Group 2 Design document **(Option A: secure, multi-party chat application)**

Tongyu Zhou, Serena Wang, Ye Won Hong

1. Overview of system architecture and operation

We are creating a secure group chat application. The basic operations of the application include creating chat groups for more than 2 participants and sending messages to the group. Our application uses end-to-end encryption methods for each message, where the server only delivers the message as we do not trust the server. Each group chat will be a session with a unique group key. All messages in the session will be encrypted and decrypted using this group key, which is generated by the chat created, transported securely to each participant, and stored in each of their password-protected files.

2. Attacker model (assumptions made about attackers relevant for your application)

- Goals of the attacker:
    - Read a message sent in the group chat
    - Forge a message sent in the group chat
    - Replay a message
    - Modify a message
    - Break the group key
- Capabilities:
    - The attacker has full control over the messages sent between participant and server, but he/she does not have access to the database mentioned later and cannot break the underlying cryptographic primitives.
    - The attacker also has access to parallel computing and can use it to brute force passwords/keys in a clever way if they are predictable, but still cannot rely on extremely computationally heavy methods.
- Note:
    - We acknowledge that the server is not trusted, and can withheld any number of messages from the participants in the group chat. The attacker may also intercept every message and withheld them. However, we cannot protect against this, as this requires directly controlling the operations of the server.
    - We assume that none of the participants acts as an attacker.

3. Security requirements identified based on the attacker model

- Protection Against Replay Attack
    - Each participant keeps track of the participant ID, public key, and a sequence numbers of all the other participants (if message is accepted, we increment sequence number), under the assumption that participants have public key certificates that are issued off-line
- Integrity and Confidentiality Protection

    - Authenticated encryption with signature: a participant can digitally sign with his/her private key and send out to all the other participants to verify with the public key of the signer
      → We chose to use a signature is over a MAC here because it is more efficient (don't need another MAC key) and more secure (one user cannot pretend to be another user)
- Guaranteed Pseudo-random Key/Nonce Generation
    - Using cryptographic PRNGs keeps our keys/nonces unpredictable
- Password Are Stored In A Protected File
    - Only users with a verified password can access the private key and group key
- Passwords cannot be guessable or brute-forced via dictionary attack
    - We do not actually use the plaintext-version of the password entered by the user, but derive another key from the password that is resistant to dictionary attacks

4. Detailed description of your system satisfying the security requirements

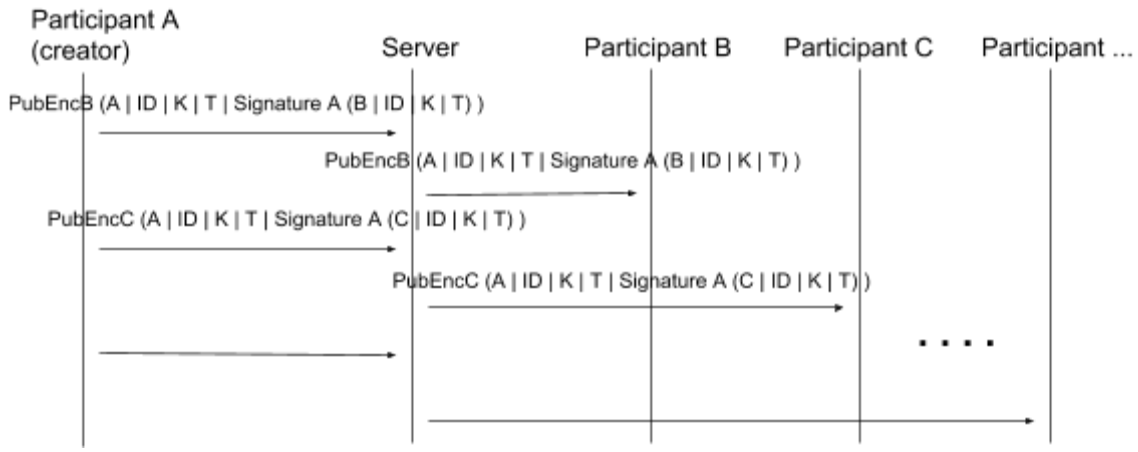
Handshake (for each new group chat):

- This protocol includes key derivation, key exchange, and establishment of the group ID.
- The group key will be derived and exchanged using public key crypto key transport (ISO 11770-3/3) as follows:
    a. Participant A generates the group key, K, and creates the chat group/sends invitations to other participants (e.g. Participant B, Participant C).
    b. Participant A will send "invitation messages" to each participant who they invite through the server. These "invitation messages" will contain K, signed with Participant A's private key, and encrypted with Participant B's public key.

       Invitation (key exchange) message format (ID = group ID):

```
                +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
PubEncB (       | A | ID |  K  |  Time   |   Signature A(B | ID | K | Time)  …|              )
                +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

       This process is repeated for each other participant in the group chat. Each message is directly transmitted through the server without server modification.



Password-protected file:

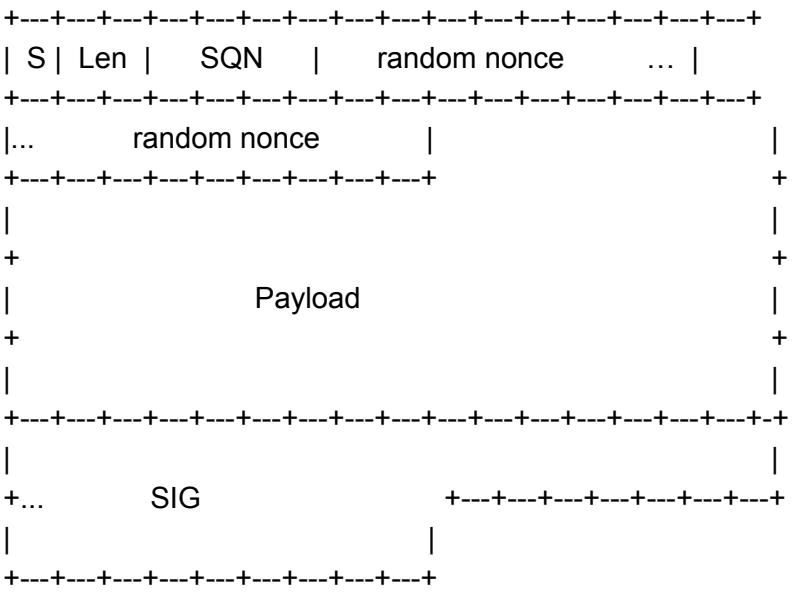
To protect the secret keys of each participant, we will store them in a password-protected file. To prevent dictionary attacks, we differentiate between participant password (P_c) and the actual password that protects the file (P_f).

- P_f = PBKDF2(PRF, P_c, salt, icount, length)
- We can use HMAC-SHA256 for the PRF
- Protects the following keys
    - Private key of each participant (from the pre-established certificate)
    - Group key

Whenever a participant logs into the application, we will ask for their password (P_c). From that password, we derive the key (P_f) using the PBKDF2 function as shown above. With that we decrypt the file which contains the private key and group key, ready for signing and encryption.

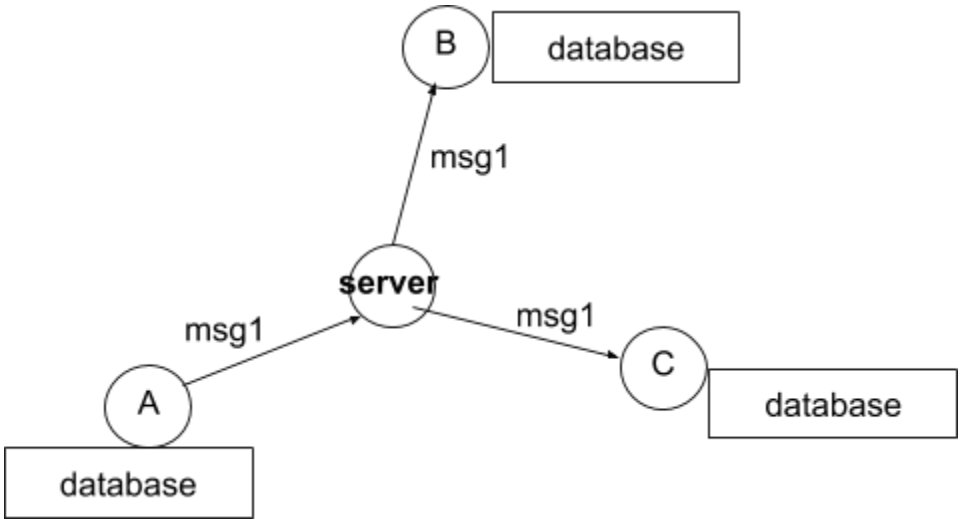
Sending and receiving messages:

1. ***ENCRYPTION***:
   When the user wants to send a chat message, they first encrypt group chat ID concatenated with chat message with CTR mode and then sign the encrypted message to obtain a signature.
    - input: plaintext message X, group key K, nonce N
    - Output in the following format:
        - Sender → S
        - Length of plaintext → Len
        - SQN_SEND → SQN_SEND + 1
        - N → nonce
        - Encrypted group ID|message → Payload
        - Signature S → SIG
    - Our final message format is therefore:



(Note that only the payload is encrypted. This is okay because even if an attacker tries to modify S to impersonate another sender, this will not work as the same sender also signs the message.)

2. **Message Transmission**: In this example, Participant A is trying to send a message to Participant B and Participant C through the server, which relays the messages without modification.



(Note: msg1 is in the message format shown in Encryption part. Database is the one mentioned below.)

3. **Sequence Number Tracking**: We will handle replay protection with sequence numbers.

- Each participants stores their own send sequence number (SQN_SEND)
- Each participant of the group chat other than the server will also maintain a database (shown below) of the last SQN_RECEIVE number seen and public key of every group member other than themselves. For example, participant Z might have the following database:

Participant ID	SQN_RECEIVE	Public Key
A	SQN_a	xxxxx
B	SQN_b	xxxxx
C	SQN_c	xxxxx
...	...	...

4. ***DECRYPTION***:
   - Check if the SQN of the message received is within a certain window of the SQN for that sender in the database. If so, continue. Else, reject.
   - Verify the signature with the public key in the database. If success, continue. Else, reject.
   - Decrypt using the message format to figure out group ID and plaintext message. If group ID belongs to the current group, accept the message. Else, reject.