Assignment # 2
Information security

Name: Munam Mustafa

Roll No. 21i-0460

Section: D

Introduction

This report outlines the design and implementation process for a **Secure Chat System**, where users can register and log in securely to a server, engage in encrypted communication, and ensure the confidentiality of messages exchanged. The system integrates essential cryptographic techniques, such as **password hashing**, **key exchange** via **Diffie-Hellman**, and **symmetric encryption** using **AES**, to safeguard user data and communications.

Objectives

The primary objectives of this system are:

1. **User Registration**: To allow users to create accounts with a unique username and password.
2. **User authentication**: To verify user credentials during login using password hashing and encryption techniques.
3. **Encrypted Communication**: To enable encrypted communication between the client and server after successful login, ensuring the confidentiality of the exchanged messages.
4. **Credential Storage**: To securely store user credentials (hashed passwords and salts) in a file.
5. **Message Confidentiality**: To ensure that all chat messages remain confidential and are only accessible by the intended recipients.

Key Components

1. **User Registration**:
    - Users can create accounts by providing a valid email, a unique username, and a password.
    - The system ensures that passwords are stored securely using a hashing mechanism (SHA-256) with added **salts** for security.
2. **User Login**:
    - During login, the server authenticates users by verifying the hashed password against the stored hash in the credentials file.
    - Both client and server use **Diffie-Hellman** for secure key exchange, followed by the use of **AES-128** encryption for securing the communication.
3. **Encrypted Chat**:
    - After successful authentication, the client and server establish an encrypted channel using a shared secret key (derived from Diffie-Hellman).
    - All chat messages are encrypted using **AES-128 CBC mode**, ensuring that the content of the messages is not exposed during transmission.

4. **Credential Storage**:
    - User credentials (username, email, hashed password, and salt) are stored securely in a file (e.g.,creds.txt) on the server. The credentials file is accessible only by the server.

Detailed Design and Functionality

Registration Process

- **Pre-Phase**: Before registration, a secret key is exchanged using the **Diffie-Hellman Key Exchange** protocol. This ensures that both the client and server agree on a mutual key.
    - Public parameters are fixed for both the client and server.
    - Both parties generate their own secret keys and compute the shared secret key k-shared.
- **User Prompt at Client Side**: The client requests the user to input:
    - A **valid email address**.
    - A **unique username**.
    - A **password**.
- **Encryption of Credentials**: The user-provided email, username, and password are encrypted using **AES-128 CBC mode** and the shared secret key (`k_shared`) derived during Diffie-Hellman key exchange.
- **Server End**:
    - The server decrypts the user credentials using the shared key (`k_shared`).
    - The server checks for **username uniqueness** by verifying if the username already exists in the credentials file (`creds.txt`).
    - If the username is unique, the server generates a **random salt** (at least 32 bits) and **hashes the password** using the **SHA-256** hashing algorithm along with the salt.
    - The server stores the following details in `creds.txt`:
        - User's **email address**,
        - **username**,
        - **hashed password**,
        - **salt value**.

Login Process

- **User Prompt**: The client prompts the user for **a username** and **password**.
- **Secret Key Exchange Phase**: The client and server exchange keys using **Diffie-Hellman** again, using the same public parameters as the registration phase but with different secret keys to compute a new shared secret key (k-shared).

- **Encrypted Login**: The client encrypts the **username** and **password** using **AES-128 CBC mode** with the shared key (k-shared) and sends the ciphertext to the server.
- **Password Verification at Server**:
  - The server decrypts the received ciphertext using the shared key (k-shared).
  - The server retrieves the stored **hashed password** and **salt** for the provided username from the creds.txt file.
  - The server hashes the entered password using the same salt and **SHA-256** and compares it with the stored hash. If they match, the login is successful.
- **Access Control**: If the password matches, the user is granted access to the chat system. If it fails, the user is prompted to try again, and a new Diffie-Hellman key exchange is performed for security.

Encrypted Chat

- **Key Exchange for Message Encryption**:
  - After login, a new Diffie-Hellman exchange is performed to calculate the shared secret key for encrypting chat messages.
  - The shared key is appended with the **username**, which is used to derive the final encryption key.
- **Message Flow**:
  - The client and server exchange encrypted chat messages. Each message is encrypted with **AES-128 CBC mode** before being sent and decrypted by the receiver.
- **Termination**: The session continues until either the client or server types bye, indicating the end of the conversation.

Credential Storage and Key Management

- **File Storage**: The server stores user credentials (username, hashed password, salt) in a secure file (Creds.txt).
- **Storing Secure Passwords**: Passwords are never stored in plaintext. They are always **hashed** using **SHA-256** along with a **salt** to prevent password recovery in the event of a file breach.
- **Retrieving Credentials**: During login, the server retrieves the hashed password and salt from the credentials file, hashes the entered password, and compares it with the stored hash to authenticate the user.

Security Considerations

1. **Password Security**:

    - **SHA-256** hashing is used to ensure passwords are stored securely.
    - **Salting** ensures that identical passwords have different hashes, protecting against **rainbow table** attacks.
2. **Confidentiality**:
    - The entire communication between the client and server, including login and chat messages, is encrypted using **AES-128 CBC mode**.
    - **Diffie-Hellman** ensures that the encryption keys are securely exchanged without them being exposed to third parties.
3. **File Security**:
    - The credential file (credts.txt) is stored with restricted access permissions to prevent unauthorized access.
    - Optionally, the creds.txt file can be encrypted for additional security.

System Architecture and Workflow

Client-Server Model

- **Client**: initiates registration and login requests and sends encrypted messages to the server.
- **Server**: Manages user accounts, performs authentication, and facilitates encrypted message exchange.

Registration Workflow

1. The client sends a registration request.
2. The server verifies the username uniqueness and encrypts the registration details.
3. The server stores the encrypted credentials and the hashed password in the credentials file.

Login Workflow

1. The client sends a login request.
2. The server decrypts the login details, verifies the password, and grants access if successful.

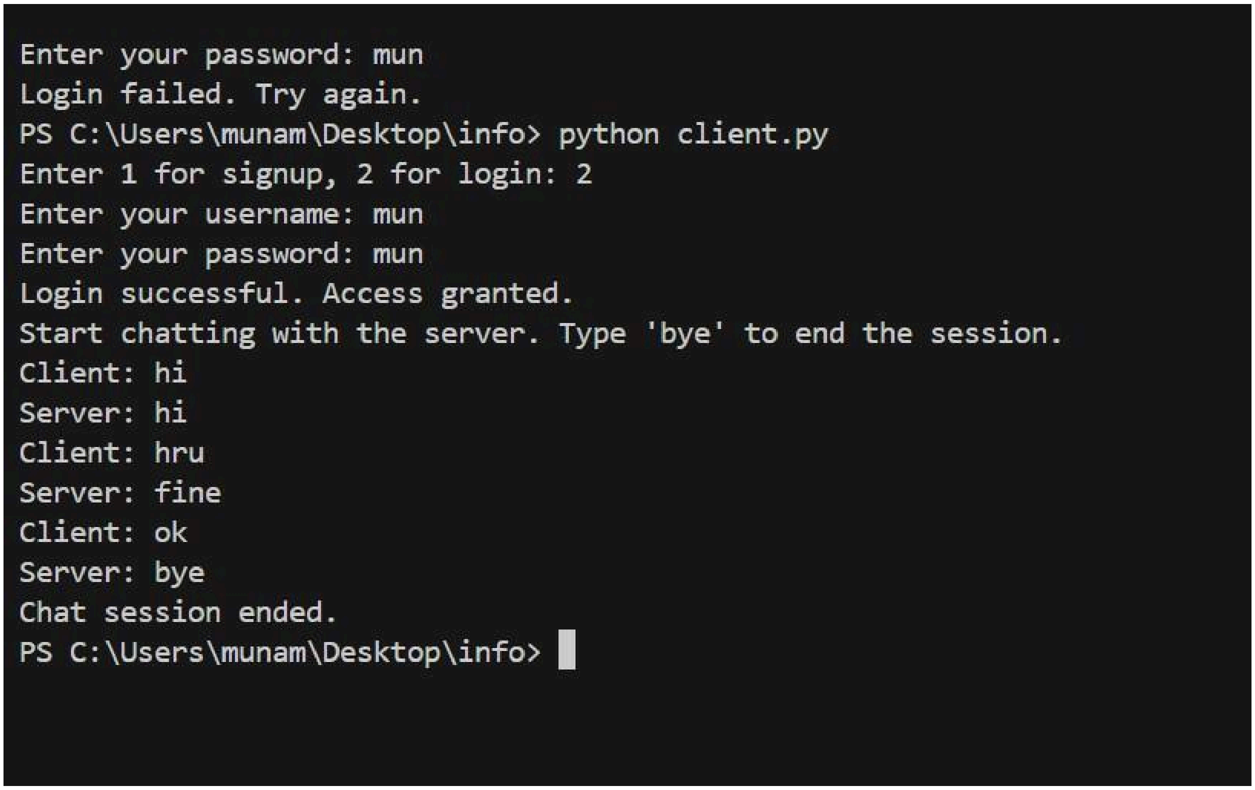
Chat Workflow

1. After successful login, the client and server exchange a shared secret key for message encryption.
2. Messages are encrypted using **AES-128 CBC mode** before transmission and decrypted upon receipt.

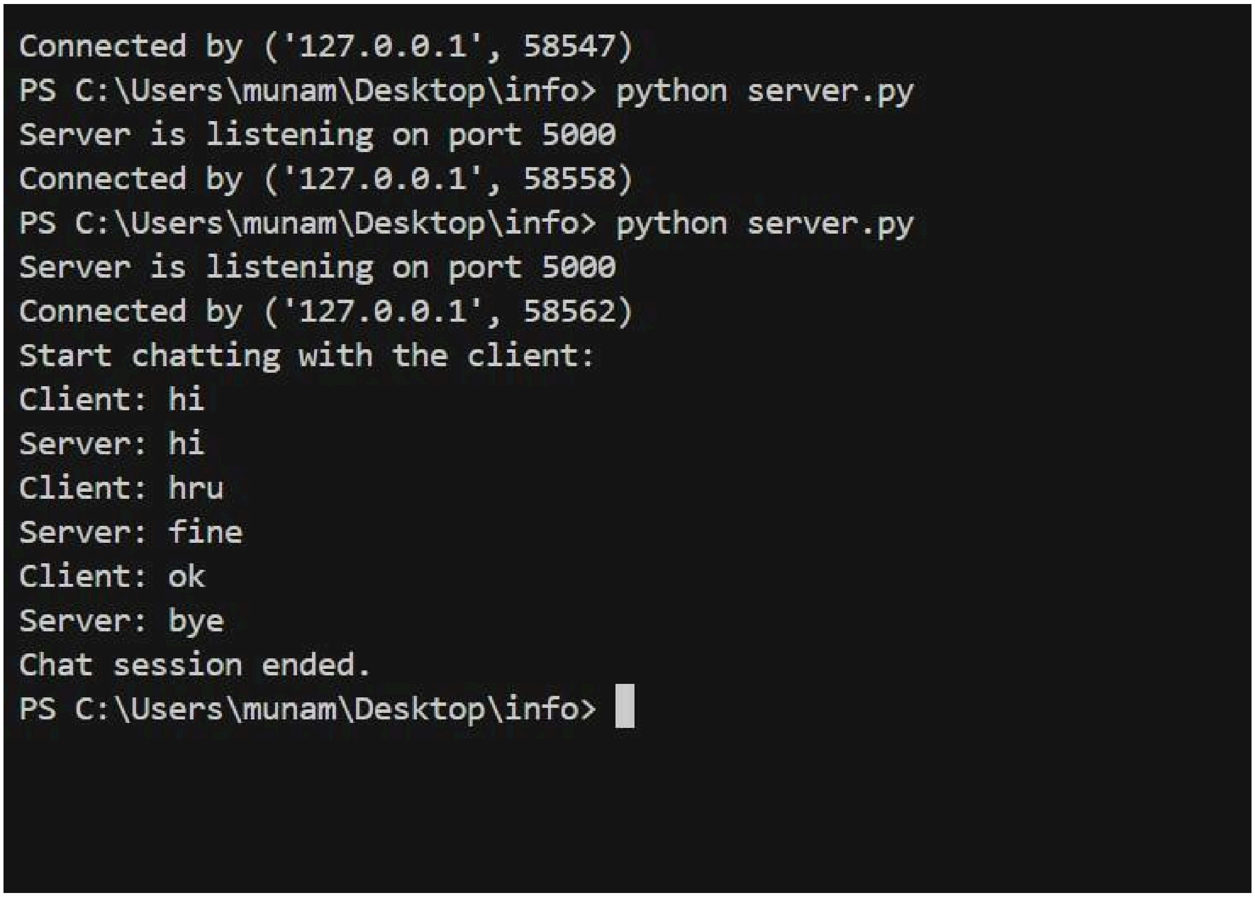
Testing and Evaluation

Functional Testing

- **Registration**: Ensure new users can register with a unique username and password. Verify that passwords are hashed and stored securely.
- **Login**: Test login with valid credentials and ensure that invalid login attempts are handled correctly.



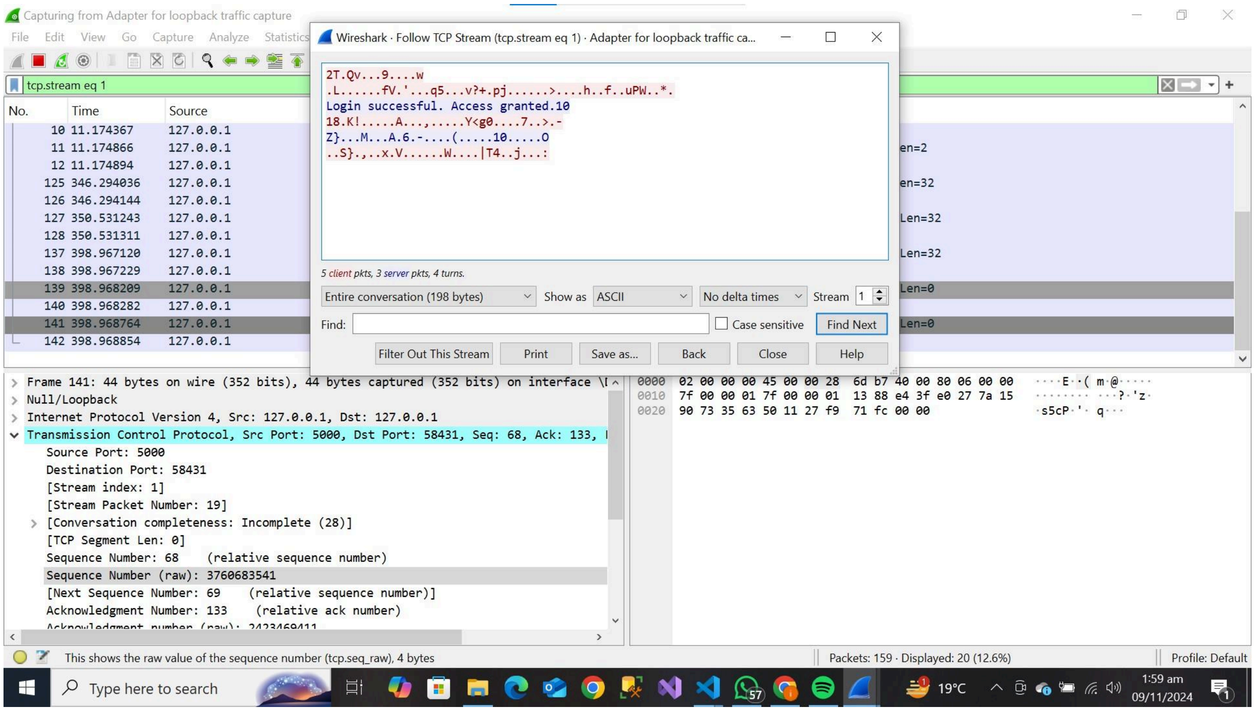
```
Enter your password: mun
Login failed. Try again.
PS C:\Users\munam\Desktop\info> python client.py
Enter 1 for signup, 2 for login: 2
Enter your username: mun
Enter your password: mun
Login successful. Access granted.
Start chatting with the server. Type 'bye' to end the session.
Client: hi
Server: hi
Client: hru
Server: fine
Client: ok
Server: bye
Chat session ended.
PS C:\Users\munam\Desktop\info>
```



```
Connected by ('127.0.0.1', 58547)
PS C:\Users\munam\Desktop\info> python server.py
Server is listening on port 5000
Connected by ('127.0.0.1', 58558)
PS C:\Users\munam\Desktop\info> python server.py
Server is listening on port 5000
Connected by ('127.0.0.1', 58562)
Start chatting with the client:
Client: hi
Server: hi
Client: hru
Server: fine
Client: ok
Server: bye
Chat session ended.
PS C:\Users\munam\Desktop\info>
```

Security Testing

- **Hash Verification**: Ensure that identical passwords generate different hashes due to salting. Verify correct password hashing during login.
- **Encryption Testing**: Use tools like **Wireshark** to capture communication between client and server and verify that all messages, including registration and login phases, are encrypted.



Capturing from Adapter for loopback traffic capture

tcp.stream eq 1

```
No.   Time        Source
   10 11.174367   127.0.0.1
   11 11.174866   127.0.0.1
   12 11.174894   127.0.0.1
  125 346.294036  127.0.0.1
  126 346.294144  127.0.0.1
  127 350.531243  127.0.0.1
  128 350.531311  127.0.0.1
  137 398.967120  127.0.0.1
  138 398.967229  127.0.0.1
  139 398.968209  127.0.0.1
  140 398.968282  127.0.0.1
  141 398.968764  127.0.0.1
  142 398.968854  127.0.0.1
```

```
en=2
en=32
Len=32
Len=32
Len=0
Len=0
```

Wireshark · Follow TCP Stream (tcp.stream eq 1) · Adapter for loopback traffic ca...

```
2T.Qv...9....w
.L......fV.'...q5...v?+.pj......>....h..f..uPW..*.
Login successful. Access granted.10
18.K!.....A...,.....Y<g0....7..>.-
Z}...M...A.6.-....(.....10.....O
..S}.,..x.V......W....|T4..j...:
```

5 client pkts, 3 server pkts, 4 turns.

Entire conversation (198 bytes) · Show as: ASCII · No delta times · Stream: 1

Find: □ Case sensitive

```
> Frame 141: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \I
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v Transmission Control Protocol, Src Port: 5000, Dst Port: 58431, Seq: 68, Ack: 133, I
    Source Port: 5000
    Destination Port: 58431
    [Stream index: 1]
    [Stream Packet Number: 19]
  > [Conversation completeness: Incomplete (28)]
    [TCP Segment Len: 0]
    Sequence Number: 68    (relative sequence number)
    Sequence Number (raw): 3760683541
    [Next Sequence Number: 69    (relative sequence number)]
    Acknowledgment Number: 133    (relative ack number)
    Acknowledgment number (raw): 2423469411
```

```
0000  02 00 00 00 45 00 00 28  6d b7 40 00 80 06 00 00   ····E··( m·@·····
0010  7f 00 00 01 7f 00 00 01  13 88 e4 3f e0 27 7a 15   ········ ···?·'z·
0020  90 73 35 63 50 11 27 f9  71 fc 00 00               ·s5cP·'· q···
```

This shows the raw value of the sequence number (tcp.seq_raw), 4 bytes

Packets: 159 · Displayed: 20 (12.6%)

Profile: Default

Conclusion

The secure chat system utilizes strong cryptographic techniques like **Diffie-Hellman** for key exchange and **AES-128 CBC mode** for encryption, ensuring the confidentiality and security of both user credentials and chat messages. The use of **SHA-256 hashing** with **salting** for password storage adds an additional layer of security, preventing password compromise. The system is designed to be robust, secure, and scalable for real-world applications where secure communication is paramount.