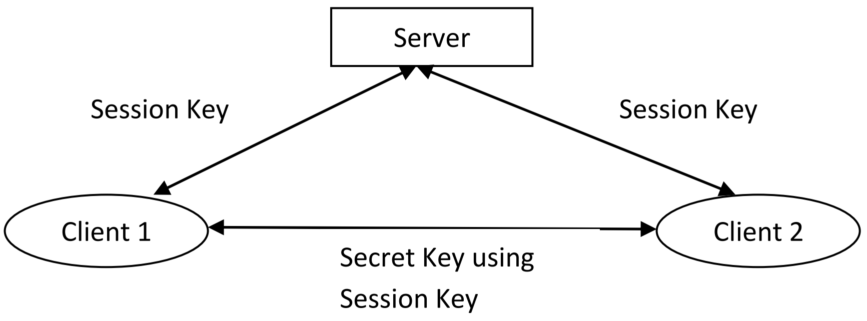
Secure Instant Messaging System

Team members:
Sourabh Suman
Sanat Chugh

Client-Server Architecture



1. Each client first authenticates to server and a session key is established between them.

2. Clients establish a secret key among themselves to talk to each other, using the session key first supplied to them by the server.

3. All asymmetric keys use RSA.

4. Hashing function used is SHA-256.

5. All symmetric keys use AES in CBC mode.

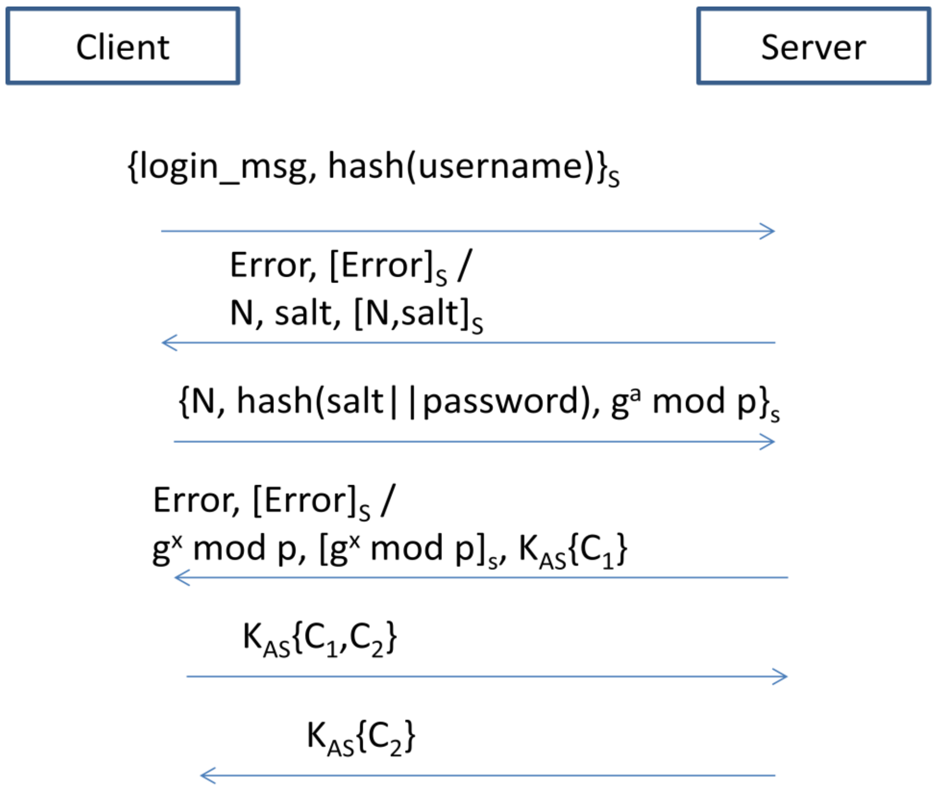
Assumptions:

1. Server is preconfigured with the hash of usernames and salt||password of all clients.

2. Server also stores its public-private key pair.

3. Client has the public key of server saved.

Client – Server Authentication:


Client
Server
$\{login\_msg, hash(username)\}_S$
$Error, [Error]_S$ /
$N, salt, [N,salt]_S$
$\{N, hash(salt||password), g^a \bmod p\}_S$
$Error, [Error]_S$ /
$g^x \bmod p, [g^x \bmod p]_S, K_{AS}\{C_1\}$
$K_{AS}\{C_1,C_2\}$
$K_{AS}\{C_2\}$

1. Client sends a login message, hash of its username.

2. Server validates the username and also checks if this user already exists in its list of online users. If successful, it sends back a nonce and salt of password for that user.

3. The client sends the hash of salt+password along with the nonce and its part of Diffie Hellman, all encrypted with server's public key.

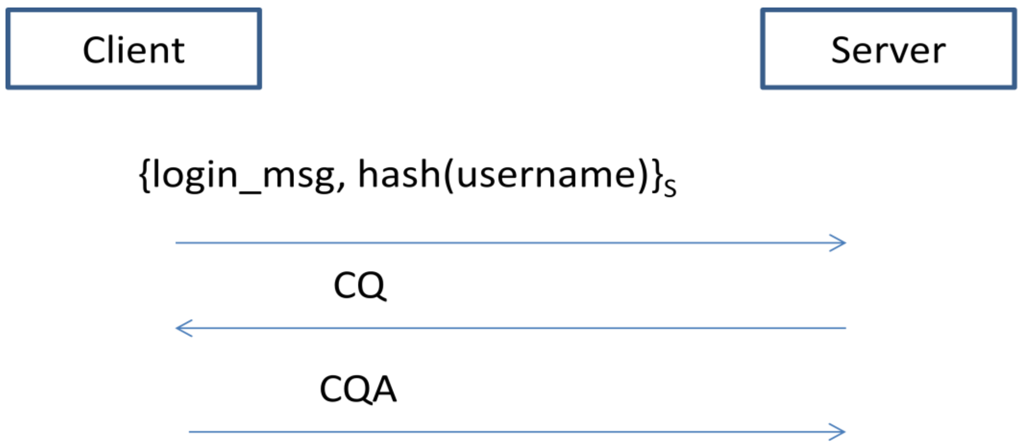
4. The server checks the validity of password against user from the information in its database and if successful, it sends the server's part of Diffie-Hellman signed by its private key along with a nonce encrypted with the DH-Key established.

5. Client and server exchange nonces encrypted with session key established to validate that the other user is legitimate.

Error cases:

1. If user already exists in online users list at server (error at step 2), server sends an error signed by its private key.

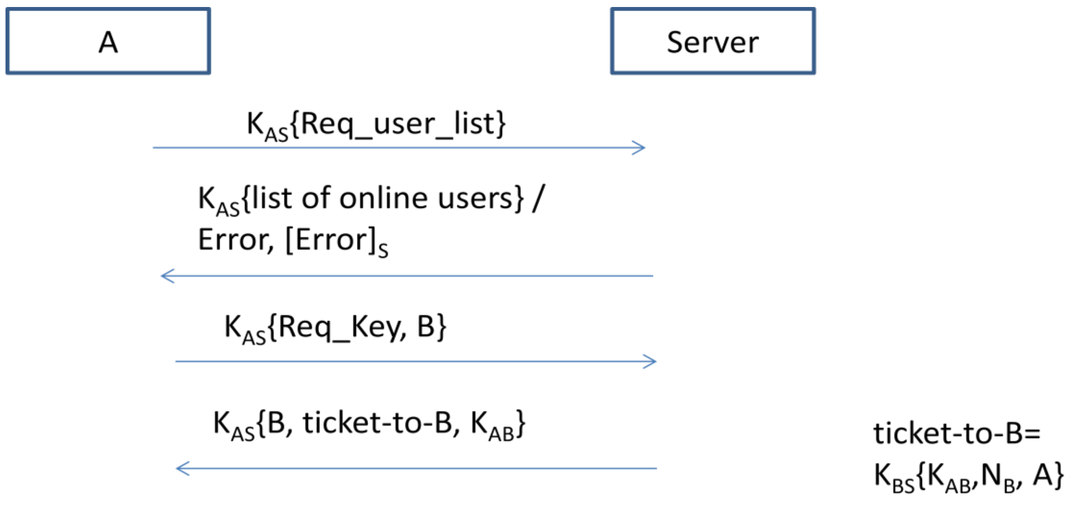
2. If username/password combination is wrong (error at step 4), server sends an error signed by its private key.



1. If server receives three consecutive incorrect username/password combinations, on the fourth request, it sends a challenge question (which involves some computation) to the client.

2. Client should send the correct answer in order to proceed.

Client to Server request for list of online users



A
Server
K_{AS}{Req_user_list}
K_{AS}{list of online users} / Error, [Error]$_S$
K_{AS}{Req_Key, B}
K_{AS}{B, ticket-to-B, K_{AB}}
ticket-to-B= K_{BS}{K_{AB},N_B, A}

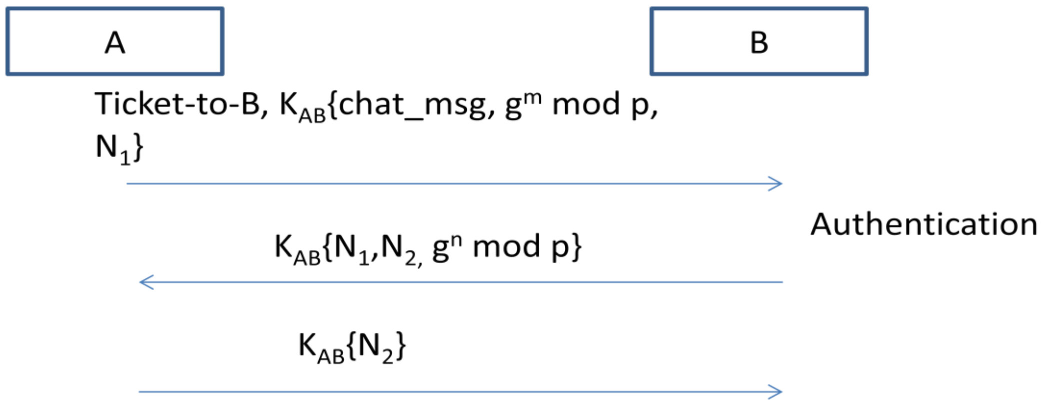
1. Client A sends a request to server to list the users currently online, encrypted with their symmetric session key.

2. Server checks the validity of message and returns a list of online users or an error accordingly.

3. After getting the list of online users, A selects the user it wants to talk to and sends the request to server.

4. Server responds with a ticket-to-B which contains a nonce, requestor A’s identity and their shared key to be used. It also sends the shared key separately so that A can use it to talk to B.

Client – Client Authentication

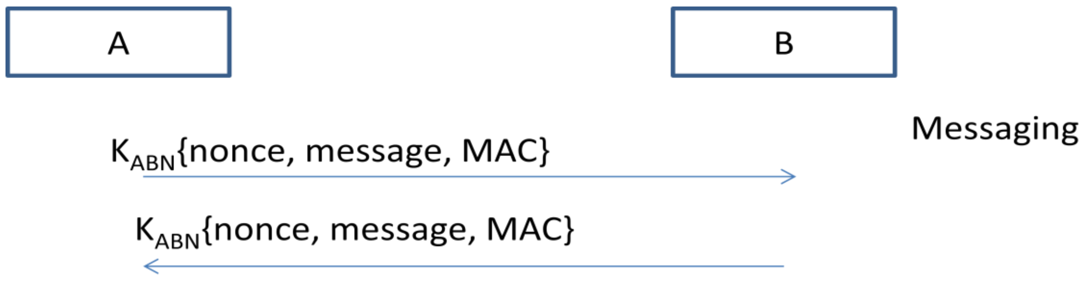

A
B
Ticket-to-B, K_{AB}\{chat_msg, g^m mod p, N_1\}
$K_{AB}\{N_1, N_2, g^n \bmod p\}$
$K_{AB}\{N_2\}$
Authentication

1. Client A sends the ticket-to-B it received from server, along with a chat_msg request, a nonce and its own part of Diffie-Hellman, all encrypted with the shared key setup by server.

2. B gets the shared key from ticket-to-B and uses it to decrypt the information sent by A. It responds with its own part of Diffie-Hellman, the decrypted nonce and a new nonce.

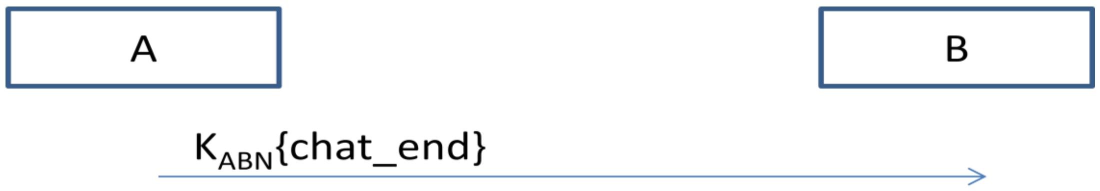
3. A responds with the decrypted Nonce and encrypts and sends it back. Thus, both clients are authenticated and they set up a new key.

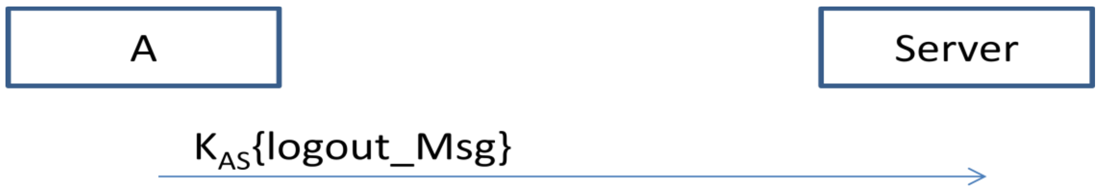
Client – Client Messaging


A
B
Messaging
K_{ABN}\{nonce, message, MAC\}
K_{ABN}\{nonce, message, MAC\}

1. Clients talk to each other after setting a new Key $K_{ABN} = g^{mn}$ mod p. Each message contains a sequentially incrementing nonce, the message and the message's MAC.

Log Out





1. The client who wants to logout sends a message to the clients it's talking to, to end the chat session. Both clients forget the keys they were using.

2. Client A sends the server a request to log him out. Both server and client forget the session keys they were using.

Services Provided

1. Mutual Authentication – both parties are authenticated each time.

2. Identity Hiding – no identity information is sent in the clear.

3. Perfect Forward Secrecy – both parties during client-server and client-client communication set up a Diffie-Hellman key and forget it after session is over.

4. Resilience to Denial of Service attacks – Server sends a challenge question which is computationally expensive for the attacker and this makes the system resilient to DoS attacks.

5. Message Integrity – Both clients send a MAC along with the message to ensure integrity.

Discussions

Q4. a. Does your system protect against the use of weak passwords? Discuss both online and offline dictionary attacks.

Ans: Our design protects against weak passwords to some extent as the password is salted before being hashed when it's stored on server. Salting makes it difficult for someone who breaks into the server's database to retrieve the password even if it's weak. Salting also helps against offline dictionary attacks as with the salt, the attacker will have to take into account the salt for each password and create a new dictionary for each one of them.

Our design also protects against online dictionary attacks as passwords sent over the network are encrypted with asymmetric keys.

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

Ans: Yes. The server sends a challenge question (which is computationally expensive) if it receives 3 consecutive incorrect username/password combinations. This slows down DoS attacks.

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

Ans: Our system provides complete end-point hiding as all user identities are encrypted by keys when being sent over the network. It also provides PFS as both client-server authentication and client-client messaging use Diffie-Hellman to establish symmetric keys on the fly and forget them after each session.

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.

Ans: In our design, the clients set up a separate symmetric key using the initial key given by server. The messages are sent using this new key and the server cannot decrypt these messages since it doesn't have the client's secret part of Diffie Hellman and so cannot find the secret key they established.

When the user trusts the application running on his system, he can simply type in his username and password. If the user doesn't trust the application, there has to be a design modification to display some kind of message to the user on the application which configured by him when he opens the application. The message should be known only to the server and the user, not the workstation.