Secure Messenger Design

By Yifei Sun, Yiting Wu

Architecture: KDC + Clients

Assumptions: KDC have preregisted list of usernames and corresponding Argon2id hashed SHA256 hashed passwords

Workflow: Password → SHA256(Password) → Argon2id → RSA4096 → Client-Server Key Exchange → Client-Client Key Exchange -> Encrypted Com

Services: Login (w/ username), List, Encrypted Messaging, Logout
- Clients and servers generate new RSA key pair for every new "server session".
- Encrypted communications between clients have signature of the communication content using RSA4096.

Login/Message/Logout Protocols
- Login: send identity (host + port), encrypted password, key exchange
- List: between client and server, encrypted with session key
- Message: client to client, first do key exchange, then encrypted with session key
- Logout: reset client public keys and session keys on server side, must redo handshake to reestablish connection

What Server Has at the Beginning of Each Session

P: a password of arbitrary length provided by client

c_t: time cost factor for Argon2id KDF (int)

c_m: memory cost factor for Argon2id KDF (int)

r: salt

$K = \text{Argon2id}(\text{SHA256}(P), c_t, c_m, r)$

The server has a dictionary of usernames and corresponding K values

Client-Server Ephemeral Session Key Generation

Assumption:

- KDC (server) generates a long-lived public/private key pair
- The key pair will stay the same for entire lifetime of the server (a new one will be generated if the server dies)
- Server and each client will have a randomly generated RSA4096 public/private key pair

Client-Server Ephemeral Session Key Generation

Step 1: A $\longrightarrow$ S: K_A, T_1

Step 2: S $\longrightarrow$ A: K_S, $\{T_1, T_2\}_{K_A}$

Step 3: A $\longrightarrow$ S: $\{A, P_A, K_{\mathrm{AS}}, T_2, T_3\}_{K_S}$ P_A is the hashed password

Step 4: S $\longrightarrow$ A: $\{\mathrm{Op}(T_3)\}_{K_{\mathrm{AS}}}$

From 6 and on: A $\longrightarrow$ S: $\{\text{type} : \text{list}, N\}_{K_{\mathrm{AS}}}$ or $\{\text{type} : \text{logout}, N\}_{K_{\mathrm{AS}}}$ …

Replies will also be encrypted with the session key + nonce

Client-Client Ephemeral Session Key Generation

Step 1: A $\longrightarrow$ B: A, K_A, T_1

Step 2: B $\longrightarrow$ A: K_B, $\{T_1, T_2\}_{K_A}$

Step 3: A $\longrightarrow$ S: $\{K_{\text{AB}}, T_2, T_3\}_{K_B}$

Step 4: S $\longrightarrow$ A: $\{\text{Op}(T_3)\}_{K_{\text{AB}}}$

From 6 and on: A $\longrightarrow$ S: $\{\text{type} : \text{message}, N\}_{K_{\text{AB}}}$

Replies will also be encrypted with the session key + nonce

Client-Client Ephemeral Session Key Generation

Originally, this was supposed to be a modified version of Kerberos, but the final implementation is simplified to a modified version of TLS handshake

Summary

Argon2id KDF

- Memory hard / Long execution time
- Prevents on-/off- line dictionary attacks

Perfect Forward Secrecy:

- Ephemeral session keys
- Server does not know the session keys between two clients

Denial of Service Attacks: Spawn more KDCs

End-points Hiding

- usernames are not exposed in the client-server communication
- Only relatively anonymous public keys are exposed in plaintext