

Secure Messenger Design

Aishwarya Vinod, Mallory Gilligan



Setup

Server-Client Model Architecture:

Our design is a server-client model where the server will be used for initial discovery, key exchange, and user authentication but messages will be transmitted between the clients (peer-to-peer). The server and clients will store the shared server-client keys for the duration that the client is logged in and the server is running.

Assumptions

Type of Keys/Who knows what:

- User only knows their password
- Client software does not store anything regarding users
- Client software is configured to connect to the server's address and port at runtime
- Client software is trustworthy
- Client will know the g and p used by the server to store $gWmodp$ at run time
 - Will be shared securely in the real world
- Server knows which users are registered and their $gWmodp$'s
- Clients and Server stores the generated server-client key only for the duration of the logged in session
- Clients temporarily store client-client generated key for the duration of the session.
- IP addresses will not change in the middle of an established session

Protocols

Authentication Protocol:

We have decided to use password-derived properties in the client, specifically Secure Remote Protocol (SRP) because it allows for mutual authentication of the server and client, protects against any eavesdropping, and eliminates the need to transmit passwords over the network. SRP provides us with a shared key between the client and the server, also allowing for future secure communication between the client and server. A new shared session key will be created every time a user logs out and logs back in. This also provides PFS because the session key is forgotten after that session and cannot be used to decrypt past communication

Registration:

Registration is dealt with through a preconfigured file.

Login:

- User provides username and password to client application (assume that the user is already registered).
- Client generates a random private key and computes $g^a \bmod p$ and sends this along with the username to the server.
- Server sends back $g^b + g^W \bmod p$, u , c_1
- Client and server compute shared key $g^{(b(a+uW)) \bmod p}$
- Client sends $c_1 - 1$ encrypted using the new shared secret key to prove to the server that it is the user and also sends c_2
- Server decrypts $c_1 - 1$ using the shared key and verifies that this is the right user and also sends back $c_2 - 1$ encrypted using the shared key.
- Once the client verifies this, mutual authentication is achieved
- AESGCM is used for encryption

Login:

User

Client

Server stores $g^W \bmod p$
Server

Username, password



chooses any a

Username, $g^a \bmod p$



$g^b + g^w \bmod p, u, c1$



Server chooses $b, c1$, and 32 bit number u .

Shared Key: $K = g^{(b(a+uW))} \bmod p$

$K\{c1\}, c2$



$K\{c2\}$



Protocols (Cont.)

Message protocol:

- Client sends a message to the server encrypted using the shared key K with the server which was created during login requesting info about another user.
- The server will send back another message encrypted with their shared key which contains another nonce, a shared key that the two clients can use for initial communication, and the IP address of client B so that client A can message them, This message will also contain another message encrypted using the shared key between the server and client B containing the client to client shared key - ticket to B
- Client A decrypts the part of the message that is encrypted with its shared key with the server and retrieves the shared key with the user it wants to talk to and their address
- Client A will send the message encrypted with the shared key between client B and the server to client B (ticket to B) which contains the shared key between the two (client A cannot decrypt this message). The client will also send its identity and a Nonce.
- Client B will decrypt the client-client shared key using the key it has shared with the server.
- Using the retrieved shared key, client B will decrypt the Nonce and send back the Nonce - 1 along with its nonce
- Client A verifies that it can decrypt the message and that the Nonce sent is the nonce that it had sent (- 1)
- Client A sends back Client B's nonce - 1
- Once Client B verifies that the nonce sent by Client A is the nonce that it had sent - 1, Mutual authentication is achieved.
- Future messages are encrypted using this shared key until the session ends

Logout Protocol:

- When the user decides to log out, the client securely deletes/forgets the session key and any other information stored during the session. So, the client application is left with storing no information relating to the user.
- The client message will also send a logout message to the server.
- The server will then forget the user's shared key, IP address, and other any information other than gWmodp. (The only thing that the server will store is the information that was exchanged during the user's initial registration.)

Client A

Server

Client B

KA{I am A, Can I get info on User B? + NonceA}



KA{NonceA - 1, KAB(shared key), B, Ticket-to-B} where ticket to B is KB{KAB}



KB{KAB}, A, Nonce_B



B decrypts {KAB} using shared key with server. This shared key will now be used for future communication once mutual authentication is achieved

KAB{NonceB-1, NonceC}



KAB{NonceC - 1}



Discussion

PLEASE REFER TO THE REPORT FOR AN ANALYSIS OF THIS SECTION!