Multi-party chat application specification

Dömötör Szilárd *AE9EC3*
Juhász Márton *GASYQY*
Székely Gábor *EDVTAZ*

Functional requirements

In addition to the requirements defined in the homework handout, we specify the functional requirements in this section.

Channels

- The size of a channel is not limited
- There can be multiple channels
- A channel can be created by a single user
- Any user in a channel can add new users to that channel
    - Users cannot leave channels, if a user needs to be removed, a new channel can be created without that specific user

General usage

- Upon starting the application, user must supply username and password (to unlock her private keys)
- If the supplied password is correct, the list of channels (that the user is part of) is displayed
- User can go back to main menu from any sub-menu
- User can create new channel or select existing one to enter
    - If new channel option is selected:
        - A prompt is displayed for the new channel name
    - If an existing channel is selected:
        - Message history is displayed
        - New messages can be sent
        - New users can be added

Attacker model

Goals of the attacker (including the server)

- Recover private key of some other participant

- Recover symmetric key of a channel she is not part of
- Add a user in a channel that the attacker is not part of
- Read or send messages in channels she is not part of
- Replay or modify messages
- Send messages in the name of someone else

We trust the server to

- Store and forward messages in the order they arrived
- Correctly dismiss messages, that have incorrect sequence numbers
  - This is needed in order to ensure that every client receives the same message, if there is a sequence number collision
  - The server can't verify the authenticity of the sequence number, that is done in every client
- Apply the current time to messages as a timestamp

Capabilities

- The attacker knows the public key of every user (as do every user)
- The attacker can be a user invited to a number of channels
  - in this case we want to protect channels that she is not part of
  - in the channel that she is part of, we ensure the authenticity of the source of the messages and the order of the messages
- The attacker has man-in-the-middle positions in every link and can view, modify and delay (for an indefinite period of time) messages
- The attacker can view all data that is available to the server except the secret keys of the server
  - If the attacker can access the secret keys of the server, the ordering and timestamps of the messages may be modified, but the content of the messages will remain hidden and won't be replayable

Security requirements

Attacker

- Can't view, modify or send messages in channels she isn't part of
- Can only send messages in her own name in the channels she is part of
- Can't invite users to channels that the attacker isn't part of

System architecture

There are two types of entities in our system: server and user. The users communicate with each other through the server. Users can communicate with each other via channels: when a user sends a message to a channel, each user that is part of that channel will receive that message through the server.

Server

Each time a client wants to send a message, it is sent to the server. The server's main message handler determines the message type, and acts accordingly: after performing some basic checks, broadcasting the message to all parties that need to receive it (i.e. the users that are part of the channel in question), and making a persistent copy of it, so offline users can later receive it.

User

A user is identified by a unique user ID. Every user has a signing and an encryption key, and the public parts of these keys are pre-shared with each user, so it is known to each participant what user IDs belong to what public keys. Messages received by the user are stored locally. The private keys of the user are stored in persistent storage, encrypted with a secret password.

Channel

A channel is identified by its channel ID. A channel holds the following information:

- IDs of users that are part of the channel
- For each user in the channel, the channel key (symmetric key), encrypted with her public encryption key
- The messages that have been sent in the channel, encrypted with the channel key

Server architecture diagram

```
  +--------+          +------------------------+
  |        |          |                        |
  | socket +<-------->+ server message handler |
  |        |          |                        |
  +--------+          +--+----------------+----+
                         |                ^
                         |                |
                         |                |
                         v                v
               +---------+---+         +--+---------+
               |             |         |            |
               | broadcaster +<--------+ persistent |
               |             |         | storage    |
               +-------------+         |            |
                                       +------------+
```

Client architecture diagram

```
                    +----------------+
+--------+          |                |
|        |          | client message |
| socket +<-------->+    handler     |
|        |          |                |
+--------+          +-------+--------+
                            ^
                            |
                            |
                            v
                  +---------+----------+       +--------------+
                  |                    |       |              |
                  | crytographic check +<------+ user message |
                  | and en/decryption  |       |     input    |
                  |                    |       |              |
                  +---+-----------+----+       +--------------+
                      ^           ^
                      |           |
                      |           |
                      |           v
             +--------+-+       +-+----------+
             |          |       |            |
             | keystore |       | persistent |
             |          |       | storage    |
             +----------+       |            |
                                +------------+
```

Cryptographic protocols

Message types

All message detailed below are sent between the clients and the server through a secure channel negotiated one-by-one. When the server sends a message to a client, it also appends a timestamp to the message, marking when that message was received from the other client. The server broadcasts messages, even to the original sender, thus serving as a confirmation, that the message was received and accepted. If the sender doesn't receive an echo of the message it sent, it can automatically retry.

The following message types are used in our protocol.

Add user to channel

$$payload = userID_A || userID_B || channelID_C || E_{pk_B}(K_{channel})$$

$$msg = payload || S_{sk_A}(payload)$$

- This message is used for creating a new channel or adding a new member to it.
- The above message is user `A` adding user `B` to channel `C`.
- The channel key is encrypted with the public key of user `B`, this way sharing it with her and only her.
- The whole message is signed by user `A`, proving that the message is originating from a user that is part of channel `C`.
- If `A == B`, and `C` doesn't exist yet, this is the creation of a new channel
  - In this case, if `C` already exists, this message is ignored
- This message is also ignored if `A` is not part of channel `C`, or if the signature is invalid.
- This message is broadcasted (by the server) to all users in channel `C`, including the new user `B`.

Communication message

$$assoc\_data = channelID_C || ChannelSeqNum_C || userID_A || UserSeqNum_A$$

$$enc\_data = message\_text$$

$$msg = E^{GCM}_{K_{channel}}\big(assoc\_data, enc\_data || S_{sk_A}(assoc\_data || enc\_data)\big)$$

- This message is used for sending a text message to channel `C`, the sender's ID being `A`.
- This message is encrypted with the channel key, in `GCM` mode (with a random IV), with `channelID`, `ChannelSeqNum` and `UserSeqNum` being the associated data so the server can determine the channel, and possible sequence number duplications. The rest of the data is encrypted. Both the encrypted data and associated data are signed with the sender's signing key, so the sender cannot be impersonated.
- The server broadcasts this message to all channel `C` participants.
- Recipients check the following, and only accepts the message if all checks are successful:
  - User `A` is part of the channel `C`
  - The $ChannelSeqNum_C$ is higher than all previous values in channel `C`
  - The $UserSeqNum_A$ is higher than all previous values associated with user `A`
  - All MACs and signatures are valid

Secure channel init message

The above mentioned secure channel is built with the following message, sent by the client to the server. `initconn` is sent by the client, and `initkey` is sent by the server.

$$initconn = userID || rnd$$
$$initkey = E_{K_{user}}\big(symkey, S_{server}(rnd, symkey)\big)$$

- when a user wants to connect to the server, she sends a fresh random value and her `userID`
- `symkey` is generated by the server, and will be used for the secure connection in GCM mode, and encrypt `initkey` message with the respective user's public key
- the client will decrypt the message and verify the signature (with the server's key) on `symkey` and `rnd, and if correct, will switch to this key on the current connection

- other

Security Analysis

We consider the following scenarios according to the previously described attacker model and capabilities.

Recover private key of user

The private key of users are stored encrypted, and they can accessed if they are correctly decrypted using the secret password of the given user. These private keys are never sent through the network.

Recover symmetric key of a channel

The channel key is generated by the user creating a given channel. Each new channel participant receives the key, encrypted by their public key. Thus, the attacker can only recover the channel key if she breaks the public key encryption or steals the private key of some channel participant.

Add user to arbitrary channel

The message that is used to add participants to a channel is signed with the private signing key of a user who is already part of the given channel, otherwise the message will be rejected. If the signature is invalid, the message will also be rejected.

Read or send messages in arbitrary channel

In order to read or send messages in a channel, the channel key is necessary to encrypt or decrypt the message. Additionally, even if the attacker can steal the channel key somehow, she will only be able to read messages, because messages need to be signed with the sender's private signing key.

Message forgery, modification and replay

Message modification is prevented by GCM mode and the sender's signature on the message, so even if the attacker knows the channel key she can not modify the message or impersonate anyone.

Message replay is prevented by the use of a channel-wide sequence number.