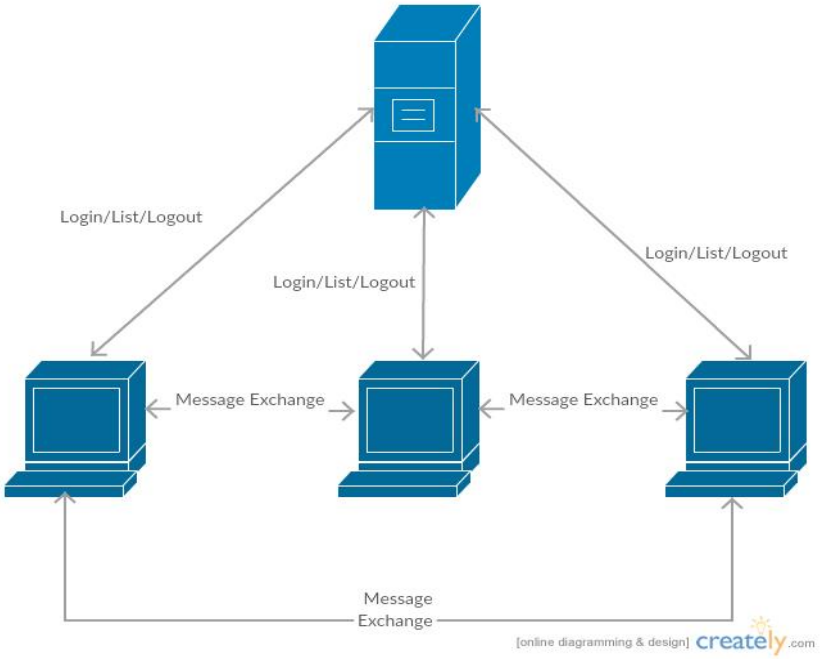
Secure Chat Application Design

Anoop Pai

Joyesh Kakkar

Architecture:



Assumptions:

- Server knows the session key required to communicate with other entities.
- Server has a public and private key of its own.

Server:

- Responsible for authenticating the client(s).
- Sets up the (initial) communication between the registered clients.
- Stores the username and hashed password for the registered users.
- Stores session key for the registered clients.

Client:

- Authenticates with server.
- Knows public key of server.
- Responsible for storing the session key to communicate with other entities.

- Exchanges messages with other clients.

Encryption:

To preserve the integrity and confidentiality of the message involved, we've used both symmetric and non-symmetric types of encryption in the form of AES-256 encryption in GCM Mode.

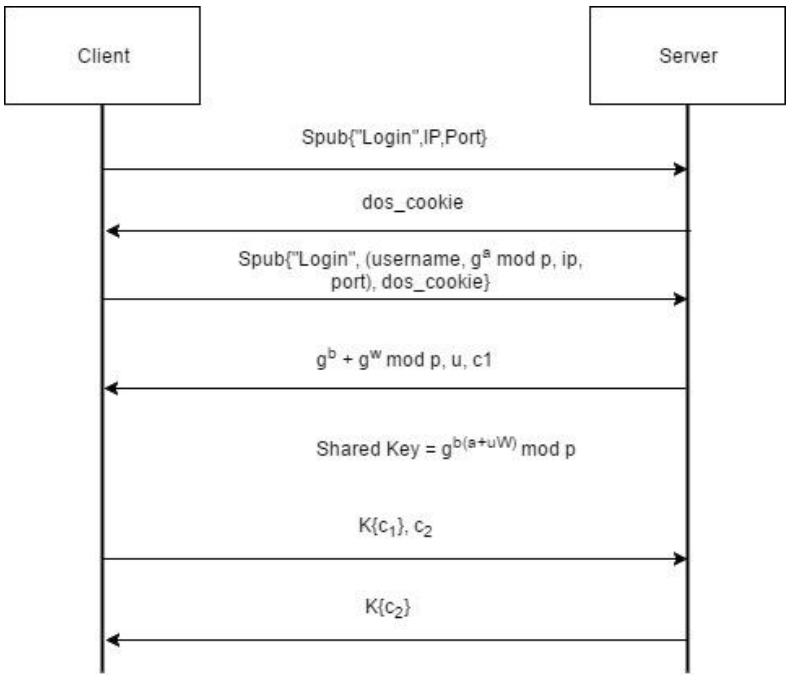
Protocols used:

Login Protocol:

- We've used SRP Protocol to provide authentication for both sides.
- DOS attacks are prevented by the means of a DOS cookie which is sent by the server.

SRP Justification:

- Usage of SRP makes it increasingly hard for the adversary to replicate the shared key unless the password is known.
- As challenge and response messages are encrypted with a shared key, confidentiality is ensured.
- Usage of SRP is also quite resilient to offline dictionary attacks. It would be, however, exposed to brute force attacks which are time-consuming and cumbersome in nature.
- SRP enables security for even weak passwords as a, b and u are generated randomly per session, and only known by the client(s) and the server.



Key Establishment Protocol

Client to Server:

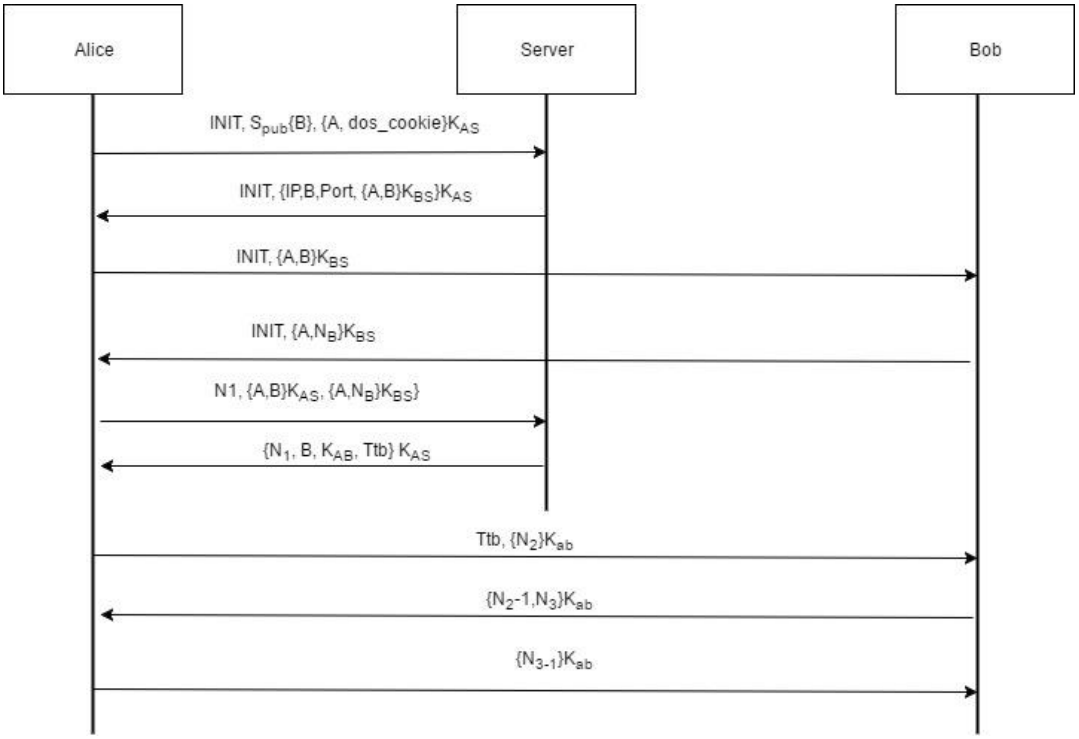
- As specified above, the shared session key will be $g^{b(a+Uw)}$ modp (via SRP). This ensures Perfect Forward Secrecy which adds extra security as it brings protection even if the password is compromised (previous communication cannot be decrypted).

Client to Client:

- The algorithm of choice for providing mutual authentication of the clients is Needham-Schroeder expanded algorithm.

Needham-Schroeder Justification:

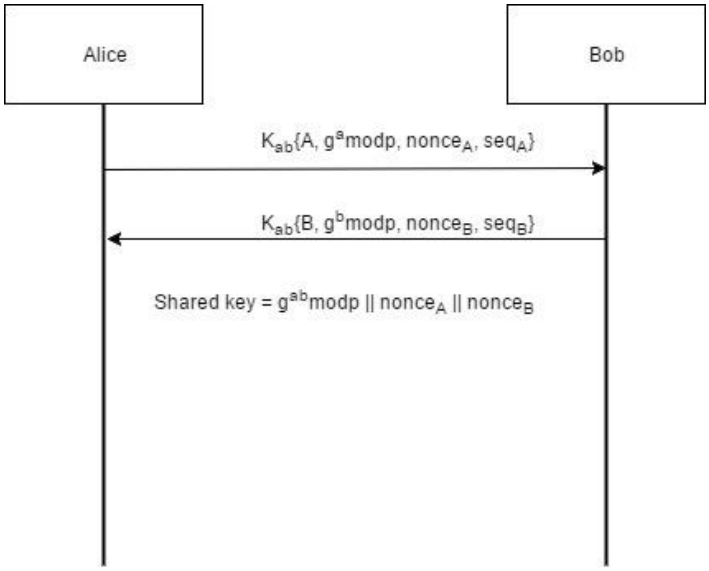
- It provides mutual authentication for the clients involved and thereby generating a session key which is shared by the client and the server for the communication ahead.
- Due to the use of nonce/salt, this algorithm is resilient to replay and reflection attacks.
- This algorithm makes sure an adversary doesn't impersonate a client involved in the communication.



where Ttb = {A, B, N_B, K_{AB}} $_{KBS}$

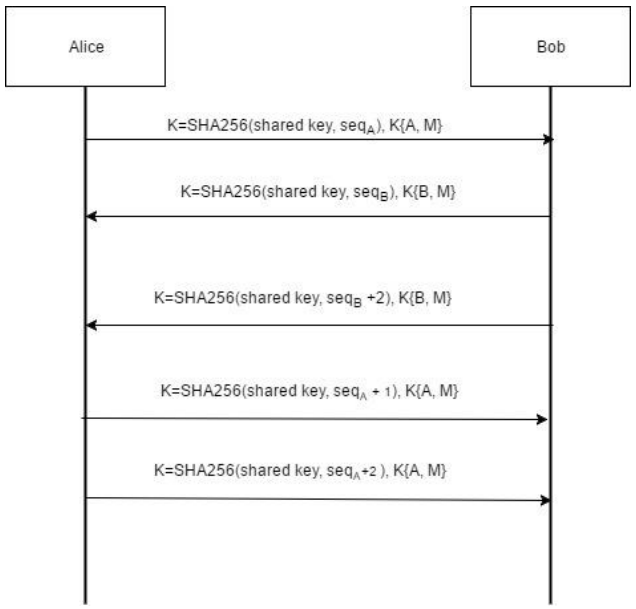
Messaging Protocol:

- We've used authenticated Diffie-Hellman key exchange by having the clients use the same session key. This way, we can ensure that any form of man-in-the-middle attacks are prevented.
- The initial key which is used for exchanging messages between the involved clients is the generated Diffie-Hellman key, where N_A and N_B being the nonce generated at Alice and Bob respectively.



Message exchange:

- As show in the below diagram, A uses DH shared key to encrypt the communication from A to B which involves a sequence number.
- Similarly, B does the same when it sends out a message; with the only exception that it uses a newly generated sequence number within the message.
- In the next exchange, A increments the sequence number of B by an order of 2 which results in the next sequence number. B, on the other hand, increments by an order of 1. (initiator sequence number by 1; receiver sequence number by 2)

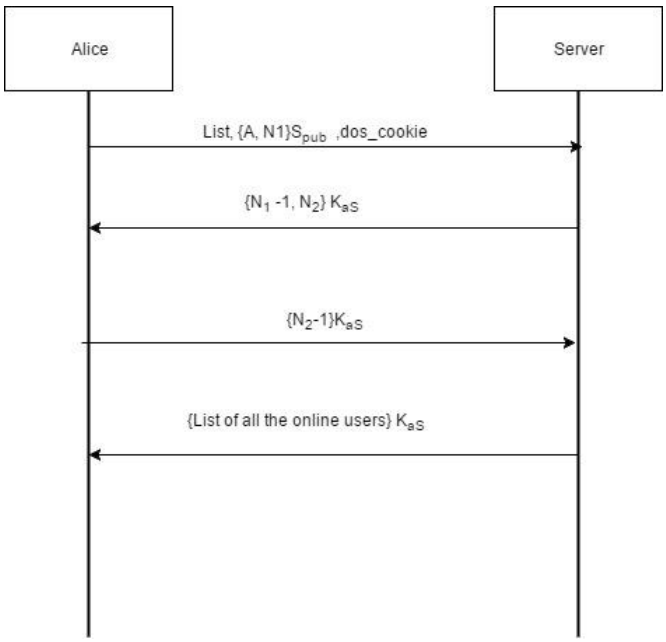


Justification:

- **Authentication:** As the secrets a and b are only known to A and B, for the adversary to generate the keys, he must solve the Diffie-Hellman problem.
- **Endpoint hiding**
- **End to end encryption**
- **Encryption:** The messages used between the parties in the communication is encrypted and therefore is resilient towards an eavesdropper, who is unable to decrypt these messages.
- **Perfect Forward Secrecy:** Although an adversary manages to gain access to g^a modp and g^b modp, it is impossible for him to attain the shared secret key.
- **Replay attack:** The adversary cannot make replay attacks by using the old messages as we have generated a new sequence number each time.

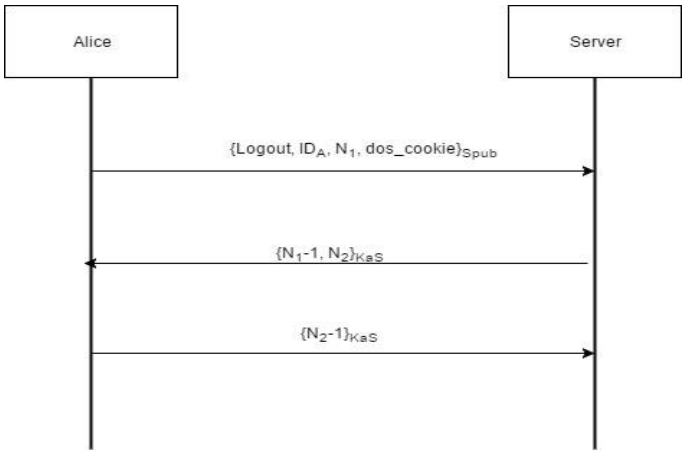
List Protocol

- First, the client sends a request to the server regarding the list command, together with a nonce and a DOS cookie. Next, this message is encrypted by the client using the server's public key.
- The server, then, decrements the nonce and sends another challenge which is encrypted using the client's shared session key.
- Next, the client sends the nonce after decrementing it, encrypting it with the shared session key.
- Finally, the server sends the list of all online clients to the client.



Logout Protocol

- First, the client sends a logout request to the server which contains the nonce and the DOS cookie. This is encrypted using the server's public key.
- The server, then, decrements the nonce and sends another challenge which is encrypted using the client's shared session key.
- Next, the client sends the nonce after decrementing it, encrypting it with the shared session key.
- The server, then, after deleting the shared session key and the DOS cookie, notifies all the connected clients.
- Finally, the clients delete their shared key.



Services

The services that are provided by our approach is as follows:

- Safety against DOS attacks: We prevent DOS attacks by the usage of a DOS cookie. A DOS cookie is attached with every request of the user, ensuring the legitimacy of an IP address.
- Endpoint hiding: The identity of the sender/receiver is never communicated without encryption. To transfer messages, the identity of the entities involved is encrypted into the tickets during the exchange of keys.
- Weak Password protection: Our choice of SRP protocol helps the system become resilient to offline attacks for the reasons stated earlier.
- Perfect forward secrecy: Although an adversary manages to gain access to g^a modp and g^b modp, it is impossible for him to attain the shared secret key, as he must solve the Diffie-Hillman problem to do so.

Changes from PS4 that was implemented in the project:

1. (Needham – Schroeder) In PS4, ticket to B was generated in a rather complicated manner to ensure endpoint hiding. In our implementation, we've preserved endpoint hiding while generating the ticket in a simplistic way that is also smaller in size. The ticket to B in our implementation being $\{A, B, N_B, K_{AB}\}_{KBS}$ which in PS4 was $\{A, B, \{A, N_B\}_{KBS}, \{A, B\}_{KBS}, K_{AB}\}_{KBS}$

2. (Messaging Protocol) In PS4, our design constituted of a single sequence number with varying Diffie-Hellman contributions from both the sender and receiver side to add extra security. However, we realized that generating several sequence numbers would be computationally easier than generating several Diffie-Hellman contributions. Therefore, this resulted in the shared key being the concatenation of the sent nonces along with the DH key.

3. (Message Exchange) Like the above change, to provide high security, our design involved changes in the DH contribution at the cost of sequence numbers. However, we found this to be quite challenging to implement in a practical environment between various clients. Hence, to simplify matters, we've used sequence numbers which offer perfect forward secrecy.