SKISSM White Paper

Gen-Cher Lee, Hsuan-Hung Kuo

April 12, 2023

version 1.2.3

Academia Sinica
中央研究院

資訊科技創新研究中心
Research Center for Information Technology Innovation

Contents

Introduction

This white paper provides a technical overview on the end-to-end encryption (E2EE) protocols and security aspects implemented by SKI software security module (SKISSM).

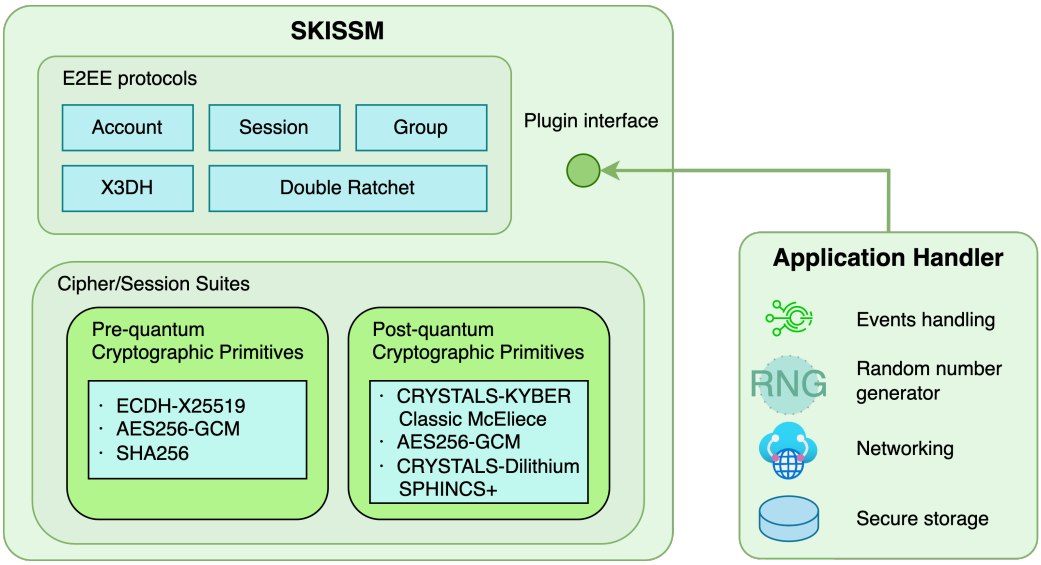

Fig. 1: SKISSM software architecture

SKISSM implements the Signal protocol [1][2][3][4] and aims to make it easy to build versatile end-to-end messaging applications. SKISSM supports asynchronous and out-of-order end-to-end message encryption scheme. It also supports one-to-one messaging and group messaging for registered user with multiple devices. The two crucial security properties are provided:

- **End-to-end encryption**

  Only sender and recipient (and not even the server) can decrypt the content.

- **Forward secrecy**

  Past sessions are protected against future compromises of keys or passwords.

The extensible software architecture[Fig. 1] implemented by SKISSM has been released as open source project[17].

Cryptographic Primitives Support

A cipher suite in SKISSM is a software interface constructed by the following cryptographic functions. SKISSM provides an implementation that utilizes the curve25519-donna[10] and mbed TLS[11] library.

- **get_crypto_param** Get the parameters of the cipher suite.
- **asym_key_gen** Generate a random key pair that will be used to calculate shared secret keys.
- **sign_key_gen** Generate a random key pair that will be used to generate or verify a signature.
- **ss_key_gen** Calculate shared secret key.
- **encrypt** Encrypt a given plaintext.
- **decrypt** Decrypt a given ciphertext.
- **sign** Sign a message.
- **verify** Verify a signature with a given message.
- **hkdf** HMAC-based key derivation function.
- **hmac** Keyed-Hashing for message authentication.
- **hash** Hash function.

SKISSM provides two cipher suites currently:

- **E2EE_CIPHER_ECDH_X25519_AES256_GCM_SHA256**

  Used cryptographic primitives:

  ECDH-X25519, AES256-GCM, SHA256

- **E2EE_CIPHER_KYBER_SPHINCSPLUS_SHA256_256S_AES256_GCM_SHA256**

  Used cryptographic primitives:

  Kyber[12], SPHINCS+-SHA-256[13], AES256-GCM, SHA256

SKISSM Plugins

SKISSM implements a plugin interface to achieve module flexibility for engaging variant application platforms. There are four kinds of plugin handlers [Fig. 2]:

- **Common handler**

  A common handler provides a set of platform dependent functions for generating time stamp, random number, and universally unique identifier (UUID).

- **Event handler**

  An event handler is used to receive events form SKISSM while performing E2EE protocol request and processing protocol messages from server. User application can make use of this efficient notification mechanism to catch the changes of states that are maintained by SKISSM.

- **Database handler**

  A database handler is provided to help SKISSM keeping data persistency. The state accessibility of user account, one-to-one sessions, and group sessions are finely implemented through a range of database functions.

- **Protocols handler**

  The fourth handler is designated to provide a layer of protocol transportation that helps SKISSM forwarding the request messages to E2EE Server. The response message of each request is then propagated back to SKISSM to maintain the states of account and sessions. SKISSM will use the functional interface of database handler to maintain data persistency of affected sessions and account.

Addressing

To specify an end point address for end-to-end encryption, SKISSM provides an E2eeAddress struct to represent user address or group address as shown in [Fig. 3]. An E2eeAddress with PeerUser type is obtained from E2EE server after user registration. The PeerUser data is specified by a user_id that is created uniquely by server and a device_id that is provided by user. An alternative E2eeAddress with PeerGroup type is obtained from E2EE server after group creation. The PeerGroup data is specified by a uniquely assigned group_id from server. The uniqueness of user_id and group_id is assured in the scope of the same server domain while the uniqueness of device_id is kept by user application.

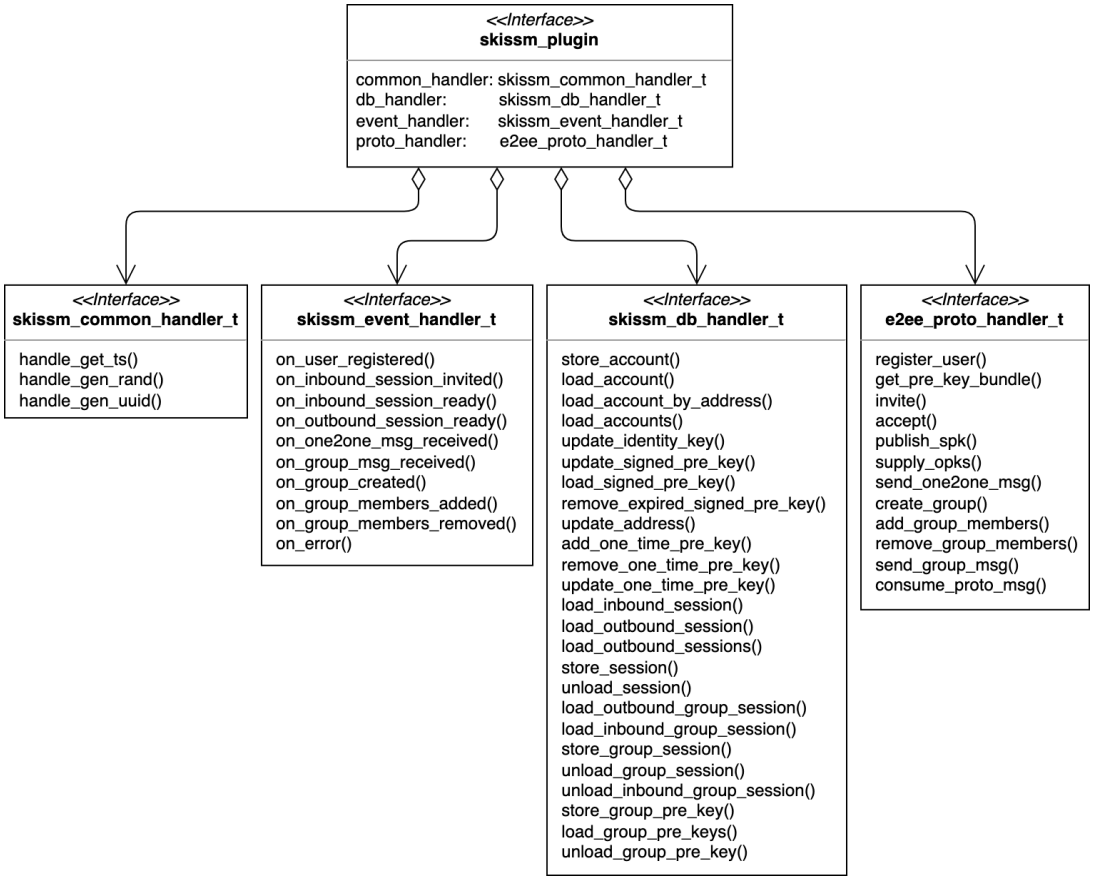

Fig. 2: SKISSM plugin interface

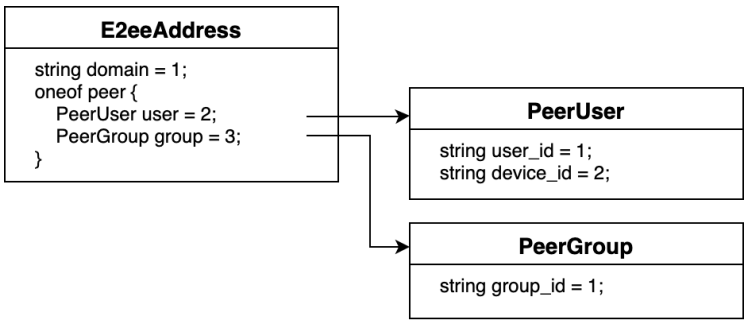

Fig. 3: E2eeAddress struct

Account

An account keeps user's address and three types of keys that are used by E2EE schemes. A complete set of keys include a long-term key-pair (IdentityKey), a mid-term key-pair (SignedPreKey), and a bunch (100 as default) of one-time used key-pairs (OneTimePreKey) [Fig. 4]. On the stage of user registration, the public part of this set of key-pairs will be uploaded to E2EE server to help together with other peers establish sessions for messaging.

Moreover, the SignedPreKey has a special time-to-live (ttl) attribute that helps remember the the next renew time. SKISSM implements 7 days as the default renewal time interval. On each begin time of SKISSM activation, the module will check this "ttl". If it is reached then the "publish signed pre-key" protocol will be requested to submit the public part of newly generated signed pre-key. E2EE server will update the key and use it subsequently to provide clients with PreKeyBundle for creating a new outbound session.

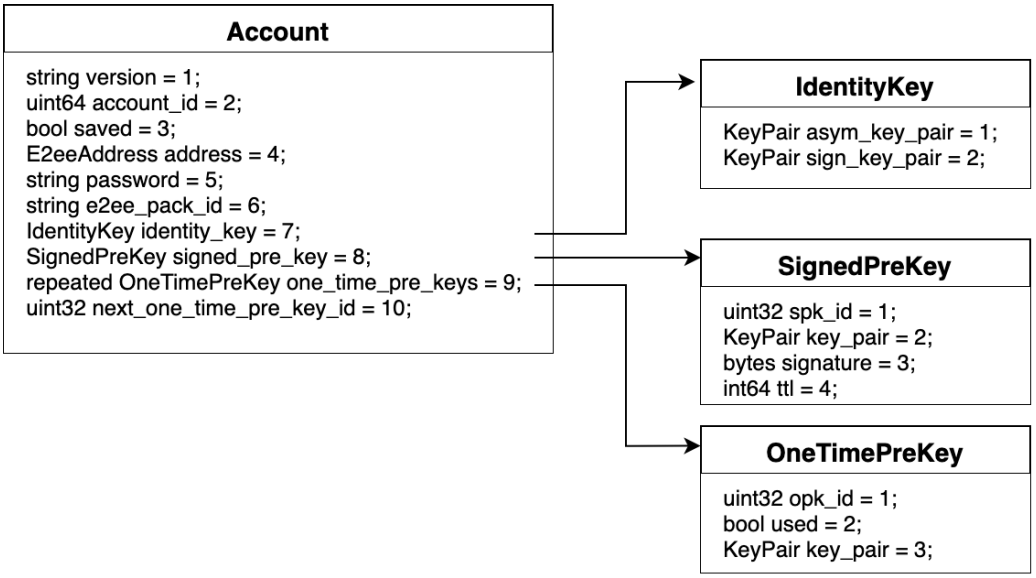

Fig. 4: Account struct

Pre-Key Bundle

Before a user application can build a outbound session, the “get pre-key bundle” protocol will be used to download a data set that encloses pre-key bundles [Fig.5]. To get the pre-key bundles of a peer user with given user_id, E2EE server will gather a list of pre-key bundles with which is related to each device_id of the same user_id. A PreKeyBundle just collects the public part of an identity key, a signed pre-key and an one-time pre-key. The E2EE server will remove the used one-time pre-keys after sending the collected pre-key bundles as a response.

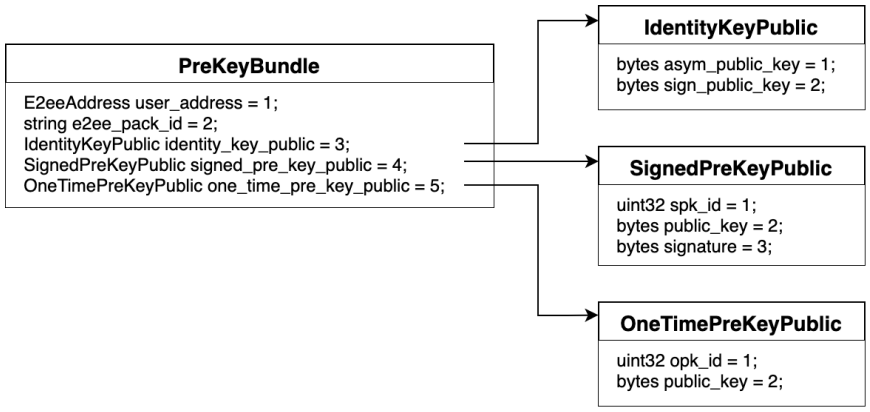

Fig. 5: PreKeyBundle struct

Session

A Session struct [Fig. 6] is used to encapsulate the states of one-to-one messaging that will be changed on each encryption or decryption. A session can be used for handling inbound messages or outbound messages alternatively by setting-up the “from” and “to” address attribute. An outbound session is used to send one-to-one encryption message to remote peer, while an inbound session is used to decrypt the incoming one-to-one message received from remote peer. Specially, An outbound session uses the attribute “responded” as a lock that will be enabled if AcceptMsg is received and complete the shared key calculation.

Each session has a “ratchet” attribute with Ratchet struct that maintains the ratchet states[3] for either inbound or outbound usage. If a ratchet is used for managing outbound session, then it will be operated with a sender chain that has ratchet_key to assign that an outbound message belongs to this chain, and a “ratchet” attribute to generate message key for encrypting outbound message. On the other hand, if a ratchet is used for managing inbound session, it will be operated with a receiver chain that has a “ratchet_key_public” attribute to identify the inbound message belongs to this chain, and a “chain_key” attribute to generate message key to decrypt inbound message. The skipped messages chain helps maintain the message key that is skipped while an inbound session is performing decryption task over receiver chain. Moreover, each chain has a max chan index 512 as default by SKISSM. If an outbound session with ratchet of sender chain reaches the limit, a new outbound session will be built as a replacement by using a new PreKeyBundle provided by server.

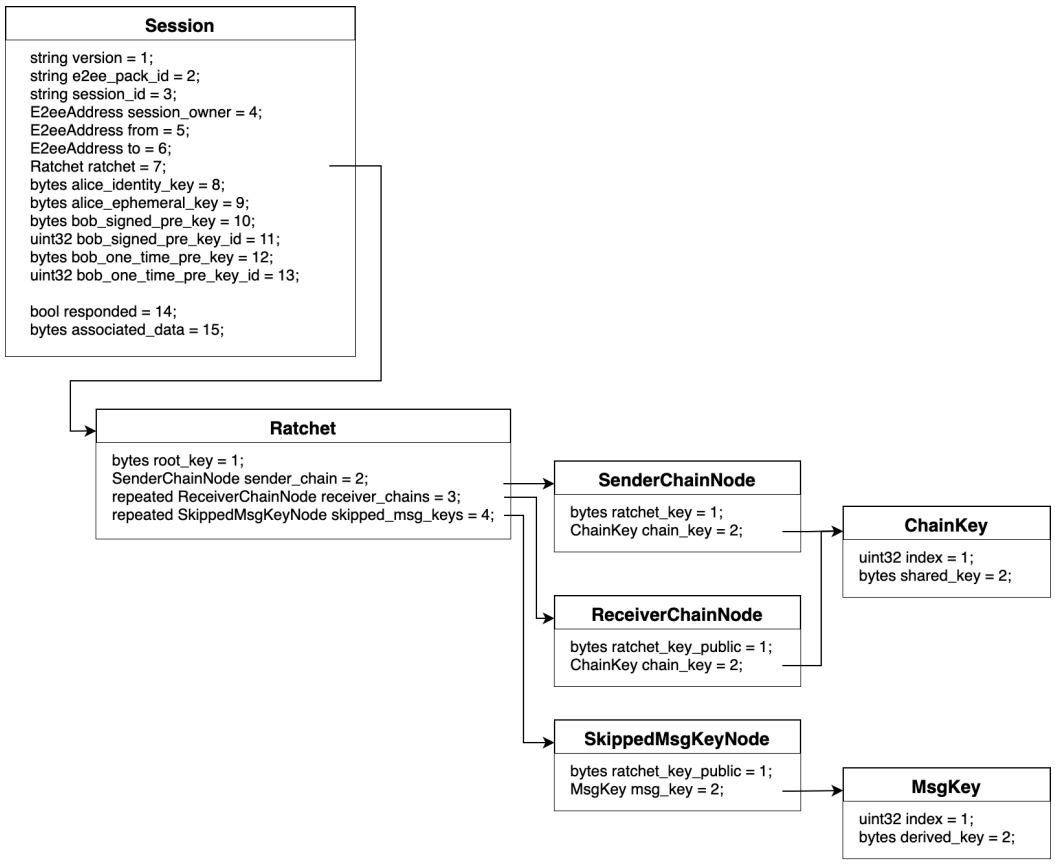

Fig. 6: Session Struct

Group Session

A GroupSession struct [Fig. 7] is used to encapsulate the states of group messaging that will be changed on each encryption or decryption. A group session can be used for handling inbound group messages or outbound group messages. In the case of outbound group session, the “signature_private_key” attribute will be created while an inbound group session only make use of “signature_public_key”.

An outbound group session is created after a success request of “create group” protocol and returning a unique group address. Then a GroupPreKeyBundle message will be packed as the payload of a Plaintext type message and delivered to each group member through one-to-one session introduced previously. E2EE server then help forwarding the one-to-one message to recipient. Each group member can build inbound group session after processing the decrypted plaintext with “group_pre_key” payload. If some one-to-one outbound is not ready for sending message, SKISSM will keep the data in database, and the saved “group_pre_key_plaintext” will be resent automatically after a respective AcceptMsg has been received and successfully create the outbound session.

The group members can be altered by requesting “add group members” and “remove group member” protocol. SKISSM will automatically rebuild the outbound group session if the group members were changed. The inbound group session of each group member will also be rebuilt as a result.

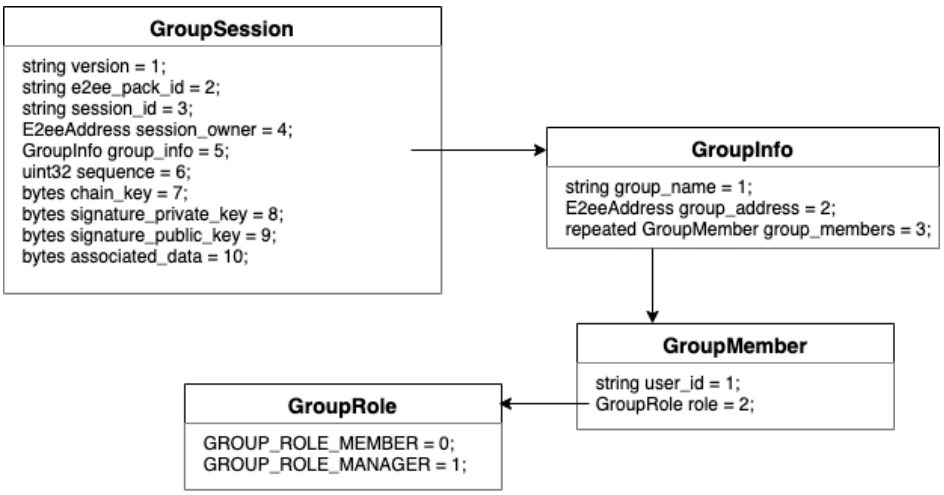

Fig. 7: GroupSession struct

Cryptographic Algorithms

Abbreviations

- **ck:** chain key
- **C**: ciphertext
- **Dec(x, y)**: decrypt message x with key y using AES256 with GCM mode
- **ss_key_gen(x, y)**: In the case of elliptic curve Diffie-Hellman key exchange with X25519 algorithm, the calculation return ECDH(x, y) where x is Alice's private key and y is Bob's public key. In the case of KEM based algorithm, the calculation return k←Decaps(x, y), where x is Alice's private key and y is came from Bob's side by calculating (y, k) ←$ Encaps(z) with Alice's public key z.
- **Enc(x, y)**: encrypt message x with key y using AES256 with GCM mode
- **ek, ek^{-1}**: ephemeral key pair
- **HKDF(IKM, salt, info)**: HKDF with SHA-256 with input key material IKM, salt, and info
- **HMAC(key, input)**: HMAC with SHA-256 with the key and the input
- **ik, ik^{-1}:** identity key pair
- **mk**: message key
- **opk, opk^{-1}**: one-time pre-key pair
- **P**: plaintext
- **rk, rk^{-1}**: ratchet key pair
- **RK**: root key
- **sig = Sign(x, y)**: sign message x with private key y and output the signature sig
- **spk, spk^{-1}**: signed pre-key pair
- **sk**: shared secret key
- **sk_priv**: signature private key
- **sk_pub**: signature public key
- **Verify(sig, k)**: verify the signature sig with the public key k

Algorithms

Invite and accept key agreement

Since the calculation of shared key as described in X3DH[2] is a kind of DH-based protocol with elliptic curve cryptography (ECC). The key agreement process can not complete at Alice's side alone in the case of applying post quantum cryptographic (PQC) primitives that mainly work with key encapsulation mechanisms (KEM)[14][15][16]. The flow for calculating the shared key for both Alice and Bob is altered by SKISSM [Fig. 8]. An invite message is sent on creating a new outbound session. The outbound session is not able to send encryption message before receiving an accept message and completing the calculation of shared key. SKISSM implements "invite" and "accept" protocols as a compromise to enable X3DH works in a uniform data flow for both pre quantum and post quantum cryptographic primitives.

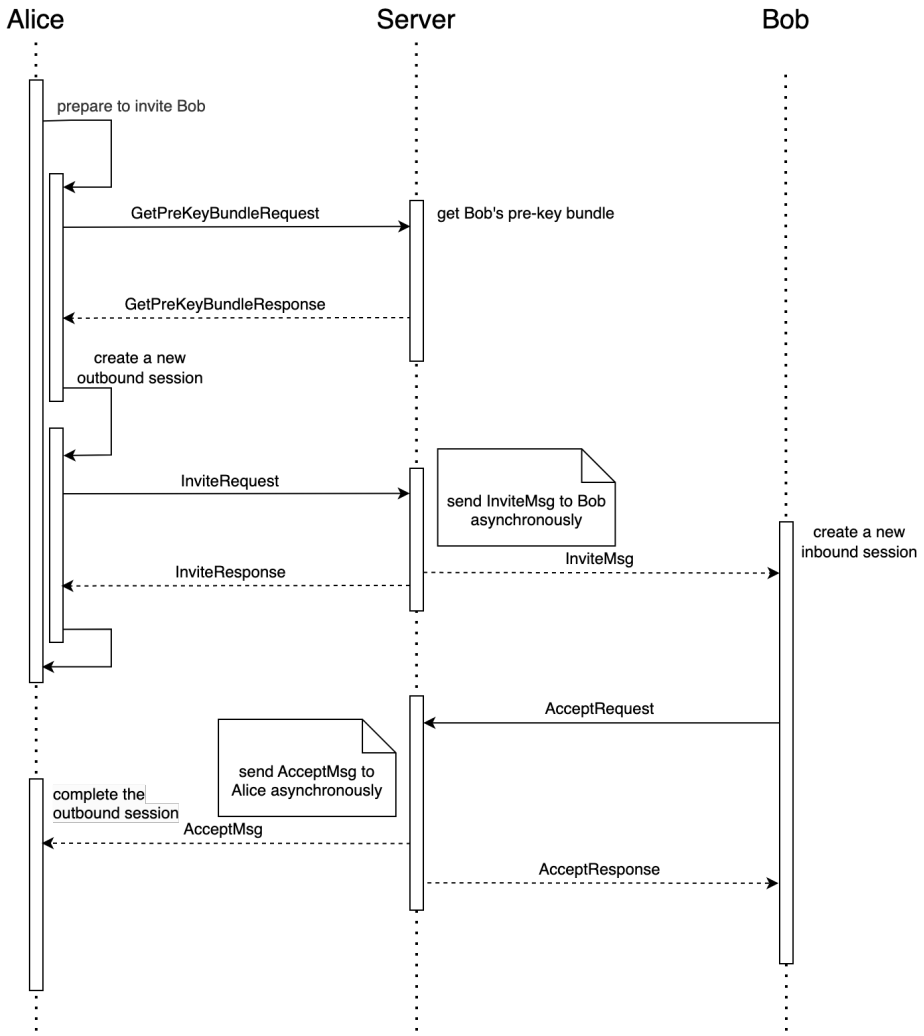

Fig. 8: Invite and accept protocol

Outbound session creation

To build a new outbound session, Alice first acquires Bob's pre-key bundle from server, then performs the following steps:

- Verify(Sig, ik_B)
- Generate ek_A (32 bytes key pair) and rk_A (32 bytes key pair) in the case of ECC.
- Start calculating the share secrets.

  k_2(32 bytes) = ss_key_gen(ek_A^{-1}, ik_B) in the case of ECC, or just calculate (c_2, k_2) ←$ Encaps(ik_B) in the case of PQC.

  k_3(32 bytes) = ss_key_gen(ek_A^{-1}, spk_B) in the case of ECC, or just calculate (c_3, k_3) ←$ Encaps(spk_B) in the case of PQC.

  k_4(32 bytes) = ss_key_gen(ek_A^{-1}, opk_B) in the case of ECC, or just calculate (c_4, k_4) ←$ Encaps(opk_B) in the case of PQC.
- Send InviteMsg with pre_share_keys: c_2, c_3, c_4.
- Complete calculating the share secret sk and complete the building of outbound session when AcceptMsg is received.

  k_1(32 bytes) = ss_key_gen(ik_A^{-1}, spk_B) in the case of ECC, or
  calculate Decaps(ik_A^{-1}, c_1) in the case of PQC where c_1 is obtained from the pre_share_keys of received AcceptMsg.

  secret(128 bytes) = k_1 ll k_2 ll k_3 ll k_4

  sk(64 bytes) = HKDF(secret, salt[32]={0}, info="ROOT")

To encrypt message by using established outbound session:

- Apply the Double Ratchet Algorithm[3]

  RK(32 bytes) = prefix 32 bytes of sk

  The first ratchet key is just the Bob's signed pre-key. That is, rk_B = spk_B

  secret_input(32 bytes) = ss_key_gen(rk_A^{-1}, rk_B) in the case of ECC, or calculate Encaps(rk_B) in the case of PQC.

  Next sk(64 bytes)

  = HKDF(secret_input, salt=RK, info="RATCHET")

  = next RK(32 bytes) ll sender_chain_key(32 bytes)

mk(48 bytes)

= HKDF(sender_chain_key, salt[32]={0}, info="MessageKeys")

C = Enc(P, mk)

- Send C and rk_A to Bob

Inbound session creation

When Bob received an InviteMsg from Alice, Bob can build a new inbound session by performing the following steps:

- Calculate share secret using X3DH

  k_1 = ss_key_gen(spk_B^{-1}, ik_A) in the case of ECC, or just calculate (c_1, k_1) ←$ Encaps(ik_A) in the case of PQC.

  k_2 = ss_key_gen(ik_B^{-1}, ek_A) in the case of ECC, or just calculate k_2 ← Decaps(ik_B^{-1}, c_2) in the case of PQC.

  k_3 = ss_key_gen(spk_B^{-1}, ek_A) in the case of ECC, or just calculate k_3 ← Decaps(spk_B^{-1}, c_3) in the case of PQC.

  k_4 = ss_key_gen(opk_B^{-1}, ek_A) in the case of ECC, or just calculate k_4 ← Decaps(opk_B^{-1}, c_4) in the case of PQC.

  secret(128 bytes) = k_1 ll k_2 ll k_3 ll k_4

  sk(64 bytes) = HKDF(secret, salt[32]={0}, info="ROOT")

- Send AcceptMsg with pre_shared_key. In the case of PQC, it will be c_1.

To decrypt message by using established inbound session:

- Apply the Double Ratchet Algorithm[3]

  RK(32 bytes) = prefix 32 bytes of sk

  secret_input(32 bytes) = ss_key_gen(rk_B^{-1}, rk_A)

  next sk(64 bytes)

  = HKDF(secret_input, salt=RK, info="RATCHET")

  = next RK(32 bytes) ll receiver_chain_key(32 bytes)

  mk(48 bytes)

  = HKDF(receiver_chain_key, salt[32]={0}, info="MessageKeys")

  P = Dec(C, mk)

Group session creation

Each group member creates an outbound group session for encrypting and sending group message. On the other hand, the other group members create inbound group session with respect to the outbound group session for decrypting received group message[Fig. 9].

- Group creator creates an outbound group session by generating random ck, and random (sk_priv, sk_pub)
- Group creator then send group pre-key (ck, sk_pub) to each group member by using one-to-one session. Each group member can build inbound group session by using group pre-key.

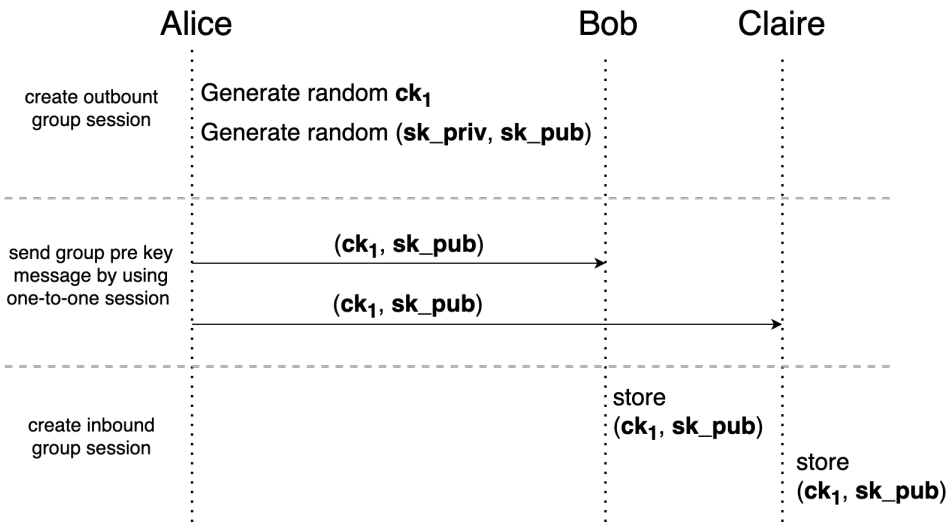

Fig. 9: Group session creation

To encrypt and send outbound group message, Alice uses the established outbound group session and performs the following steps[Fig. 10]:

- mk = HKDF(ck)
- C = Enc(P, mk)
- Sig = Sign(C, sk_priv)
- Send (C, Sig) to each group member

Alice uses the outbound group session to ratchet ck for the next encryption.

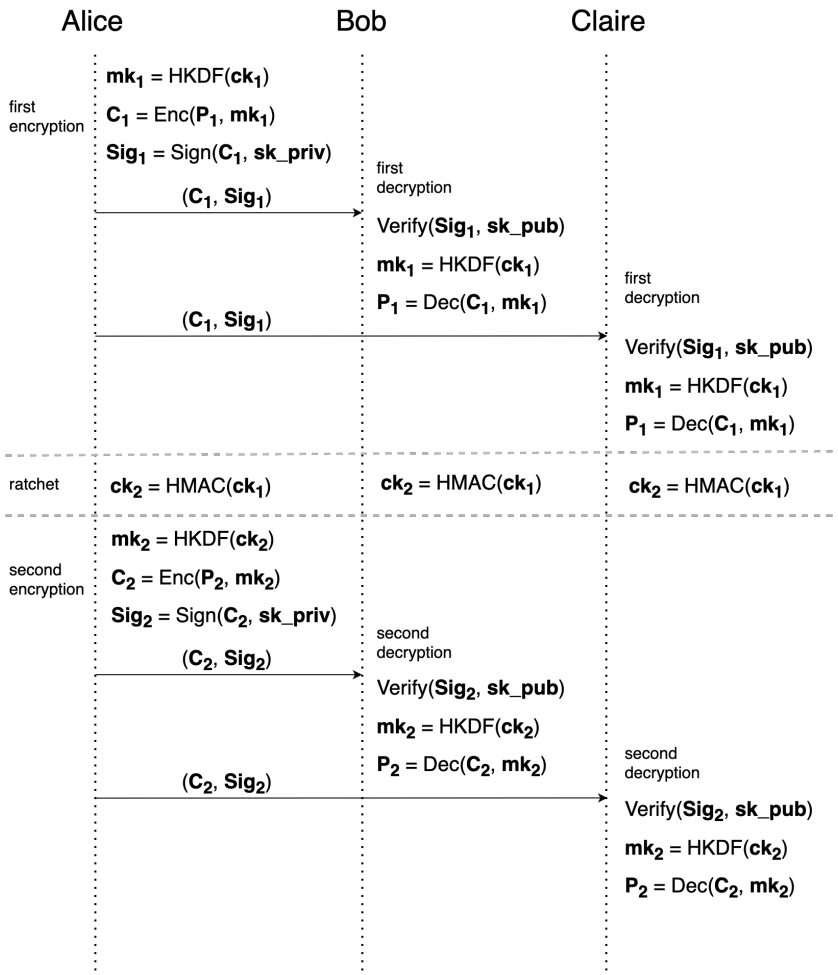

Fig. 10: Group message delivery

To decrypt a received inbound group message, Bob and other group members use the established inbound group session and perform the following steps[Fig. 10]:

- Verify(Sig, sk_pub)
- mk = HKDF(ck)
- P = Dec(C, mk)

Each group member uses their own inbound group session to ratchet ck for the next decryption.

Add or remove group members

When some group members are added or removed, the group member who makes the changed event first rebuild the outbound group session, then notify the affected group members to rebuild the corresponding inbound group session, and subsequently rebuild each group member's outbound group session. As a result, all outbound and inbound group sessions will be renewed, and the removed group members has no information about the updated group sessions.

E2EE Protocols

SKISSM provides a set of request-response protocols and supports the handling of server-sent messages[Fig. 11]. These request-response protocols give a direct control and message exchange mechanism to interact with E2EE server for better integration with user application. The server-sent messages on the other hand help SKISSM efficiently notified and keep the states of account and sessions updated with the E2EE messaging schemes.

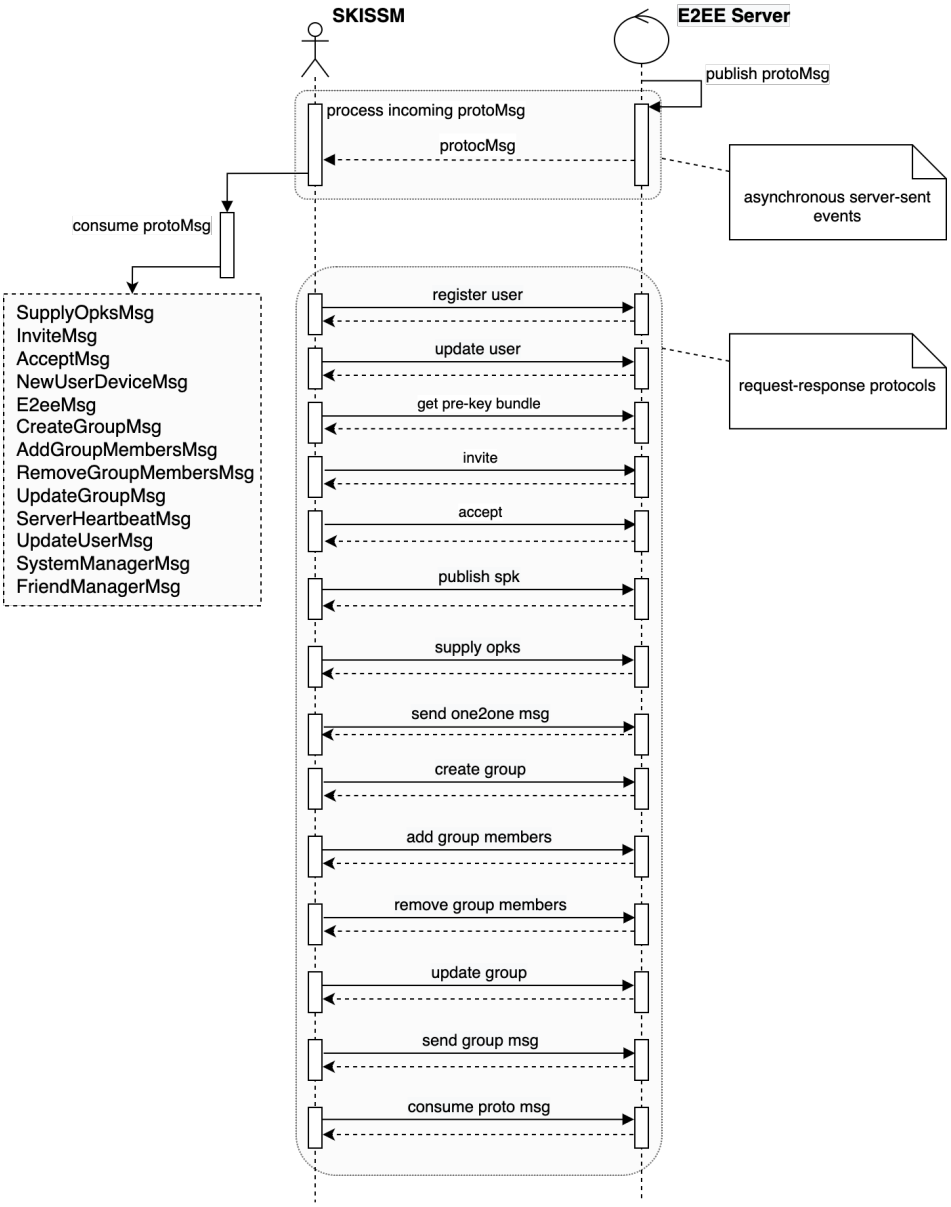

Fig. 11: E2EE protocols

Request-response protocols

Response code

A response code indicates the response state from server for requesting a resource from client[Fig. 12].

- RESPONSE_CODE_UNKNOWN

  The client get a requested response with unknown state.

- RESPONSE_CODE_OK

  The request succeeded, and some resources were read or updated.

- RESPONSE_CODE_CREATED

  The request succeeded, and some new resources were created as a result.

- RESPONSE_CODE_ACCEPTED

  The request has been received but not yet acted upon.

- RESPONSE_CODE_NO_CONTENT

  There is no content to send for this request, and some resources were deleted.

- RESPONSE_CODE_BAD_REQUEST

  The server cannot or will not process the request due to something that is perceived to be a client error.

- RESPONSE_CODE_UNAUTHORIZED

  The client is not authenticated to get the requested response.

- RESPONSE_CODE_FORBIDDEN

  The client is authenticated but does not have access rights to the content.

- RESPONSE_CODE_NOT_FOUND

  The server can not find the requested resource.

- RESPONSE_CODE_INTERNAL_SERVER_ERROR

  The server has encountered a situation it does not know how to handle.

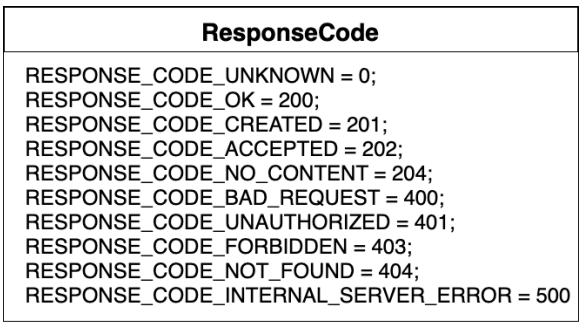

Fig. 12: Response code

Register User

The register user protocol[Fig. 13] helps create a new account in SKISSM by sending RegisterUserRequest data. A unique user address will be returned in a successful response from E2EE server.

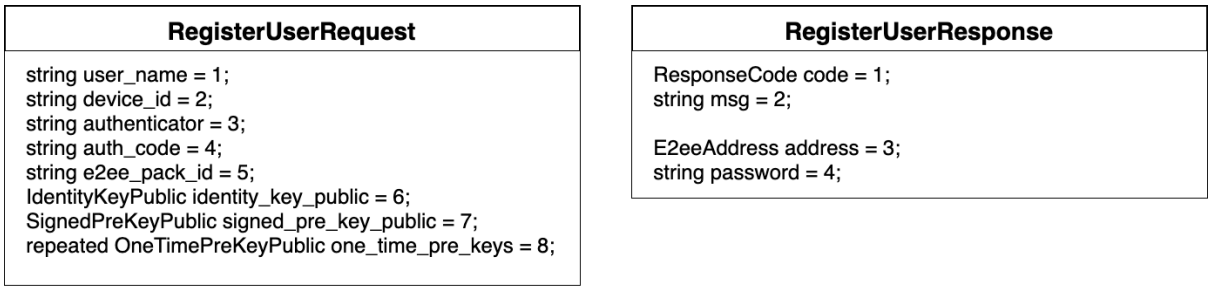

Fig. 13: Register user protocol

Update User

The update user protocol[Fig. 14] helps user update user's information by sending UpdateUserRequest data. E2EE server authenticate the user_id and publish a ProtoMsg to the server-sent messaging channel by packing the UpdateUserMsg data. The peer users who have applied "invite" protocol to this user_id will receive this message if this channel is subscribed.

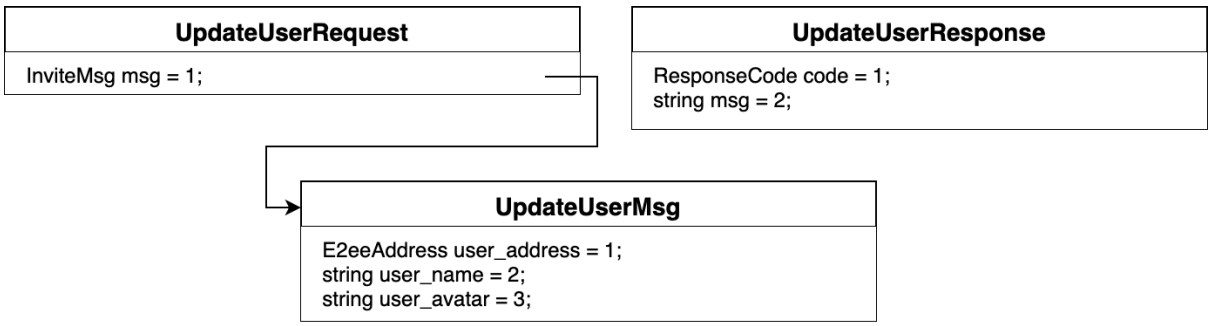

Fig. 14: Update user protocol

Get pre-key bundle

The get pre-key bundle protocol[Fig. 15] helps download PreKeyBundle for creating a new outbound session. By sending a GetPreKeyBundleRequest data with “user_adress”, E2EE server will return “pre_key_bundles” as an array of PreKeyBundle data. SKISSM will process them and create respective outbound session fro each PreKeyBundle data.

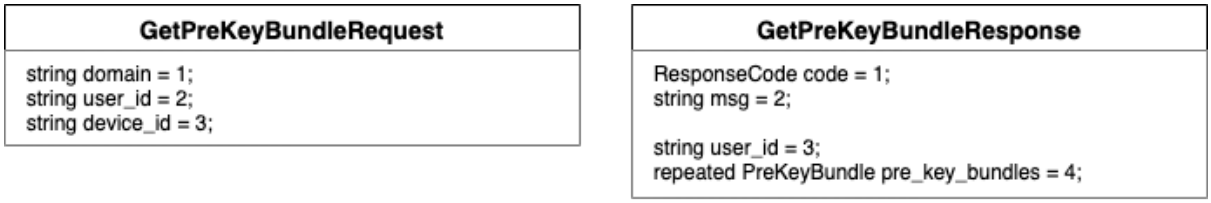

Fig. 15: Get pre-key bundle protocol

Invite

The invite protocol[Fig. 16] helps send InviteMsg to a peer user while build a new outbound session. E2EE server will publish a ProtoMsg to the server-sent messaging channel by packing the InviteMsg data. The peer user will receive this message if this channel is subscribed.

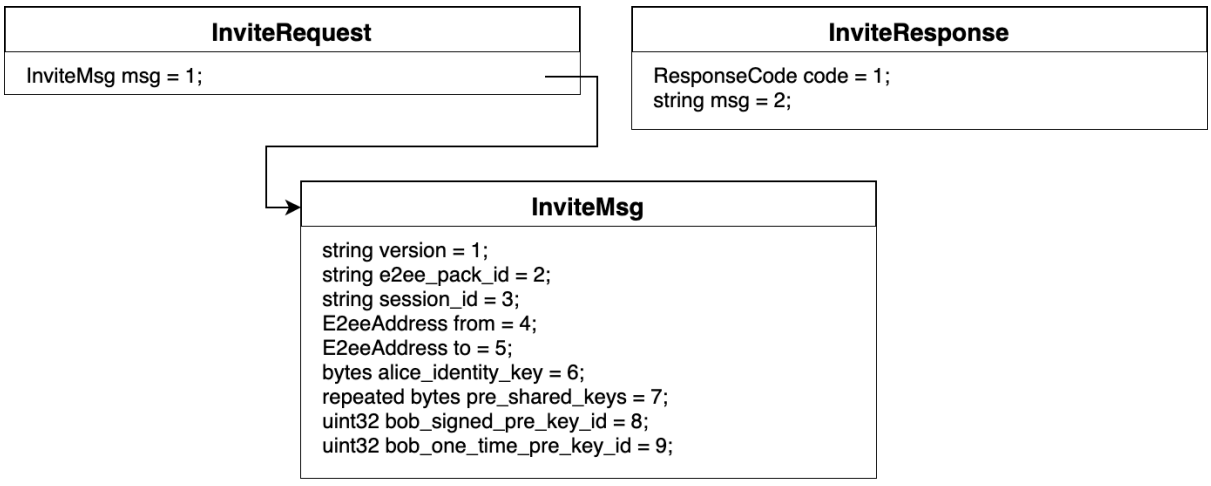

Fig. 16: Invite protocol

Accept

The accept protocol[Fig. 17] helps send AcceptMsg to a peer user after successfully builds a new inbound session. E2EE server will publish a ProtoMsg to the server-sent messaging channel by packing the AcceptMsg data. The peer user will receive this message if this channel is subscribed.

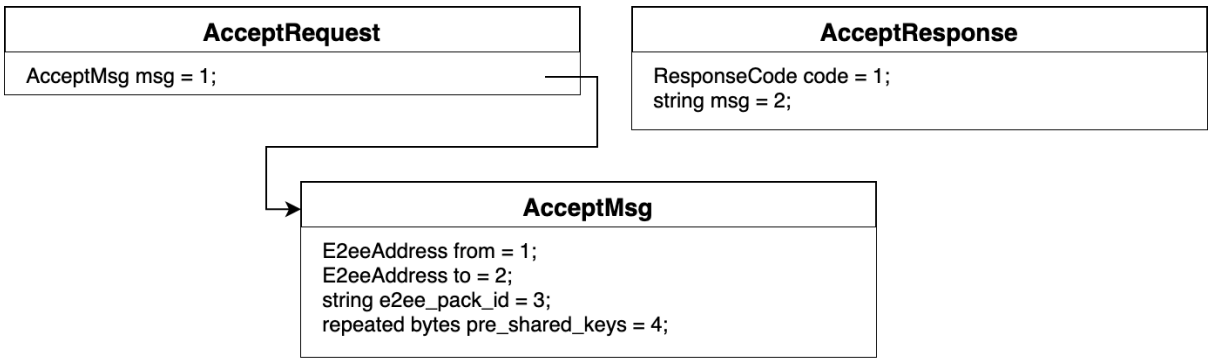

Fig. 17: Accept protocol

Publish signed pre-key

The publish signed pre-key protocol[Fig. 18] helps submit a new signed pre-key to server when the 7 days renew time is exceed that is managed by Account in SKISSM. E2EE server will keep and replace the old signed pre-key and use the new key to serve the request of “get pre-key bundle” protocol.

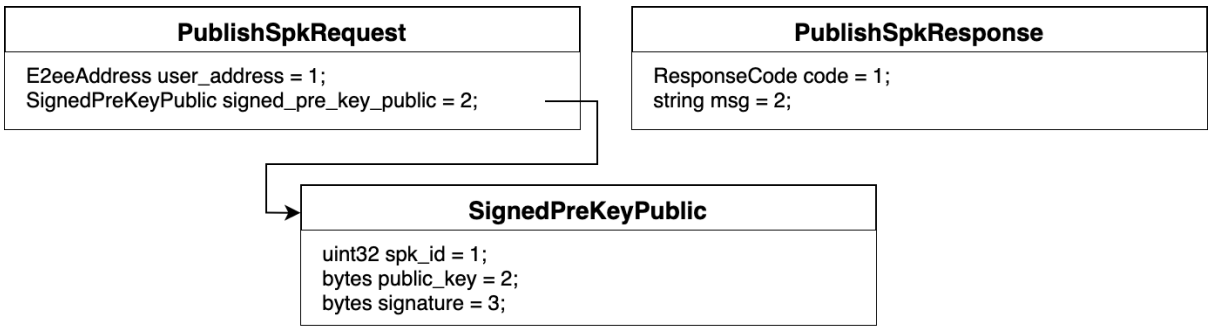

Fig. 18: Publish signed pre-key protocol

Supply one-time pre-key

The supply one-time pre-key protocol[Fig. 19] helps submit a set of one-time pre-key public parts to E2EE server. This is normally triggered by receiving a SupplyOpkMsg and notifying that server is running out of one-time pre-keys. SKISSM will create 100 new one-time pre-keys and apply “supply one-time pre-key” protocol to complete the job.

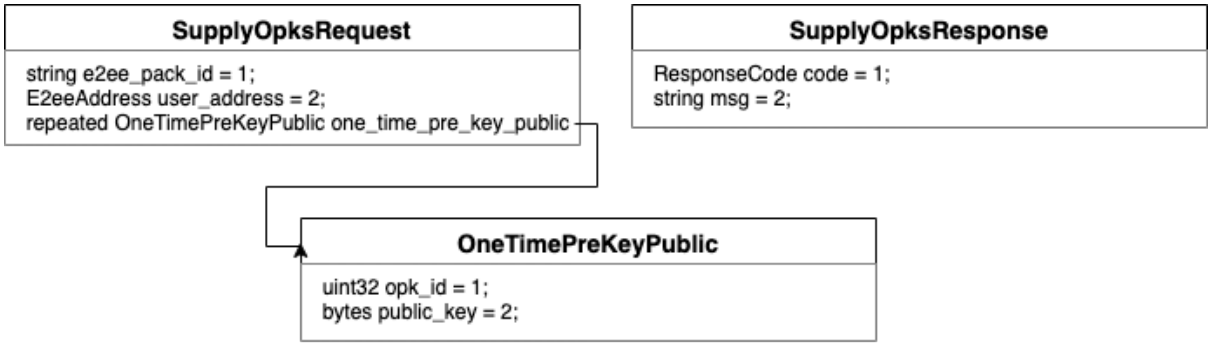

Fig. 19: Supply one-time pre-key protocol

Send one-to-one message

The send one-to-one message protocol[Fig. 20] helps send a E2eeMsg data that has “one2one_msg” as its payload to a remote peer user. E2EE server will publish a ProtoMsg to the server-sent messaging channel by packing the E2eeMsg data. The peer user will receive this message if this channel is subscribed.

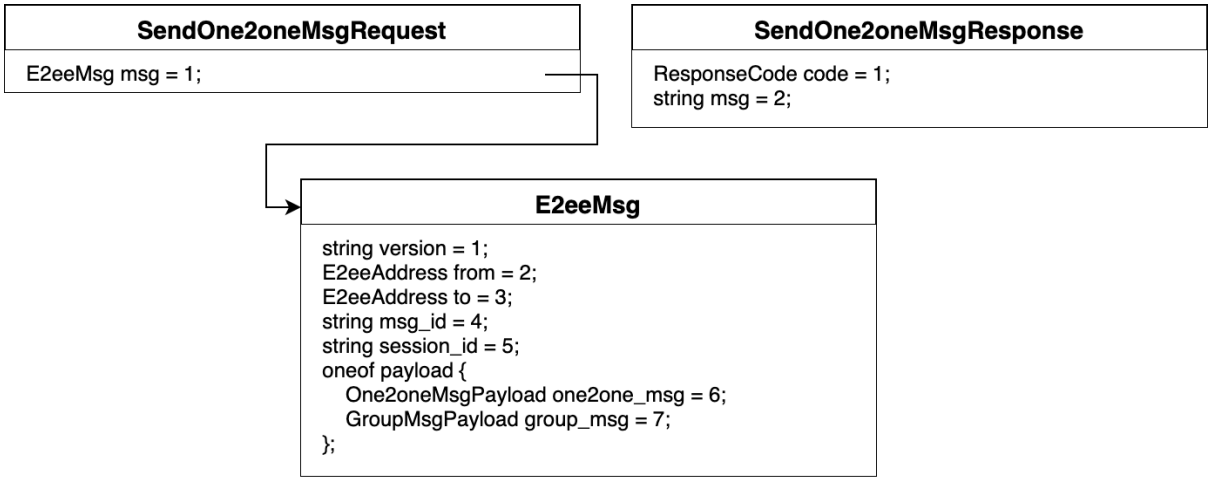

Fig. 20: Send one-to-one message protocol

Create group

The create group protocol[Fig. 21] helps send the CreateGroupMsg data while SKISSM is creating a new outbound group session. E2EE server will publish a ProtoMsg to the server-sent messaging channel by packing the CreateGroupMsg data. The peer user will receive this message if this channel is subscribed. SKISSM will send GroupPreKeyBundle data through one-to-one session to other group members after receiving a successful response. On the other hand, SKISSM will help each group member who receives CreateGroupMsg data by creating new outbound group session automatically.

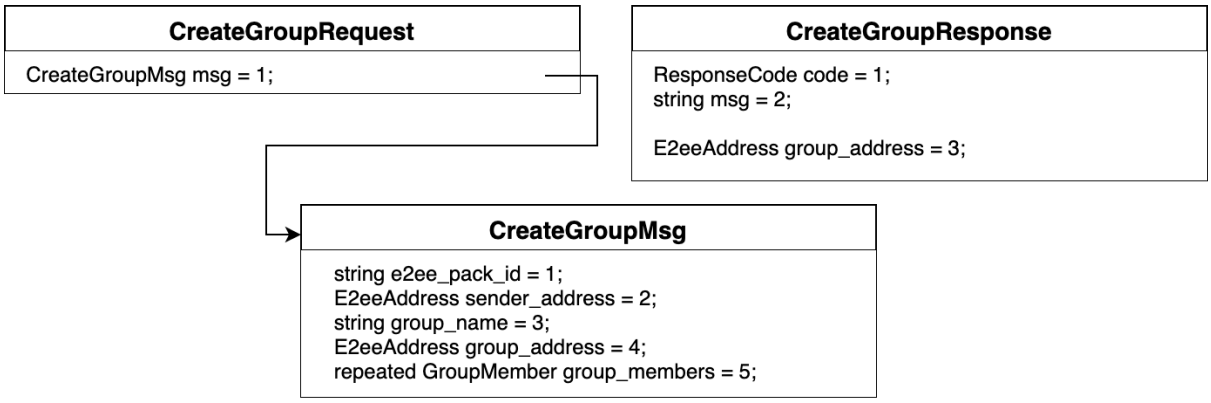

Fig. 21: Create group protocol

Add group members

The add group members protocol[Fig. 22] helps send the AddGroupMembersMsg data to other group members. If E2EE server verifies the user who send this request is a group member with manager role, a ProtoMsg will be published to the server-sent messaging channel by packing the AddGroupMembersMsg data. The peer user will receive this message if this channel is subscribed. A new outbound group session will be rebuilt after a successful response received. On the other hand, all other group members will also rebuild a new outbound group session on receiving the AddGroupMembersMsg data.

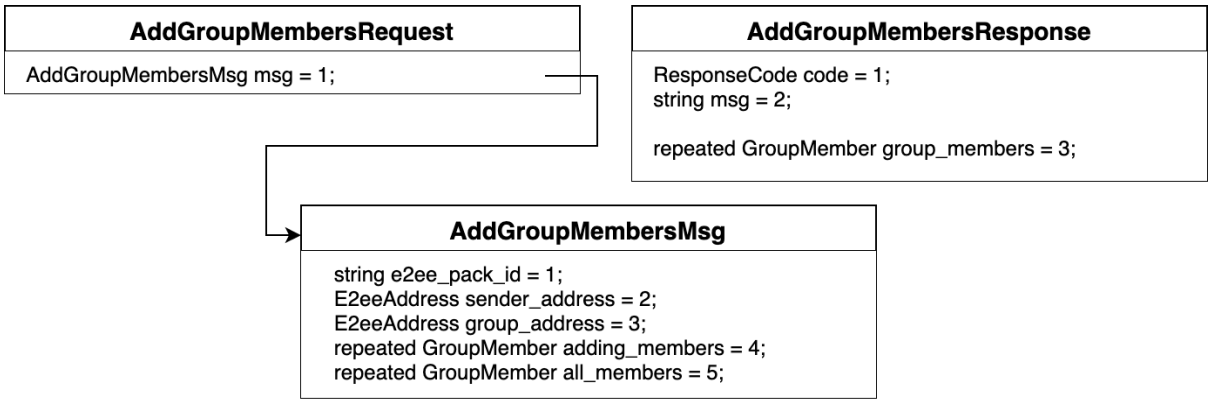

Fig. 22: Add group members protocol

Remove group members

The remove group members protocol[Fig. 23] helps send the RemoveGroupMembersMsg data to other group members. If E2EE server verifies the user who send this request is a group member with manager role, a ProtoMsg will be published to the server-sent messaging channel by packing the RemoveGroupMembersMsg data. The peer user will receive this message if this channel is subscribed. A new outbound group session will be rebuilt after a successful response received. On the other hand, all other group members will also rebuild a new outbound group session on receiving the RemoveGroupMembersMsg data.

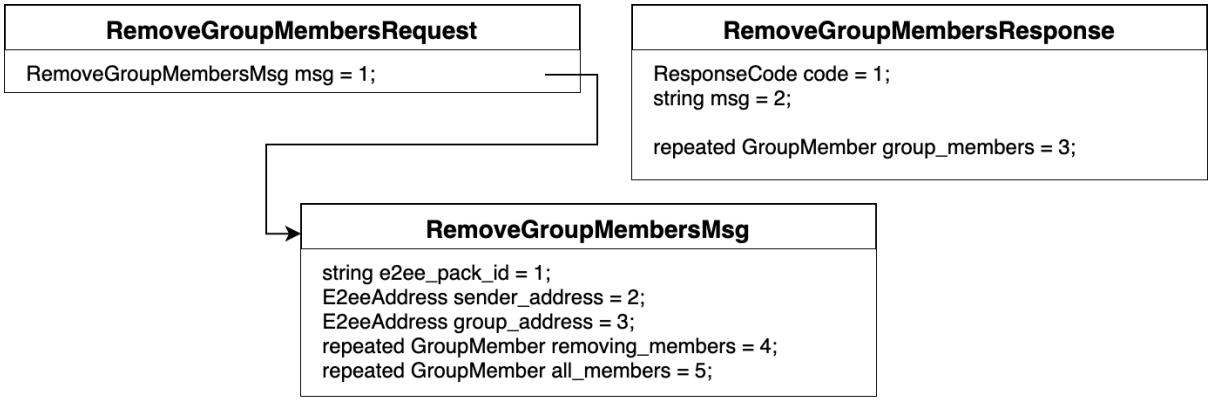

Fig. 23: Remove group members protocol

Update group

The update group protocol[Fig. 24] helps send the UpdateGroupMsg data to other group members. If E2EE server verifies the user who send this request is a group member with manager role, a ProtoMsg will be published to the server-sent messaging channel by packing the UpdateGroupMsg data. The peer user will receive this message if this channel is subscribed. The group sessions with the same group address should be updated after a successful response received. All other group members will also update the group session of the same group address on receiving the UpdateGroupMsg data.

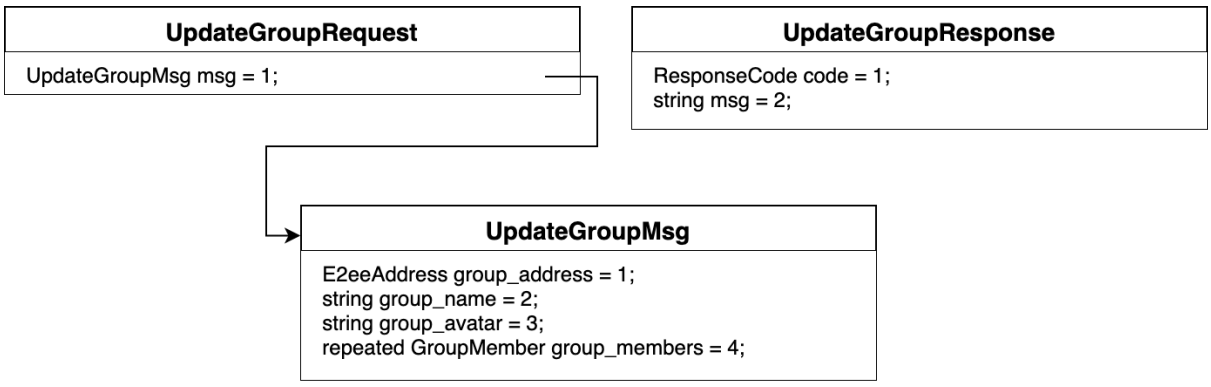

Fig. 24: Update group protocol

Send group message

The send group message protocol[Fig. 25] helps send a E2eeMsg data that has “group_msg” as its payload to a remote peer user. E2EE server will create a ProtoMsg by packing the E2eeMsg data and replicate it for each address of all other members. Then publish each E2eeMsg to the server-sent messaging channel. The peer user will receive this message if this channel is subscribed.

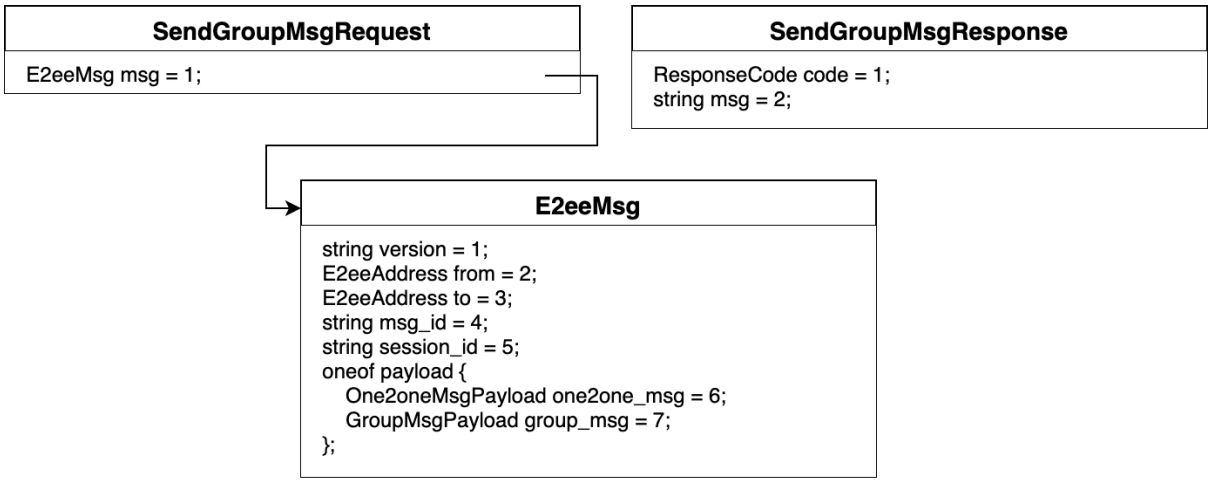

Fig. 25: Send group message protocol

Consume ProtoMsg

The consume ProtoMsg protocol[Fig. 26] helps notify E2EE server that a ProtoMsg with pro_msg_id has been successfully processed by SKISSM. E2EE server will remove the ProtoMsg that is stored in server database and return a successful response.

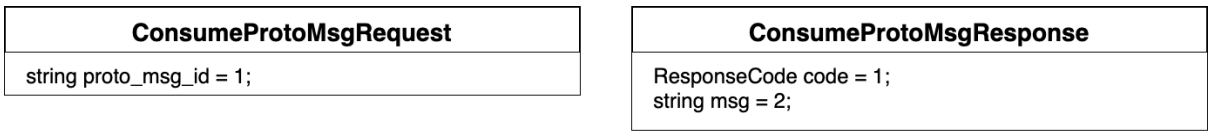

Fig. 26: Consume ProtoMsg protocol

Server-Sent messages

The message sent from E2EE server through the server-sent messaging channel is encapsulated in ProtoMsg struct[Fig. 27]. In addition to “from” and “to” address, a ProtoMsg is also tagged with a unique protocol message id and a time stamp by server. The payload of a ProtoMsg is specified by a variety of message types that help SKISSM to manage and update the respective session and account states.

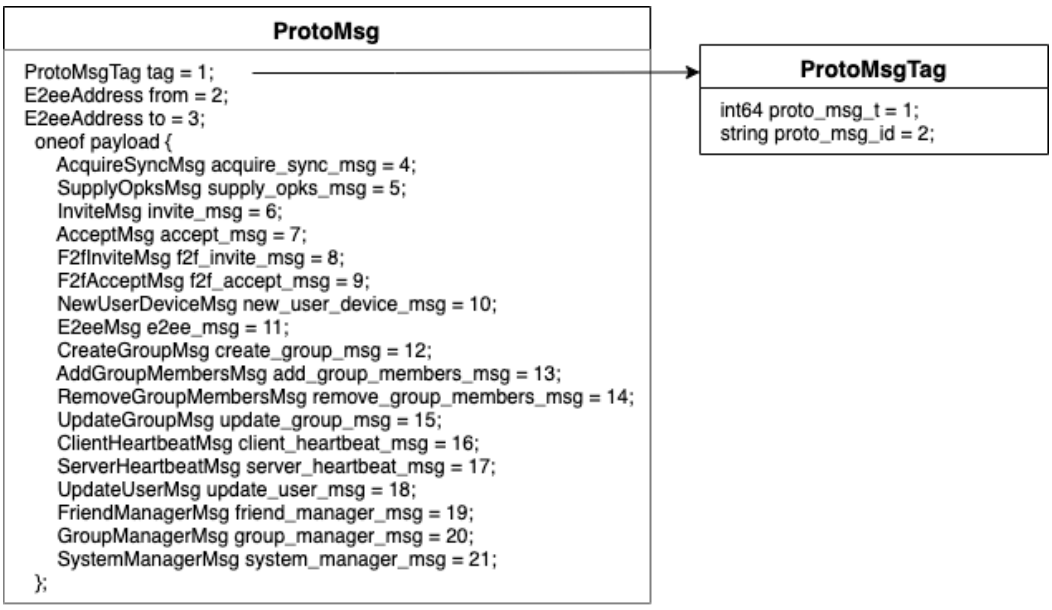

Fig. 27: Message struct of ProtoMsg

SupplyOpkMsg

A ProtoMsg with SupplyOpkMsg payload[Fig. 28] is sent from E2EE server when a user's one-time pre-keys are running out. SKISSM will apply "supply one-time pre-key" protocol on receiving this message, then apply "consume ProtoMsg" protocol to report a successful server-sent message consumption.

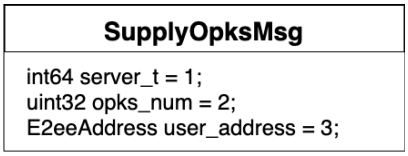

Fig. 28: SupplyOpkMsg

InviteMsg

A ProtoMsg with InviteMsg payload[Fig. 29] is received from server-sent channel when some user apply "invite" protocol to E2EE server. SKISSM will create an inbound session and apply "accept" protocol on receiving this message, then apply "consume ProtoMsg" protocol to report a successful server-sent message consumption.

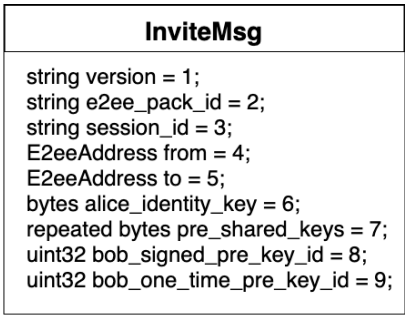

Fig. 29: InviteMsg

AcceptMsg

A ProtoMsg with AcceptMsg payload[Fig. 30] is received from server-sent channel when some user apply “accept” protocol to E2EE server. SKISSM will completing the creation of an outbound session on receiving this message, then apply “consume ProtoMsg” protocol to report a successful server-sent message consumption.

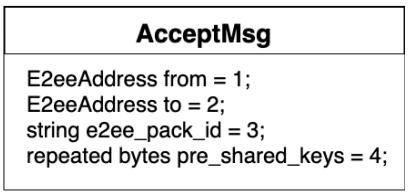

Fig. 30: AcceptMsg

NewUserDeviceMsg

A ProtoMsg with NewUserDeviceMsg payload[Fig. 31] is received from server-sent channel when some user apply “register user” protocol with new “devide_id” for a registered user_id to E2EE server. After a successful authentication, E2EE server will replicate and send this type of message to all the addresses that have been apply “invite” protocol to the same user_id. SKISSM will create a new outbound session by applying “invite” protocol on receiving this message, then apply “consume ProtoMsg” protocol to report a successful server-sent message consumption.

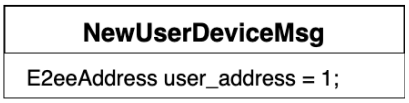

Fig. 31: NewUserDeviceMsg

E2eeMsg

A ProtoMsg with E2eeMsg payload[Fig. 32] is received from server-sent channel when some user apply “send one2one message” or “send group message” protocol to E2EE server. SKISSM establish E2eeMsg as the main message struct to carry out the transmission of E2EE messages. An E2eeMsg data has a “session_id” attribute that is related to the working session. The session can be type of an outbound session, inbound session, outbound group session, or inbound group session. In addition to the “to” and “from” attributes that specify the source and destination address, a payload attribute can be chosen from two types:

- One2oneMsgPayload

  In the case of one2one_msg payload, the message contains “sequence”, “ratchet_key”, and “ciphertext” attributes that are related to an outbound session or inbound session.

- GroupMsgPayload

  In the case of group_msg payload, the message contains “sequence”, “signature”, and “ciphertext” attributes that are related to an outbound group session or inbound group session.

The ciphertext is managed with Double Ratchet Algorithm[3]. SKISSM implement a Plaintext message to mediate the transmission of common message data from user application and group pre-key data from SKISSM. A user application should use Plaintext with “common_msg” payload. SKISSM will create an inbound group session on receiving a E2eeMsg with GroupPreKeyBundle data as its payload. In this case of E2eeMsg with payload in GroupMsgPayload type should only carry a ciphertext that is encrypted from a Plaintext message with “common_msg” payload.

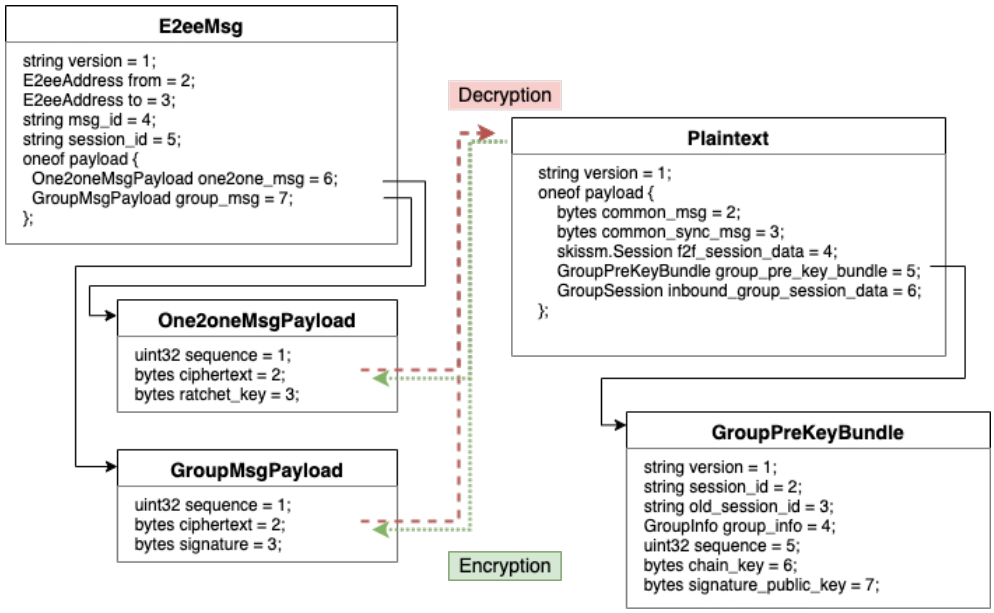

Fig. 32: E2eeMsg struct

CreateGroupMsg

A ProtoMsg with CreateGroupMsg payload[Fig. 33] is received from server-sent channel when some user apply “create group” protocol to E2EE server. SKISSM will create a new outbound group session on receiving this message, then apply “consume ProtoMsg” protocol to report a successful server-sent message consumption.

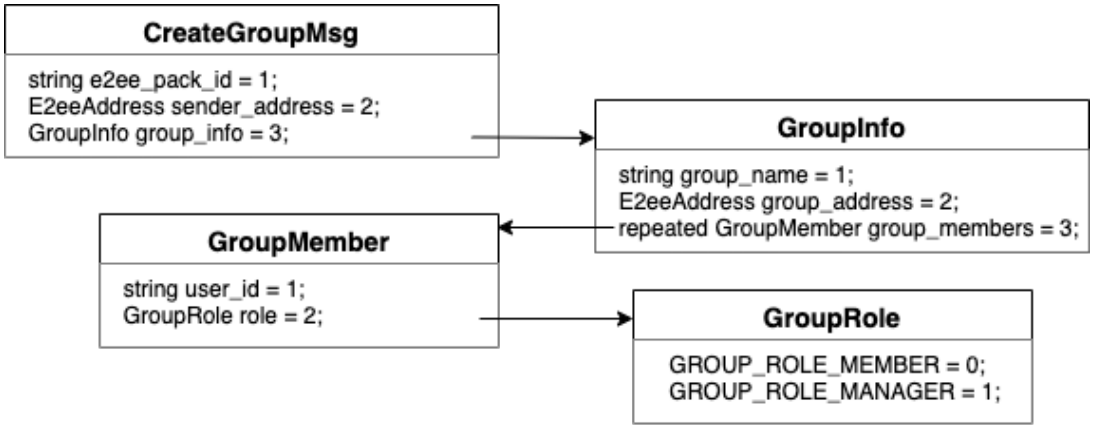

Fig. 33: CreateGroupMsg

AddGroupMembersMsg

A ProtoMsg with AddGroupMembers payload[Fig. 34] is received from server-sent channel some group member with manager role apply “add group members” protocol to E2EE server. SKISSM will create a new outbound group session on receiving this message, then apply “consume ProtoMsg” protocol to report a successful server-sent message consumption.

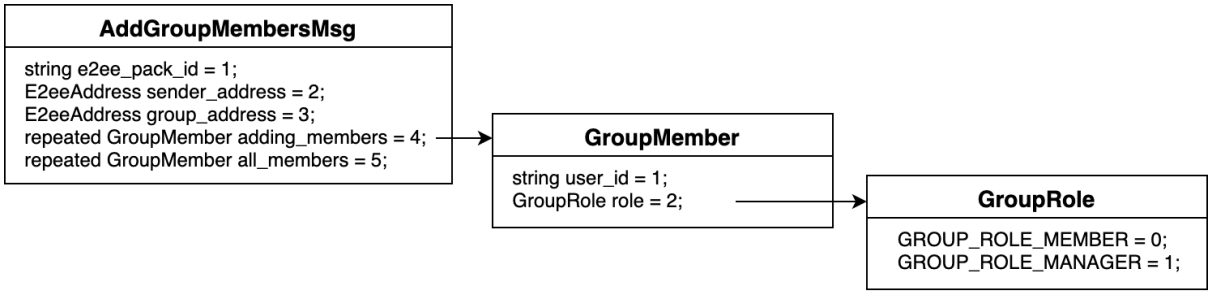

Fig. 34: AddGroupMembersMsg

RemoveGroupMembersMsg

A ProtoMsg with RemoveGroupMembers payload[Fig. 35] is received from server-sent channel when some group member with manager role apply “remove group members” protocol to E2EE server. SKISSM will create a new outbound group session on receiving this message, then apply “consume ProtoMsg” protocol to report a successful server-sent message consumption.

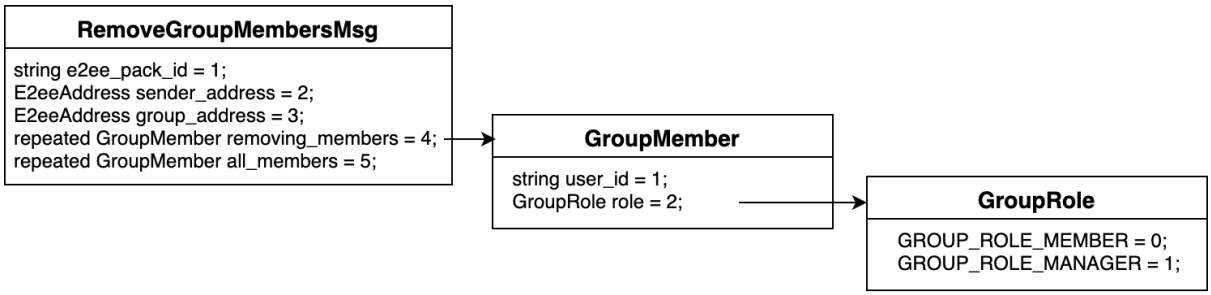

Fig. 35: RemoveGroupMembersMsg

UpdateGroupMsg

A ProtoMsg with UpdateGroupMsg payload[Fig. 36] is received from server-sent channel when some group member with manager role apply “update group” protocol to E2EE server. SKISSM will update the group session with the same group address on receiving this message, then apply “consume ProtoMsg” protocol to report a successful server-sent message consumption.

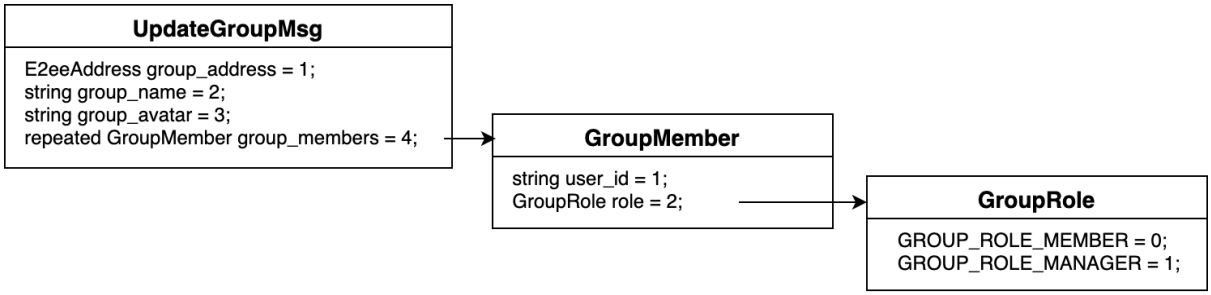

Fig. 36: UpdateGroupMsg

ServerHeartbeatMsg

A ProtoMsg with ServerHertbeatMsg payload[Fig. 37] is received from server-sent channel in a time interval periodically. User application can keep noticed about service availability. It is not needed to report the consumption state of this message type.

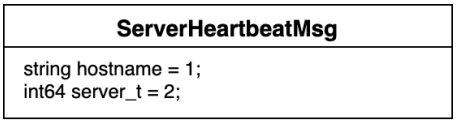

Fig. 37: ServerHeartbeatMsg

UpdateUserMsg

A ProtoMsg with UpdateUserMsg payload[Fig. 38] is received from server-sent channel when some user apply “update user” protocol to E2EE server. All the users that have apply “invite” protocol to this user will receive this message. User application just update the user information on receiving this message, then apply “consume ProtoMsg” protocol to report a successful server-sent message consumption.

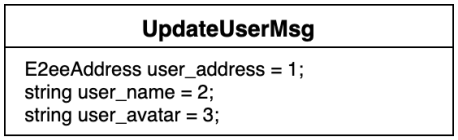

Fig. 38: UpdateUserMsg

SystemManagerMsg

A ProtoMsg with SystemManagerMsg payload[Fig. 39] is sent from E2EE server when there is a system notification to be delivered. User application will get notified on receiving this message, then apply “consume ProtoMsg” protocol to report a successful server-sent message consumption.

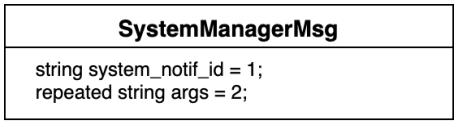

Fig. 39: SystemManagerMsg

FriendManagerMsg

A ProtoMsg with FriendManagerMsg payload[Fig. 40] is sent from E2EE server when there is a friend operation to be delivered. User application will get notified on receiving this message, then apply “consume ProtoMsg” protocol to report a successful server-sent message consumption.

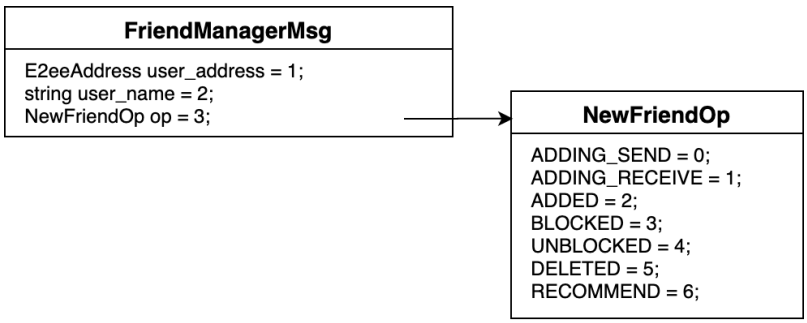

Fig. 40: FriendManagerMsg

References

[1] Trevor Perrin (editor) "The XEdDSA and VXEdDSA Signature Schemes", Revision 1, 2016-10-20. https://signal.org/docs/specifications/xeddsa/

[2] Moxie Marlinspike, Trevor Perrin (editor) "The X3DH Key Agreement Protocol", Revision 1, 2016-11-04. https://signal.org/docs/specifications/x3dh/

[3] Moxie Marlinspike, Trevor Perrin (editor) "The Double Ratchet Algorithm", Revision 1, 2016-11-20. https://signal.org/docs/specifications/doubleratchet/

[4] Moxie Marlinspike, Trevor Perrin (editor) "The Sesame Algorithm: Session Management for Asynchronous Message Encryption", Revision 2, 2017-04-14. https://signal.org/docs/specifications/sesame/

[5] Proto3 Language Guide, https://developers.google.com/protocol-buffers/docs/proto3

[6] A. Langley, M. Hamburg, and S. Turner, "Elliptic Curves for Security.", Internet Engineering Task Force; RFC 7748 (Informational); IETF, Jan-2016. http://www.ietf.org/rfc/rfc7748.txt

[7] S. Josefsson and I. Liusvaara "Edwards-Curve Digital Signature Algorithm (Ed- DSA)", Internet Engineering Task Force; RFC 8032 (Informational); IETF, Jan- 2017. https://tools.ietf.org/html/rfc8032

[8] J. Salowey, A. Choudhury, and D. McGrew, "AES Galois Counter Mode (GCM) Cipher Suites for TLS", Internet Engineering Task Force; RFC 5288 (Standards Track); IETF, August 2008. https://www.ietf.org/rfc/rfc5288.txt

[9] H. Krawczyk and P. Eronen "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", Internet Engineering Task Force; RFC 5869 (Informational); IETF, May-2010. https://tools.ietf.org/html/rfc5869

[10] A collection of implementations of curve25519, an elliptic curve Diffie Hellman primitive "curve25519-donna", https://github.com/agl/curve25519-donna/tree/master

[11] ARM mbed "mbed TLS", https://tls.mbed.org

[12] Kyber, https://pq-crystals.org/kyber/index.shtml

[13] SPHINCS+, https://sphincs.org

[14] Jacqueline Brendel , Marc Fischlin , Felix Günther , Christian Janson , Douglas Stebila Authors Info & Claims “Towards Post-Quantum Security for Signal's X3DH Handshake”, Selected Areas in Cryptography: 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected PapersOct 2020 Pages 404–430.

[15] Jacqueline Brendel and Rune Fiedler and Felix Günther and Christian Janson and Douglas Stebila, "Post-quantum Asynchronous Deniable Key Exchange and the Signal Handshake", IACR-PKC, 2022.

[16] Keitaro Hashimoto and Shuichi Katsumata and Kris Kwiatkowski and Thomas Prest, "An Efficient and Generic Construction for Signal's Handshake (X3DH): Post-Quantum, State Leakage Secure, and Deniable", IACR-JOC, 2022.

[17] SKISSM opensource project, https://github.com/ziv-e2eelab-org/skissm