SecureCom: Integrating Client Authentication, Message Encryption and Verification

Name: Tejas Jamdade
Roll No: B20MT049

Client Authentication

- As of now, I have created a dummy database (using a .csv file) that consists of the registered users. The database consisted of 'id', 'username' and 'token'. The token is created using a password hashing algorithm, `SHA-256`. The hash is calculated for the credentials of the user, i.e. the username and password.

	A	B	C
1	id	user_id	token
2	x01	Alice	d6db90c1ad25a4102859da1c9048f91eee5589ab4095a4f19a95c4b0d3470348
3	x02	Bob	360aaaa6d867cf44c745853fb4e94259fc244f327288ffd0bb47eb100e59f689
4	x03	Jack	34592cd468eba551260025f13fed7341b5e31c6a190bbcf5e79fdff8fbe41f8d

Key Generation

- For key generation, I have used the Diffie Hellman Key Exchange protocol.
- This key-exchange scheme is based on the factorisation problem:
  - Public Key:

```
from Crypto.Util.number import getPrime

# Generate a 256-bit prime number for N
p = getPrime(128)
q = getPrime(128)
n = p*q

g = int.from_bytes(get_random_bytes(16), byteorder='big') % N
```

    - n → Product of two primes, p and q.
    - g → Generator of the Group Z_N

- To factorize, one needs to iterate over all the prime numbers less than N.
  - To calculate the number of primes, we can use the Prime Number theorem, which is defined as follows:
  - $\boxed{\pi(n) = \frac{n}{ln(n)}}$
- In our case, n is 256 bits; and therefore $\pi(n) \approx 2^{248}$ bits, which is not a searchable key space. Hence, preventing a brute force attack.

◦ Private Key:
  - Alice generates a random number 'i_1' of 16 bytes i.e. 128 bits, and sends g^{i_1} to the Server.
  - Similarly, the Server generates a random number, 'j' and sends g^j to Alice.
  - Shared secret: $\boxed{g^{i_1 * j}}$

- Based on the Diffie Hellman Problem, we know that knowing g and g^i, one can't find i [as it is a **discrete log problem**]

Key Exchange

Client-Server

- Just after the client is online and connects to the server, the client performs a key exchange with the server.
- The Public Key, i.e. `g` and `n`, are already known to both the server and the client.
- `Client_1` or `Alice` generates i_1 (of size - 16 bytes, i.e. 128 bits) and sends g^{i_1} to the server.
- The server generates j (of size - 16 bytes) and sends g^j to Alice.
- This is how the shared secret is generated: $g^{i_1 * j}$
- Using this shared secret key, the AES key is generated that will be used to encrypt and decrypt the communication between Alice and the server

💡 **Note**:

- **The server stores g^i of each client, which will be later used for client-client key exchange.**

Client-Client

Client 1:

- Whenever `Client_1` wishes to communicate with other clients, it first needs to exchange keys with that client.
- So, in the sending mode, `Client_1` enters the 'username' of the receiver client then Server sends the g^i of that client.
- For e.g. `Client_1` wants to communicate with `Client_2`, then the server will send g^{i_2} to `Client_1`, and then Client 1 can generate the key using the shared secret.
- Using this shared secret, Client 1 will send the encrypted messages for Client 2 to the server.
    - Although, it must be noted that this encrypted message {encrypted by the key between Client 1 and Client 2} is again encrypted by the Key between `Client_1` and the server.

Client 2:

- The server will send g^i of `Client_1` to this client along with the encrypted messages.
- `Client_2` will decrypt the message, and this is how end-to-end encryption is ensured.

💡 **Note**:

- **Since the server doesn't have access to the key between `Client_1` and `Client_2` i.e., $\boxed{g^{i_1 * i_2}}$. The server won't be able to decrypt and manipulate the ciphertext.**
- **So any Man-in-the-middle attack can be prevented.**

AES-GCM mode

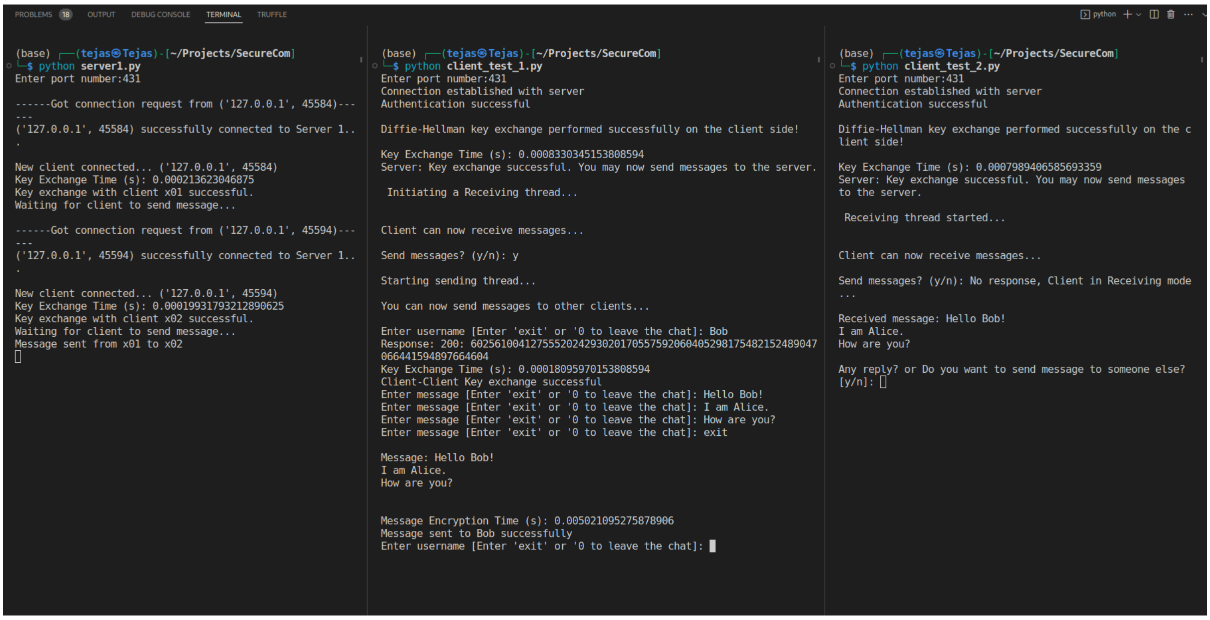
Message Encryption

- For message encryption, I have used the GCM mode of AES implemented by PyCryptodome.
- The GCM or Galois/Counter mode is defined by NIST SP 800-38D.
- Key size: 16 bytes or 128 bits.

Message Verification

- AES GCM mode uses an authentication tag to verify the integrity of the message.
- The authentication tag is generated during the encryption process and is appended to the ciphertext.

Outputs



Server:

```
(base) ┌──(tejas㉿Tejas)-[~/Projects/SecureCom]
└─$ python server1.py
Enter port number:625

------Got connection request from ('127.0.0.1', 44302)------
('127.0.0.1', 44302) successfully connected to Server 1...
```

```
New client connected... ('127.0.0.1', 44302)
Key Exchange Time (s): 0.0001735687255859375
Key exchange with client x01 successful.
Waiting for client to send message...

------Got connection request from ('127.0.0.1', 39826)------
('127.0.0.1', 39826) successfully connected to Server 1...

New client connected... ('127.0.0.1', 39826)
Key Exchange Time (s): 0.0001933574676513672
Key exchange with client x02 successful.
Waiting for client to send message...
Message sent from x01 to x02
```

Client 1 [Alice]

```
(base) ┌──(tejas㉿Tejas)-[~/Projects/SecureCom]
└─$ python client_test_1.py
Enter port number:625
Connection established with server
Authentication successful

Diffie-Hellman key exchange performed successfully on the client side!

Key Exchange Time (s): 0.0007619857788085938
Server: Key exchange successful. You may now send messages to the server.

 Initiating a Receiving thread...


Client can now receive messages...

Send messages? (y/n): y

Starting sending thread...

You can now send messages to other clients...

Enter username [Enter 'exit' or '0 to leave the chat]: Bob
Response: 200: 4571245969679352963874562863653476580164616981900411687129652052745963544331‌4
Key Exchange Time (s): 0.00021219253540039062
Client-Client Key exchange successful
Enter message [Enter 'exit' or '0 to leave the chat]: Hello Bob!
Enter message [Enter 'exit' or '0 to leave the chat]: I am Alice.
Enter message [Enter 'exit' or '0 to leave the chat]: How are you?
Enter message [Enter 'exit' or '0 to leave the chat]: exit

Message: Hello Bob!
I am Alice.
How are you?


Message Encryption Time (s): 0.00840902328491211
Message sent to Bob successfully
Enter username [Enter 'exit' or '0 to leave the chat]:
```

Client 2 [Bob]

```
(base) ┌──(tejas㉿Tejas)-[~/Projects/SecureCom]
└─$ python client_test_2.py
Enter port number:625
Connection established with server
Authentication successful

Diffie-Hellman key exchange performed successfully on the client side!

Key Exchange Time (s): 0.0008063316345214844
Server: Key exchange successful. You may now send messages to the server.

 Receiving thread started...


Client can now receive messages...

Send messages? (y/n): No response, Client in Receiving mode...

Received message: Hello Bob!
I am Alice.
How are you?

Any reply? or Do you want to send message to someone else? [y/n]:
```