

# Information Security **Project**

---

Talha Sohail

21i-0374

May 7, 2025



# Table of Contents

<b>Introduction:</b>	<b>2</b>
Event Response Tables	9
Registration Process	9
Login Process	10
Chatting	11
1. System Architecture	11
2. Credential Storage and Key Management	12
3. Security Considerations	12
4. Testing and Validation Report	13
● Functional Testing:	13
● Security Testing:	16
5. Conclusion	21

## 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):**

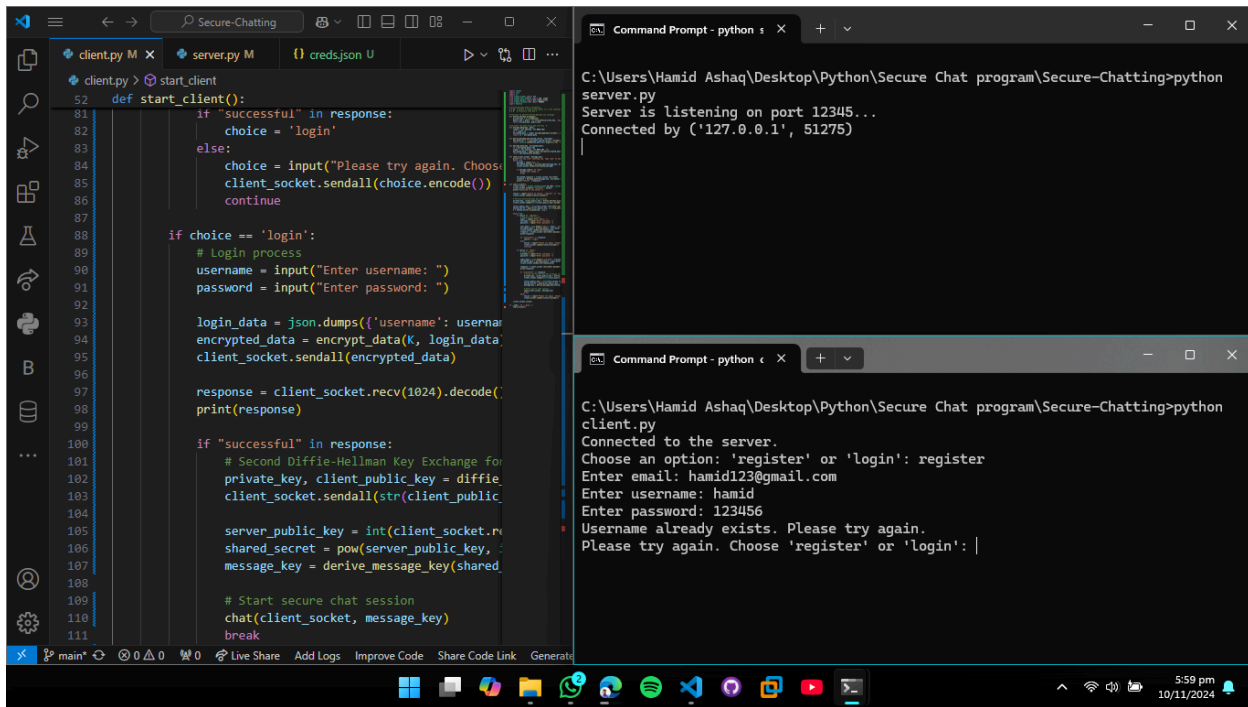
- Messages between the client and server are encrypted using AES in CBC mode, which requires a unique initialization vector (IV) for each message. The IV is prepended to each encrypted message, allowing the receiver to decrypt the data correctly.

- 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.



```
def start_client():
    if "successful" in response:
        choice = "login"
    else:
        choice = input("Please try again. Choose 'register' or 'login': ")
        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)

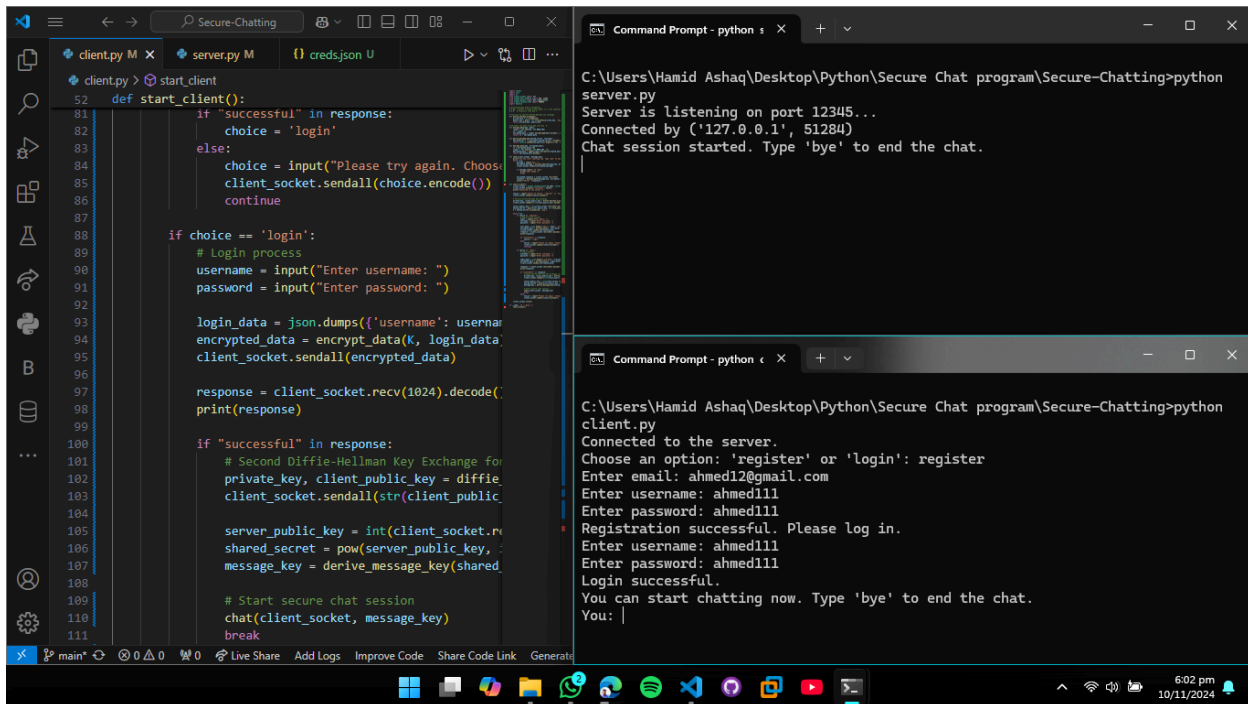
        if "successful" in response:
            # Second Diffie-Hellman Key Exchange for shared secret
            private_key, client_public_key = diffie_hellman(private_key, client_public_key)
            client_socket.sendall(str(client_public_key).encode())

            server_public_key = int(client_socket.recv(1024).decode())
            shared_secret = pow(server_public_key, private_key)
            message_key = derive_message_key(shared_secret)

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

```
C:\Users\Hamid Ashaq\Desktop\Python\Secure Chat program\Secure-Chatting>python server.py
Server is listening on port 12345...
Connected by ('127.0.0.1', 51275)

C:\Users\Hamid Ashaq\Desktop\Python\Secure Chat program\Secure-Chatting>python client.py
Connected to the server.
Choose an option: 'register' or 'login': register
Enter email: hamid123@gmail.com
Enter username: hamid
Enter password: 123456
Username already exists. Please try again.
Please try again. Choose 'register' or 'login': |
```



```
def start_client():
    if "successful" in response:
        choice = "login"
    else:
        choice = input("Please try again. Choose 'register' or 'login': ")
        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 shared secret
            private_key, client_public_key = diffie_hellman(private_key, client_public_key)
            client_socket.sendall(str(client_public_key).encode())

            server_public_key = int(client_socket.recv(1024).decode())
            shared_secret = pow(server_public_key, private_key)
            message_key = derive_message_key(shared_secret)

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

```
C:\Users\Hamid Ashaq\Desktop\Python\Secure Chat program\Secure-Chatting>python server.py
Server is listening on port 12345...
Connected by ('127.0.0.1', 51284)
Chat session started. Type 'bye' to end the chat.

C:\Users\Hamid Ashaq\Desktop\Python\Secure Chat program\Secure-Chatting>python client.py
Connected to the server.
Choose an option: 'register' or 'login': register
Enter email: ahmed12@gmail.com
Enter username: ahmed111
Enter password: ahmed111
Registration successful. Please log in.
Enter username: ahmed111
Enter password: ahmed111
Login successful.
You can start chatting now. Type 'bye' to end the chat.
You: |
```

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





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

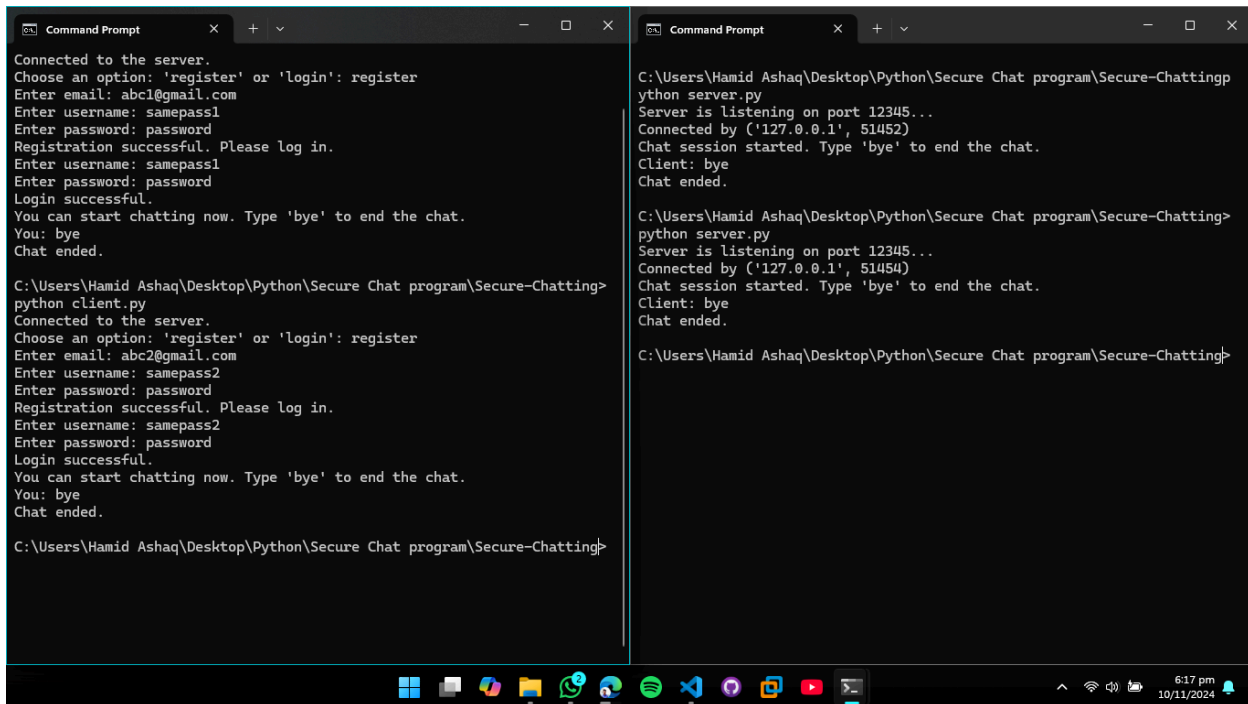
```
client.py M server.py M creds.json U
65 server():
108 if choice == 'login':
109     # Login process
110     encrypted_data = conn.recv(1024)
111     decrypted_data = json.loads(decrypt_data(K, encrypted_data)
112     username = decrypted_data['username']
113
114     if username in credentials:
115         user_data = credentials[username]
116         salt = user_data['salt']
117         hashed_password = hashlib.sha256((decrypted_data['pass
118
119         if hashed_password == user_data['password']:
120             conn.sendall(b"Login successful.")
121
122             # Perform second Diffie-Hellman Key Exchange for s
123             client_public_key = int(conn.recv(1024).decode())
124             private_key, server_public_key = diffie_hellman_ex
125             conn.sendall(str(server_public_key).encode())
126
127             shared_secret = pow(client_public_key, int.from_by
128             message_key = derive_message_key(shared_secret, us
129
130             # Start secure chat session
131             chat(conn, message_key)
132             break
133         else:
134             conn.sendall(b"Invalid password. Please try again.")
135     else:
136         conn.sendall(b"Username not found. Please try again.")
137
138 if __name__ == '__main__':
139     main()

C:\Users\Hamid Ashaq\Desktop\Python\Secure Chat program\Secure-Chatting>
python client.py
Connected to the server.
Choose an option: 'register' or 'login': login
Enter username: hamdd
Enter password: 123456
Username not found. Please try again.
Please try again. Choose 'register' or 'login': login
Enter username: hamid
Enter password: 122222
Please try again. Choose 'register' or 'login': login
Enter username: |
```

## ● Security Testing:

### ○ Hash Verification

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



```
Command Prompt
Connected to the server.
Choose an option: 'register' or 'login': register
Enter email: abc1@gmail.com
Enter username: samepass1
Enter password: password
Registration successful. Please log in.
Enter username: samepass1
Enter password: password
Login successful.
You can start chatting now. Type 'bye' to end the chat.
You: bye
Chat ended.

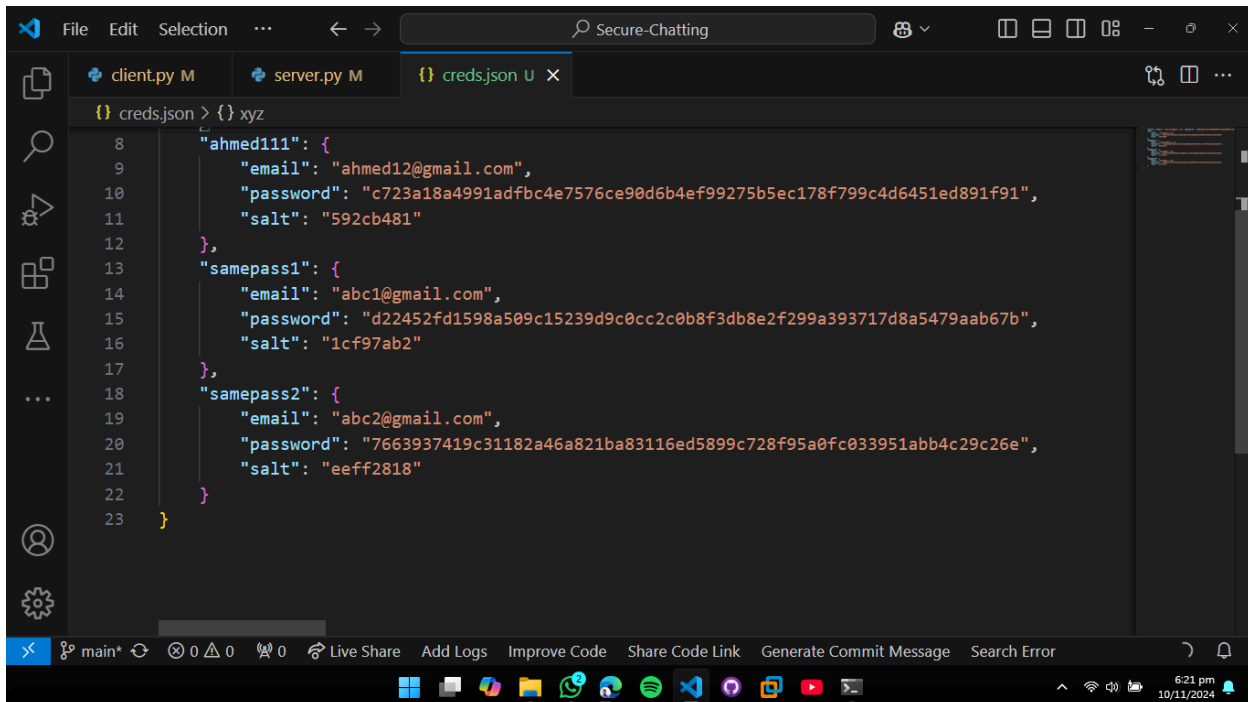
C:\Users\Hamid Ashaq\Desktop\Python\Secure Chat program\Secure-Chatting>
python client.py
Connected to the server.
Choose an option: 'register' or 'login': register
Enter email: abc2@gmail.com
Enter username: samepass2
Enter password: password
Registration successful. Please log in.
Enter username: samepass2
Enter password: password
Login successful.
You can start chatting now. Type 'bye' to end the chat.
You: bye
Chat ended.

C:\Users\Hamid Ashaq\Desktop\Python\Secure Chat program\Secure-Chatting>

Command Prompt
C:\Users\Hamid Ashaq\Desktop\Python\Secure Chat program\Secure-Chatting>
python server.py
Server is listening on port 12345...
Connected by ('127.0.0.1', 51452)
Chat session started. Type 'bye' to end the chat.
Client: bye
Chat ended.

C:\Users\Hamid Ashaq\Desktop\Python\Secure Chat program\Secure-Chatting>
python server.py
Server is listening on port 12345...
Connected by ('127.0.0.1', 51454)
Chat session started. Type 'bye' to end the chat.
Client: bye
Chat ended.

C:\Users\Hamid Ashaq\Desktop\Python\Secure Chat program\Secure-Chatting>
```



```
File Edit Selection ... Secure-Chatting
client.py M server.py M creds.json U x
creds.json > {} xyz
8  "ahmed111": {
9      "email": "ahmed12@gmail.com",
10     "password": "c723a18a4991adfb4e7576ce90d6b4ef99275b5ec178f799c4d6451ed891f91",
11     "salt": "592cb481"
12 },
13 "samepass1": {
14     "email": "abc1@gmail.com",
15     "password": "d22452fd1598a509c15239d9c0cc2c0b8f3db8e2f299a393717d8a5479aab67b",
16     "salt": "1cf97ab2"
17 },
18 "samepass2": {
19     "email": "abc2@gmail.com",
20     "password": "7663937419c31182a46a821ba83116ed5899c728f95a0fc033951abb4c29c26e",
21     "salt": "eeff2818"
22 }
23 }
```

- Verify user passwords are verified during the login phase.

```
65 def start_server():
108     if choice == 'login':
109         # Login process
110         encrypted_data = conn.recv(1024)
111         decrypted_data = json.loads(decrypt_data(K, encrypted_data))
112         username = decrypted_data['username']
113
114         if username in credentials:
115             user_data = credentials[username]
116             salt = user_data['salt']
117             hashed_password = hashlib.sha256((decrypted_data['password'] + salt).encode()).hexdigest()
118
119             if hashed_password == user_data['password']:
120                 conn.sendall(b"Login successful.")
121
122                 # Perform second Diffie-Hellman Key Exchange for secure messaging
123                 client_public_key = int(conn.recv(1024).decode())
124                 private_key, server_public_key = diffie_hellman_exchange()
125                 conn.sendall(str(server_public_key).encode())
126
127                 shared_secret = pow(client_public_key, int.from_bytes(private_key, 'big'), P)
128                 message_key = derive_message_key(shared_secret, username)
129
130                 # Start secure chat session
131                 chat(conn, message_key)
132                 break
133             else:
134                 conn.sendall(b"Invalid password. Please try again.")
135         else:
136             conn.sendall(b"Username not found. Please try again.")
137
```

- 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

The image displays two windows side-by-side. The left window is Wireshark, showing a packet capture from the adapter for loopback traffic. The right window is a Command Prompt running a Python script for a secure chat program.

**Wireshark Packet Capture:**

No.	Source	Destination	Protocol	Length	Info
04053	127.0.0.1	127.0.0.1	TCP	49	51579 → 12345 [PSH, ACK] Seq=1
04140	127.0.0.1	127.0.0.1	TCP	44	12345 → 51579 [ACK] Seq=1 Ack=
04298	127.0.0.1	127.0.0.1	TCP	46	51579 → 12345 [PSH, ACK] Seq=6
04312	127.0.0.1	127.0.0.1	TCP	44	12345 → 51579 [ACK] Seq=1 Ack=
04402	127.0.0.1	127.0.0.1	TCP	46	12345 → 51579 [PSH, ACK] Seq=1
04428	127.0.0.1	127.0.0.1	TCP	44	51579 → 12345 [ACK] Seq=8 Ack=
00614	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host
99418	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host
37642	127.0.0.1	127.0.0.1	TCP	108	51579 → 12345 [PSH, ACK] Seq=8
37671	127.0.0.1	127.0.0.1	TCP	44	12345 → 51579 [ACK] Seq=3 Ack=
38101	127.0.0.1	127.0.0.1	TCP	61	12345 → 51579 [PSH, ACK] Seq=3
38139	127.0.0.1	127.0.0.1	TCP	44	51579 → 12345 [ACK] Seq=72 Ack=
38372	127.0.0.1	127.0.0.1	TCP	46	51579 → 12345 [PSH, ACK] Seq=7
38411	127.0.0.1	127.0.0.1	TCP	44	12345 → 51579 [ACK] Seq=20 Ack=
38540	127.0.0.1	127.0.0.1	TCP	46	12345 → 51579 [PSH, ACK] Seq=2
38569	127.0.0.1	127.0.0.1	TCP	44	51579 → 12345 [ACK] Seq=74 Ack=
501817	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host

**Command Prompt - python server.py:**

```
C:\Users\Hamid Ashaq\Desktop\Python\Secure Chat program\Secure-Chatting
python server.py
Server is listening on port 12345...
Connected by ('127.0.0.1', 51579)
Chat session started. Type 'bye' to end the chat.
```

**Command Prompt - python client.py:**

```
C:\Users\Hamid Ashaq\Desktop\Python\Secure Chat program\Secure-Chatting
python client.py
Connected to the server.
Choose an option: 'register' or 'login': login
Enter username: hamid
Enter password: 123456
Login successful.
You can start chatting now. Type 'bye' to end the chat.
You:
```

## ■ Chatting Communication

The image displays two windows side-by-side. The left window is Wireshark, showing a packet capture from the adapter for loopback traffic. The right window is a Command Prompt running a Python script for a secure chat program.

**Wireshark Packet Capture:**

No.	Source	Destination	Protocol	Length	Info
152153	192.168.201.1	224.0.0.251	MDNS	75	Standard query 0x0000 PTR _mic
152268	192.168.0.104	224.0.0.251	MDNS	75	Standard query 0x0000 PTR _mic
163019	192.168.56.1	224.0.0.251	MDNS	75	Standard query 0x0000 PTR _mic
163339	192.168.137.1	224.0.0.251	MDNS	75	Standard query 0x0000 PTR _mic
163522	192.168.201.1	224.0.0.251	MDNS	75	Standard query 0x0000 PTR _mic
163683	192.168.0.104	224.0.0.251	MDNS	75	Standard query 0x0000 PTR _mic
000833	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host
994991	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host
503960	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host
493690	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host
999833	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host
005036	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host
991089	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host
997131	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host
137856	127.0.0.1	127.0.0.1	TCP	76	51579 → 12345 [PSH, ACK] Seq=7
137989	127.0.0.1	127.0.0.1	TCP	44	12345 → 51579 [ACK] Seq=22 Ack=

**Command Prompt - python server.py:**

```
C:\Users\Hamid Ashaq\Desktop\Python\Secure Chat program\Secure-Chatting
python server.py
Server is listening on port 12345...
Connected by ('127.0.0.1', 51579)
Chat session started. Type 'bye' to end the chat.
Client: hello
Server: |
```

**Command Prompt - python client.py:**

```
C:\Users\Hamid Ashaq\Desktop\Python\Secure Chat program\Secure-Chatting
python client.py
Connected to the server.
Choose an option: 'register' or 'login': login
Enter username: hamid
Enter password: 123456
Login successful.
You can start chatting now. Type 'bye' to end the chat.
You: hello
```

The top screenshot shows a Wireshark capture of a chat session. The packet list table is as follows:

No.	Source	Destination	Protocol	Length	Info
503972	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host ...)
496944	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host ...)
317516	127.0.0.1	127.0.0.1	TCP	92	51579 → 12345 [PSH, ACK] Seq=1
317554	127.0.0.1	127.0.0.1	TCP	44	12345 → 51579 [ACK] Seq=102 Ac
990844	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host ...)
996746	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host ...)
996579	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host ...)
997663	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host ...)
215538	127.0.0.1	127.0.0.1	TCP	92	12345 → 51579 [PSH, ACK] Seq=1
215585	127.0.0.1	127.0.0.1	TCP	44	51579 → 12345 [ACK] Seq=186 Ac
895838	127.0.0.1	127.0.0.1	TCP	76	51579 → 12345 [PSH, ACK] Seq=1
895876	127.0.0.1	127.0.0.1	TCP	44	12345 → 51579 [ACK] Seq=150 Ac
895349	127.0.0.1	127.0.0.1	TCP	44	51579 → 12345 [FIN, ACK] Seq=2
895366	127.0.0.1	127.0.0.1	TCP	44	12345 → 51579 [ACK] Seq=150 Ac
895451	127.0.0.1	127.0.0.1	TCP	44	12345 → 51579 [FIN, ACK] Seq=1
895478	127.0.0.1	127.0.0.1	TCP	44	51579 → 12345 [ACK] Seq=219 Ac

The bottom screenshot shows the Command Prompt output for the chat session:

```
python server.py
Server is listening on port 12345...
Connected by ('127.0.0.1', 51579)
Chat session started. Type 'bye' to end the chat.
Client: hello
Server: hi
Client: how are you
Server: i am fine, what about you?
Client: great! what you doing?
Server: chatting with you :)
Client: bye
Chat ended.

C:\Users\Hamid Ashaq\Desktop\Python\Secure Chat program\Secure-Chatting
>
```

## ■ Registration Communication

The top screenshot shows a Wireshark capture of registration communication. The packet list table is as follows:

No.	Source	Destination	Protocol	Length	Info
406112	127.0.0.1	127.0.0.1	TCP	56	51624 → 12345 [SYN] Seq=0 Win=
406167	127.0.0.1	127.0.0.1	TCP	56	12345 → 51624 [SYN, ACK] Seq=0
406232	127.0.0.1	127.0.0.1	TCP	44	51624 → 12345 [ACK] Seq=1 Ack=
029150	127.0.0.1	127.0.0.1	TCP	52	51624 → 12345 [PSH, ACK] Seq=1
029220	127.0.0.1	127.0.0.1	TCP	44	12345 → 51624 [ACK] Seq=1 Ack=
029292	127.0.0.1	127.0.0.1	TCP	45	51624 → 12345 [PSH, ACK] Seq=9
029300	127.0.0.1	127.0.0.1	TCP	44	12345 → 51624 [ACK] Seq=1 Ack=
029514	127.0.0.1	127.0.0.1	TCP	46	12345 → 51624 [PSH, ACK] Seq=1
029535	127.0.0.1	127.0.0.1	TCP	44	51624 → 12345 [ACK] Seq=10 Ack=
504601	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host ...)
503748	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host ...)
366382	127.0.0.1	127.0.0.1	TCP	156	51624 → 12345 [PSH, ACK] Seq=1
366422	127.0.0.1	127.0.0.1	TCP	44	12345 → 51624 [ACK] Seq=3 Ack=
368056	127.0.0.1	127.0.0.1	TCP	83	12345 → 51624 [PSH, ACK] Seq=3
368080	127.0.0.1	127.0.0.1	TCP	44	51624 → 12345 [ACK] Seq=122 Ac
999523	192.168.201.1	192.168.201.1	ICMP	128	Destination unreachable (Host ...)

The bottom screenshot shows the Command Prompt output for the registration process:

```
C:\Users\Hamid Ashaq\Desktop\Python\Secure Chat program\Secure-Chatting
>python server.py
Server is listening on port 12345...
Connected by ('127.0.0.1', 51624)

C:\Users\Hamid Ashaq\Desktop\Python\Secure Chat program\Secure-Chatting
>python client.py
Connected to the server.
Choose an option: 'register' or 'login': register
Enter email: test3@gmail.com
Enter username: encryptiontest
Enter password: wireshark
Registration successful. Please log in.
Enter username:
```



## 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.

