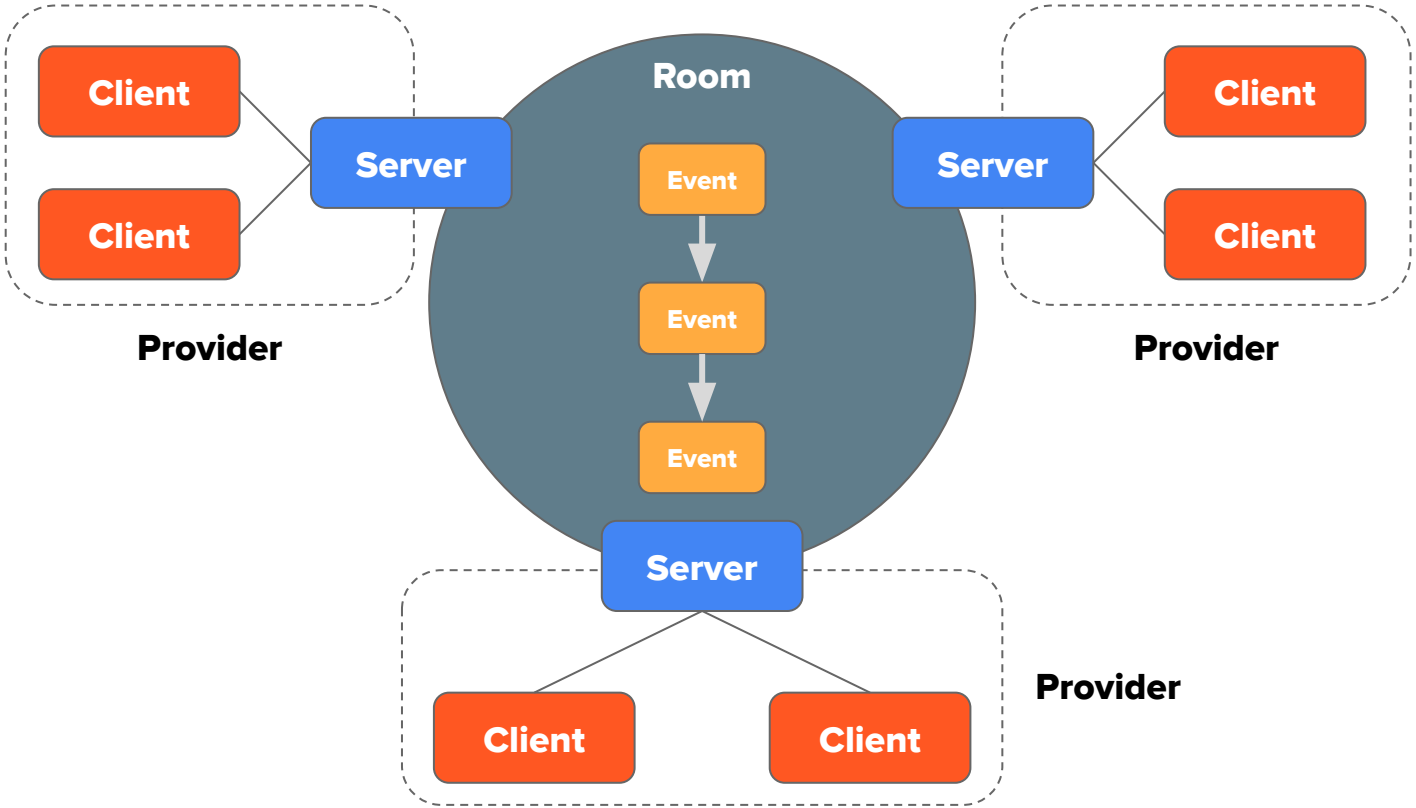
draft-ietf-mimi-arch

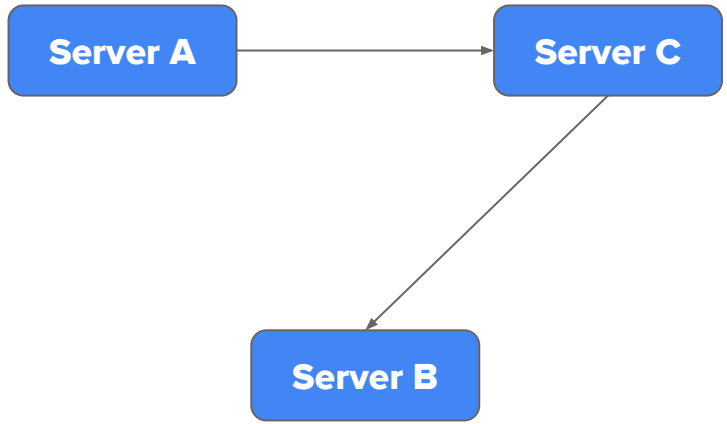
Richard Barnes
IETF 121, November 2024

CSSSC


Client
Client
Server
Provider
Room
Event
Event
Event
Server
Client
Client
Provider
Server
Client
Client
Provider

CSSSC

- Each room is hosted at a hub server
- All communications go via the hub
  - Messages
  - State changes
  - Ancillary things like KeyPackage fetches
- Anticipate that not every server will be willing to talk to every other server
  - A and B might not be willing to talk directly
  - But if users from A and B are in a room hosted at C...
  - A needs to be able to talk to B via C



Terminology

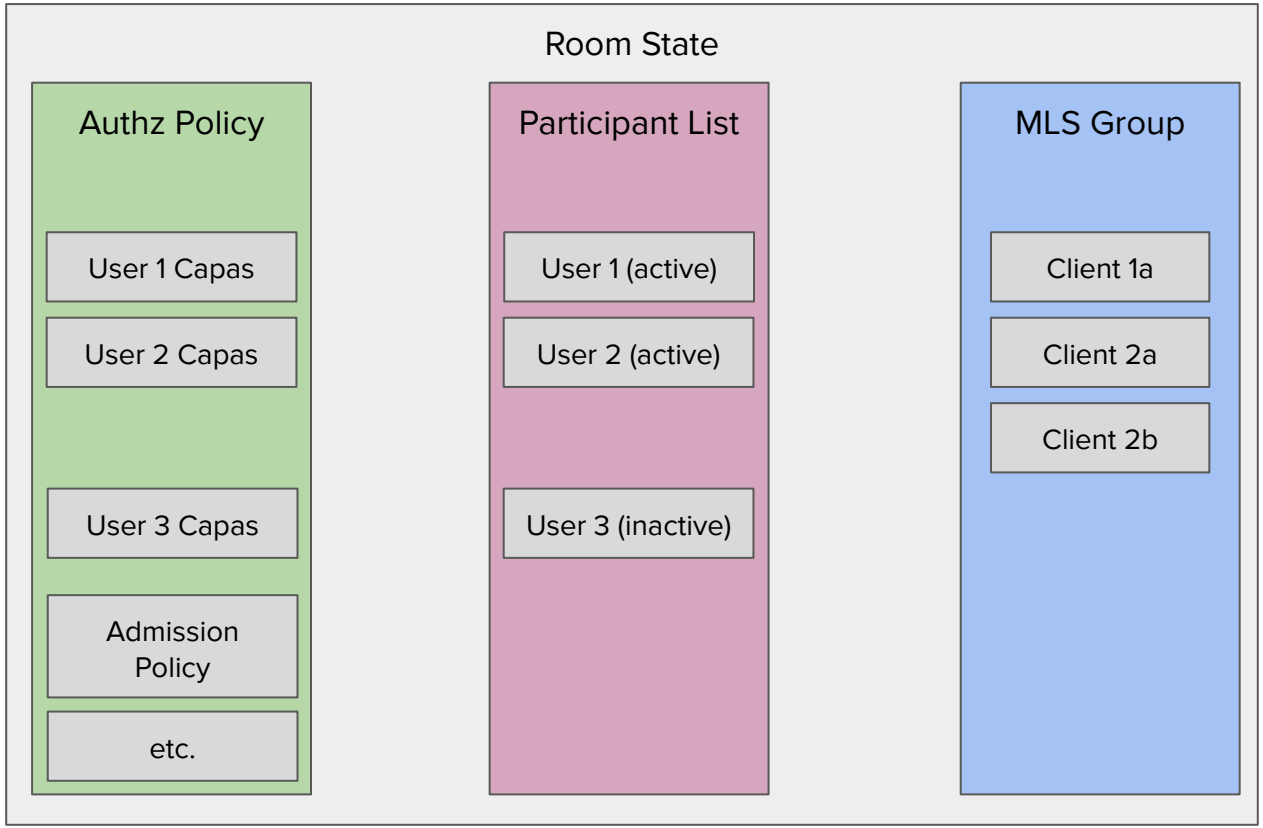
The unit of MIMI functionality is a **room**

A room has a **state**, which has a few components:

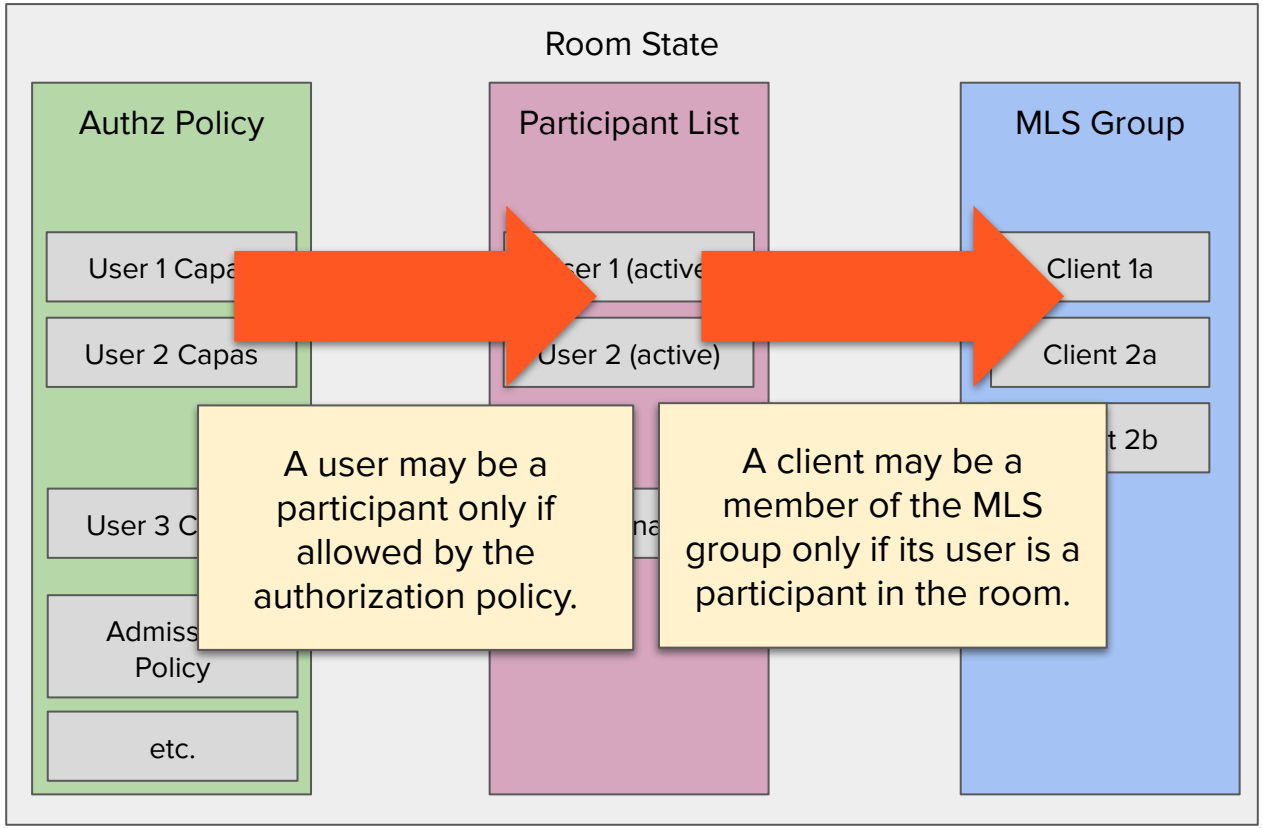
- An **authorization policy**
- A list of **users** who are **participants** in the room
- An MLS group, including a list of **clients** who are **members** of the group

Participants can either be **active** or **inactive**. Active participants have at least one client that is a member of the MLS group.

Terminology Illustrated



Preemption


Room State
Authz Policy
Participant List
MLS Group
User 1 Capa
User 2 Capas
User 3 C
Admiss
Policy
etc.
er 1 (active
User 2 (active)
Client 1a
Client 2a
t 2b
A user may be a participant only if allowed by the authorization policy.
A client may be a member of the MLS group only if its user is a participant in the room.

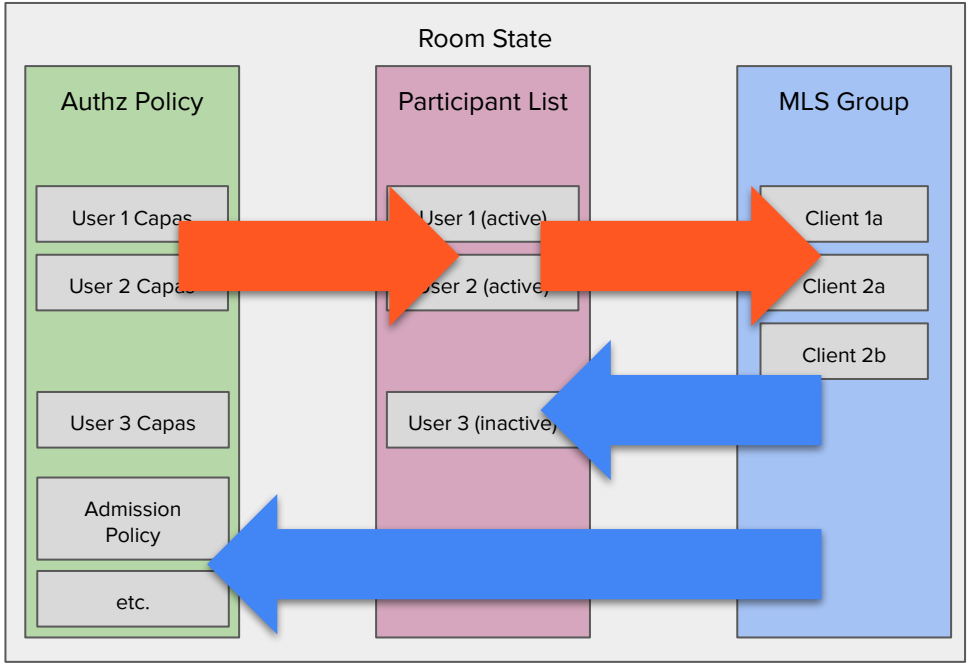
Confirmation

Different aspects of the state are managed via separate control mechanisms

To ensure everyone agrees on state, it is included in the MLS key schedule

- … as a GroupContext extension
- If clients don't agree, they can't communicate

Each Commit must reflect the current room state.

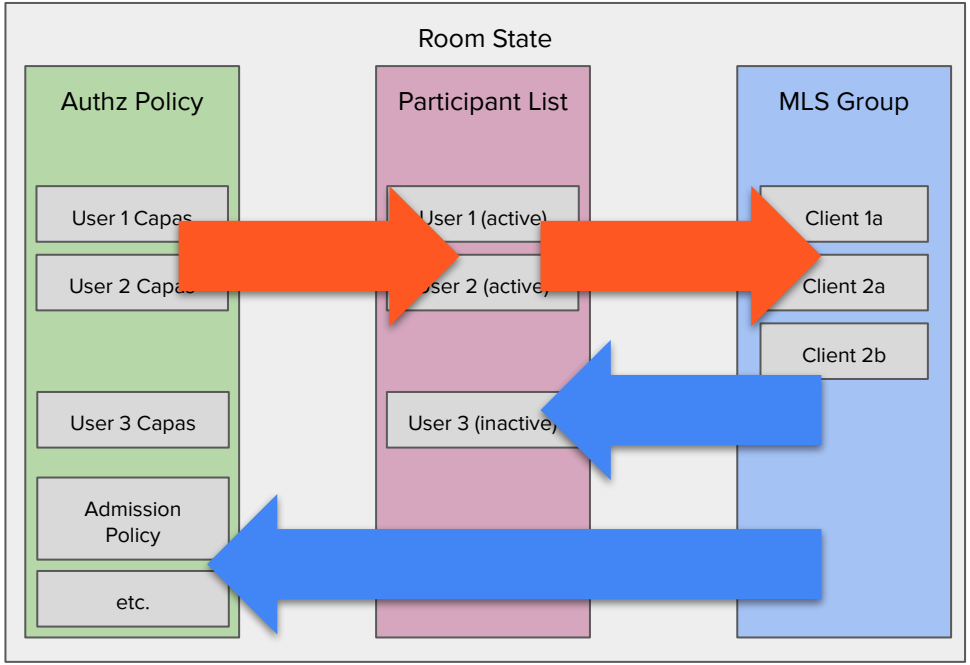


Order of Operations

Preemption + Confirmation require some coupling between control mechanisms

- User must be on PList before clients added to MLS
- Clients must be removed from MLS before user leaves PList

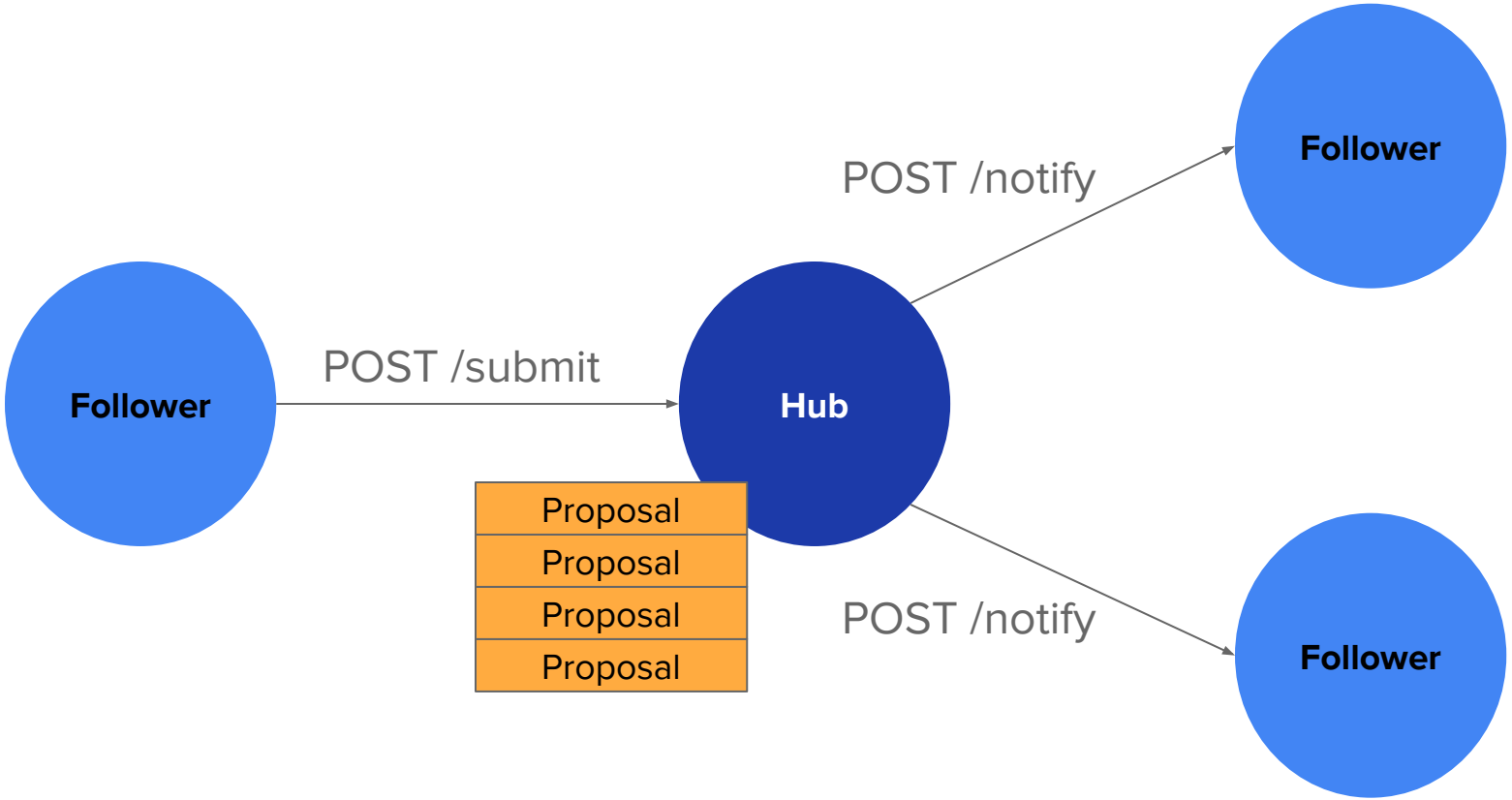
Sending control messages together will help keep things organized.



draft-ietf-mimi-protocol

IETF 121, November 2024

Overview


Follower
POST /submit
Hub
Proposal
Proposal
Proposal
Proposal
POST /notify
Follower
POST /notify
Follower

Server-to-Server Communications

```
POST /notify HTTP/1.1
Host: hub.example
From: mimi@follower.example
Content-Type: application/mimi
```

HTTPS over mutually-authenticated TLS

RFC 6125 framework of **presented** vs. **reference** identifiers

TLS certificates contain a set of presented identifiers for each endpoint

Server reference identity in Host / :authority

Client reference identity in From

MLS and AppSync

MIMI uses MLS for three things:

1. Protecting application messages
2. Managing which devices have access to the group
3. Ensuring agreement on room state

(1) and (2) are features of base MLS

(3) uses the AppSync extension (draft-barnes-mls-appsync) – basically a key/value store of application data that the devices in a room all agree on

Two types of sync'ed state: Policy + Participant List

Policy and Participants

Policy Framework:

- Granular capabilities
- Hub-defined roles = sets of capabilities

Participant list associates user identifier to role

> Here, we assume that Alice uses ClientA1 to create a room with the following base policy properties:
>
> - Room Identifier: `mimi://a.example/r/clubhouse`
> - Roles: `admin = [canAddUser, canRemoveUser, canSetUserRole]`
>
> And the following participant list:
>
> - Participants: `[[mimi://a.example/u/alice, "admin"]]`

HTTP endpoints

Direction	Endpoint	Purpose
*	`GET /.well-known/mimi-protocol-directory`	Learn the endpoints for a server
F ➡ H ➡ F	`POST /keyMaterial/{targetUser}`	Get initial key material for a user
F ➡ H	`POST /update/{roomId}`	Send Proposals / Commit updating the room
F ➡ H	`POST /submitMessage/{roomId}`	Submit a message for delivery
H ➡ F	`POST /notify/{roomId}`	Receive events / messages
F ➡ H	`POST /groupInfo/{roomId}`	Fetch the GroupInfo for the room
F ➡ F	`POST /requestConsent/{targetDomain}` `POST /updateConsent/{requesterDomain}`	Request, grant, revoke, and cancel consent
F ➡ H	`POST /identifierQuery/{domain}`	Find identifier for a user in the target domain
C ➡ H	`POST /reportAbuse/{roomId}`	Report alleged abuse

Since IETF 120

PRs since 120

🧑‍🎨 Diagram the extended Associated Parties Key Schedule to derive franking_context_secret (PR#87)

🔐 Encrypt GroupInfo and ratchet_tree in response to /groupInfo endpoint (PR#86)

🆙 Support more efficient ways to upload ratchet_tree and GroupInfo (PR#85)

💌 Add abuse reporting mechanism with message franking (PR#83)

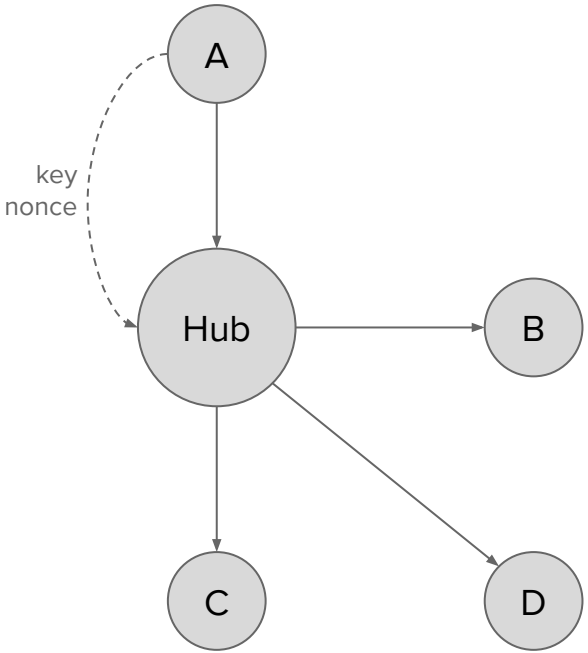
🫣 Semi-private handshake can use an MLS SemiPrivateMessage (PR#80)

Privacy Protection Levels

Metadata such as room policies / participant lists are exposed to...

1. ~~Clients, hub server, and non-hub servers~~ ← unnecessarily over-sharing
2. **Clients and hub server** ← **Baseline - Done**
3. **Clients and hub server (pseudonymously)** ← **Opt-in - WIP**
4. ~~Clients~~ ← too much functionality loss

MLS SemiPrivateMessage



Members of the group agree on a set of **external recipients** (e.g., hub servers)

When a message is encrypted, the encryption key and nonce are encrypted to the external recipients

Commits (=> tree, membership) are available to the hub, but not other servers

Syntactically limited to Commits/Proposals

Encrypted GroupInfo / ratchet tree

- When fetching GroupInfo / ratchet tree they are returned encrypted for the requesting client — External joiner's local provider can't see the GroupContext and members
- Welcome has GroupInfo encrypted to new joiner
- When sending a Commit, if the Hub can construct the ratchet tree and GroupContext from Commits, commiter's local provider doesn't see them
- Follower servers get Proposals and Commits but not the GroupInfo or ratchet tree

Abuse Reporting / Franking Requirements

- Receiver of a message can report it as allegedly abusive to the Hub provider
  - Because it knows the policy
  - Because all messages go through the hub
  - Sends the plaintext of the allegedly abusive message(s) to the Hub
- Hub can't read the content of messages unless they are reported
- Receiver can't attribute a bogus abusive message to a sender
- Sender can't deny a message they sent
- Follower servers do not learn sender/content of any message **
- Hub servers need to know the sender of a message in order to enforce policies related to which participants can send messages.
- Receiving client does not need to store ciphertext indefinitely to report a message (could decrypt/store, report later)
- Hubs don't need to store ciphertext to verify they received a particular message
- Not especially heavy to implement for client or server.
- Doesn't add extra round trips before sending a message.
- There may be additional requirements (Hub reporting its findings to sender's provider) that can be implemented as additional endpoints that do not have any client implementation implications. Probably handled with SLAs

Franking

- Incoming messages are Franked
- Sender
  - Creates a unique per-message secret
  - Encodes the key, its identity, the room URI inside the message
  - Generates a franking tag covering the message and puts it in the AAD
- Hub
  - Hashes the context (sender URI + room URI + timestamp) with a franking_context_key exported using associated parties (members + Hub)
  - Encrypts a hash of the tag and the context hash
  - Forwards to receiver
- Receiver
  - Verifies the tag and context are correct or throws away (sender didn't lie)

Reporting

- Report is sent by receiver to the Hub
  - Contains decrypted message and frank
  - Hub can verify that it franked THIS MESSAGE from the same sender
  - Hub can verify the message presented by the reporter matches a message sent by the Sender.

- How Hub communicates with other providers (ex: sender provider) is not covered. Out of scope? Likely to be covered by SOWs.

- **Note:** Franking introduced a requirement that the receiving client receives the hub accepted time

You asked for it.
We added it.
Do we like it?

Metadata privacy level 2 (SemiPrivate / Assoc Parties)

Pros:

- Provides an admirable baseline level of metadata privacy

Cons:

- Relies on two new MLS extensions (SemiPrivateMessage and associated parties)
- Adds non-trivial complexity. Could result in slower / fewer implementations

Do we want to keep this mandatory or make it optional?

Next Topics

What's left

Downloading files (#75 / attachments draft) – Holding pattern; need to do, but respecting privacy

Privacy from the Hub (ex: pseudonyms, MIMIMI) — Protocol should be agnostic to these additional privacy measures.

Room Policy definitions — separate document?

Baseline Identity / Credential definitions — separate document?

Downloading Files

General pattern pretty clear and widely used:

- Sender encrypts file, uploads it somewhere
- Sender sends link + encryption key to the group

Practical concern: Who will host the files in a federated chat?

Privacy concern: File hosting provider can see who downloads (IP addresses, etc.)

PR provides **proxy download**, client's provider can ask the hub to download a file on their behalf

Pseudonyms / MIMIMI covered later