

PROBLEM SET 3

1. Propose an architecture and design for a secure instant messaging system. The architecture can have a server (but not necessarily, it is up to you to make the decision). If a server is used the messages between the users should not go through the server (with the exception of the first discovery messages).

**Ans:** The proposed architecture is as shown below. It is mainly divided into two parts. The first part deals with the communication between client and server and then the second deals with the communication between the two clients.

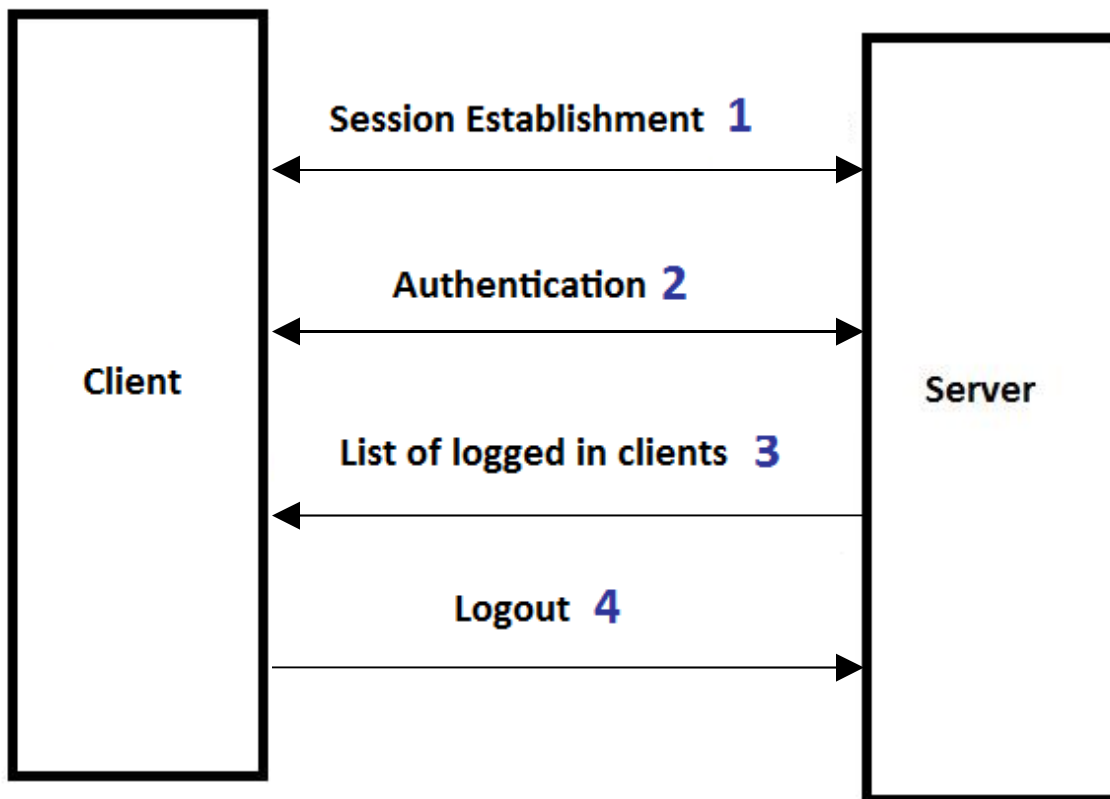
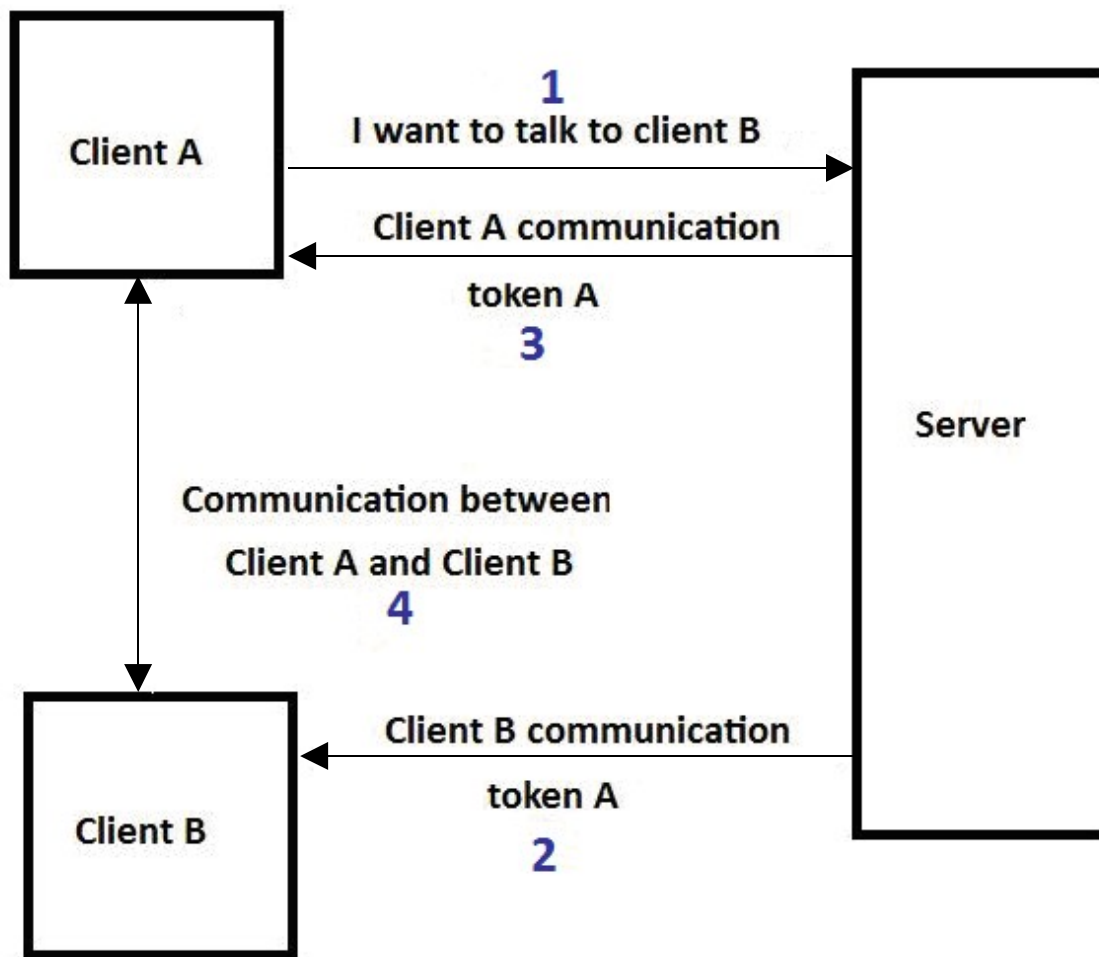


fig a. Client – to – server Communication



**fig b. Client – to – client communication**

As shown above we have implemented the use of a server in our architecture. The server is mainly used to establish a session with the clients and between them; however, it does not receive any communication that occurs between the clients. The following steps occur while initiating communication between client A and client B.

- 1) First the client A communicates with the server and registers itself to the server. If any client request comes up, it establishes a session key between the clients from the server.
- 2) If client A wants to talk with client B it requests the server using the session key established in step 1. The server on receiving this request, issues a token to both client A and client B as shown in diagram in step 2 and 3 respectively.
- 3) On receiving this token both clients A and B establish a new session key between the clients and from here on the communication between A and B occurs directly without the server using this new session key.

**2. Assume that the users only need to remember a single password. Your system should provide mutual authentication (user and server), message integrity and confidentiality.**

**Ans:** We are proposing a system in which the user only needs to remember his username and password. First by knowing the password, the quantity  $W$  is derived by hashing the password. Then quantity  $W'$  is derived by adding salt to “ $W$ ”. This is done to improve security against database attacks, as  $W$  is in its pure form would be weak and it is stored in the database.

By encrypting each communication with the session key we are ensuring confidentiality of the data. Our Design provides mutual authentication between server and client and between clients. Mutual Authentication ensures from both clients are talking to each other.

We are encrypting the messages with  $W'$  which includes hashing function, providing message integrity and confidentiality. Also, we are providing message integrity by signing the authentication details and then sending the hash of the message along with it. The authentication details are also securely encrypted to maintain its confidentiality and the hash protects from someone tampering the message which could be verified at the other end. Details of how the protocol works would be explained in answer to question 3.

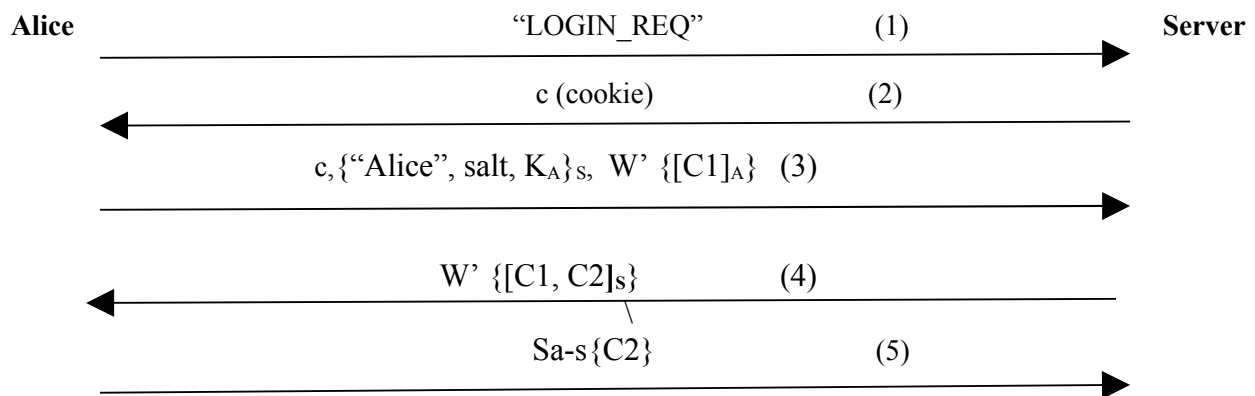
**3. Describe in detail the security protocols that you are proposing for the whole system. Remember that you are not allowed to use SSL or a complete existing protocol. You are allowed to use cryptographic libraries that provide encryption, hashing, etc.**

**Ans:** The protocol implemented has 4 phases:

- a. Session establishment and mutual authentication,
- b. Request for online users
- c. Client to Client Communication
- d. Logout

**a) Session Establishment and Mutual Authentication**

**The new design**



**$W = \text{hash}(\text{password})$**

**$W' = f(W, \text{salt})$**

**$C1, C2 : \text{Nonce}$**

**$S_{a-s} : \text{Session key between client and server}$**

**$S_{a-s} : f(C1, C2)$**

Step1: The client who wants to login sends a login request to the server.

Step2: The server sends a cookie to the client to prevent DOS attack and verifies if he is the person who requested for login.

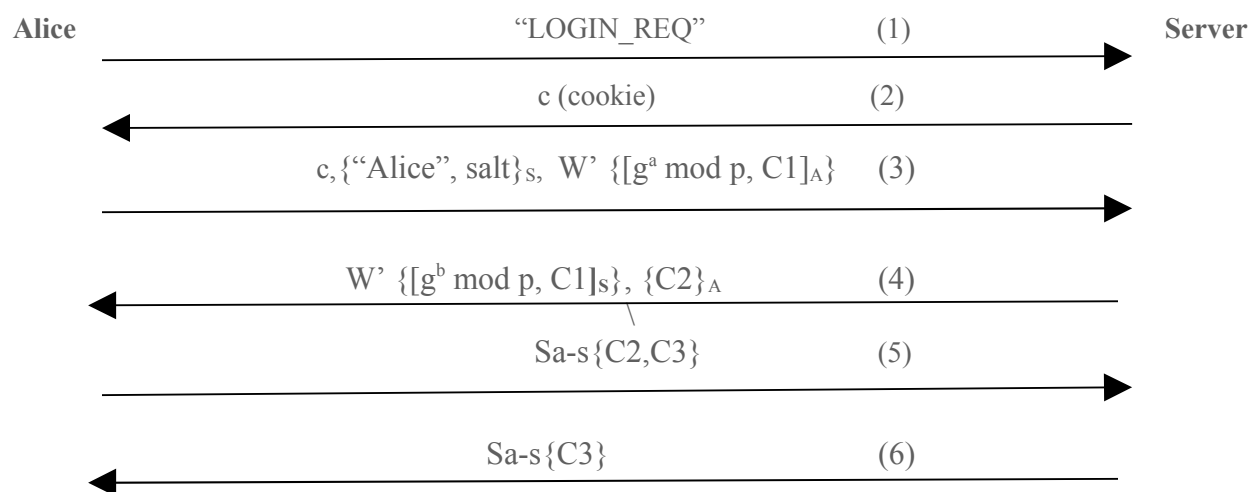
Step 3: The client returns the cookie and also sends his identity along with a salt encrypted with server's public key. He also sends a nonce (C1) digitally signed and further encrypted with  $W'$  (a function of  $W$  and salt). He also sends his public key so that the server can store it for client-client communication.

Step 4: The server replies with Nonce (C1) digitally signed and encrypted with  $W'$ . The server also sends another Nonce C2, for the client to authenticate back.

By providing C1, server authenticates to clients request.

Step 5: The session key is generated at the Server and the client which is a function of C1 and C2. Also, we can say that both the parties are actively involved in key establishment as both contribute for  $S_{a-s}$ . Client authenticates itself and thereby completing Mutual authentication by returning C2 securely encrypted with Session key.

### ***Old design:***



$W = \text{hash}(\text{password})$

$W' = f(W, \text{salt})$

$C1, C2, C3$  : Nonce

$Sa-s$  : Session key between client and server

$Sa-s : g^{ab} \bmod p$

Step1: The client who wants to login sends a login request to the server.

Step2: The server sends a cookie to the client to prevent DOS attack and verifies if he is the person who requested for login.

Step 3: The client returns the cookie and also sends his identity along with a salt encrypted with server's public key. He also sends the first part of diffie Hellman and a nonce (C1) digitally signed and further encrypted with  $W'$  (a function of  $W$  and salt).

Step 4: The server replies with the second half of Diffie Hellman and Nonce (C1) digitally signed and encrypted with  $W'$ . The server also sends another Nonce  $C2$ , encrypted with public key of the Client.

By providing  $C1$ , server authenticates to clients request.

Step 5: At the server and client generate a common session key by Diffie Hellman. The client authenticates itself by providing  $C2$  and the nonce  $C3$  to the server within the session key.

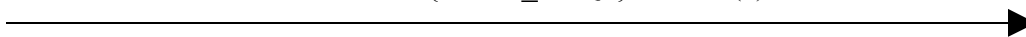
Step 6: To avoid any hijacking of the session we again request the server to send nonce  $C3$  within the session key.

### **b) Request for Online users (Remains Same)**

Alice

Server

$Sa-s\{\text{"LIST\_REQ"}\}$  (1)

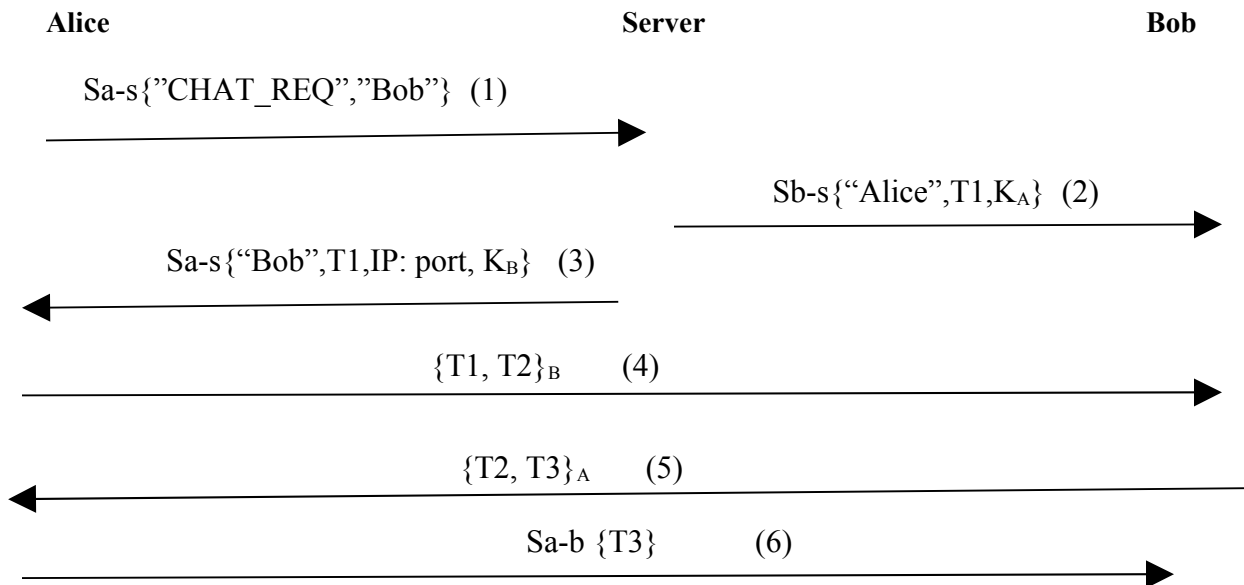


Alice asks for a current list of online users from the server for communication. The server responds with the online user list to Alice. The server comes to know that the request has come from Alice after looking at the IP address and Port details stored. Hence, we need not explicitly mention Alice inside the Session Key.

### **c) Client to Client communication**

In this method we discuss the protocol for talking between 2 clients without server figuring out the conversation.

## The new Design



$S_{b-s}$  : Session key establish between client 2 (Bob) and Server

$K_B$  : Public key of Client 2 (Bob)

$K_A$  : Public key of Client 1 (Alice )

$\{\}_A$  : Encrypted with public key of A

$T1, T2, T3$  : Tokens (Similar to Nonces)

$S_{a-b}$  : Session key establish between client 1 (Alice) and client 2 (Bob)

$S_{a-b} : f(T2, T3)$

Step 1: Alice sends a chat request to server specifying the client with whom she would like to chat within a session key between Server and Alice  $S_{a-s}$ .

Step 2: Server sends a token to Bob specifying that Alice would like to talk to you and also sends public key of Alice within the established session key between Bob and the Server  $S_{b-s}$ .

Step 3: The server sends the same token along with Bob's network details and Bob's public key within the session key  $S_{a-s}$ .

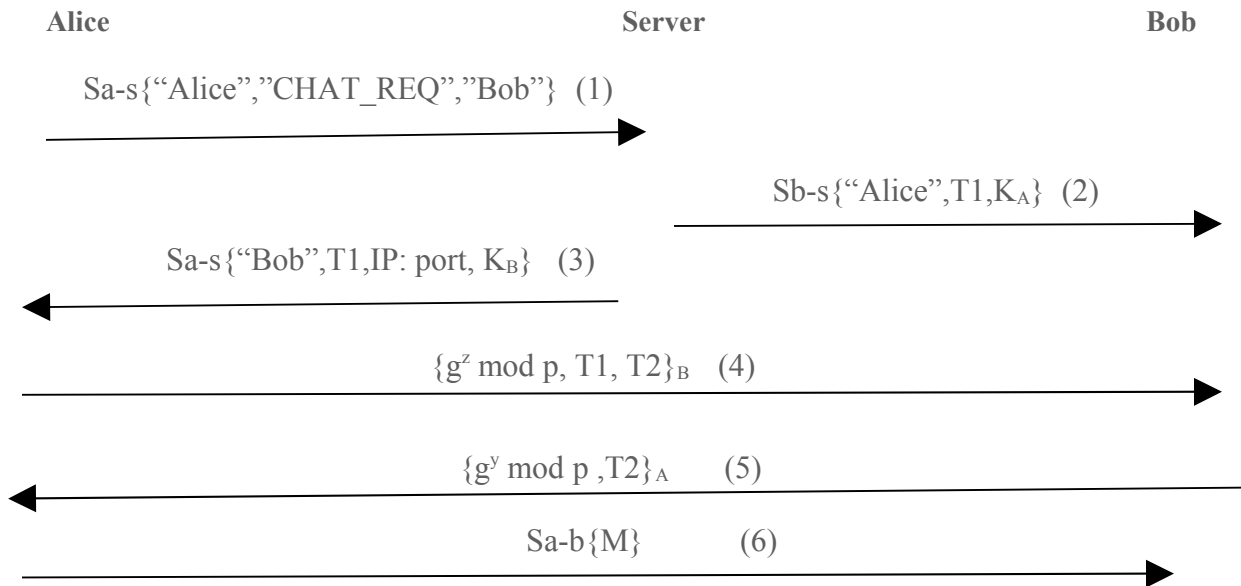
Step 4: Alice encrypts the Token  $T1$ , with Bobs Public key. The token  $T1$  is sent to Bob so that he can verify that the client he is talking is Alice. Also,  $T2$  is sent so that Bob can authenticate himself back.

Step 5: Token  $T2$  is encrypted with public key of Alice. Bob authenticates himself by providing

T2 within the public key of Alice. Also, T3 is sent so that session key can be generated from that and also to authenticate without any server component.

Step 6: Now session key is generated on both sides which is a function of T2 and T3, which is unknown to the server, and they communicate with each other. At this point Alice and Bob have mutually authenticated each other.

### *The old Design*



Sb-s : Session key establish between client 2 (Bob) and Server  
 $K_B$  : Public key of Client 2 (Bob)  
 $K_A$  : Public key of Client 1 (Alice )  
 $\{\}_A$  : Encrypted with public key of A  
 T1, T2 :  
 Sa-b : Session key establish between client 1 (Alice) and client 2 (Bob)  
 $Sa-b : g^{zy} \text{ mod } p$

Step 1: Alice sends a chat request to server specifying the client with whom she would like to chat within a session key.

Step 2: Server sends a token to Bob specifying that Alice would like to talk to you and also sends public key of Alice within the established session key.

Step 3: The server sends the same token along with Bob's network details and his public key

within the session key.

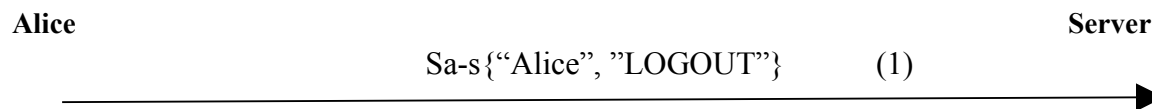
Step 4: Alice encrypts the Token T1, first part of Diffie Hellman and a token T2 with Bob's public key, received from the server. The token T1 is sent to Bob so that he can verify that the client he is talking is Alice.

Step 5: Bob sends second part of Diffie Hellman and Token T2 encrypted with public key of Alice. Bob authenticates himself by providing T2 within the public key of Alice. At this point Alice and Bob have mutually authenticated each other.

Step 6: Now session key is generated on both sides with Diffie Hellman, which is unknown to the server, and they communicate with each other.

The Server is only involved in the initial setup phase for providing Token and public keys and cannot take part in any communication between 2 clients.

#### **d) Logout Request (Remains Same)**



Alice no longer wishes to communicate and sends a logout request. The server verifies Alice within the session key and terminates the session with Alice.

#### **ASSUMPTIONS FOR THE PROTOCOL:**

- The client is pre registered and the hash of the password is stored at the server.
- For this IM within the class we are assuming that the server is the most secured authority.
- Server knows the list of clients and also their public keys (Firstly, the users have to forward their public keys each session)
- Client know the public key of the server
- Client can have only one session at a time with the server.

#### **4) Discuss the following issues:**

##### **a. Does your system protect against the use of weak passwords? Discuss both online and offline dictionary attacks.**

As we are using  $W'$  to encrypt, which is a function of  $W$  (weak secret derived from password) and salt. If there is an offline dictionary attack trial made on the value encrypted by  $W'$ , Intruder will not be successful as he does not know the salt which is generated by the client. The session key is a strong secret because an attacker would both have to guess the password and break the Session key.

**b. Is your design resistant to denial of service attacks?**

Our design is resistant to Denial of service attacks as we have used a feature called cookies. When a connection initiation request is received, server chooses a random number  $c$  which is unpredictable, is sent to the source IP address of the request. Server does not compute anything until it receives back the cookie  $c$  from that IP address.

Another method is to use a timeout session if the client fails to enter the password in 3 attempts. The time out session would be 10 min.

**c. To what level does your system provide end-points hiding, or perfect Forward secrecy?**

At any stage of the System design, the identity of the end point is not revealed in clear text and use of strong encryption to protect identities of all the clients. In case of Server getting compromised, identity of client is disclosed which is a rare case. Perfect forward secrecy is provided by generating Public and Private keys for each session. So even if anyone gets the keys he can only decrypt information for that session if recorded and hence, perfect forward secrecy is achieved.

**d. If the users do not trust the server can you devise a scheme that prevents the server from decrypting the communication between the users without requiring the users to remember more than a password? Discuss the cases when the user trusts (vs. does not trust) the application running on his workstation**

In our System when the client A wants to initiate a contact with Client B, Server passes same token to Client A and Client B. Both the clients generate a common session key using the token. Here the server does not know the session key. This session key is used for interaction between the clients. Thus, none of the conversation is passed through the server and the trust factor is resolved.