Final Project

Secure IM Client

Course Name: Network Security

Team members: Akshay Nayak & Ankit Malhotra

Assumptions:

- Users are pre-registered.
- Client-Server Architecture.
- Client App cannot remember passwords.
- Username and Password are the only details provided by the user.

Architecture:

Authentication protocol – This protocol is for the client to login to the server.

Before a client can communicate with server it generates its public and private keys for that login session.

1. **A → S:** I need to talk
   A does not send its identity during this step. Only it's IP and port info is sent.
2. **S → A:** cookie
   The server sends a stateless cookie to the client. This stateless cookie is generated using the source IP, port and a secret known to the server. This step has been added to minimize DoS attacks.
3. **A → S:** cookie, $\{h(A), h(h^n(pwd) \| T), P_A, C_1, T\}_{server}$
   A sends back the stateless cookie along with authentication information. This authentication info includes:
   - Hash of the username
   - Hash of ((password hashed n times) concatenated with the timestamp)
   - A's public key
   - Challenge
   - Timestamp

   All of the above info is encrypted with the server's public key. n is an integer that is unique to each password. It is equal to the length of the password. It is used to protect against weak passwords.
4. **S → A:** $[\{R_{c1}, C_2, T\}_A]_{server}$
   The server sends:
   - Response to A's Challenge
   - Its own Challenge
   - Timestamp

   This information is encrypted using A's public key and signed by the servers private key.
5. **A → S:** $[\{R_{c2}, T\}_{server}]_A$
   A sends the following information:

- Response to server's Challenge
- Timestamp

This information is encrypted using the server's public key and signed by A's private key.

The shared session key between the server and A is derived as:

$\mathbf{K_{AS} = R_{C1} \oplus R_{C2}}$

<u>List protocol</u> – This is to list the users that are currently online, so that A can chat with them.

When A is logged in and types "LIST":

1. **A → S:** $\{$"LIST", T$\}_{KAS}$, HMAC
2. **S → A:** $\{$online user list, T$\}_{KAS}$, HMAC

<u>Key Exchange protocol</u> – This is to exchange keys between the users wanting to chat with each other.

1. **A → S:** $\{A, B, C_1, T\}_{KAS}$, HMAC

   When A wants to start a new chat with B it sends the following information to the Server:
   - A's username
   - B's username
   - A's challenge for B
   - Timestamp

   This information is encrypted with the shared session key K_{AS}.
2. **S → A:** $\{P_B, p, T_6\}_{KAS,}$ HMAC

   The server sends the following information to A:
   - B's public key
   - The most recent value of p
   - Timestamp

   The information is encrypted with the shared session key K_{AS}.

3. **S → B:** $\{A, P_A, C_1, p, T\}_{KBS,}$ HMAC

   The server send the following information to B:
   - A's username
   - A's public key
   - Challenge that it got from A
   - The most recent value of p
   - Timestamp

   The information is encrypted with the shared session key K_{BS}.

4. **B → A:** $[\{B, R_{c1}, g^y \bmod p, C_2, T\}_A]_B$
   B sends the following information to A:
   - B's username
   - Response to A's Challenge
   - $g^y \bmod p$
   - Challenge for A
   - Timestamp

   This information is encrypted with A's public key and signed by B's private key.
5. **A → B**: $[\{A, R_{c2}, g^x \bmod p, T\}_B]_A$

A sends the following information to B:
   - A's username
   - Response to B's Challenge
   - $g^x \bmod p$
   - Timestamp

   This information is encrypted with B's public key and signed by A's private key.

The session key between the A and B is derived as:
$K_{AB} = g^{xy} \bmod p$

<u>Messaging protocol</u> – This will be used for the chat messages between clients once the shared key is obtained.

1. A → B: $\{A, \text{message}, T\}_{K_{AB}}$, HMAC
2. B → A: $\{B, \text{message}, T\}_{K_{AB}}$, HMAC

<u>Logout protocol</u>

1. **A → S:** $\{$"LOGOUT", $T\}_{K_{AS}}$, HMAC
2. **S → A:** $\{\text{Challenge}, T\}_{K_{AS}}$, HMAC
3. **A → S**: $\{\text{Response}, T\}_{K_{AS}}$, HMAC

<u>**Implementation details and Specifications**</u>:

- All RSA keys – 2048 bits.
- RSA key pairs are generated by the client software.
- For symmetric encryption, AES-CBC with key size of 256 bits and random 128 bit IV.
- SHA-256 used for hashing (with salt being used for passwords).

- The server's secret used for the cookie is a 80 bit number.
- Challenges and responses will be 128 bit random numbers.
- The skew within which timestamps are accepted is taken as 300 seconds or 5 minutes.
- g is taken as 2.
- p will be generated every time the server is started and subsequently, every 24 hours, in case the server is on for a longer time. p is a safe prime number of 3072 bits.
- All symmetric keys such as K_{AS} and K_{AB} are hashed (using SHA-256) to make sure a 256 bit key is generated for AES-CBC.
- All keys (except the server's public key) are valid only for that single login session and destroyed after it ends.
- x and y are unique to each set of users communicating with each other.

Changes during implementation:

- In the third step of authentication, we used the <u>username instead of the hash of username</u>.
- In the third step of authentication, since the size of the packet exceeds the RSA key size, we adopted a <u>hybrid encryption approach</u>. We generated a temporary 256 bit key and 128 bit IV used to encrypt (AES-CBC symmetric encryption) the data. The key and IV are then encrypted with the servers public key. After this the both the cipher-texts are concatenated and sent to S.
- In the 4th and 5th step of the key exchange protocol, the size of the packets exceeds the RSA key. Just like in the 3rd Authentication step, we used a <u>hybrid encryption approach</u>. We generated a temporary 256 bit key and 128 bit IV used to encrypt (AES-CBC symmetric encryption) the data. The temporary keys and IVs are then encrypted by the receiver's public key and the whole cipher text is signed by the sender's private key.
- In the logout protocol, we added a <u>'not-me' packet</u>, that is sent from the client to server, in case the client receives a challenge in spite of not initiating the logout.
- In the 2nd step of the key exchange protocol, B's username and address (ip, port) is also provided.
- In the 3rd step of the key exchange protocol, A's address (ip, port) is provided in addition to its username.

Limitations in implemented system

- Only one user can login at a time.
- Only login, list and logout functions were implemented completely. Key exchange between clients and the messaging part could not be implemented properly.
- Very limited exception handling.
- There is no mechanism against online brute force attacks i.e. address blacklisting or account lockout.

Known Vulnerabilities:

- If the server or client private keys are compromised, all the session keys will be compromised.
- Replay attacks are possible if a packet is replayed within the acceptable time window (5 minutes).
- Since HMACS are used most for most of the messages, it is possible that a user can deny sending a message. (Non-Repudiation fails).
- Due to bugs in the code, DoS messages are possible.

Secure Services Provided:

- Confidentiality
- Integrity
- Authenticity
- Non-Repudiation (for some of the authentication and key exchange messages, where signature is used)
- Perfect Forward Secrecy (ephemeral keys)
- Identity hiding
- Protection against replay and reflection attacks
- Limited DoS protection