CS 6349.001 - Network Security
Project Design Report
Instant Messaging (IM) System

Group Members -

Amogh Yatnatti	-	ary180001
Avani Gumudavelli	-	axg200002
Manasi Gosavi	-	mvg200001
Shatavari Shinde	-	svs210001

Programming language -

Python

System Design -

1) **Client Authentication**

- To authenticate and establish client logins on the server, we are using a RSA key pair where the server knows the client's public key and only the client knows its own private key.
- To begin, we have created public and private key pairs for our clients using python's rsa library.
- Here, the client has access to their own public and private keys while the server only has access to the client's public key.
- When the IM opens, the client is asked for their username. Then the server generates a random nonce and sends it to the client. After this, the client will sign this nonce with its own private key and send the signature to the server.
- Once the server receives this signature, it will verify it using the client's public key. If the nonce matches the one originally sent by the server, the user is authenticated and will be able to start chatting with other clients. If not, they will see an error message.
- To avoid any replay attacks, we are sending a newly generated random nonce each time we want to authenticate a user. This way an attacker cannot send the same nonce again and try to get authenticated.

2) **Client - Server Communication**

- The client communicates with the server to authenticate themselves, view the list of clients, and to establish secure sessions with other clients.
- Once a client is authenticated, the server generates a session key and sends it to the client after encrypting the session key using the public key of the client.
- The client then decrypts the session key using its private key.

- At this point, both the server and the client will derive 1 key for encryption /decryption, 1 key to be used as an initialization vector (IV) and 1 key for integrity using their shared session key.
- All communication from this point on is now encrypted using a keyed-hash (Message Digest) in a mode similar to CFB mode of operation.
- Thus, making all server-client communication confidential.

3) Client - Client Communication

- When client A is ready to start chatting, it will run the "!session host" command on the IM which will let the server know that it is open for communication.
- Now client B can run "!session A" and begin speaking with client A.
- At this point, both the clients are authenticated and are able to start communicating with each other.
- Now, the server will send two versions of the client-client session key to the host client (so client A in this case) :
    - One which is encrypted with client A's client-server encryption key that we had derived from the session key after authentication.
    - One which is encrypted with client B's client-server encryption key that we had derived from the session key after authentication.
- Client A will now decrypt the response using its own client-server encryption key and send the other encrypted key in the response to client B.
- Client A will now generate the keys for encryption/decryption, initialization vector and a key for integrity which is used for all further communication between the two clients.
- Client B will decrypt the key sent by Client A using its own client-server session key and then generate its sets of keys for encryption/decryption, initialization vector and a key for integrity (will be same as what client A generates for itself) which is then used for all further communication between the two clients.
- Here on, all is encrypted using a keyed-hash (Message Digest) in a mode similar to CFB mode of operation.
- Thus, making all client-client communication confidential.

Security Features -

- **Authentication**
  - Client-Server authentication is done using RSA digital signature scheme (sign + verify)
  - This is done using an RSA key pair where the client knows its own private key and the server knows the client's public key.
  - Once a client signs a random nonce sent by the server, the server can verify it with the client's public key and if it matches, it will authenticate the client.
- **Confidentiality**
  - All messages are encrypted using keys derived from a shared session key.
  - Session Key Generation:
    - Session key is generated using python's secret library which generates cryptographically strong random numbers for security tokens.
    - Session keys for client-client as well as client-server communication are generated at the server side, encrypted and sent to clients.
  - Every client-server pair and client-client pair will have their own shared session key, with which they will derive their own set of keys for encryption/decryption.
    - Python's key derivation library is used to derive keys from the session key at both ends. This way, the actual keys used for encryption and integrity are never sent from client to server or client to client and it's difficult for an attacker to guess the keys as the attacker does not know the function used to derive these keys.
  - Encryption Process:
    - For encryption, Every message is divided into MD-length blocks called p1, p2 ... pi. An intermediate value bi is calculated which is a message digest or a keyed hash as shown below from which the ciphertext blocks c1, c2 .. ci is generated. Here, Kab and IV are the keys that we generated using the shared session key.
    - A list of these ciphertext blocks are sent during communication instead of the plain text message.

$$
\begin{array}{ll}
b_1 = \mathrm{MD}(K_{\mathrm{AB}}|\mathrm{IV}) & c_1 = p_1 \oplus b_1 \\
b_2 = \mathrm{MD}(K_{\mathrm{AB}}|c_1) & c_2 = p_2 \oplus b_2 \\
\vdots & \vdots \\
b_i = \mathrm{MD}(K_{\mathrm{AB}}|c_{i-1}) & c_i = p_i \oplus b_i \\
\vdots & \vdots
\end{array}
$$

- Decryption Process:
  - For decryption, the receiver receives the list of ciphertext blocks. The intermediate values bi are calculated back which is a message digest or a keyed hash as shown below. The plaintext blocks pi are then calculated using ci and bi. Here, Kab and IV are the keys that we generated using the shared session key. Thus, with the knowledge of Kab, IV and ci, the messages can be decrypted easily.

$$
\begin{aligned}
&b_n = MD(K_{AB}|c_n - 1)\\
&b_{n-1} = MD(K_{AB}|c_{n-2})\\
&\dots\\
&b_{n-i} = MD(K_{AB}|c_{n-i-1})\\
&\dots\\
&b_2 = MD(K_{AB}|c_1)\\
&b_1 = MD(K_{AB}|IV)\\
\\
&p_n = c_n \oplus b_n\\
&p_{n-1} = c_{n-1} \oplus b_{n-1}\\
&\dots\\
&p_1 = c_1 \oplus b_1
\end{aligned}
$$

- **Integrity**
  - The hash of the message is appended to the original message (message+hash) and encrypted before it is sent.
    - A hash of the message or a message digest is calculated using SHA-256 algorithm using one of the keys derived from the shared session key of 2 clients (C12). The hmac and hashlib libraries are used to do so.
  - On the receiver's end, the entire response (message+hash) is decrypted and the hash of the received message is calculated using the same derived key for integrity .
    - If the hash calculated is the same as the hash received, we know the integrity of the message is intact. If not, we discard the message because it has been tampered with.
- **Freshness:**
  - Freshness is achieved during authentication by sending a random nonce. That way if the wrong nonce is received by the server, it will know that the signature is either tampered with or is outdated.
  - For client-client communication, a timestamp is added with the message and message hash which is all then encrypted together and sent. Once received, it is decrypted and compared with the current time. If the difference between the two timestamps is longer than two minutes, it is discarded.

Threat Model -

The Instant Messaging System will prevent the following attacks-

- **Passive attack**
    - The client and the server use a private session key as well as IV that is discarded after every session. All messages exchanged between client to server and client to client are encrypted as well as message hash is encrypted and attached.
    - Thus, the attacker cannot generate the keys to decrypt the message. Attackers cannot use the keys of these recorded messages later as the keys get discarded once the session is closed.
- **Active attack**
    - The attacker cannot guess the session key or the initialization vector. Message hash is also attached and encrypted. Because of this, the message cannot be altered since the attacker cannot generate the key to decrypt it and even if the message gets decrypted, then any change made to the messages will be detected by hash.
- **Reflection attack**
    - The attacker cannot perform a reflection attack because a timestamp is added to every message and the message cannot be reused later since the timestamp will detect the change.
- **Replay attack**
    - Replay attacks cannot happen during authentication because we are sending a new random nonce.
    - During communication, the attacker cannot record the messages and use them in another session because the initialization vector will be different for different sessions between the client and the server and between any two clients.
    - Timestamp is also added to all messages making any replay attack ineffective.

Assumptions -

- The server already has the public keys for all the clients trying to connect.
- The server and the client are safely storing the public, private and session keys and are not vulnerable for attacks like a database attack.