Name: Sameed Ahmed Siddiqui

Roll-no: 22i-8223

Section: B

Course: Information Security

SecureChat

The SecureChat project implements a basic secure chat client-server architecture using Python. Communication is performed over plain TCP, but security is achieved using Diffie-Hellman (DH) key exchange for session key derivation and AES-128 for encrypting messages. Additionally, RSA signatures are used to verify client authenticity during login.

Key Components

1. Client (`client.py`)

- Connects to server over TCP.
- Performs a handshake with the server to receive a random challenge and server DH public key.
- Derives AES session key from DH key exchange.
- Encrypts password using the session key and sends it along with a signed challenge to authenticate.
- Can send and receive encrypted messages using the AES session key.

Important Classes & Functions:

- `SecureClient`
- `connect()`
- `handshake()`
- `establish_session_key()`
- `login()`
- `send_message()` / `receive_message()`

---

2. Server (`server.py`)

- Listens for incoming TCP connections.
- Performs handshake with the client, sending a random challenge and DH public key.
- Receives login message, verifies challenge signature using client public key.
- Derives AES session key from DH exchange.
- Decrypts password using AES session key to verify login.
- Handles encrypted messaging between clients.

Important Classes & Functions:

- `SecureServer`
- `handle_client()`
- `send_json()` / `recv_json()`
- `run()`

Identified Issues and Fixes

1. `\*\* model missing \*\*`

2. Original model:

```
class ServerHello(BaseModel):
    server_version: str = "1.0"
    challenge: str
```

- Fix:

```
class ServerHello(BaseModel):
    server_version: str = "1.0"
    challenge: str
    dh_pub: str
```

- This ensures the client can retrieve the server DH public key to derive the session key.

- **Client login before session key derivation**

- Original sequence: client tried login before deriving session key.

- Fixed sequence:

    1. Handshake to get `challenge` and `dh_pub`
    2. Establish session key using `dh_pub`
    3. Login using AES-encrypted password

- **AES password encryption/decryption**

- Password is encrypted on the client side using AES and the DH-derived session key.

- Server decrypts the password using the same session key before verifying against the database.

- **Error Handling**

- Added check for `NoneType` when accessing `dh_pub` from `ServerHello`.
- Fixed AttributeError caused by missing `dh_pub`.

Final Workflow

Client:

1. Connect to server.
2. Handshake: send username, receive `ServerHello` (challenge + DH pub key).
3. Derive AES session key using DH.
4. Login: encrypt password using session key, sign challenge, send to server.
5. Send and receive encrypted messages.

Server:

1. Accept client connection.
2. Handshake: generate DH keypair, send `ServerHello` with challenge + DH pub key.
3. Receive login message.
4. Verify challenge signature.
5. Derive session key using DH.
6. Decrypt password using session key.
7. Verify password against database.
8. Receive and relay encrypted messages.

Notes

- All messages between client and server are JSON-encoded.
- Base64 encoding is used to safely transmit binary data (AES ciphertext, DH keys, signatures).
- Transcript logging is performed for both client and server for auditing purposes.
- This report fixes the `NoneType` error and establishes the correct order of operations for secure login.