Secure Instant Messaging System

By Sumeet Dubey and Shaurya Katiyar

Architecture:

1. One server, multiple clients
2. Server maintains a list of every online client
3. To communicate, a client need to send a message to the server
4. This message will ask for the IP and port of the destination server
5. Server has a public/private key pair
6. Provides mutual authentication by using passwords and message signing
7. Message data is not sent through server to protect against malicious server

Assumptions:

1. Every client knows server's public key
2. Every client knows the IP and port of server (through a config file)
3. Every client remembers his own ID and password

Protocols:

1. Communication will happen over UDP connections between clients
2. Every client will generate a Diffie-Hellman (DH) key with the server
3. Server generates shared secret key for clients willing to communicate
4. Encryption protocol – AES 256 with CBC and random IV
5. Hashing protocol – SHA256

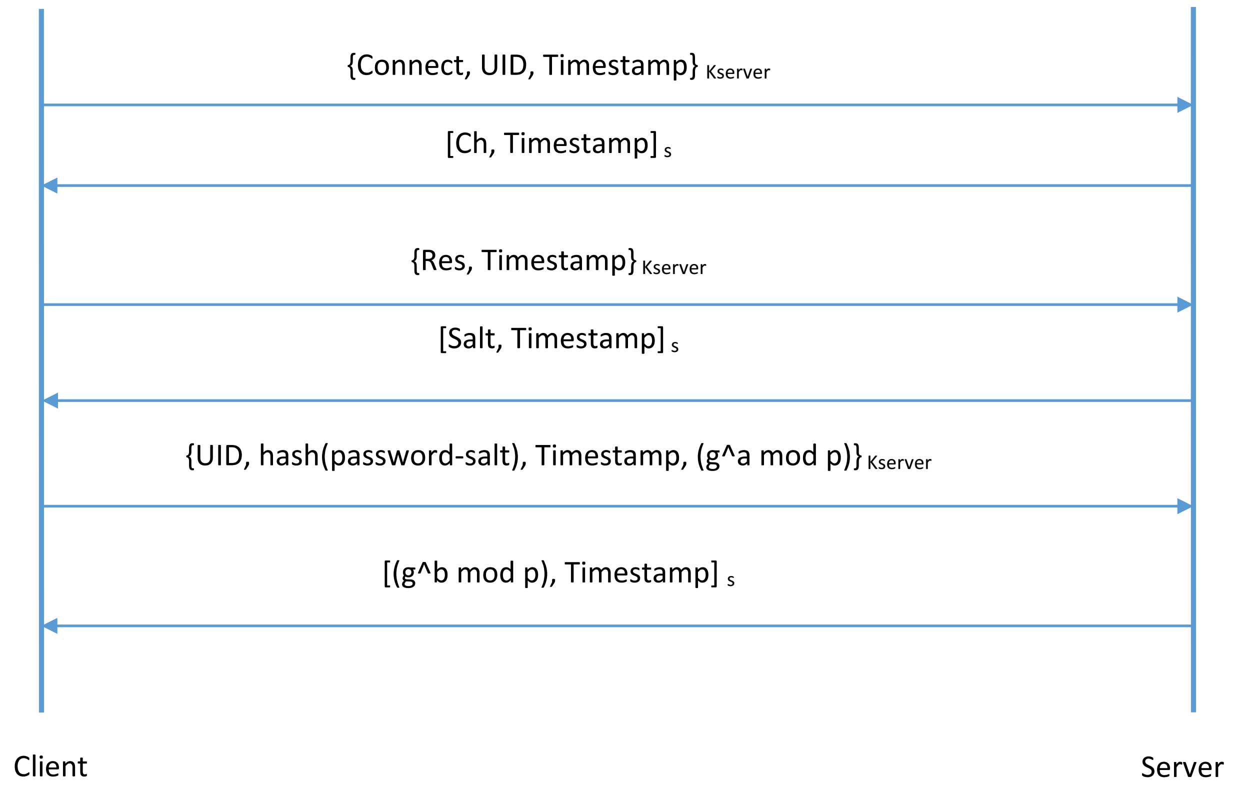
Changes compared to initial design:

The overall design remains almost the same. Below are all the design changes we made in the course of developing our system:

1. The server to client communication now takes place over TCP. Client to client communication is over UDP as previously designed. This design change was due to the following reasons:
    a. We can now remove a client from the list of online clients (when a client disconnects) more elegantly. The server does not need to ping the client every minute.
    b. Easier to keep track of clients and their respective shared secret keys with the server.
2. A timer of 1 minute to try entering a password again after 5 wrong attempts (instead of 3) opposed to 5 minutes as in previous design.
3. We are now using the concept of KDC to generate shared keys between the server and the client. Instead of server sending keys to both clients, the server sends it just to the initiating client and they forward it to the other client. This decreases the load at the

server to send two messages at every message exchange. A new key is generated for every message exchange. Details are explained below.

Client-Server Communication for authentication:

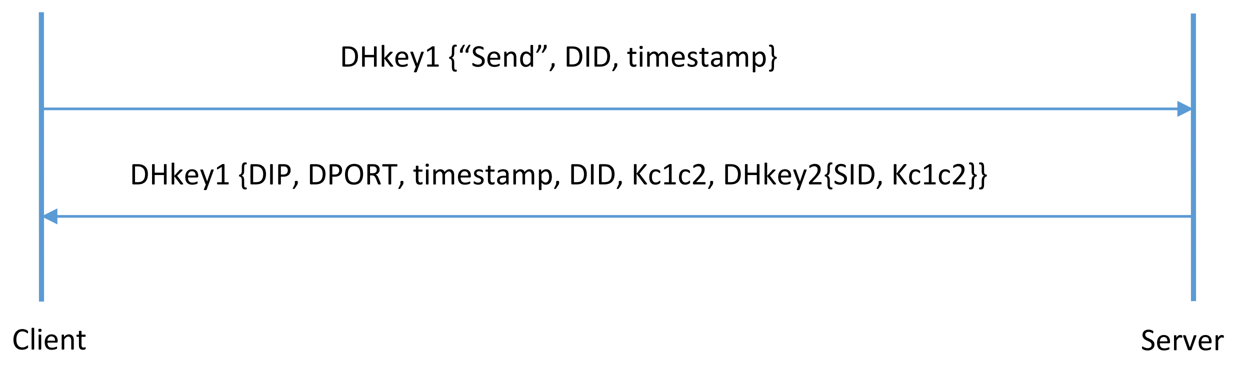

{Connect, UID, Timestamp} Kserver
[Ch, Timestamp] s
{Res, Timestamp} Kserver
[Salt, Timestamp] s
{UID, hash(password-salt), Timestamp, (g^a mod p)} Kserver
[(g^b mod p), Timestamp] s
Client
Server

Shared Secret Key: (g^ab mod p)

Legend:

1. Salt – Random string stored on server to prevent dictionary attacks
2. a – Client's DH secret
3. b – Server's DH secret
4. g, p – Publically knows primes
5. Kserver – Server's public key
6. s – Server's private key
7. timestamp – Unique time value for freshness
8. Ch: Challenge from server (two random numbers)
9. Res: Response from client (multiplication of two numbers given in Ch)

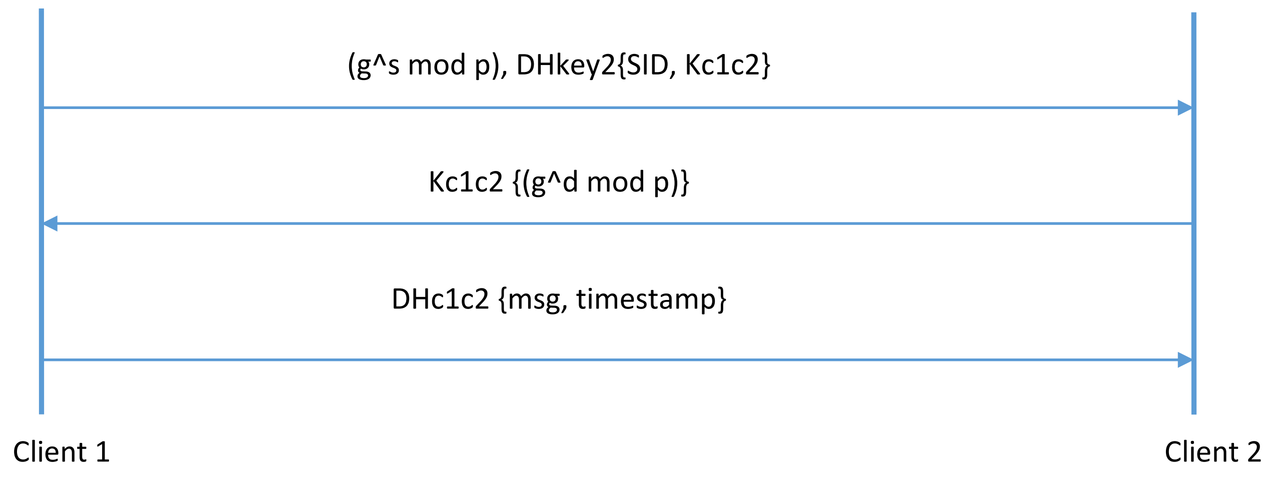
Client-Server Communication for requesting IP and port of destination:


DHkey1 {“Send”, DID, timestamp}
DHkey1 {DIP, DPORT, timestamp, DID, Kc1c2, DHkey2{SID, Kc1c2}}
Client
Server

Legend:

1. DID – User ID of destination
2. DHkey1 – Shared secret DH key between source client and server
3. DHkey2 – Shared secret DH key between destination client and server
4. DIP – Destination IP address
5. DPORT – Destination Port number
6. Kc1c2 – Generated key for source and destination clients
7. SID – User ID of source
8. timestamp – Unique time values for freshness

Client-client communication for exchanging messages:



Legend:

1. g, p: Publically known primes
2. s – Client 1's secret
3. d – Client 2's secret
4. Kc1c2 – Shared secret key between c1 and c2 generated by server
5. DHc1c2 – DH key generated as (g^st mod p)
6. SID – Source user ID
7. msg – message to be sent
8. timestamp – Unique time values for freshness

Client Signoff:

1. TCP connection is immediately closed when a client exits the program
2. Name is deleted from the list of online clients
3. Shared key with the particular client is deleted

Discussion of possible attacks:

1. Online password attacks: A timer of about 1 minute after 5 wrong password attempts. Therefore, the client cannot try multiple passwords continuously.
2. Offline password attacks: SHA256 hashing prevents password from being revealed to a 3rd party entity. Salting helps protecting against dictionary attacks when 2 users have the same password
3. Perfect Forward Secrecy: As we are using Diffie-Hellman keys, even if the user is able to compromise the server and get his private key in the future, he still cannot decode the previous messages.
4. Replay Attack: Protocol is immune to replay attacks as all the messages are attached with timestamps. Both the client and server programs will be checking the freshness of the message by using the timestamp. The message will be ignored if the timestamp is older than a threshold window.
5. DOS: As we are using UDP protocol between clients, the chances of DOS are less as the protocol does not do any resource allocation for messages. Server has challenge-response to protect against DOS.

Flaws:

We could identify one minor flaw in our protocol and a possible fix for it. During the message exchange between two clients, the DH parameter of the first client is sent in the open. An adversary can do a man in the middle attack and change this value which will cause the key at the destination client to be different that the one at source client. The attacker still cannot decrypt any communication between client, but he can deny communication between the two clients.

A fix for this problem would be to encrypt the parameter using the shared key between the two clients. But for this we need the client at the destination to know which key to use to decrypt the incoming message (as he has multiple keys for multiple online clients). For getting this information, the client would need some kind of identification of the source client. He can obtain an identification by asking the server for source client’s IP and PORT address and match it with the IP and PORT of received message. This way he knows which user wants to communicate with him.