbox.yview)

bottom_frame = Frame(chat_frame, bg='#1e1e1e')
bottom_frame.pack(fill=X, side=BOTTOM)

message_entry = Entry(bottom_frame, bg='#3a3a3a', fg='#ffffff')
message_entry.pack(fill=X, side=LEFT, expand=True, padx=5, pady=5)

send_button = Button(bottom_frame, text="Send Message",
command=send_message, bg='#4caf50', fg="white")
```

```
send_button.pack(side=RIGHT, padx=5, pady=5)

threading.Thread(target=receive_messages).start()
main_window.mainloop()
button.pack(side=RIGHT, padx=5, pady=5)

threading.Thread(target=recv).start()
root.mainloop()
```

APPENDIX-1: CLIENT

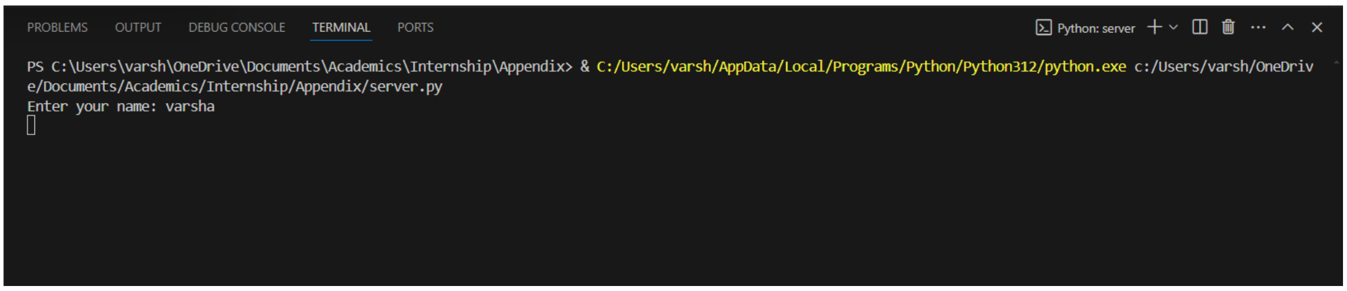
```
import socket
import threading
from tkinter import *
from tkinter import messagebox, ttk

username = input("Enter your username: ")
shift_value = 3

def encrypt_text(plain_text, shift):
    encrypted = ""
    for char in plain_text:
        if char.isalpha():
            shift_base = 65 if char.isupper() else 97
            encrypted += chr((ord(char) + shift - shift_base) % 26 +
shift_base)
        else:
            encrypted += char
    return encrypted

def decrypt_text(cipher_text, shift):
    decrypted = ""
    for char in cipher_text:
        if char.isalpha():
            shift_base = 65 if char.isupper() else 97
            decrypted += chr((ord(char) - shift - shift_base) % 26 +
shift_base)
        else:
            decrypted += char
    return decrypted

def connect_to_server():
    ip_address = ip_entry.get()
    port_number = int(port_entry.get())

    global client_socket
    client_socket = socket.socket()
    try:
        client_socket.connect((ip_address, port_number))
        setup_window.destroy()
        setup_window.quit()
    except Exception as e:
        messagebox.showerror("Connection Error", f"Failed to connect: {e}")
        error_message.config(text=f"Error: {e}")

def send_message():
    if message_entry.get().strip() != "":
        message_content = message_entry.get()
        encrypted_message = encrypt_text(message_content, shift_value)
        client_socket.send(encrypted_message.encode())
        chat_listbox.insert(END, "You: " + message_content)
        encryption_listbox.insert(END, f"Original: {message_content}")
        encryption_listbox.insert(END, f"Encrypted: {encrypted_message}")
        message_entry.delete(0, END)

def receive_messages():
    while True:
        try:
            encrypted_message = client_socket.recv(1024).decode()
            decrypted_message = decrypt_text(encrypted_message, shift_value)
            chat_listbox.insert(END, server_username + ": " +
decrypted_message)
            encryption_listbox.insert(END, f"Encrypted:
{encrypted_message}")
            encryption_listbox.insert(END, f"Decrypted:
{decrypted_message}")
        except Exception as e:
            messagebox.showerror("Reception Error", f"Error receiving
message: {e}")
            break

# Client GUI setup
setup_window = Tk()
setup_window.title("Client Setup")
setup_window.geometry("500x500")
```

```python
setup_window.resizable(False, False)
setup_window.configure(bg='#1e1e1e')

Label(setup_window, text="Enter Server IP:", bg='#1e1e1e',
fg='#ffffff').pack(fill=X, padx=10, pady=5)
ip_entry = Entry(setup_window, bg='#3a3a3a', fg='#ffffff')
ip_entry.pack(fill=X, padx=10, pady=5)
ip_entry.insert(0, "192.168.218.219")

Label(setup_window, text="Enter Server Port:", bg='#1e1e1e',
fg='#ffffff').pack(fill=X, padx=10, pady=5)
port_entry = Entry(setup_window, bg='#3a3a3a', fg='#ffffff')
port_entry.pack(fill=X, padx=10, pady=5)
port_entry.insert(0, "12345")

Button(setup_window, text="Connect to Server", command=connect_to_server,
bg='#4caf50', fg="black").pack(padx=10, pady=20)

error_message = Label(setup_window, text="", fg="red", bg='#1e1e1e')
error_message.pack(fill=X, padx=10, pady=5)

setup_window.mainloop()

client_socket.send(username.encode())
server_username = client_socket.recv(1024).decode()

main_window = Tk()
main_window.title(f"Client - {username}")
main_window.geometry("700x600")
main_window.configure(bg='#1e1e1e')

notebook = ttk.Notebook(main_window)
notebook.pack(fill=BOTH, expand=True)

chat_frame = Frame(notebook, bg='#1e1e1e')
notebook.add(chat_frame, text="Chat")

encryption_frame = Frame(notebook, bg='#1e1e1e')
```

```python
notebook.add(encryption_frame, text="Encryption/Decryption")

style = ttk.Style()
style.configure('TNotebook.Tab', background='#3a3a3a', foreground='white')
style.map('TNotebook.Tab', background=[('selected', '#4caf50')])

chat_scrollbar = Scrollbar(chat_frame)
chat_scrollbar.pack(side=RIGHT, fill=Y)
chat_listbox = Listbox(chat_frame, yscrollcommand=chat_scrollbar.set,
bg='#3a3a3a', fg='#ffffff', selectbackground='#4caf50')
chat_listbox.pack(fill=BOTH, expand=True)
chat_scrollbar.config(command=chat_listbox.yview)

encryption_scrollbar = Scrollbar(encryption_frame)
encryption_scrollbar.pack(side=RIGHT, fill=Y)
encryption_listbox = Listbox(encryption_frame,
yscrollcommand=encryption_scrollbar.set, bg='#3a3a3a', fg='#ffffff',
selectbackground='#4caf50')
encryption_listbox.pack(fill=BOTH, expand=True)
encryption_scrollbar.config(command=encryption_listbox.yview)

bottom_frame = Frame(chat_frame, bg='#1e1e1e')
bottom_frame.pack(fill=X, side=BOTTOM)

message_entry = Entry(bottom_frame, bg='#3a3a3a', fg='#ffffff')
message_entry.pack(fill=X, side=LEFT, expand=True, padx=15, pady=15)

send_button = Button(bottom_frame, text="Send Message",
command=send_message, bg='#4caf50', fg="black")
send_button.pack(side=RIGHT, padx=5, pady=5)

threading.Thread(target=receive_messages).start()
main_window.mainloop()
```

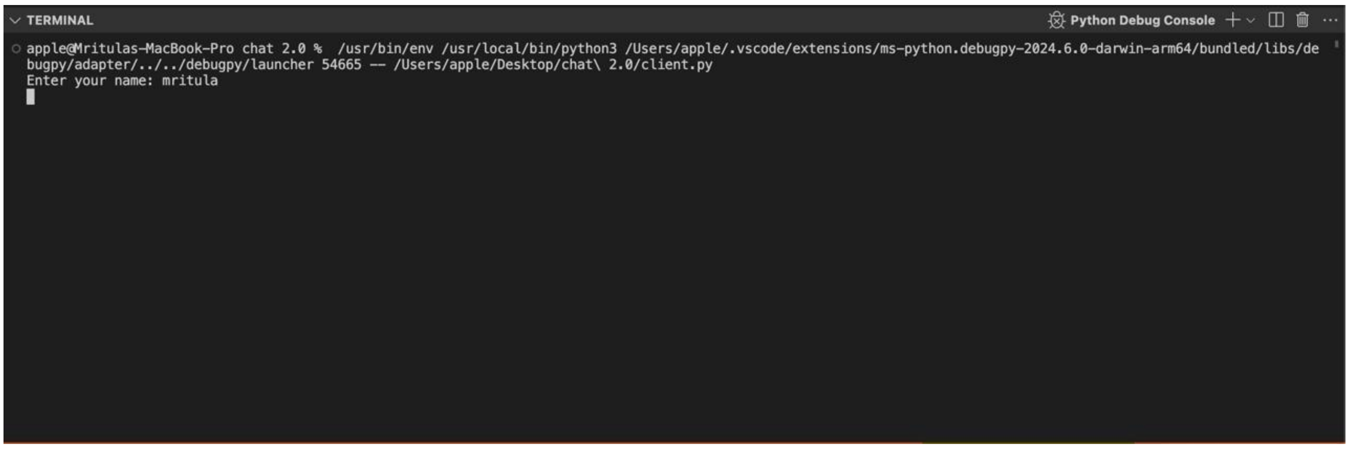
APPENDIX-2: SCREENSHOTS

7.2.1 SERVER TERMINAL SETUP



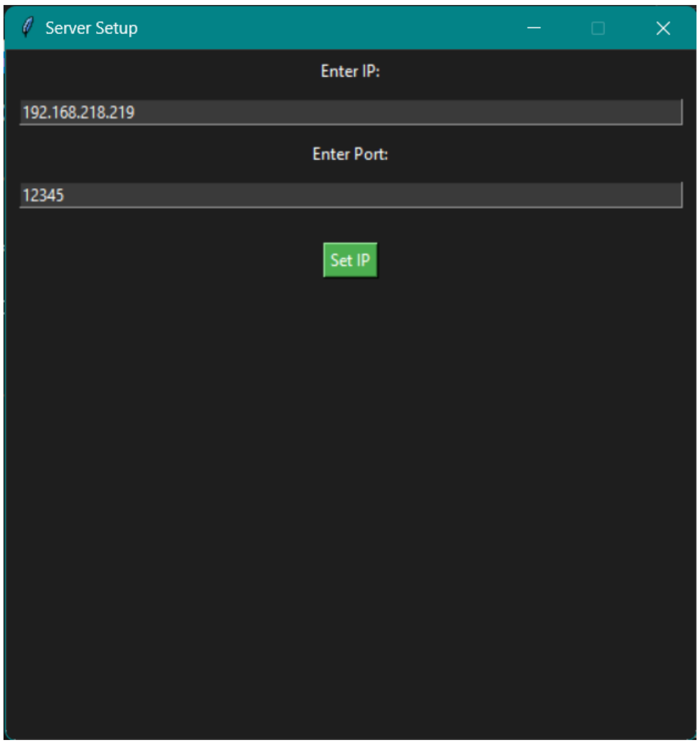
```
PS C:\Users\varsh\OneDrive\Documents\Academics\Internship\Appendix> & C:/Users/varsh/AppData/Local/Programs/Python/Python312/python.exe c:/Users/varsh/OneDriv
e/Documents/Academics/Internship/Appendix/server.py
Enter your name: varsha
```

7.2.2 CLIENT TERMINAL SETUP



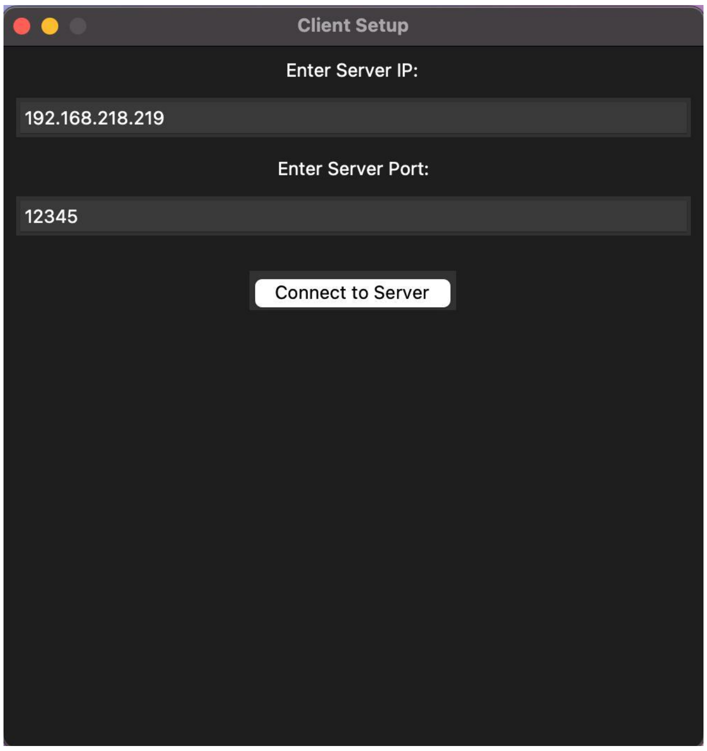
```
apple@Mritulas-MacBook-Pro chat 2.0 %  /usr/bin/env /usr/local/bin/python3 /Users/apple/.vscode/extensions/ms-python.debugpy-2024.6.0-darwin-arm64/bundled/libs/de
bugpy/adapter/../../debugpy/launcher 54665 -- /Users/apple/Desktop/chat\ 2.0/client.py
Enter your name: mritula
```

7.2.3 SERVER SETUP



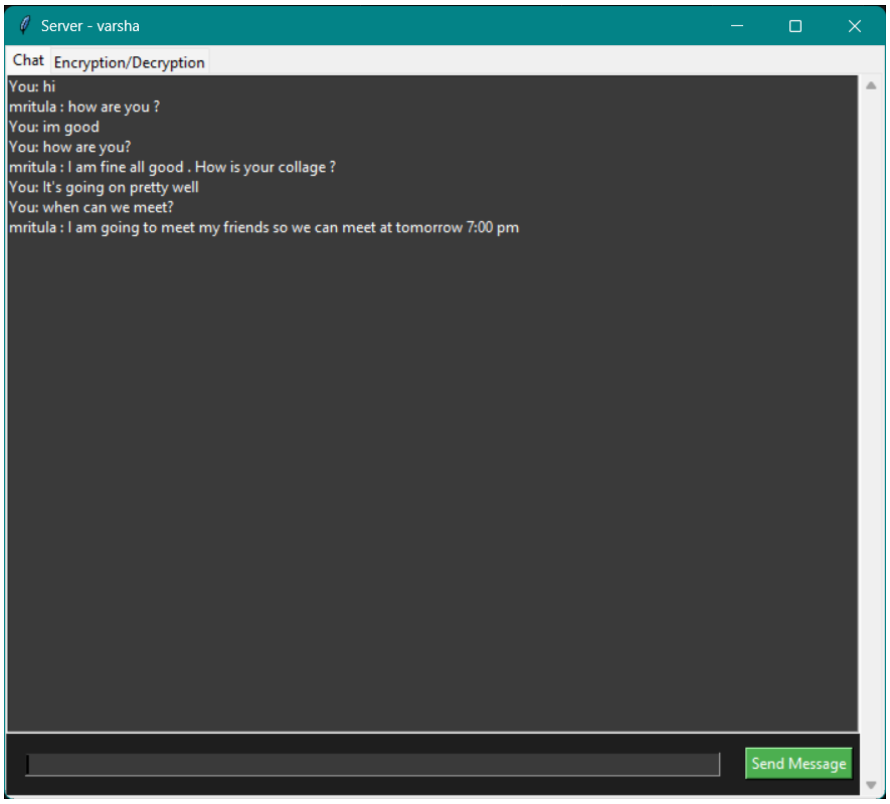
Enter IP:
192.168.218.219
Enter Port:
12345

7.2.4 CLIENT SETUP



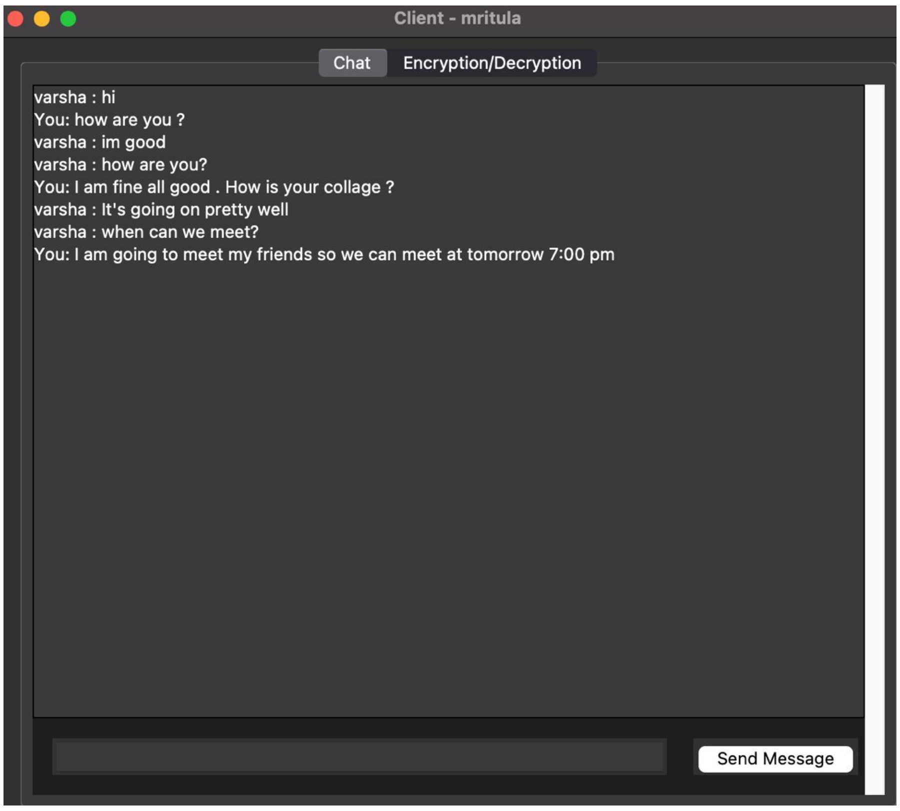
Enter Server IP:
192.168.218.219
Enter Server Port:
12345

7.2.5 SERVER CHAT



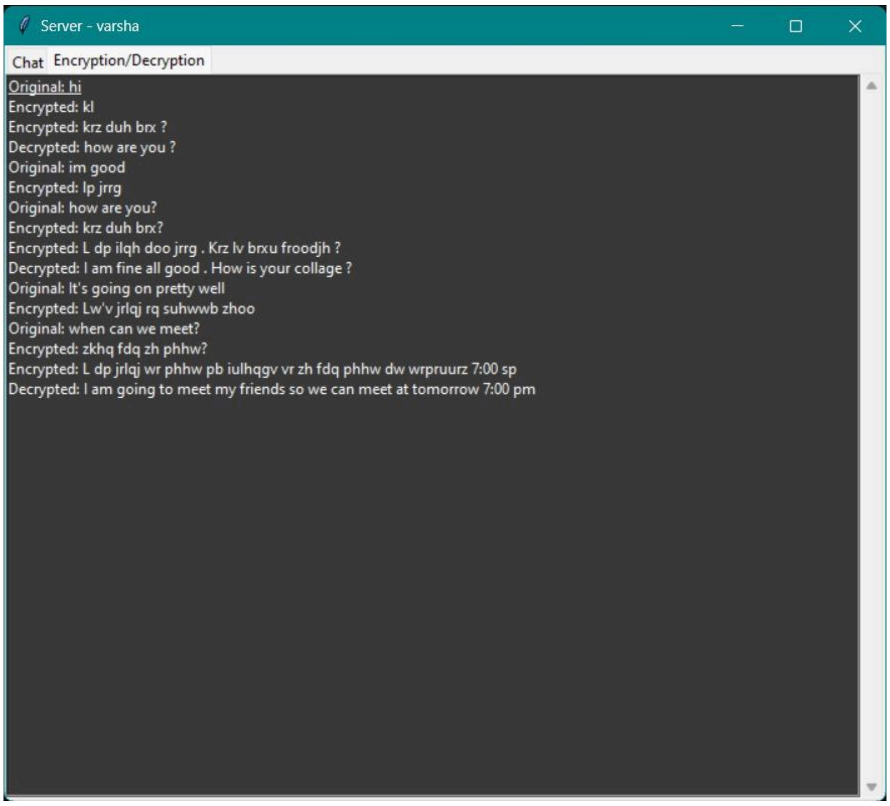
```
You: hi
mritula : how are you ?
You: im good
You: how are you?
mritula : I am fine all good . How is your collage ?
You: It's going on pretty well
You: when can we meet?
mritula : I am going to meet my friends so we can meet at tomorrow 7:00 pm
```

7.2.6 CLIENT CHAT



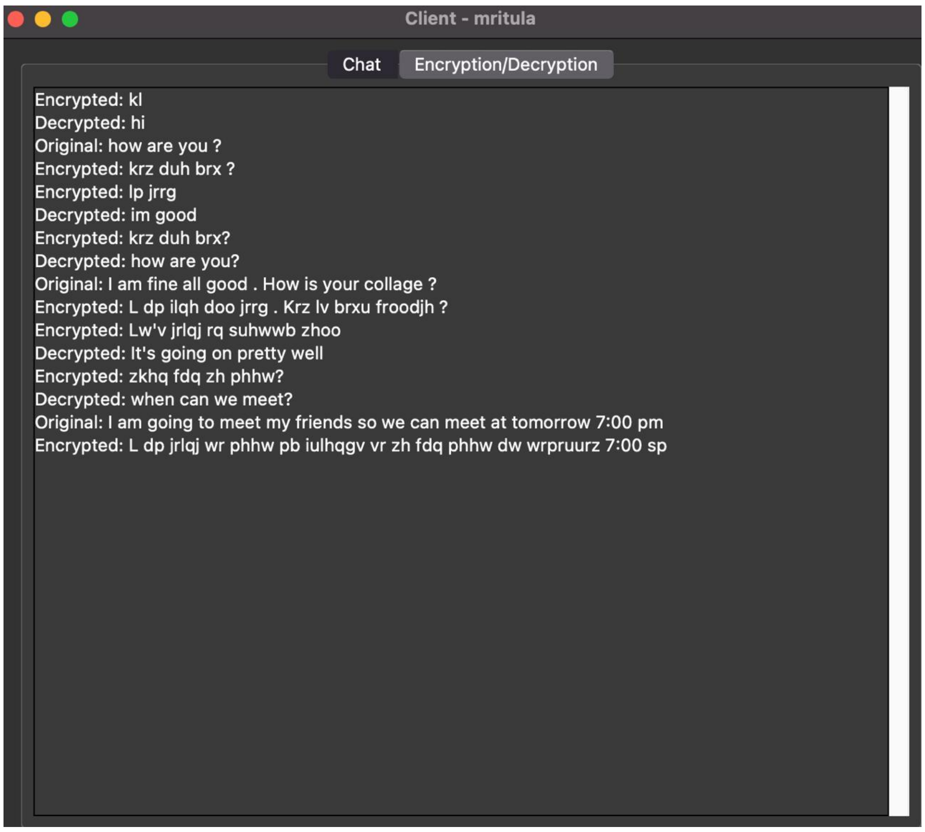
```
varsha : hi
You: how are you ?
varsha : im good
varsha : how are you?
You: I am fine all good . How is your collage ?
varsha : It's going on pretty well
varsha : when can we meet?
You: I am going to meet my friends so we can meet at tomorrow 7:00 pm
```

7.2.7 SERVER ENCRYPTION/DECRYPTION



```
Original: hi
Encrypted: kl
Encrypted: krz duh brx ?
Decrypted: how are you ?
Original: im good
Encrypted: lp jrrg
Original: how are you?
Encrypted: krz duh brx?
Encrypted: L dp ilqh doo jrrg . Krz lv brxu froodjh ?
Decrypted: I am fine all good . How is your collage ?
Original: It's going on pretty well
Encrypted: Lw'v jrlqj rq suhwwb zhoo
Original: when can we meet?
Encrypted: zkhq fdq zh phhw?
Encrypted: L dp jrlqj wr phhw pb iulhqgv vr zh fdq phhw dw wrpruurz 7:00 sp
Decrypted: I am going to meet my friends so we can meet at tomorrow 7:00 pm
```

CLIENT ENCRYPTION/DECRYPTION



```
Encrypted: kl
Decrypted: hi
Original: how are you ?
Encrypted: krz duh brx ?
Encrypted: lp jrrg
Decrypted: im good
Encrypted: krz duh brx?
Decrypted: how are you?
Original: I am fine all good . How is your collage ?
Encrypted: L dp ilqh doo jrrg . Krz lv brxu froodjh ?
Encrypted: Lw'v jrlqj rq suhwwb zhoo
Decrypted: It's going on pretty well
Encrypted: zkhq fdq zh phhw?
Decrypted: when can we meet?
Original: I am going to meet my friends so we can meet at tomorrow 7:00 pm
Encrypted: L dp jrlqj wr phhw pb iulhqgv vr zh fdq phhw dw wrpruurz 7:00 sp
```

REFERENCES

Journal References:

1. Singh, S. The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography. Anchor Books.
2. Stallings W. Cryptography and Network Security: Principles and Practice. Pearson.
3. Python Network Programming Cookbook" by Pradeeban Kathiravelu and Dr. M. O. Faruque Sarker: Focuses on various aspects of network programming in Python.
4. "Python Crash Course" by Eric Matthes: A great book for beginners that covers a wide range of Python topics including working with GUIs and network programming.
5. Ferguson, N., Schneier, B., & Kohno, T. Cryptography Engineering: Design Principles and Practical Applications. Wiley Publishing. This book bridges the gap between theoretical cryptography and practical implementation.
6. Katz, J., & Lindell, Y. Introduction to Modern Cryptography. CRC Press. This textbook provides a comprehensive introduction to modern cryptographic methods.
7. Rescorla, E. SSL and TLS: Designing and Building Secure Systems. Addison-Wesley. This book covers secure communication protocols used in modern applications.

Web References:

1. GeeksforGeeks:

Tutorial: Socket Programming in Python

Details: This tutorial explains the basics of socket programming in Python, with examples of client-server communication.

2. Programiz:

Tutorial: Python Socket Programming

Details: A beginner-friendly guide to socket programming in Python, covering both TCP and UDP protocols.

3. Python.org:

Documentation: Tkinter — Python Interface to Tcl/Tk

Details: The official Tkinter documentation provides detailed information on using Tkinter to create graphical user interfaces in Python.

WORKLOG

Day	Date	Task Done
Day 1	04/06/2024	Visualising the project, preparation of Power Point for Review – 1.
Day 2	05/06/2024	Review – 1.
Day 3	06/06/2024	Studying concepts, preparing and implementing Caesar Cipher.
Day 4	11/06/2024	Preparing, integrating Caesar Cipher in GUI Application.
Day 5	13/06/2024	Creating the Chat Application.
Day 6	18/06/2024	Implementing all the visualised parts in the application, working on all the editions required.
Day 7	20/06/2024	Finalising the Output.
Day 8	25/06/2024	Preparation of Power Point for Review – 2.
Day 9	27/06/2024	Starting to work on the Project Report.
Day 10	01/07/2024	Review – 2.
Day 11	02/07/2024	Working on Project Report.
Day 12	04/07/2024	Completion, submission of Project Report for Guide's approval.
Day 13	08/07/2024	Preparation of Power Point for Review – 1.