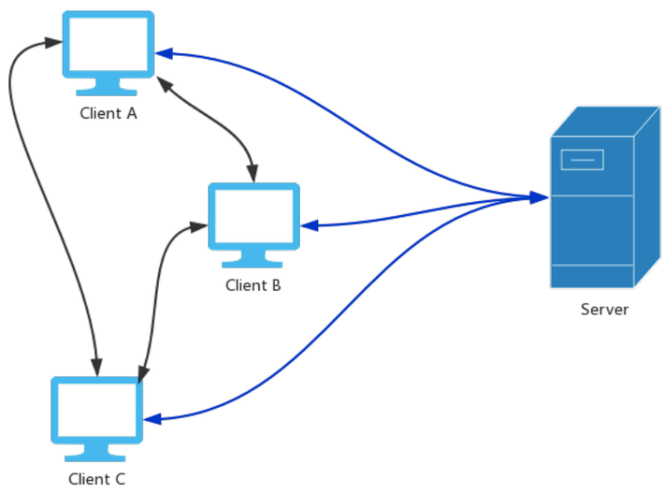
Secure IM Design

Chi Zhang & Yuyang Zhang

1. Architecture:

The system is based on client-server architecture.

- Server listens to a port with a TCP server socket. It can manage online and offline users, and help them to authenticate each other.
- Client can build a TCP connection with the server. Based on the connection, the client can login, logout, get user list from the server and get authorization to talk to other clients.
- Client also listens to a port with a UDP server socket. Based on it, clients can exchange messages with each other.
- Services like PFS, identity hiding, DoS Protection, etc. are provided.



2. Assumptions:

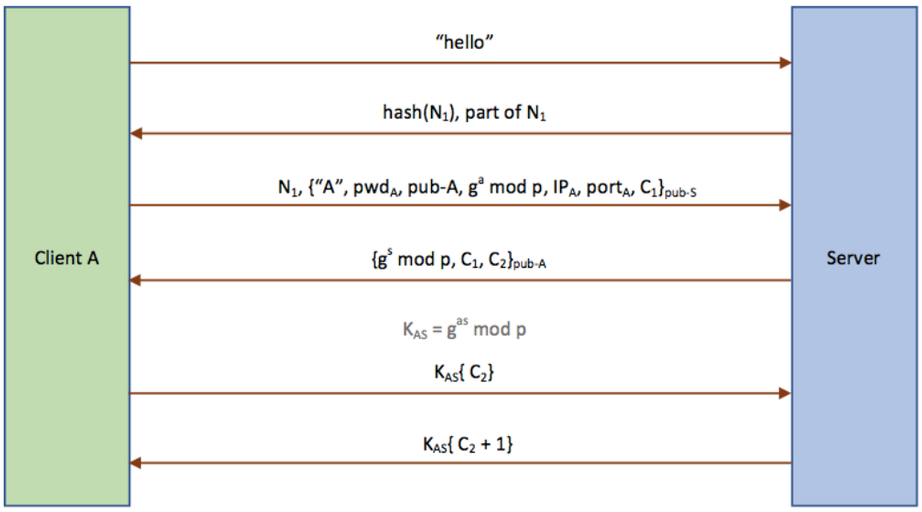
- Each user knows his/her password
- All clients know the server's public key
- Server knows all users' names, along with salted hashes of their passwords and salts
- Everyone knows g and p, which are used for Diffie-Hellman key exchange

3. Cryptographic Algorithms and Key Sizes

- Symmetric Encryption Algorithm: AES
- Symmetric Encryption Key Size: 256 bits
- Symmetric Encryption Block Cipher Mode: CTR
- Asymmetric Encryption Algorithm: RSA
- Asymmetric Encryption Key Size: 2048 bits
- Hashing Algorithm: SHA256
- Key Exchange Algorithm: Elliptic Curve Diffie-Hellman

4. *Protocols*:

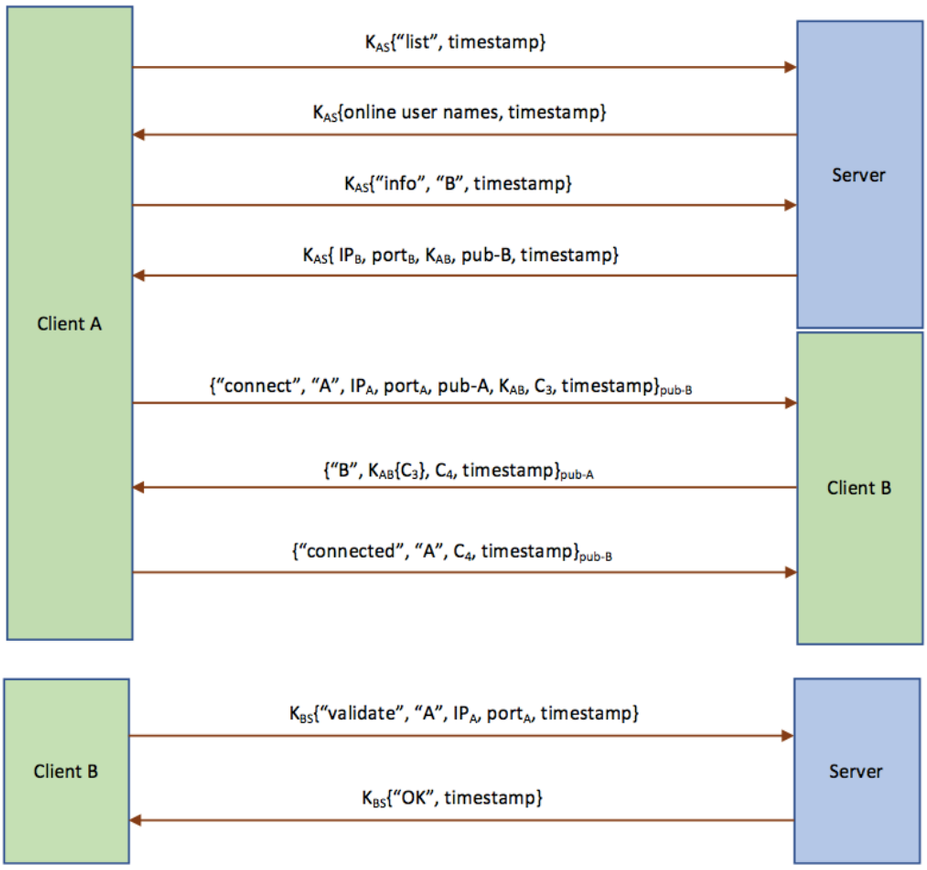
a. Authentication protocol (login)



When client A tries to login the server, it will start the following process:

1) Client A sends a "hello" message to the server.
2) Server generates a 128-bit nonce N_1, sends back part of it (the 21-128 bits) and its hash value.
3) Client A finds N_1 through computations and generates a DH component a.
   Then it generates a RSA key pair (pri-A and pub-A), sends N_1 and ("A", pwd_A, pub-A, g^a mod p, IP_A, $port_A$, C_1) encrypted with the server's public key to the server.
4) The server first checks if the value of N_1 is right. If it's right, it decrypts the data with its private key.
   Then it uses the $salt_A$ and pwd_A to compute the hash value, and compares with the value stored in the database to check if the pwd_A is right.
   If pwd_A is right, it generates a DH component s, and sends back (g^s mod p, C_1, C_2) encrypted with the public key of A.
5) Client A uses its private key to decrypt the message.
   Now client A and server shares the session key $K_{AS} = g^{ab}$ mod p.
   Then it sends C_2 encrypted with K_{AS} to the server.
6) The server decrypts the message with K_{AS}, and confirms C_2 is right.
   Then it sends C_2+1 encrypted with K_{AS} back to the client.
7) Client A decrypts the message with K_{AS}, and confirms it's equal to C_2+1.
   By now, client A successfully logins to the server.

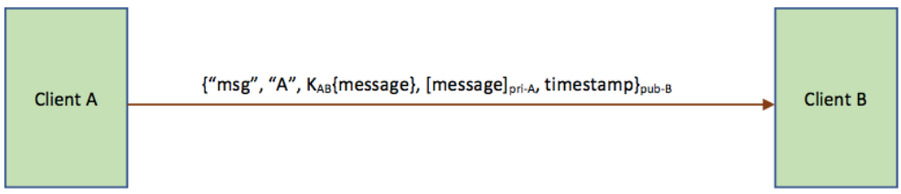
b. Key establish protocol (and authentication with peers)



When client A wants to authenticate with client B, it will start the following process:

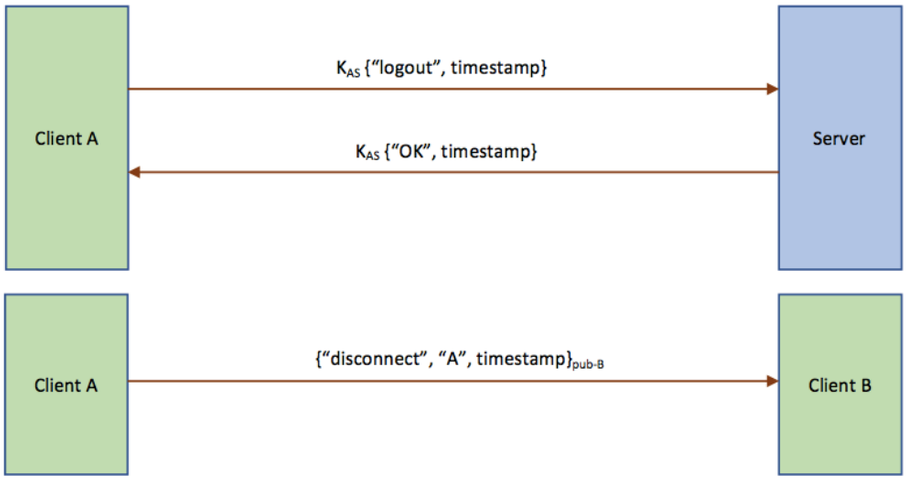
1) Client A sends a "list" message to the server, encrypted with K_{AS}
2) Server responds with all online user names, encrypted with K_{AS}
3) Client A sends a "info" message specifying "B" to the server, encrypted with K_{AS}
4) Server generates K_{AB} for A and B, and send it back together with (IP_B, $port_B$, pub-B), encrypt with K_{AS}
5) Client A creates a "connect" message including ("A", pub-A, K_{AB}, C_3), encrypts it with pub-B, and sends it to B
6) Client B decrypts the message with its private key, generates C_4 and encrypts with K_{AB}, and sends it back along with C_3.
7) Client A decrypts the message with its private key, confirms C_3 is right, and adds B into the authenticated client list. Then it sends back a "connected" message including its information and C_4 back to client B.
8) Client B decrypts the message with its private key, confirms C_4 is right. Note that now client B cannot make sure it's the real client A, since other clients like C can also claim itself to be A and initiate the above connection request. So, client B sends a "validate" message containing A's information to the server and confirms its identity. If it is real, then B adds A into the authenticated client list.

c. Messaging protocol



The messaging protocol is relatively easy:
Client A specifies the type as "msg", add "A" and the message encrypted with K_{AB}, encrypts the whole message with pub-B, and sends it to client B.

d. Logout protocol



When client A tries to logout from the server, it will start the following process:

1) Client A sends a "logout" message to the server, encrypted with K_{AS}
2) Then the server removes A from the online user list, deletes all A's information including K_{AS}, pub-A, etc., and sends backs "OK"
3) Client A notifies all connected client (in this case only client B) a "disconnect" message and then exits. Client B receives this message and remove A from the authenticated client list.

5. *Security Services*:

a. Protection from DoS attacks:
Before a client authenticates to the server, it must solve a challenge from the server, which takes about 2 seconds and is helpful to prevent the DoS attacks.
PS: we don't provide DoS attack protection for the clients. But since single client is not as important as the server, we think this service can be omitted.

b. Protection from Replay attacks:
Since every message in this protocol contains either nonce or timestamp, which are used to check if the given message is legal. So, attackers cannot simply send any recorded message to perform replay attacks.

c. Identities Hiding:
Since all identities are encrypted and cannot by seen be attackers in clear, identities hiding service is provided.

d. Perfect Forward Secrecy:
We use Elliptic Curve Diffie-Hellman Key Exchange algorithm to generate the session keys between clients and server, so PFS is provided.

e. Protection from Password Guessing attacks:
In the server side, users' information is saved in a .csv file. For each password, it is hashed along with a random salt value, which is helpful to prevent the off-line password guessing attack.
In the client side, each user can only try 3 times for incorrect password input, which is helpful to prevent the on-line password guessing attack.

Modification Since PS4

1. Since UDP is stateless, we need to send the sender's information to the receiver every time so that the receiver can know whom to reply. So we changed the three client-client messages in the key establish protocol to include the sender's information, like user name, IP and port number.
2. In the end of the authentication protocol, we added a message $K_{AS}\{C_2+1\}$, which is used by the server to prove that it also knew the generated session key K_{AS}.

Known Vulnerabilities

In our design, we assume the server to be secure and let the server to generate the session key for clients before they authenticate to each other. In such case, if the server is compromised or impersonated, the conversation between clients could be possibly decrypted by the attacker.
To fix this issue, we can improve the protocol as mentioned in the answer to question 4 – perform a separate Diffie-Hellman key exchange between clients.