Information Security

Project

Talha Sohail
21i-0374
May 7, 2025

Table of Contents

Introduction:

This report summarizes the design, development, and testing of a secure two-way communication system with encrypted registration, login, and chat functionalities, implemented using Python. The core features include credential storage, key management, secure communication, and validation through network analysis.

Client.py source code:

```
import socket
import json
import os
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes
from Crypto.Protocol.KDF import PBKDF2
import hashlib

# Diffie-Hellman public parameters
P = 23  # Choose a large prime number in a real implementation
G = 5   # Primitive root mod P

# Function to perform Diffie-Hellman key exchange
def diffie_hellman_exchange():
    private_key = os.urandom(16)
    public_key = pow(G, int.from_bytes(private_key, 'big'), P)
    return private_key, public_key

# Encrypts the data using AES with key `K`
def encrypt_data(key, data):
    cipher = AES.new(key, AES.MODE_CBC)
    iv = cipher.iv
    encrypted_data = cipher.encrypt(pad(data.encode(), AES.block_size))
    return iv + encrypted_data

def derive_message_key(shared_secret, username):
    key_material = f"{username}{shared_secret}".encode()
    return hashlib.sha256(key_material).digest()[:16]  # AES-128 bit key
```

```
def decrypt_data(key, encrypted_data):
    iv = encrypted_data[:16]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypted_data = unpad(cipher.decrypt(encrypted_data[16:]),
AES.block_size)
    return decrypted_data.decode()

def chat(client_socket, message_key):
    print("You can start chatting now. Type 'bye' to end the chat.")
    while True:
        message = input("You: ")
        encrypted_message = encrypt_data(message_key, message)
        client_socket.sendall(encrypted_message)

        if message.lower() == "bye":
            print("Chat ended.")
            break

        encrypted_response = client_socket.recv(1024)
        response = decrypt_data(message_key, encrypted_response)
        print(f"Server: {response}")

def start_client():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('localhost', 12345))
    print("Connected to the server.")

    choice = input("Choose an option: 'register' or 'login':
").strip().lower()
    client_socket.sendall(choice.encode())

    # Perform Diffie-Hellman Key Exchange
    private_key, client_public_key = diffie_hellman_exchange()
    client_socket.sendall(str(client_public_key).encode())

    server_public_key = int(client_socket.recv(1024).decode())
    shared_secret = pow(server_public_key, int.from_bytes(private_key,
'big'), P)
    K = shared_secret.to_bytes(16, 'big')

    while True:
```

```
        if choice == 'register':
            # Registration process
            email = input("Enter email: ")
            username = input("Enter username: ")
            password = input("Enter password: ")

            user_data = json.dumps({'email': email, 'username': username,
'password': password})
            encrypted_data = encrypt_data(K, user_data)
            client_socket.sendall(encrypted_data)
            response = client_socket.recv(1024).decode()
            print(response)

            if "successful" in response:
                choice = 'login'
            else:
                choice = input("Please try again. Choose 'register' or
'login': ").strip().lower()
                client_socket.sendall(choice.encode())
                continue

        if choice == 'login':
            # Login process
            username = input("Enter username: ")
            password = input("Enter password: ")

            login_data = json.dumps({'username': username, 'password':
password})
            encrypted_data = encrypt_data(K, login_data)
            client_socket.sendall(encrypted_data)

            response = client_socket.recv(1024).decode()
            print(response)

            if "successful" in response:
                # Second Diffie-Hellman Key Exchange for Message Encryption
                private_key, client_public_key = diffie_hellman_exchange()
                client_socket.sendall(str(client_public_key).encode())

                server_public_key = int(client_socket.recv(1024).decode())
                shared_secret = pow(server_public_key,
```

```
int.from_bytes(private_key, 'big'), P)
                message_key = derive_message_key(shared_secret, username)

                # Start secure chat session
                chat(client_socket, message_key)
                break
            else:
                choice = input("Please try again. Choose 'register' or
'login': ").strip().lower()
                client_socket.sendall(choice.encode())

    client_socket.close()

if __name__ == "__main__":
    start_client()
```

Server.py source code:

```
import socket
import json
import os
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes
from Crypto.Protocol.KDF import PBKDF2
import hashlib

# Diffie-Hellman public parameters
P = 23  # Same as client
G = 5   # Same as client

# Diffie-Hellman Key Exchange
def diffie_hellman_exchange():
    private_key = os.urandom(16)
    public_key = pow(G, int.from_bytes(private_key, 'big'), P)
    return private_key, public_key

# AES decryption
def decrypt_data(key, encrypted_data):
    iv = encrypted_data[:16]
```

```
    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypted_data = unpad(cipher.decrypt(encrypted_data[16:]),
AES.block_size)
    return decrypted_data.decode()

def derive_message_key(shared_secret, username):
    key_material = f"{username}{shared_secret}".encode()
    return hashlib.sha256(key_material).digest()[:16]  # AES-128 bit key

def encrypt_data(key, data):
    cipher = AES.new(key, AES.MODE_CBC)
    iv = cipher.iv
    encrypted_data = cipher.encrypt(pad(data.encode(), AES.block_size))
    return iv + encrypted_data

def chat(conn, message_key):
    print("Chat session started. Type 'bye' to end the chat.")
    while True:
        encrypted_message = conn.recv(1024)
        message = decrypt_data(message_key, encrypted_message)
        print(f"Client: {message}")

        if message.lower() == "bye":
            print("Chat ended.")
            break

        response = input("Server: ")
        encrypted_response = encrypt_data(message_key, response)
        conn.sendall(encrypted_response)

# Store or load credentials from JSON
CREDENTIALS_FILE = "creds.json"

def load_credentials():
    if os.path.exists(CREDENTIALS_FILE):
        with open(CREDENTIALS_FILE, "r") as file:
            return json.load(file)
    return {}

def save_credentials(credentials):
    with open(CREDENTIALS_FILE, "w") as file:
```

```
        json.dump(credentials, file)

def start_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('localhost', 12345))
    server_socket.listen(2)
    print("Server is listening on port 12345...")

    conn, addr = server_socket.accept()
    print(f"Connected by {addr}")

    try:
        credentials = load_credentials()

        while True:
            choice = conn.recv(1024).decode().strip().lower()

            # Perform Diffie-Hellman Key Exchange
            client_public_key = int(conn.recv(1024).decode())
            private_key, server_public_key = diffie_hellman_exchange()
            conn.sendall(str(server_public_key).encode())

            shared_secret = pow(client_public_key,
int.from_bytes(private_key, 'big'), P)
            K = shared_secret.to_bytes(16, 'big')

            if choice == 'register':
                # Registration process
                encrypted_data = conn.recv(1024)
                decrypted_data = json.loads(decrypt_data(K,
encrypted_data))
                username = decrypted_data['username']

                if username in credentials:
                    conn.sendall(b"Username already exists. Please try
again.")
                else:
                    salt = os.urandom(4).hex()
                    password_hash =
hashlib.sha256((decrypted_data['password'] + salt).encode()).hexdigest()
                    credentials[username] = {
```

```
                        'email': decrypted_data['email'],
                        'password': password_hash,
                        'salt': salt
                    }
                    save_credentials(credentials)
                    conn.sendall(b"Registration successful. Please log
in.")
                    choice = 'login'  # Switch to login mode after
registration success

            if choice == 'login':
                # Login process
                encrypted_data = conn.recv(1024)
                decrypted_data = json.loads(decrypt_data(K,
encrypted_data))
                username = decrypted_data['username']

                if username in credentials:
                    user_data = credentials[username]
                    salt = user_data['salt']
                    hashed_password =
hashlib.sha256((decrypted_data['password'] + salt).encode()).hexdigest()

                    if hashed_password == user_data['password']:
                        conn.sendall(b"Login successful.")

                        # Perform second Diffie-Hellman Key Exchange for
secure messaging
                        client_public_key = int(conn.recv(1024).decode())
                        private_key, server_public_key =
diffie_hellman_exchange()
                        conn.sendall(str(server_public_key).encode())

                        shared_secret = pow(client_public_key,
int.from_bytes(private_key, 'big'), P)
                        message_key = derive_message_key(shared_secret,
username)

                        # Start secure chat session
                        chat(conn, message_key)
                        break
```

```
                    else:
                        conn.sendall(b"Invalid password. Please try
again.")
                else:
                    conn.sendall(b"Username not found. Please try again.")

    except Exception as e:
        print("Error:", e)
    finally:
        conn.close()

if __name__ == "__main__":
    start_server()
```

Event Response Tables

Registration Process

Event	Client Response	Server Response
Client selects "register"	Sends "register" command to the server	Waits for registration data from the client
Client inputs email, username, and password	Sends encrypted registration data to the server	Receives and decrypts registration data
Server checks if username exists	None	Checks credentials file for existing username
Username exists	Receives "Username already exists" message	Sends "Username already exists" to client
Username is available	None	Registers user, saves to credentials file, and sends "Registration successful" message

Registration successful	Receives "Registration successful" message	Asks client to proceed with login
Registration unsuccessful	Receives "Username already exists" message	Awaits client retry or exit

Login Process

Event	Client Response	Server Response
Client selects "login"	Sends "login" command to the server	Waits for login data from the client
Client inputs username and password	Sends encrypted login data to the server	Receives and decrypts login data
Server checks if username exists	None	Looks up username in credentials file
Username not found	Receives "Username not found" message	Sends "Username not found" to client
Username exists, password is correct	Receives "Login successful" message	Sends "Login successful" to client, starts chat session initiation
Username exists, password is incorrect	Receives "Invalid password" message	Sends "Invalid password" to client, allows retry
Login successful	Initiates second Diffie-Hellman exchange and proceeds with chat setup	Proceeds with second Diffie-Hellman key exchange for secure chat
Login unsuccessful	Receives retry prompt	Awaits new login attempt

Chatting

Event	Client Response	Server Response
Chat session initiation	Completes second Diffie-Hellman key exchange	Completes second Diffie-Hellman key exchange
Chat message input by client	Encrypts and sends message	Receives and decrypts message, displays it
Chat message input by server	Receives and decrypts message	Encrypts and sends message
Client types "bye" to end chat	Encrypts and sends "bye" message	Receives and decrypts "bye", ends chat
Server types "bye" to end chat	Receives and decrypts "bye", ends chat	Encrypts and sends "bye" message
Client connection closes	Disconnects socket and ends program	Detects client disconnection, closes socket
Server connection closes	Detects server disconnection	Disconnects socket and ends program

1. System Architecture

The system consists of two main components:

- **Client**: Handles user registration, login, and message exchange with the server.
- **Server**: Manages user registration, credential verification, and encrypted message exchange with clients.

Communication between client and server is over TCP on port 12345. The client initiates the connection, and secure exchanges are established through a Diffie-Hellman (DH) key exchange, followed by encrypted communication using AES in CBC mode.

2. Credential Storage and Key Management

- **Credential Storage:**
    - During registration, the client sends an encrypted username and a SHA-256 hashed password (salted) to the server.
    - Credentials are stored in a file, creds.json, in the format:
      `{"username": {"email": "", "password": "", "salt": ""}`
    - Storing hashes, rather than plaintext passwords, protects against data leakage.
- **Key Management:**
    - During both registration and login, Diffie-Hellman (DH) key exchange is used to generate a shared secret key (K) between client and server, ensuring that only these two parties can derive the key.
    - For message encryption after login, a second DH key exchange occurs, generating a new shared key, which is appended to the username (e.g., john_doe19). This concatenated key is then used as the AES encryption key.
    - Each session's key is unique, ensuring that key reuse is minimized and that prior communication remains confidential if a future key is compromised.

3. Security Considerations

To achieve confidentiality, integrity, and authentication, the following security measures were implemented:

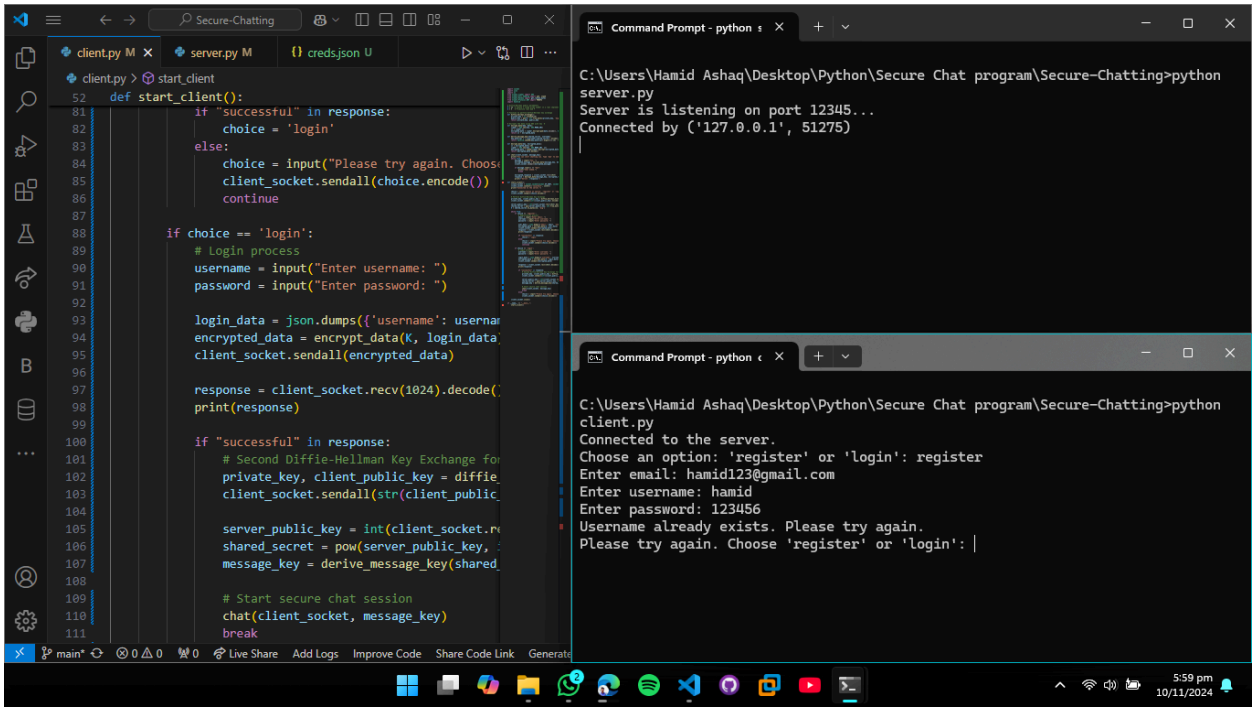
- AES Encryption (CBC Mode):
    - Messages between the client and server are encrypted using AES in CBC mode, which requires a unique initialization vector (IV) for each message. The IV is prepended to each encrypted message, allowing the receiver to decrypt the data correctly.

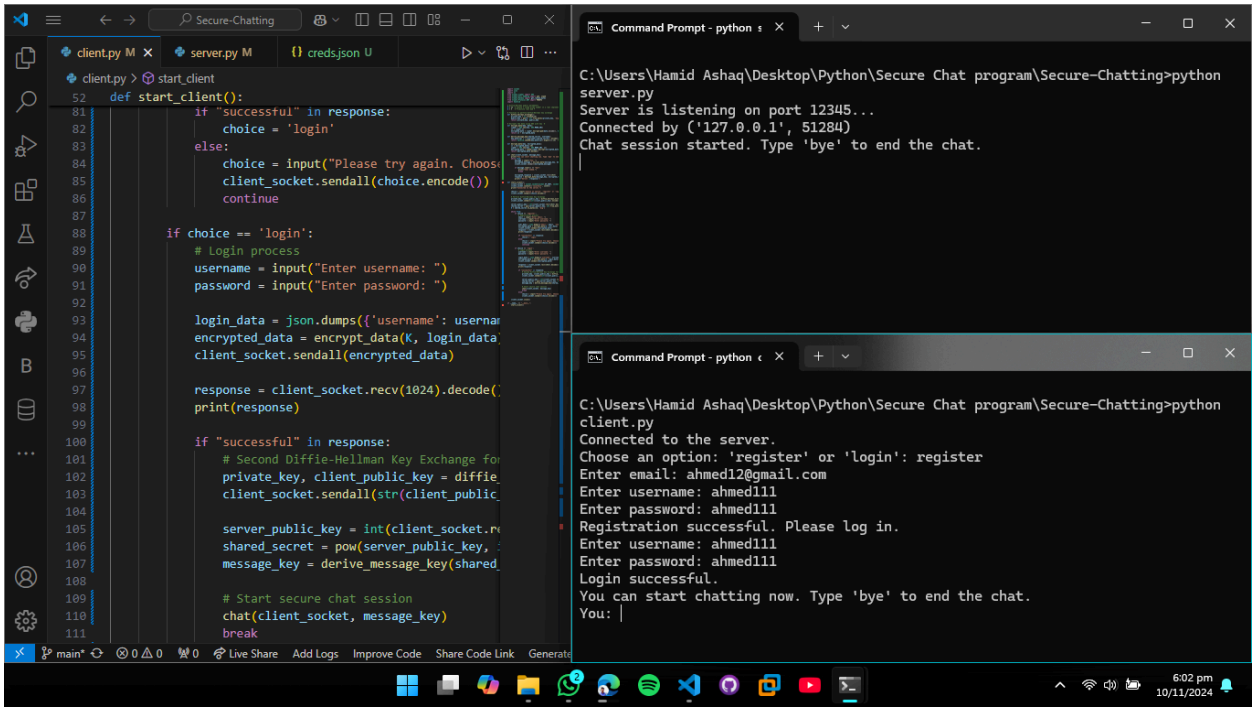
- Diffie-Hellman Key Exchange:
    - DH key exchange ensures secure key generation without directly sharing the encryption key over the network. It provides forward secrecy, meaning each session has a unique key, and previous sessions cannot be decrypted if future keys are compromised.
- Password Hashing with Salt:
    - SHA-256 hashing with a unique salt ensures that stored passwords are resistant to rainbow table attacks. Salting increases the complexity of brute-force attacks by making each password hash unique.
- Access Control:
    - Only users who successfully log in with matching credentials (username and hashed password) are granted access to the chat system. Failed login attempts can be restricted by reusing the same session key, with an option to include a timer for key freshness.

4. Testing and Validation Report

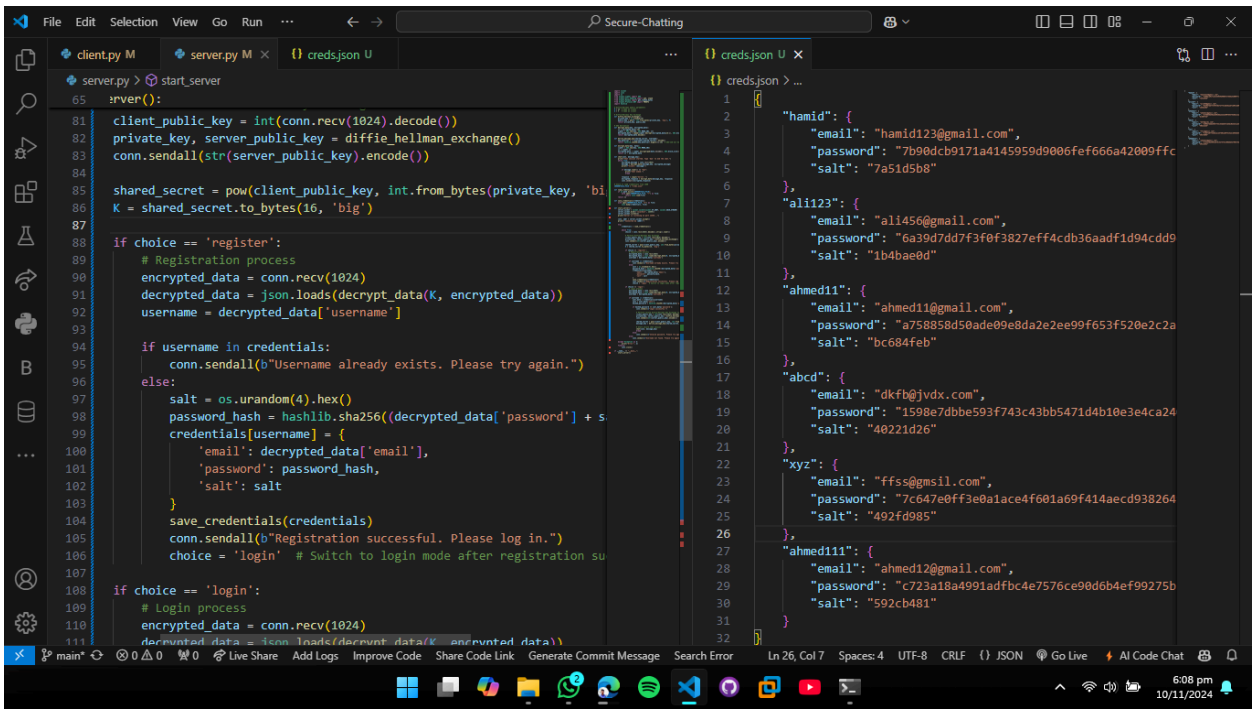
To validate the security and functionality of the chat system, several tests were performed:

- Functional Testing:
    - Registration
        - Verify that new users can register with a unique username and password.

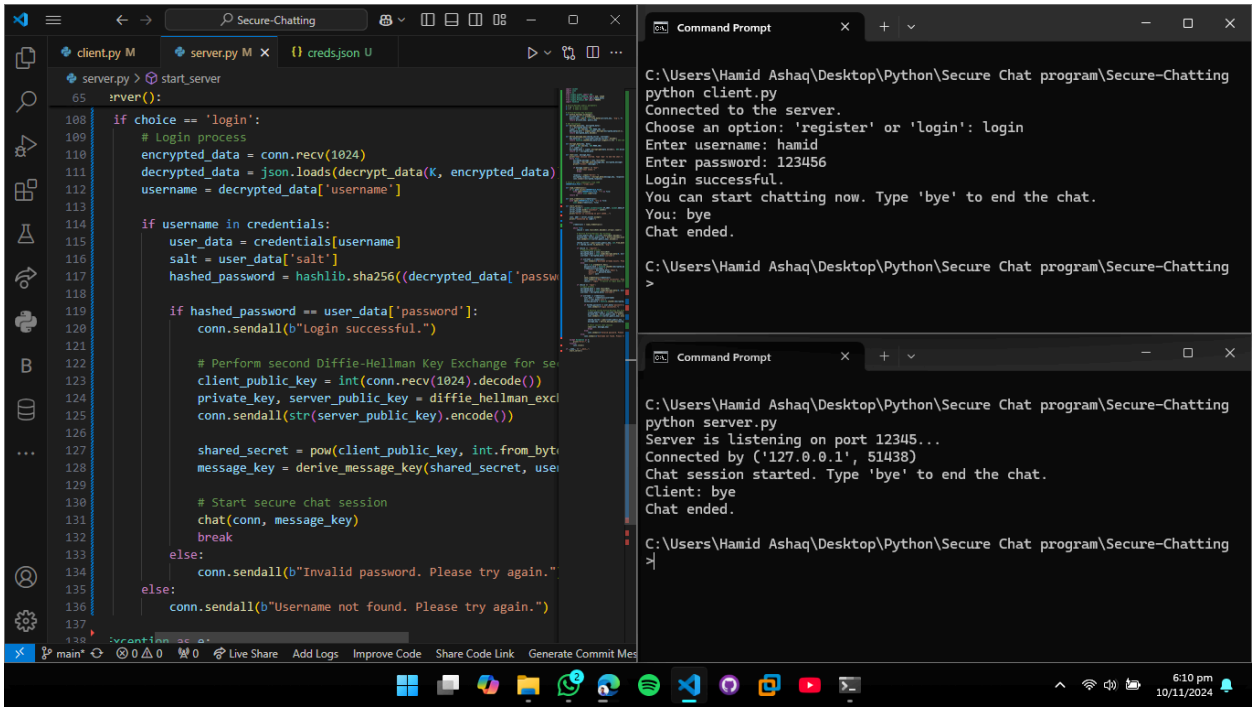




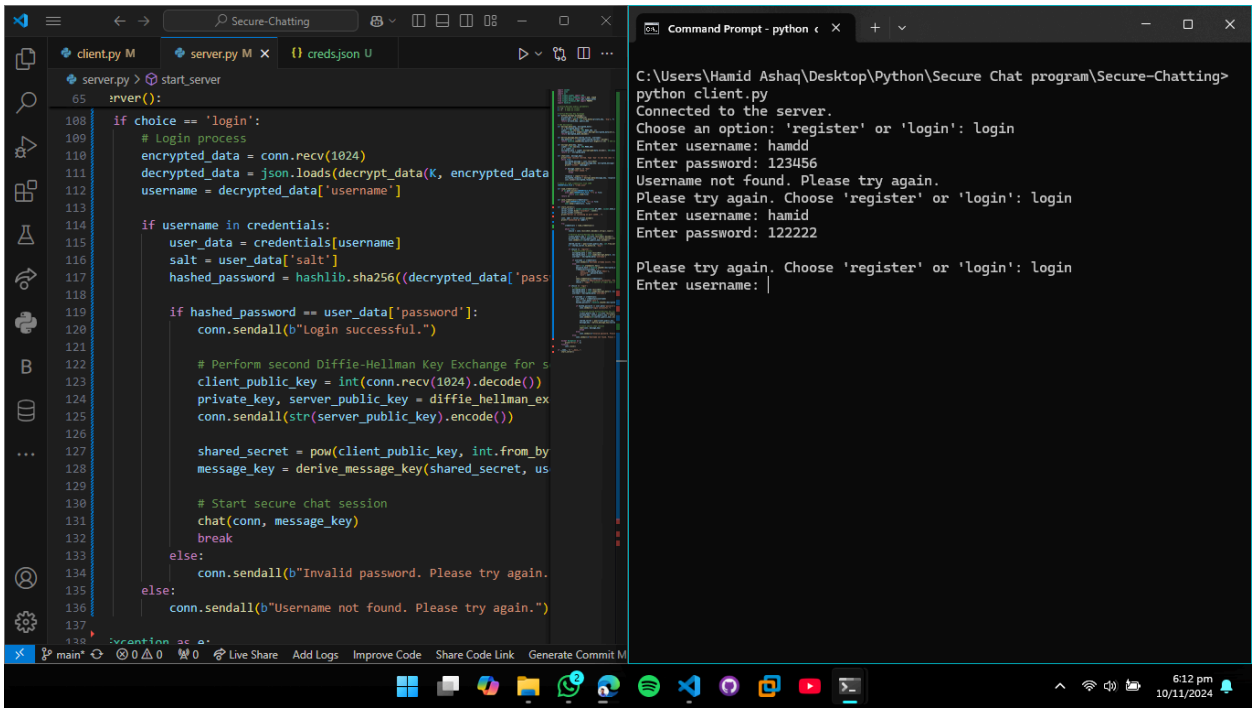
- Ensure that the password is hashed correctly using SHA-256 and stored securely in the credential file along with salt value.



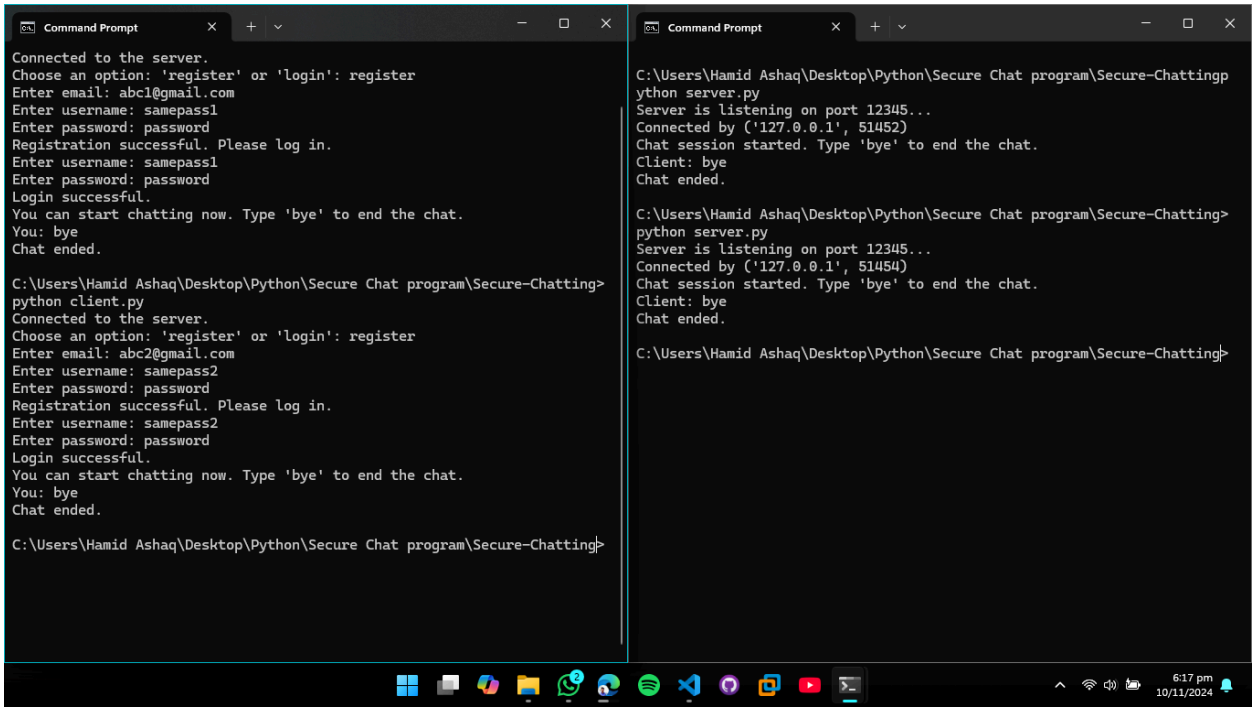
- Login
  - Test successful logins with valid credentials. (25 marks)

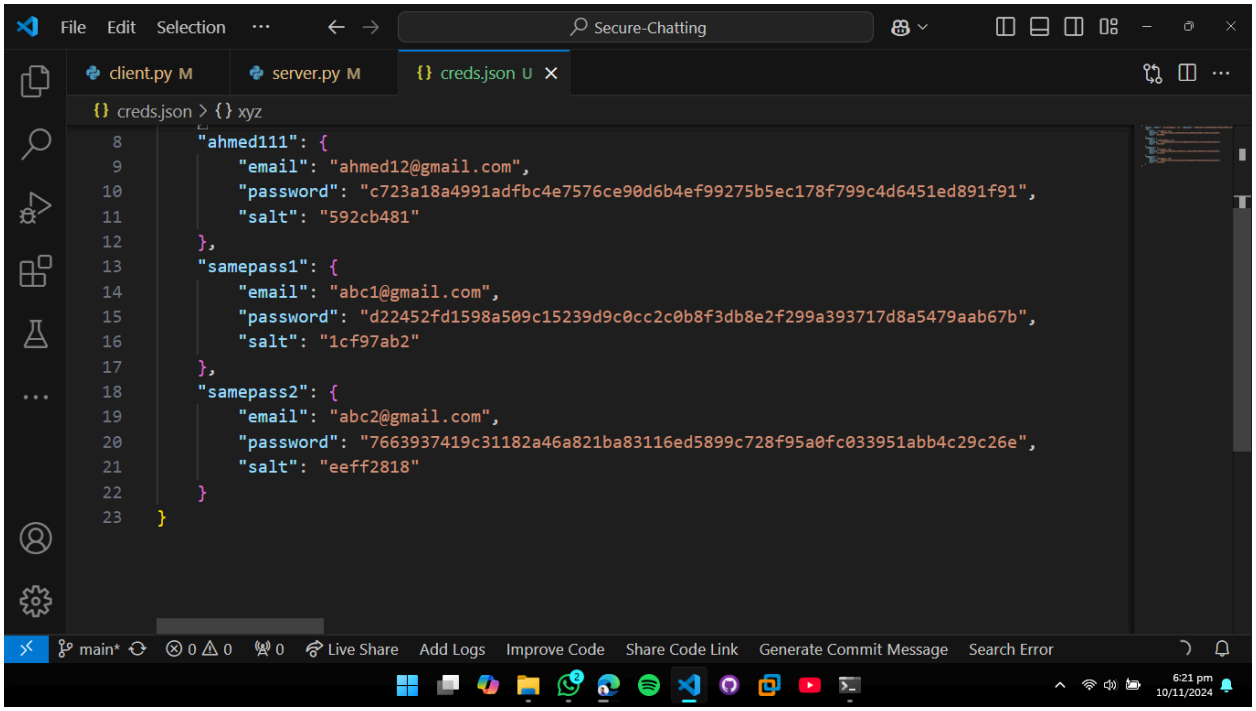


- Test failed logins with incorrect credentials and ensure proper error messages are displayed. (25 marks)

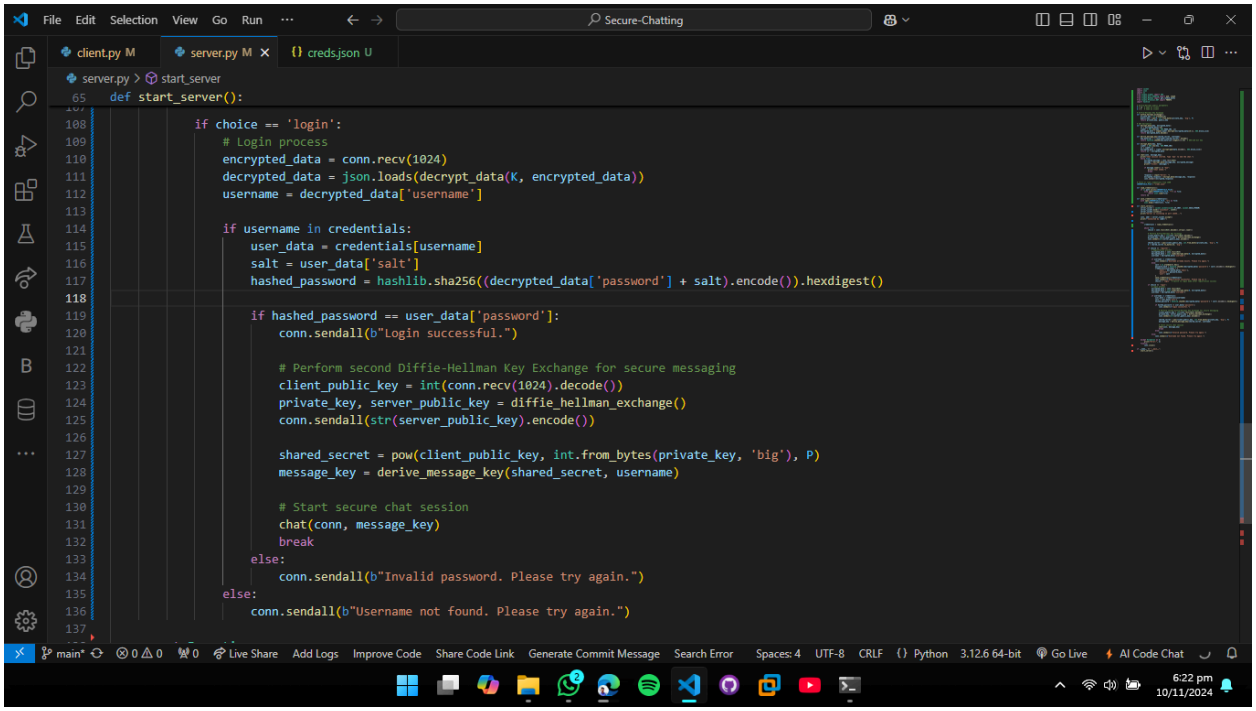


- Security Testing:
  - Hash Verification
    - Verify that identical passwords generate different hashes due to random salting with SHA-256. (25 marks)



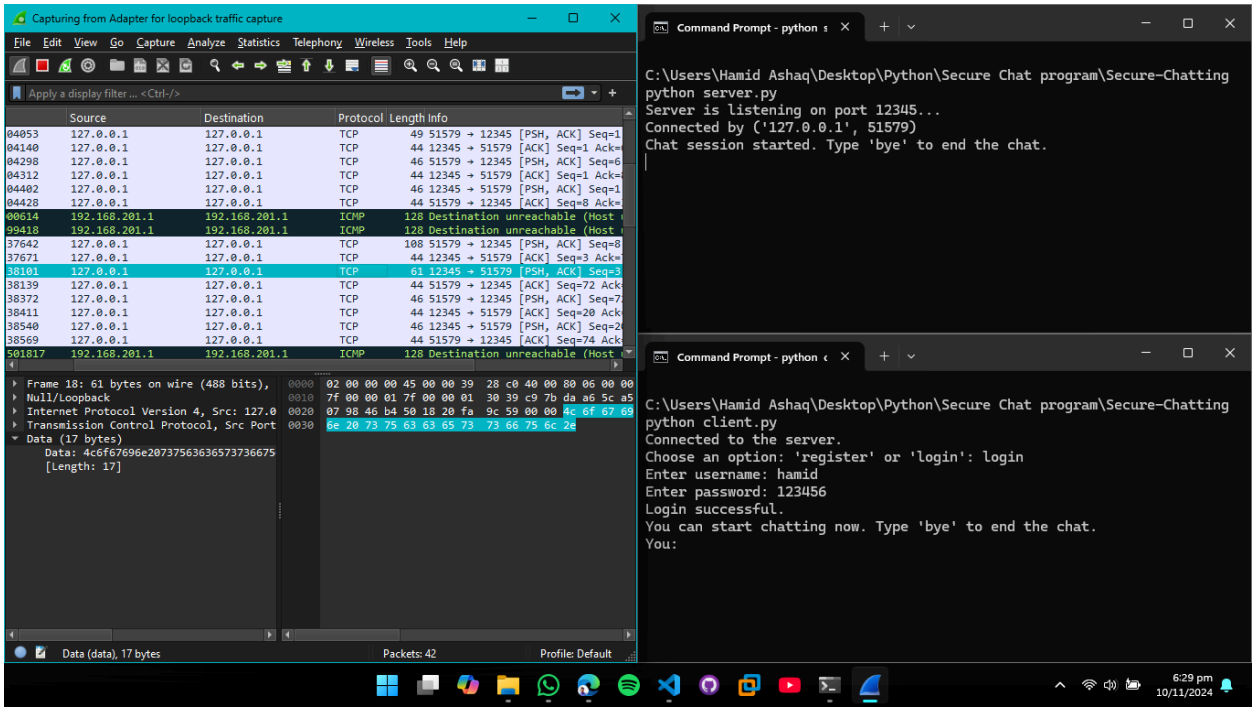


- Verify user passwords are verified during the login phase.

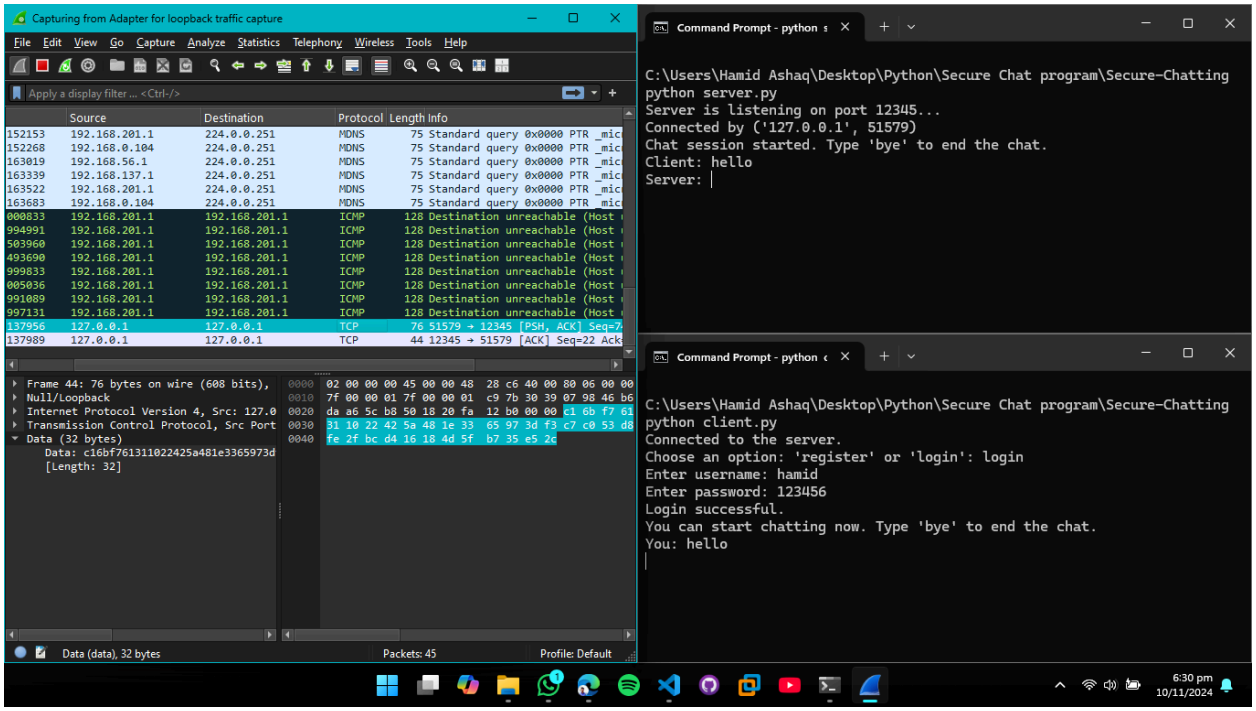


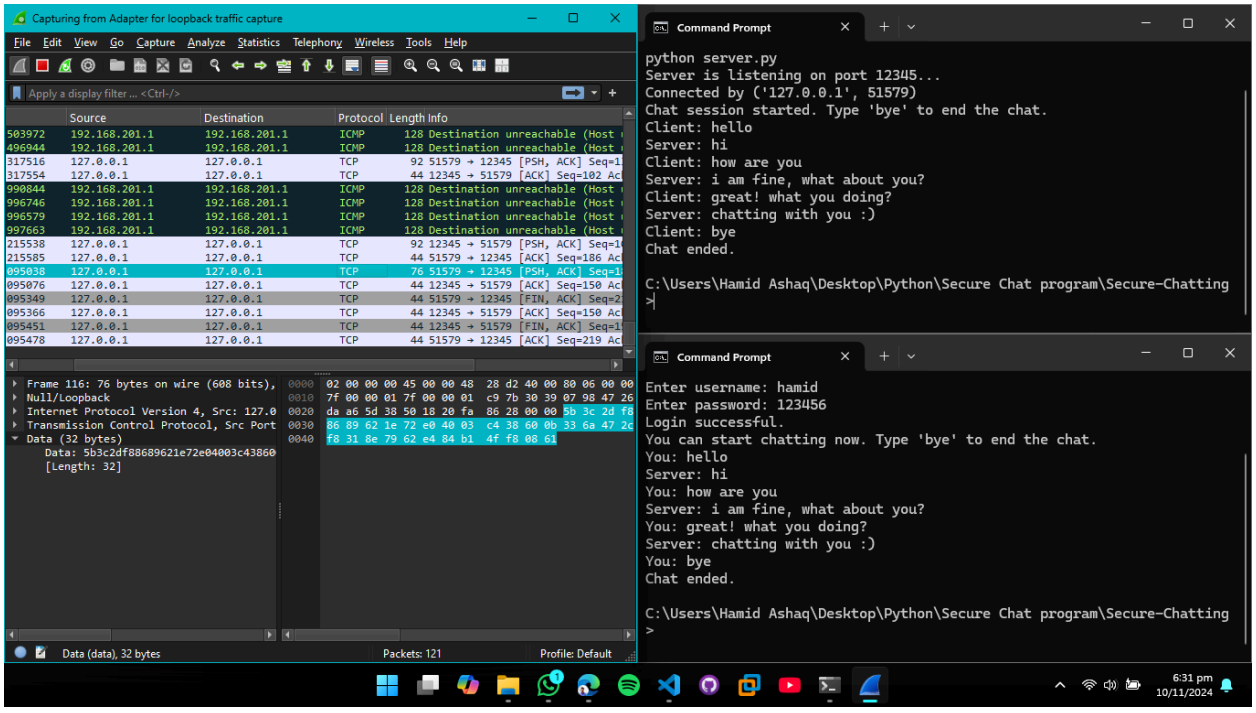
- Encryption Testing
  - Capture the communication between the client and server using network analysis tools (e.g., Wireshark) and verify that all messages are encrypted in both the Registration and Login phases including all messages exchanged.

- Login Communication

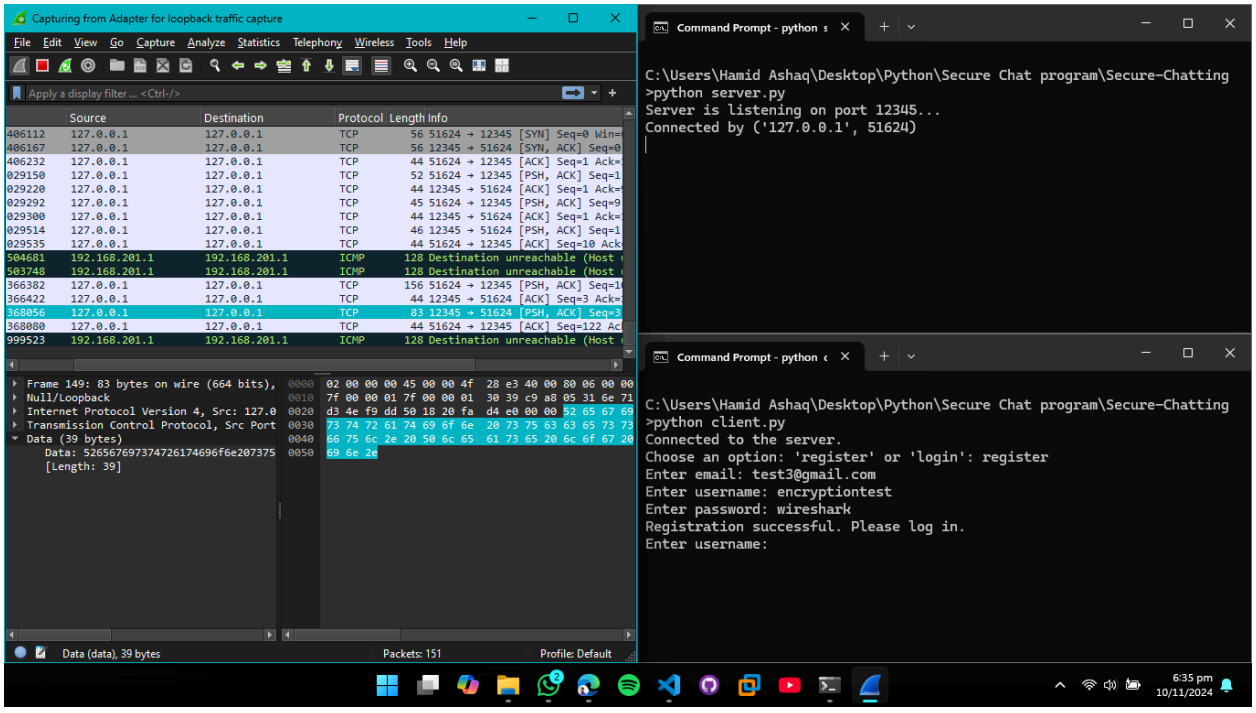


- Chatting Communication





- Registration Communication



5. Conclusion

The secure chat system successfully demonstrates a robust design for two-way encrypted communication, incorporating secure key management, credential storage, and access control. By using Diffie-Hellman key exchange and AES encryption with unique session keys, the system maintains the confidentiality and integrity of messages. Testing with Wireshark verified the encryption of all messages, meeting the security requirements. Future improvements could include implementing a timeout for session keys to further enhance key freshness and prevent replay attacks.