PS4- Secure IM Design

TEAM MEMBERS:

- ASHOK KODURU
- PARUL SINGH
- TONY ZALAKET

The architecture of the messenger consists of a server and multiple clients with the use of both symmetric and public key cryptography. Symmetric key encryption is done using AES and the public key encryption is done using RSA 2048 bit keys. All the communications happen over UDP.

ASSUMPTIONS

WHO KNOWS WHAT

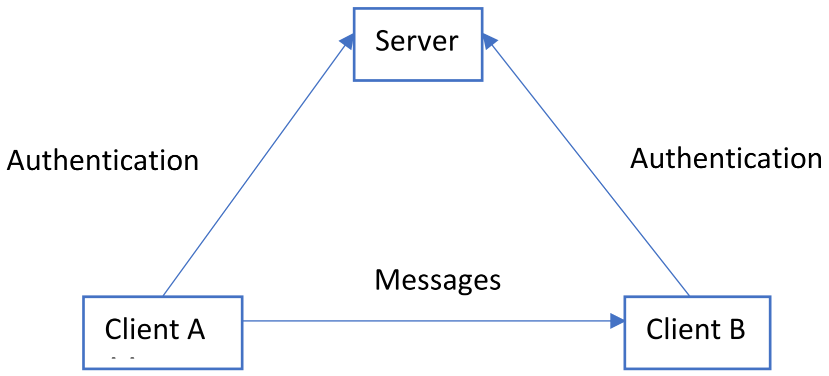
- The user knows the username and the password which he uses to login in to the application
- Since signup process is assumed to be completed, the server is assumed to have a config file on its side which stores the map of all the valid usernames and its respective password hashes.
- Passwords are assumed to be strong according to a particular-predefined standard.

CLIENT

- Every client knows the IP address of the server which It reads from a config file stored in the application.
- Client knows the public key of the server and its private key which is generated for every session

SERVER

- Upon successful login, the server stores the IP address and port of every client.
- Server stores the username, public key and valid time to live value (which expires after a certain idle time) for every active client whose login is successful.



TERMINOLOGY

Shared symmetric key between A and B	→ K_{AB}
Symmetric key encryption between A and B	→ $K_{AB}\{\ldots\}$
Public key encryption by A(encryption)	→ $\{\ldots\}_A$
Private key encryption by A (signing)	→ $[\ldots]_A$
Concatenate the messages M_1 and M_2	→ $M_1 \| \| M_2$
Standard Message Packet	→ <…>
Public Key and Private Key pair of A	→ A_{PUB} and A_{PRIV}

LOGIN PROTOCOL

- A → S: <LOGIN, Username>
- S → A: <Challenge>   // To prevent DOS Attacks
- A → S: $\{$ <Response> || N_1 || PasswordHash || A_{PUB}, || g || p $\}_S$
- S → A: $\{ N_1 \| \| N_2, \| \|\ g^s \bmod p \}_A$
- A → S: $\{ g^a \bmod p \| \| N_2 \}_S$
- Shared key for the session, K_{AS} : $g^{as} \bmod p$

USER LIST QUERY

- A → S: $K_{AS}\{$ <LIST> || N_1 $\}$
- S → A: $K_{AS}\{$ <UserList> || N_1 $\}$

PEER TO PEER CONEECTION

- A → S: $K_{AS}\{$ <Request for B> || $N_1\}$
- S → A: $K_{AS}\{ N_1 \| \| B_{PUB} \| \|$ <B_{IP} :: B_{PORT}> || $K_{BS}\{ A_{PUB} \| \|$ <A_{IP} :: A_{PORT}> } }
- A → B: { <HELLO> || $K_{BS}\{ A_{PUB} \| \|$ <A_{IP} :: A_{PORT}> } || N_1 $\}_B$
- B → A: $\{ N_1 \| \| N_2 \| \| g \| \| p \| \| g^b \bmod p\}_A$
- A → B: $\{ N_2 \| \| g^a \bmod p \}_B$
- Shared key for the session, K_{AB} : $g^{ab} \bmod p$

LOGOUT PROTOCOL

- A → S: $K_{AS}\{$ <LOGOUT> || N_1 $\}$
- S → AllUsers: $K_{(client)S}\{$ <EXIT MESSAGE OF A> || N_1 $\}$
- All the data at A and Details of A at S are deleted except the mapping of username and PasswordHash.

DESIGN CONSTRAINTS

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

   Since the registration process is assumed to be done beforehand, it is assumed that password is strong and consists of at least 10 characters with at least 1 uppercase letter, 1 lowercase letter, and 1 special character. This prevents the offline attacks, since we only store the hashed and salted version of the password if at all needed to store in the implementation during the active session. To prevent online brute force attack, we have kept a window of 3 attempts to login. Upon 3 incorrect login attempts, we will ban the username to login further for 30 minutes. This is better than banning the IP address which can be changed easily.

2. Is your design resistant to denial of service attacks?
   The design is resistant to denial of service attacks. The server presents a challenge (which will be decided) to all the incoming login requests from the clients. Only upon completing the challenge successfully which takes the computing resources from the client, the server starts authenticating procedure with the client. We will design the challenge in such a way that is resistant against the spoofing the messages.

3. To what level does your system provide end point hiding, or perfect forward secrecy?
   End point hiding is implemented in our design, since neither usernames nor passwords or their hashes are being transmitted without encryption. Perfect forward secrecy is also being implemented in our design with the help of Diffie Hellman keys, which are being used as symmetric keys once authentication is complete. Once a session between two parties is complete all the keys used in the process along with the constants will be dumped and therefore impossible to recover. So even if someone gets hold of a conversation or a key, he will not be able to decrypt the previous or the future communications. Also, the Diffie Hellman keys are signed public key cryptography which prevents the usual Man in the Middle attacks.

4. If the users do not trust the server can you devise a scheme that prevents the server 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 users trust the application running on his workstation.
   According to our design, the role of the server is only to authenticate the clients and providing them with a ticket to another authenticated client. After that client A authenticates itself to client B and they together set up a Diffie Hellman shared key which they use for further communication once they are mutually authenticated (This shared key is also dumped once either client goes offline and a new key will be generated when needed). Thus, even if the server is compromised, it cannot decrypt the communication between A and B if the clients are authentic. 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 also. So, the whole application depends on the premise that application running on the client's workstation is trustworthy.