Final Design Report

Team Members:

Kashif Hussain	kashif.hussain@utdallas.edu
Rupin Jairaj	rupin.jairaj@utdallas.edu
Preetham Rao Gottumukula	preethamrao.gottumukula@utdallas.edu

For CS 6349 - Network Security

1. Overview

Designing and implementing an end-to-end encrypted instant messaging (IM) system using Java as the programming language. The system consists of a server and a set of clients. The server authenticates legitimate clients and helps them connect securely with each other.

2. Functional requirements

2.1. Server

2.1.1. Maintains a record for each client consisting of: Client ID and Client Public Key

2.1.2. Maintains the state of each client: Busy or Idle

2.1.3. Facilitates clients in establishing a secure session by issuing a session key for the clients to use in a secure communication channel.

2.2. Client

2.2.1. Authenticates itself to the server.

2.2.2. Requests and obtains a list of other clients from the server.

2.2.3. Communicates with one client at a time.

2.2.4. Can be either in a Busy state (involved in an IM communication with another client) or in an Idle state (waiting for other clients to communicate with it).

2.2.5. Both clients need to be idle first to establish a session with each other.

2.2.6. Once a session begins, clients in that session communicate directly with each other until the session ends.

2.2.7. Only two clients participate in a single session, but there may be multiple sessions in parallel across different pairs of clients.

3. Security Features

3.1. Logins

3.1.1. The client login operation allows a client to authenticate itself to the server and obtain a session key. This session key is used for any future client-server communications until the client logouts.

3.1.2. Each client has a public-private key pair.

3.1.3. Each client sends their Client ID, nonce, client hostname and port for incoming connections. Each value is signed using the client's private-key.

3.1.4. The server's response contains the client-server session key encrypted using the client's public key.

3.1.5. The client can now utilize the session key going forward.

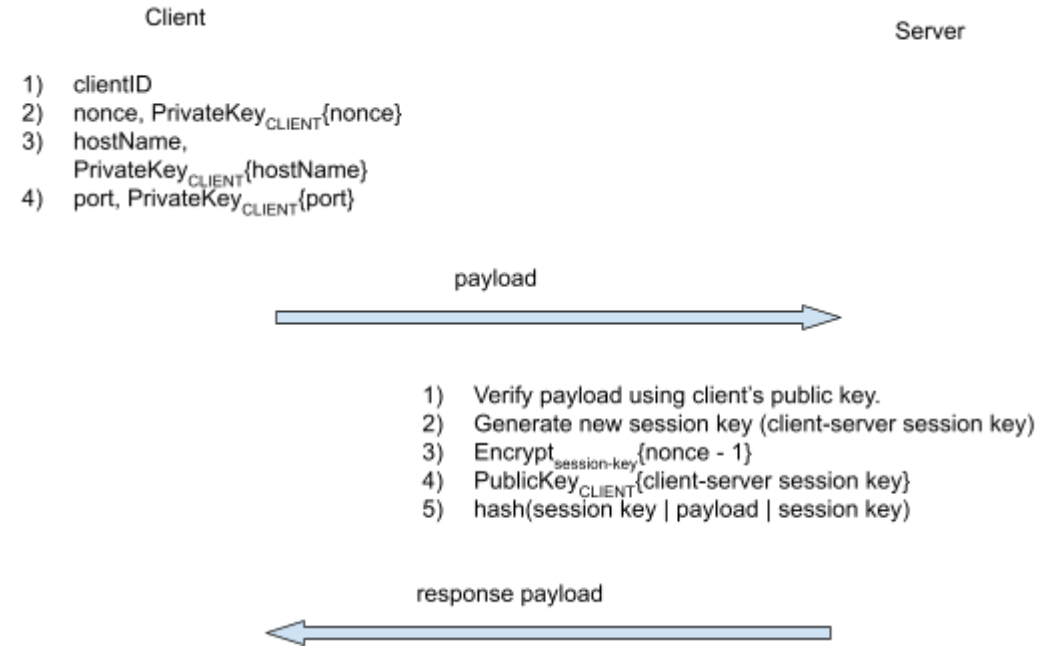
Client to server authentication request payload:

|0|clientId|randomNumber|signedRandomNumber]hostname|signedHostName|port|signedPort|

Server to client authentication(successful) response payload:

|0|encrypted(randomNumber-1)|clientPublicKey(sessionKey)|iv|hash(sessionKey|payload|sessionKey)|

Auth flow diagram



- Authentication: ClientID is used to authenticate the client to the server along with the client's digital signature.

- Confidentiality: The server's response payload containing the session key, is encrypted with the client's public key which safeguards it from being eavesdropped and thus confidentiality is achieved.

- Freshness: The nonce is generated using a fairly large number (1024 bits) and randomly generated. This guarantees freshness for the session key for the client as received in the server response payload for verification.

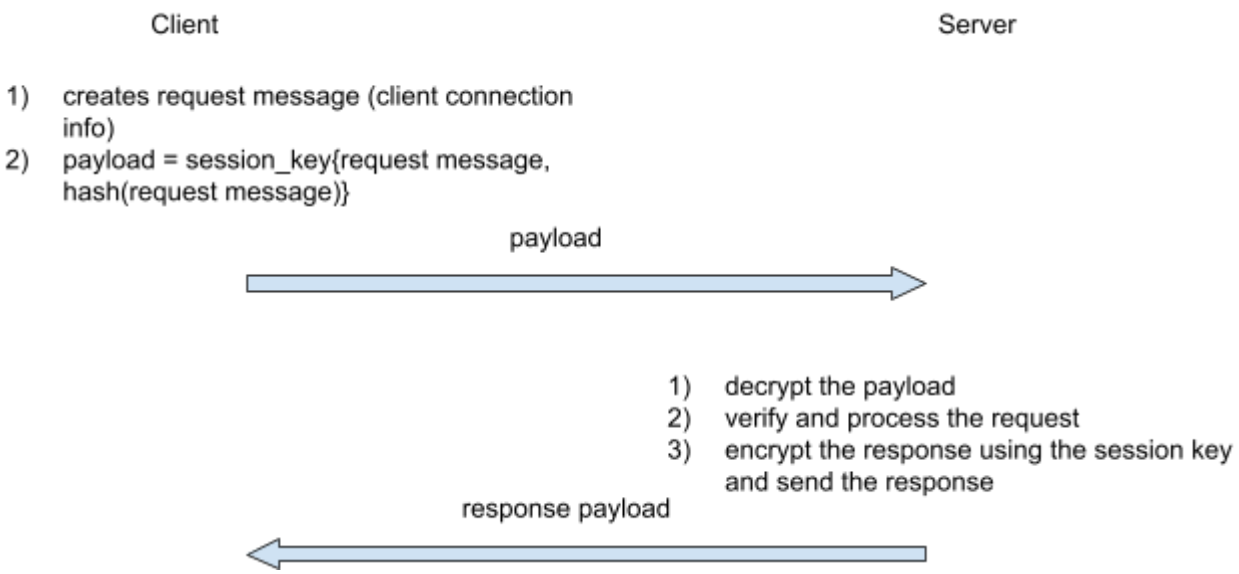
- Integrity: The server is hashing the plain text payload by surrounding it with the session key. This ensures the payload hasn't been tampered with.

3.2. Client - Server Communication

3.2.1. All client server communication is encrypted using the session key obtained during the login flow.

3.2.2. The session key is an AES key.

Client - Server flow diagram



Client to server peer list request payload:

|1|clientId|

Server to client peer list response payload:

|1|encryptedClientList|iv|hash(sessionKey|payload|sessionKey)|

Client to server peer connection/session key request payload:

|2|clientId|encryptedPeerId|iv|hash(sessionKey|payload|sessionKey)

Server to client peer connection/session key response payload:

|2|sourceClientEncrypt(incomingIv|destPeerId|destHostName|destHostPort|p2pSessionKey|destPeerTicket|destIv)|sourceIv|hash(sessionKey|payload|sessionKey)|

destPeerTicket - destClientEncrypt(p2pSessionKey|sourcePeerId|expirationTime)

Client to server status update payload:

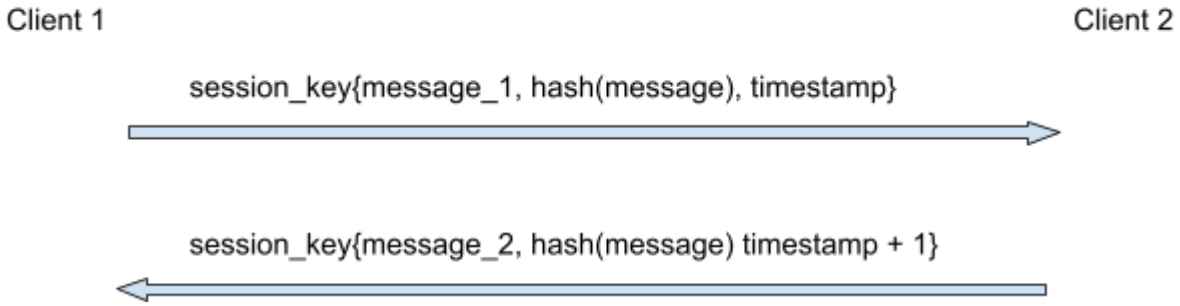
|5|clientId|encrypted(status)|iv|hash(sessionKey|payload|sessionKey)

- Authentication: The session key used by the client to send requests to the server as the means for authenticating the client as legitimate since it could only have the session key based on being authenticated successfully and securely by the server.

- Confidentiality: All communication between client and server is encrypted with the session key that is valid only for that particular session and it changes every time the client logs out and logs back in, ensuring the channel is secure from eavesdropping.
- Integrity: Each encrypted payload request from a client will contain the request along with a hash of the request which will be checked against the server's own computed hash of the request to ensure integrity of the client request.

3.3. Client - Client Communication

3.3.1. Client to client communication is encrypted using the session key the clients obtain from the server.



Peer to peer connection/session key setup payload:

|2|ticketForPeer|iv|p2pSessionKeyEncrypted(timestamp)|timeEncIv

Peer to peer connection/session key accepted payload:

|3|p2pSessionKeyEncrypt(originalChallenge+1)|iv

Peer to peer chat message payload:

|4|encChatMsg|iv|checkSum

- Authentication: The session key used between the two clients authenticates both of them as it was issued to client 1 by the server with proper credentials for client 2 and given that client 2 receives it from client 1 with proper credentials from the server, client 2 is guaranteed to be talking to a genuine client with the session key.
- Confidentiality: All communication between clients 1 and 2 are encrypted with the session key that only the two legitimate clients know, which makes it a secure channel impervious to eavesdropping.
- Freshness: The timestamps in each encrypted message between the two clients ensures freshness for each client as they will know the messages sent are indeed fresh communications and not replays.

- Integrity: Each encrypted communication sent from one client to another contains the message along with a hash of the message so that the receiver can verify what the sender intended to send is indeed what has been sent in the encrypted message.

4. What can go wrong

4.1. Server database reading attack

4.1.1. An attacker could potentially gain access to the server and perform a disk dump obtaining all the server's private information. Since we are using symmetric key encryption for client-server communication, the server has to safely keep the session keys for each client that authenticates with it.

4.2. Eavesdropping

4.2.1. Preliminary messages (messages not encrypted with a session key) are signed using the sender's private key. Since the client private-key signature does not prevent eavesdropping, there is a risk of a passive attacker seeing the client authentication payload.

4.2.2. Once session keys are in play, an eavesdropping attack is difficult to accomplish. All session keys are AES keys. Since we do not plan on reusing keys across sessions we can guarantee that replay attacks are not possible since the duration of a session is much smaller than the time taken to break an AES key using any practical attacks.

4.3. Man-in-the-middle (MITM) attack

4.3.1. The only case where an MITM attack is possible is if the attacker takes over the server and captures the session key generated for each client for their client-server communication. This would allow the attacker to decrypt messages or send a custom payload to the participating clients only for the current session.

5. Measures to manage risk

5.1. Preliminary client to server messages which are just signed by the client can be read by an attacker but cannot be modified. They can be replayed but the response is of no help to an attacker unless they have the client's private key.

5.2. Ensure session keys are not reused throughout the server's lifetime.

5.3. Use a secure random number generator to create hard-to-predict nonces for freshness guarantees.

5.4. Add an expiration time to session keys for p2p communications. An attacker cannot impersonate a peer using an old session key.

6. Member contributions

We designed the payloads for various scenarios (auth, p2p chat etc.) together. Preetham took the lead in all RSA related tasks (sign/verify, encrypt/decrypt). Kashif worked on the keyed hash encryption and helped build various helper methods for purposes like generating session keys, initialization vectors etc. Rupin built the socket programming classes that helped achieve non-blocking IO.

We worked on the integration part together.Since none of us are java programmers, it took all of us to thoroughly test and debug the source code files.