draft-ietf-mimi-arch

Richard Barnes
IETF 120, July 2024
● 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

Room State

Authz Policy
- User 1 Capas
- User 2 Capas
- User 3 Capas
- Admission Policy
- etc.

Participant List
- User 1 (active)
- User 2 (active)
- User 3 (inactive)

MLS Group
- Client 1a
- Client 2a
- Client 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.

Preemption
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.
Overview

Hub

POST /submit

Follower

POST /notify

Follower

Follower

POST /notify

Proposal
Proposal
Proposal
Proposal
Server-to-Server Communications

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

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

<table>
<thead>
<tr>
<th>Direction</th>
<th>Endpoint</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>GET /well-known/mimi-protocol-directory</td>
<td>Learn the endpoints for a server</td>
</tr>
<tr>
<td>F ➔ H ➔ F</td>
<td>POST /keyMaterial/{targetUser}</td>
<td>Get initial key material for a user</td>
</tr>
<tr>
<td>F ➔ H</td>
<td>POST /update/{roomId}</td>
<td>Send Proposals / Commit updating the room</td>
</tr>
<tr>
<td>F ➔ H</td>
<td>POST /submitMessage/{roomId}</td>
<td>Submit a message for delivery</td>
</tr>
<tr>
<td>H ➔ F</td>
<td>POST /notify/{roomId}</td>
<td>Receive events / messages</td>
</tr>
<tr>
<td>F ➔ H</td>
<td>POST /groupInfo/{roomId}</td>
<td>Fetch the GroupInfo for the room</td>
</tr>
</tbody>
</table>
Since IETF 119
PRs since 119

🎉 Converted to WG document (PR#76)
🚛 Moved appsync to its own spec (PR#82)
🔍 Added find-internal endpoint (PR#74)
🆗 Added a basic consent mechanism (PR#78)
❓ Bunch of discussion about invalid commits, no PRs
Open PRs

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

Abuse reporting (#83) – Adding message franking to support

Two approaches to hiding metadata from non-hub servers

   Using an MLS SemiPrivateMessage construct (#80)

   Using a non-MLS mechanism to reveal the required info to the hub (#79)
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
Metadata Privacy Questions

On the one hand: Desirable to deny servers metadata that they do not need

On the other hand: Desirable to have servers participate in policy actions

- Bans / deleting accounts (Remove), new devices / join links (Add)

  In both cases, seeing the corresponding Commit to verify that change applied

Three levels of protection: Proposals/Commits are visible to...

1. Clients, hub server, and non-hub servers ← today
2. Clients and hub server ← PR#79 / 80
3. Clients ← draft-mcmillion-... draft-kohbrok... later on agenda
Abuse Reporting*

Endpoint for reporting abusive messages to the hub (not follower servers)

Reporter sends to endpoint on hub:

- MLS AuthenticatedContent ← message content + sender signature
- MAC over message by hub ← proves to hub that the message transited the hub

Requires the hub to know the group’s ratchet tree

* not actually in the PR right now, result of Rohan + Richard chatting this week