PS4: Secure Instant Messaging - Architecture and Design

CS 6740

Medhavi Mahansaria

Naomi Joshi

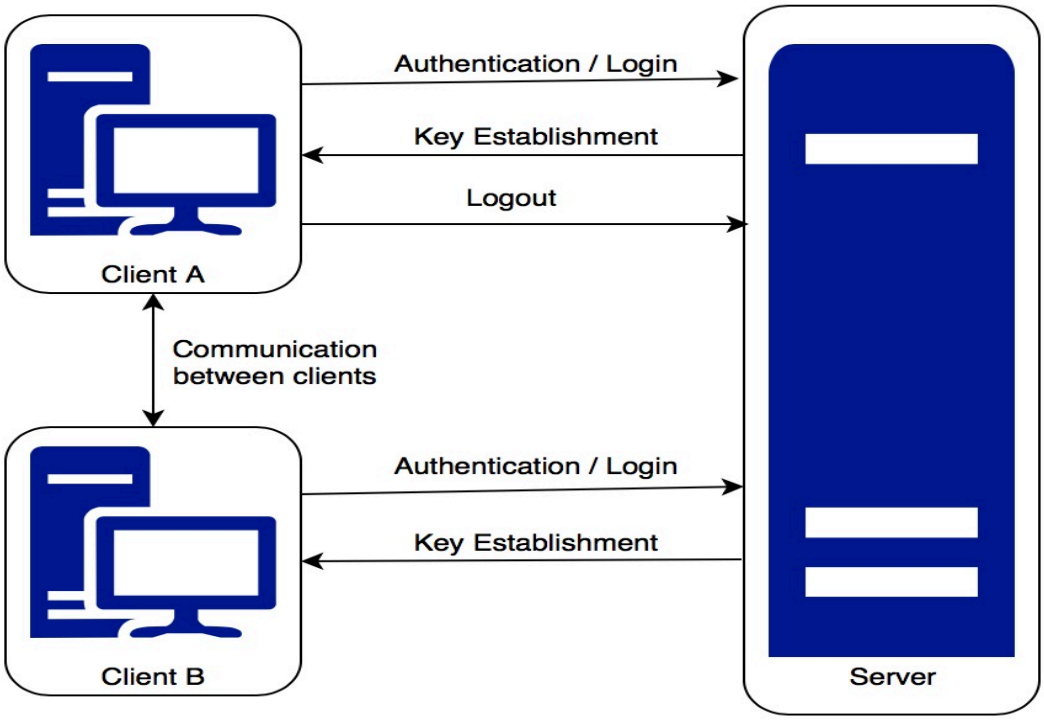
1. *Assumptions*

a. Registered users know the public key of the server.
b. Server does not know the public keys of the clients until login.
c. The user account creation process is assumed to be complete and the server knows list of registered users and the hash of the users' salted passwords.
d. The user only needs to remember a single password.

2. *Constraints (based on PS4)*

a. Messages between the users should not go through the server
b. The users only need to remember a single password.
c. The client application should not remember the user passwords.
d. The client application cannot remember the private/public key of the users.

3. *Architecture*



4. Terminology

- A: Client A's identity (A_{IP} and A_{PORT})
- B: Client B's identity (B_{IP} and B_{PORT})
- $A_{PUB/PRIV}$: Client A's public/private key
- $\{\}_S$: Encrypt with server's public key,
- $[]_{S:}$ Sign with server's private key
- K_{AS}: Symmetric key shared between Client A and Server = g^{as} mod p
- K_{AB}: Session shared key between Client A and B = g^{ab} mod p
- N_i: Nonce, random number
- N_T: Nonce, timestamp
- M: Message
- Encryption Algorithms: We will be using RSA asymmetric algorithm and AES with CBC mode symmetric algorithm.

5. Protocols

a. Login Protocol

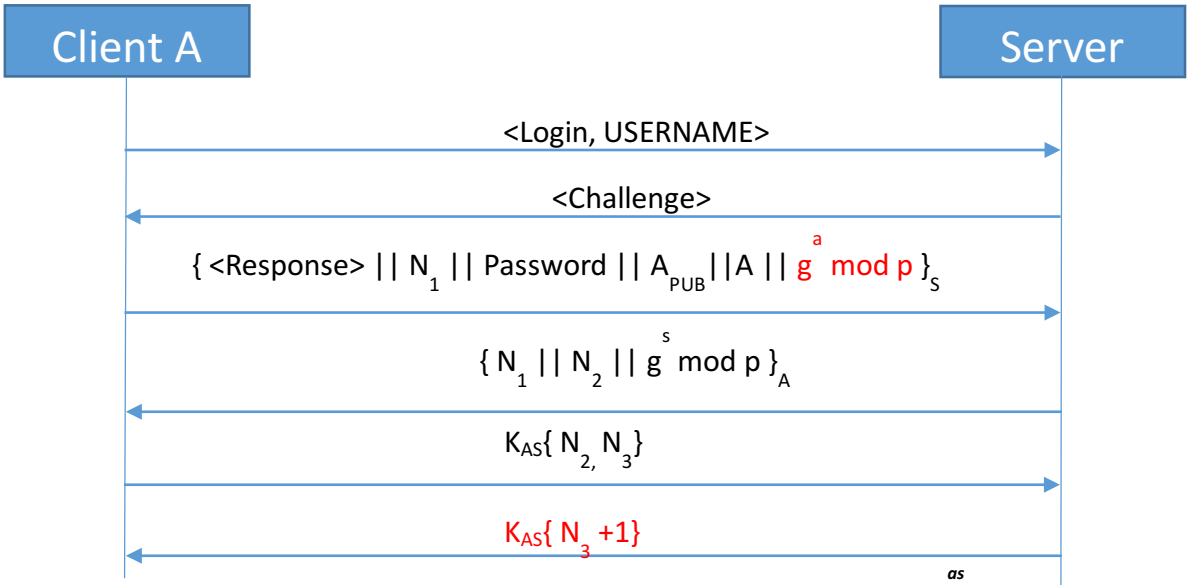
Step 1: Client A sends a Login message with its username to the server to initiate the Login request.

Step 2: The sever then throws a challenge to the client and expects client to do some work. With the above two steps we prevent the DoS attacks because the server is just not going to entertain any client without the client doing some work.

Step 3: In this step, assuming that we know the server's public key, we encrypt: response to the challenge thrown, identity of A, A's password, A_{PUB} the public key of A and nonce N1, to prevent replay attacks of these messages in future time and start Diffie Hellman shared key exchange by sending A's Diffie Hellman public key.

Step 4: In this step if it is the actual server and not an adversary, it should be able to decrypt the message using its private key, server send back N1, a new nonce N2 and provide server's Diffie Hellman public key.

Step 5: Client replies with its Diffie Hellman shared key component and a new Nonce N3 and this completes the mutual authentication between parties and a shared DH key K_{AS} is established to maintain Perfect Forward Secrecy.



Shared key for the session between Client A and the Server, $K_{AS} : g^{as} \bmod p$

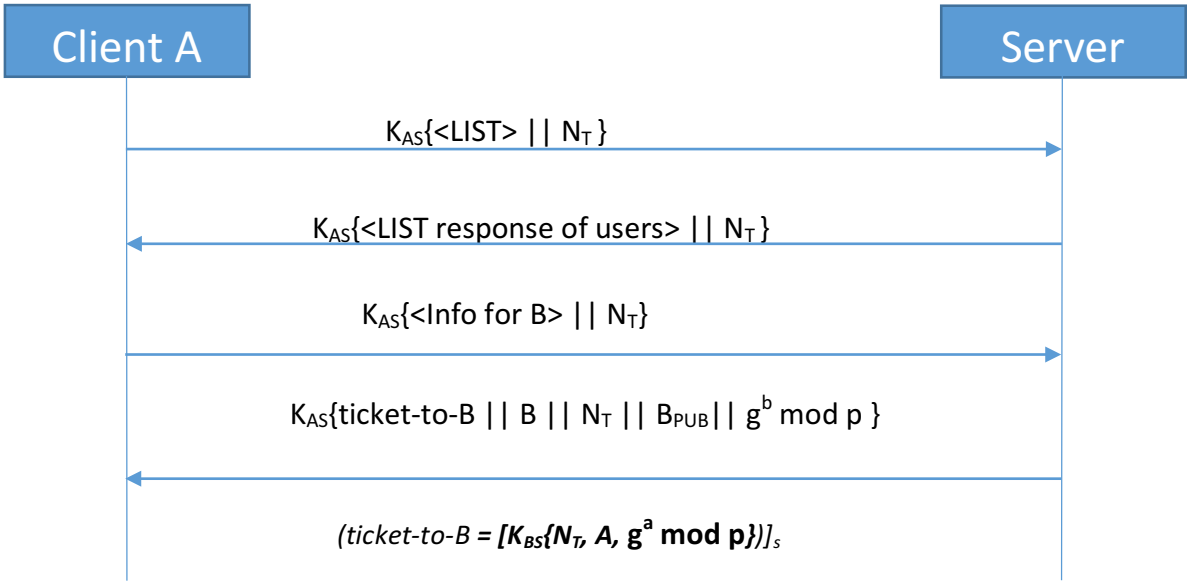
b. LIST query and Key Establishment Protocol

Step 1: After A and the server authenticated each other, A is going to request for a list of users from the server. A encrypts the request with K_{AS}, the nonce sent by the server, and a new nonce. Then sends the message to the server.

Step 2: The server encrypts the list of users and a new nonce. Then sends the message to A.

Step 3: A sends the request to connect to B

Step 4: The server sends ticket-to-B, B's identity, a new nonce that the server created for A and B to share. The ticket-to-B is encrypted with server and B's shared session key. Inside of the encryption, it has the shared nonce, A's identity, and A's public key. At this point A does not know B's public key, but A has ticket-to-B. The nonce N_{AB1} is used by B to prove B's identity.



c. Messaging Protocol

Step 1: The client A receives a ticket-to-B from the server and can use this to communicate with client B. Client A send this ticket to B.

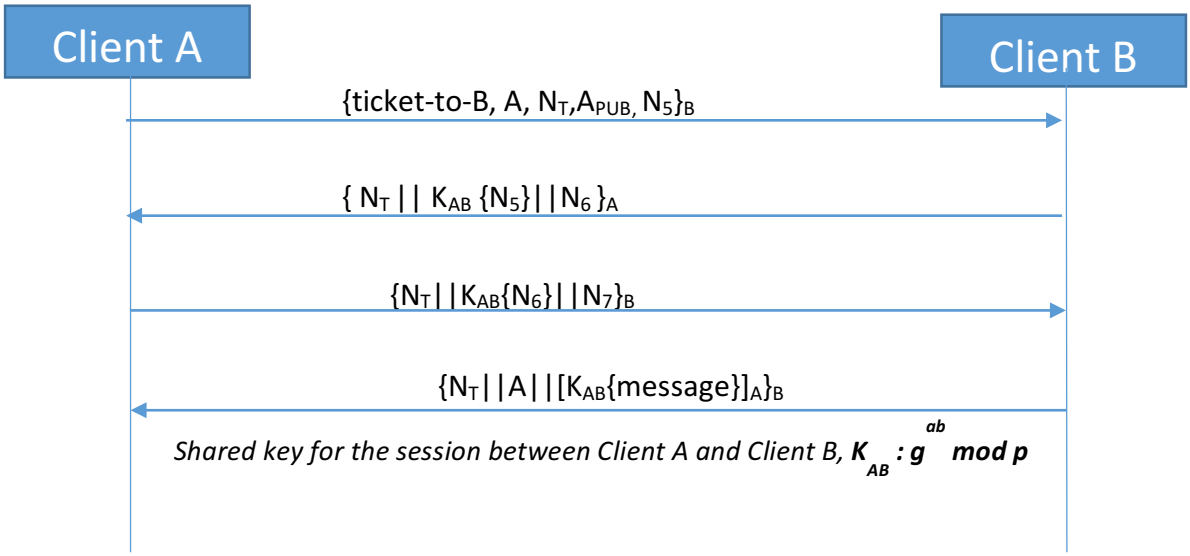
Step 2: Upon receiving the ticket, B decrypts this message since it is encrypted using its own public key, extracts A's identity, A's public key and the common nonce N_{AB1}. It then sends a new message using A's public key with N_{AB1}, a new nonce and B's public key.

Step 3: At this point both the clients know each other's identity and continue with the Diffie-Hellman key exchange process for perfect forward secrecy. A computes its share and sends it to B along with the nonce received in Step 2 and a new nonce, all encrypted using B's public key.

Step 4: B now computes its share of Diffie-Hellman and sends it back to A along with respective nonce.

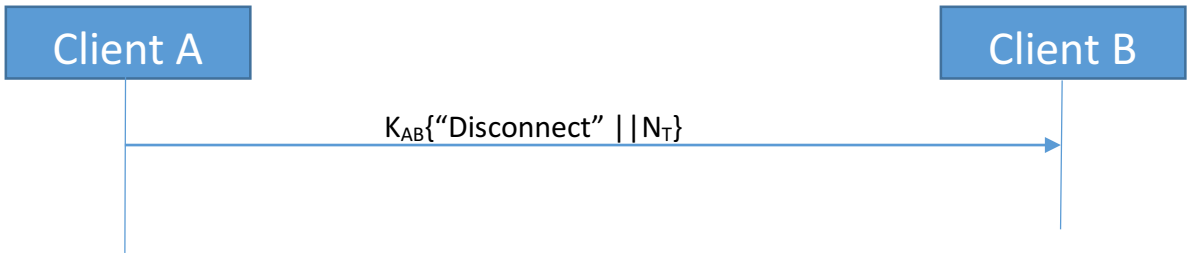
After this point both ends now have a secret key for the session and can now encrypt actual messages.

Step 5 and 6: The actual chat messages between A and B are sent in the following manner, take hash of the message and sign it using private key (to provide message integrity), along with the original message and nonce to prevent replay attacks.



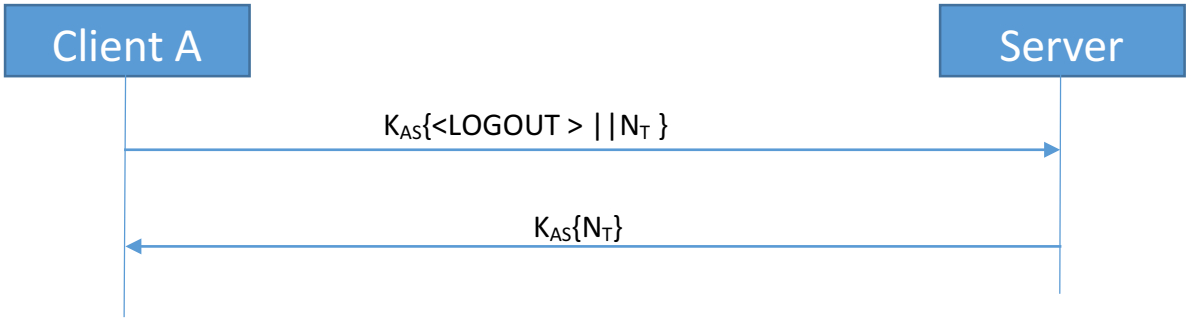
d. Logout Protocol

Step 1: A sends B a closing request the same way they have been communicating. The nonce is used to prevent replay attack. They encrypt the message with their session key. That way, any random person will not be able to terminate their connection.



Step 1: A sends the server a closing request. The nonce is used to prevent replay attack. They encrypt the message with their session key. That way, any random person will not be able to terminate their connection.

Step 2: The server sends A back the verification nonce. Connection terminates. All the data at A and details of A at S are deleted except the mapping of username and PasswordHash.



6. Services

a. Mutual authentication
b. Perfect Forward Secrecy
c. DoS attack protection
d. Message integrity protection
e. Replay attack protection

7. Security

a. Protection against weak passwords

- To prevent online brute force attack, we will have a window of 3 attempts to login. Upon 3 incorrect login attempts, we will lock the username from logging in for the next 15 minutes.
- Our system also protects against offline dictionary attacks because we will only store the hashed and salted version of the password if at all needed to store in the implementation during the active session

b. Denial of Service (DoS) Protection

- To prevent DoS attacks the server presents a challenge to all the incoming login requests from the clients.
- The client has to do some work before the connection establishment.
- This way the server can discard any client that does not provide the correct response to the challenge.

c. Perfect Forward Secrecy

- We will use Diffie-Hellman to establish the session key, which will be used as symmetric keys once authentication is complete. Thus, implementing Perfect Forward Secrecy.
- Once a Logout request is sent, all the keys used in the process along with the constants will be dumped and therefore impossible to recover.

d. End Point Hiding

- End point hiding is implemented in our design, since neither usernames nor passwords or their hashes are being transmitted without encryption.

e. Untrusted Server

- The role of the server in our design is only to authenticate a client and provide a ticket for another authenticated user.
- There is a session key that is established between two clients using Diffie-Hellman, which will be used as symmetric keys for encryption.
- Since the untrusted server cannot compute this key, and hence will not be able to decrypt any messages
- Hence, even if the server is untrusted or compromised, the communication between two clients cannot be intercepted.

f. Untrusted Workstation

- If the user was to trust the application on his workstation, it implies that he trusts the server, since the server is contacted by the application based on the config file residing inside it.
- If the application is malicious, the attacker can modify it in a way that the client connects to malicious server and in which case the attacker can provide false details to A who wants to talk to B, thereby impersonating B as well.
- So, the whole application depends on the premise that application running on client's workstation is trustworthy.