PS4: Secure Instant Messaging System

Team Members:
Kenan Anil Akisik - akisik.k@husky.neu.edu
Hao Wu - wuhao@ccs.neu.edu

The setup:

Architecture:

Type of Messages:

1. Login
    a. Clients send login requests to server
    b. Server Initiates protocols
2. Logout
    a. Clients send logout requests to server
    b. Client notifies other clients about logging out
    c. Other clients update their address book
    d. Server updates its address book
3. List
    a. Client request the list of online users
    b. Server responds with the list of users
4. Send
    a. Client notifies server to send a message to another client
    b. Server attaches the necessary information for the communication between two entities to its response

Note: Detailed explanation can be found under Protocols

Server:

1. The server will store an address book.
    a. format <username(unique), hash of the password (SHA256), IP address, Session Key (K_{AS})>
2. The server will not store any password hashing, instead, it will store the 2^W% p (W is a function of password)
3. Prime number p is calculated based on client's password and being sent to the server during registration.

Client:

1. Address book
    a. Format <username-of-another-client(unique), IP-of-another-client, Session Key (K_{AB})
2. The prime number p is a 512-bit “Sophie-Germain” prime((p-1)/ 2 is also prime) derived from the user’s password f(password).

Assumptions:

Methods used for encryption and integration:

1. Asymmetric encryption: 2048-bit RSA
2. Symmetric encryption: AES
    a. Block cipher mode: GCM (preferred), CTR if GCM is not available
    b. Block size: 128-bit
    c. Key size: 128-bit
3. Hash function: SHA-256

Assumptions about the client (user and work station):

1. The user do not remember anything other than her username and password
2. The workstation knows the IP address of the server.
3. The workstation knows server’s public key.
4. The workstation does not store session key, private/public key for each user.
    a. It generates new RSA keys whenever a client logs in and forget it once the user logs out.

Assumptions about the Server:

1. All of the users are pre-registered in the server.
2. Server saves hashmap <username, 2^W mod p, p, IP address> for each user.

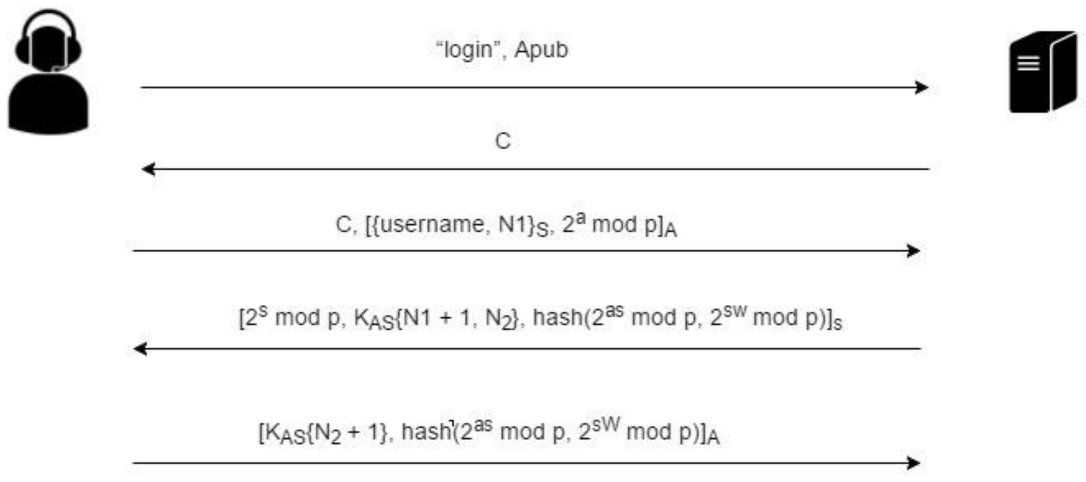
Protocols:

Notations:

x mod y: The remainder of x divided by y.
hash(x) = SHA-256(x)

hash'(x) = SHA-512(x)
W = hash(password)
A = Alice(Client)
B = Bob(Client)
S = Server
A_{pub} = RSA public key of Alice (generated at login)
C = f(IP | secret) = Cookie
K_{AS} = DH session key between A and Server
K_{AB} = DH session key between A and B
K_{AB}\{m\} = AES encryption using session key K_{AB}
a, **s, b** = 512-bit random secrets that will be forgotten after establishing session keys K_{AB} and K_{AS}
$N_{1\text{-}12}$ = nonces (128-bit integer generated by pseudo-random generator)

Authentication Protocol (Login):

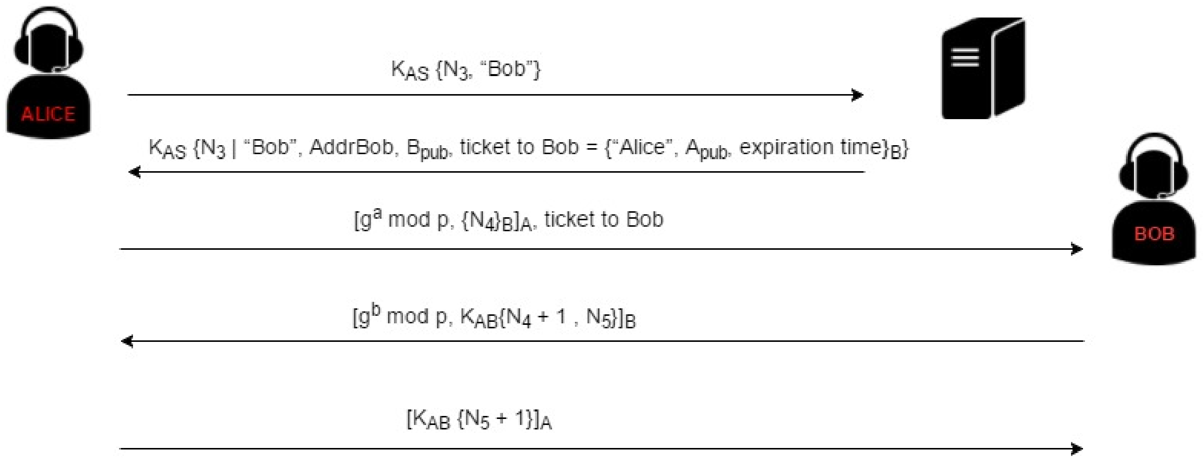

"login", Apub
C
C, [{username, N1}$_S$, 2^a mod p]$_A$
[2^s mod p, K_{AS}\{N1 + 1, N_2\}, hash(2^{as} mod p, 2^{sw} mod p)]$_s$
[K_{AS}\{N_2 + 1\}, hash'(2^{as} mod p, 2^{sW} mod p)]$_A$

Protocol Walkthrough:

1. The client "Alice" first generates her RSA key pairs, and send her public key along with the "login" command to the server in clear text.
2. The server will send a **stateless cookie** "C" (C=hash(Alice's IP address, secret) back to Alice.
    a. The secret will be a 64-bit nonce generated by the server. It is unique to the server and regenerated whenever the server restarts.
    b. The stateless cookie is aim to prevent Denial-of-Service attack towards the server.

3. Alice will then reply to the server:
    a. C, the stateless cookie
    b. Alice’s username and a nonce which is encrypted using server’s public key.
        i. Username is for server to lookup the matching password variate (2^a mod p).
    c. Alice will select a secret integer “a” and calculate “2^a mod p.” Then Alice sends the Diffie-Hellman partial key(2^a mod p) for the server to generate K_{AS} session key.
    d. The important parts of the message are being signed using A’s private key to guarantee the message integrity and preventing man-in-the-middle attacks.
4. Server response, complete Diffie-Hellman key exchange:
    a. The server picks its secret integer “b” and calculate “2^s mod p” for its part of Diffie-Hellman key.
    b. The server then sends encrypted nonces, N1+1 and N2 to Alice to counter replay attacks.
    c. The server hashes 2^{AS} mod p (the session key), and 2^{SW} mod p (to authenticate itself to Alice) using **SHA-256**.
    d. Server signs the above messages to ensure the integrity, authentication and nonrepudiation.
5. Alice authenticate herself to the server:
    a. Alice sends the incremented nonce using the session key.
    b. Alice hashes the 2^{AS} mod p, and 2^{SW} mod p using **SHA-512.** This step proves that Alice knows the password hash W to the server.
    c. The server calculates the hash’(2^{AS} mod p, 2^{SW} mod p). If it does not match with the version which Alice sent, the server will reduce the attempt times left for Alice to enter the password.

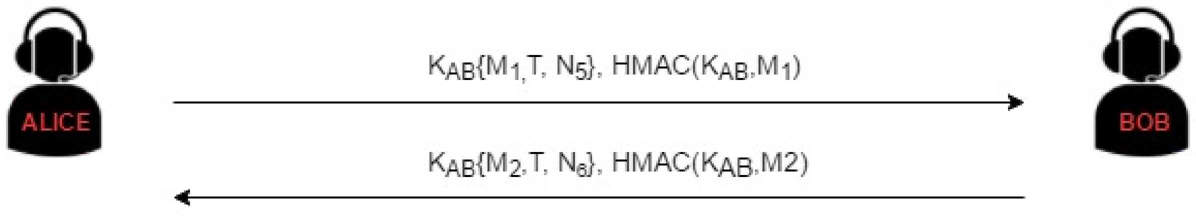
Key establishment protocol (and authentication with peers)


ALICE
BOB
K_{AS} {N_3, “Bob”}
K_{AS} {N_3 | “Bob”, AddrBob, B_{pub}, ticket to Bob = {“Alice”, A_{pub}, expiration time}$_B$}
[g^a mod p, {N_4}$_B$]$_A$, ticket to Bob
[g^b mod p, K_{AB}{N_4 + 1 , N_5}]$_B$
[K_{AB} {N_5 + 1}]$_A$

Protocol Walkthrough:

1. The client Alice wants to talk with Bob. Alice sends N_3 and “Bob” and encrypt these with her session key with the server (K_{AS}).
    a. N_3 is a 64-bit nonce created by Alice and it is unique to every session.
2. Server then replies to Alice and encrypt all with session key:
    a. N_3 | “Bob”, response to Alice's challenge
    b. IP address of “Bob”
    c. RSA public key of Bob
    d. Ticket to Bob which consists of Alice’s username(“Alice”), RSA public key of Alice and expiration time of the ticket. Ticket is encrypted with Bob’s public key.
3. Alice sends her contribution to DH, N_4(challenge for Bob), and ticket to Bob
    a. Alice signs DH contribution with her private key to provide perfect forward secrecy.
4. Bob computes session key as $K_{AB} = g^{ab}$ mod p and replies with N_4 + 1 and a challenge N_5 for Alice encrypts these with the session key and then sign them together with the session key with his private key.
    a. N_4 and N_5 are 64-bit nonces and are encrypted with session key between the clients.
5. Alice computes session key as $K_{AB} = g^{ab}$ mod p sends her response to the challenge N_5 encrypts it with session key and signs it.

P2P Messaging Protocol:


ALICE
BOB
K_{AB}{M_1,T, N_5}, HMAC(K_{AB},M_1)
K_{AB}{M_2,T, N_6}, HMAC(K_{AB},M2)

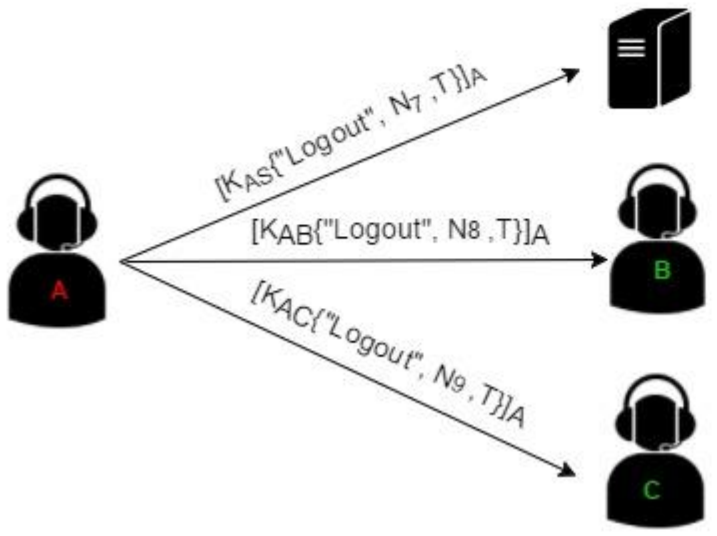
Protocol Walkthrough:

1. Alice sends message M_1 and time-stamp(T), encrypts them with session key, computes HMAC(K_{AB},M_1) and sends it along with the encrypted message. HMAC uses SHA-256.
2. Bob sends message M_2 and time-stamp(T), encrypts it with session key, computes HMAC(K_{AB},M_2) and sends it along with the encrypted message. HMAC uses SHA-256.

The system will be implemented using AES-GCM mode, which provides both **confidentiality** and **integrity** of the data. The HMAC sign explicitly specify the integrity function for other modes other than GCM.

If A or B being idle for more than 10 minute, the session key will be expired. The next person initiate the conversation needs to establish the new session key.

Logout Protocol:

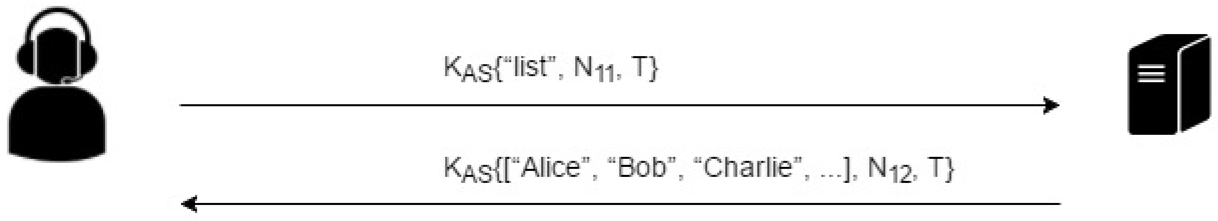


Protocol Walkthrough:

1. Alice sends “logout” and time-stamp(T) and encrypts these with shared session key with server and signs it with her private key.
    a. Alice forgets the session key shared with the server
    b. Server updates its address book, forgets all information related to Alice
2. Alice sends the same message that she sent to server to all clients that she has a session with
    a. Other clients forget the session keys that they share with Alice and update their address book.

Use Cases:

“list” command:


K_{AS}{“list”, N_{11}, T}
K_{AS}{[“Alice”, “Bob”, “Charlie”, ...], N_{12}, T}

After the user logged in to the server, she is eligible to send a ‘list’ command to the server to get all the online users’ username.
The detailed process of the “list” command are as follow:

1. User type in “list” in the terminal
2. The client will assemble a packet with the command, a timestamp and a nonce. The packet will be encrypted with the session key between the client Alice and server.
    a. Message format: K_{AS}{“list”, N_{11}, T}
    b. T is the timestamp for countering **replay attack** against the server. If the timestamp has more than 1 minute differences with the server’s current time, then the server will simply ignore the list request.
    c. N_1 is the nonce for countering **offline dictionary attack** from eavesdroppers. Since the message content is quite simple (“list”, current time), it is possible for an eavesdropper to record message and perform dictionary attack. The nonce adds much more complexity to the message body, so that dictionary attack would also be less possible to success.
3. The server will serialize all the usernames in its address table, then assemble a response packet to the list request.
    a. Message format: K_{AS}{[“Alice”, “Bob”, “Charlie”, ...], N_{12}, T}

“send” command (send USER MESSAGE):

1. Once a user types in a complete send command, its workstation will first perform a lookup on its address book. If the receiver exists in the address book (which means the receiver is online and sender has the session key), the client would simply send the message to receiver’s ip.
2. If the receiver is not currently in sender’s address book, then the client has to go through the “Key Establish Protocol” to gain the session key with receiver. Then use the session key to talk to the receiver.

Services:

1. **Protection against the use of weak passwords**
    a. Augmented form of PDM (Password Derived Moduli)

        i. PDM as a strong password protocol prevents eavesdroppers and impersonators from doing offline dictionary attack.

   b. Online
        i. Clients can only enter a wrong password 3 times and after that the account will locked the account for an hour.

   c. Offline
        i. The server does not store client's password or password-equivalent; instead, it stores the $2^W \% p$ (where W is the hash of the password). So the attacker who has stolen the database only gains 2^W and p. This feature prevents offline-password attack by relying on the difficulty of solving "discrete logarithm problem."

2. **Resistance to DoS attacks**
   This is provided with the addition of the cookie(C). C is a value created by the server and it is C = hash(IP address, secret). When server receives a login request from an IP source address, server sends this cookie in clear to the source IP address. Server doesn't initiate the login protocol unless it receives the same C from same IP source address. This assures that person sending login requests can receive packets sent to the IP address that he/she claims to have. Therefore this method protects against someone trying to do a DoS attack by sending overwhelming number of login requests by spoofing IPs.

3. **End-point hiding**
   Usernames of the endpoints are never sent in clear and always encrypted with either a symmetric session key or an asymmetric key. Therefore an eavesdropper won't be able to tell who is talking with whom.

4. **Perfect forward secrecy**
   Diffie Hellman is used for session key generation and the DH contribution of both sides are signed with their private keys. Since it is not possible to break Diffie Hellman in reasonable time and since contributions are signed it is not possible for someone who recorded to whole conversation and compromised both ends to decrypt the conversation.

5. **Man-in-the-middle**
   Usage of signatures ensures that nobody else can forge the same message without having the private key of the message owner.

6. **Reflection Attack**
   The only message sent in cleartext in our system would be at the beginning 3 messages of the Login Protocol. However, the eavesdropper/attacker would not able to find out the content within the encrypted message ($\{\text{username}, N_1\}_S$) unless he obtained the client's private key. It is very unlikely that the attacker could deduce the content of the message because of the nonce.

7. **Data Integrity**
   For the login protocol and key establishment protocol, the integrity of the message are ensured by the public key signatures from both client and server. The receiver will verify the signature using RSA public key for the message.

   Once the session keys are established (client to server or client to client), then the data integrity will be provided by AES-GCM mode or HMAC (for other cipher modes). For GCM mode, the receiver could verify the message integrity. For other cipher modes, the receivers have to calculate the HMAC and compare it with the HMAC in the message.

PS4 Required Questions:

Q : Does your system protect against the use of weak passwords? Discuss both online and offline dictionary attacks.

A: Answered in Services.1

Q: Is your design resistant to denial of service attacks?

A: Answered in Services. 2

Q: To what level does your system provide end-points hiding, or perfect forward secrecy?

A: Answered in Services.3 and Services.4

Q: If the users do not trust the server can you devise a scheme that prevents the server from decrypting the communication between the users without requiring the users to remember more than a password? Discuss the cases when the user trusts (vs. does not trust) the application running on his workstation.

A1: In our design, the server does not provide the symmetric key between client A and B. Client A and B establishes their own symmetric keys based on an alternation of Diffie-Hellman key exchange protocol. Thus, the server will not be able to decrypt messages between A and B.

A2:

a. If client trusts the workstation, workstation can store all the session keys a client currently shares with others and RSA keys that are generated when a client logs in.
b. If client doesn’t trust the workstation, then the system will still provide Perfect Forward Secrecy, and the client knows that her message will not be decrypted by an

eavesdropper. However, the system is not able to prevent the malware on client's workstation to record user's inputs.

Possible Changes for the Design for Implementation:

Due to the limited time and computation power that given for the project, here are some minor changes to the design for the actual implementation:

1. In the login protocol, we designed the prime number p to be a 512-bit "Sophie-Germain" prime. We might not able to find a large Sophie-Germain within reasonable time frame in our developing environment. So we will start developing with regular prime number.
2. In the login protocol, the server needs to calculate 2^{sW} mod p as part of the design (W is password hash integer). If W is too large, then our laptop may not able to handle the modular exponentiation. We might have to choose a hash function to generate shorter hash value of the password (e.g. SHA-128).

Changes:

1. For login protocol, instead of using "Augmented Strong Password Protocol" we used the "Basic Form" which are given in text book Part2: Authentication – 12.3. Strong Password Protocols. We used the basic form because it was taking long for clients to login. This was because for DH key exchange it was computing an additional 2^{bW}modp along with 2^{ab}modp.

2. In key establishment protocol we realized that to promote PFS we only need to sign DH contributions instead of signing the whole message with client's public keys. Thus we only signed the DH contribuions.