MIMI Content Protocol Room Policy

Rohan Mahy — rohan.ietf@gmail.com

24-July-2025

with contributions from:
Timo Kösters — timo@koesters.xyz

https://github.com/ietf-wg-mimi/draft-ietf-mimi-content/

MIMI Content

IETF 123, Madrid

Rohan Mahy — rohan.ietf@gmail.com
24-July-2025

Changes since -06

- Removed message status format (Issue #62/PR #65) and submitted as draft-mahy-mimi-message-status
- Fixed construction of Message ID
- Fixed extension keys in the Examples (PR #56 from bitbltr)
- Fixed text so the salt is always 16-octets (Issue #58/ PR#63)
- Updated change log
- Regenerated all the examples (PR#57)

Issues

- width and height for attachments (#59): wrote AV metadata draft (draft-mahy-mimi-av-metadata)
- inconsistent timestamp format (#60): propose won't fix
- message ID calculation question (#61): do we need separators?
- move delivery status to separate spec (#62): (already discussed)
- IANA registry of dispositions? (#64)
- message ID interaction replying to edits (#67):

Inconsistent Timestamps #60

- Hub timestamp (only in the **mimi-protocol** and **mimi-message-status**) is by defaults uint8 in msecs since the start of the UNIX epoch. In mimi-message-status can also use CBOR time tags for more granularity.
- Expires time in **mimi-content** is currently a uint4 in seconds since the start of the UNIX epoch. Will wrap in 2106.
  - Do nothing - (I sincerely hope we have a new version of MIMI content or a replacement by 2106)
  - Extend it to uint8 and keep it in seconds
  - Extend it to uint8 in milliseconds
  - Try to make consistent with hub timestamp
- Proposal: do nothing.

Message ID calculation #60

- Concern that because of the concatenation below, an attacker could generate two valid messages with the same message ID.

```
hash_output = hash( senderUri || roomUri || message || salt )
messageId = hashAlg || hash_output[0..30]
```

- Salt is fixed length, message always starts with 0x87 which is not legal in a room URI. Could have different sender URI + room URI combinations which are the same. However, this would require likely unprecedented freedom to manipulate the sender URI and room URI.
- Options:
  - document the possibility of attack and warn hub and client to validate sender and room URI
  - disallow `mimi://` in the path part of a sender URI or room URI
  - add a separator which is not legal in URIs (ex: `|`) between sender URI and room URI
  - frame the hash input with something like CBOR

Replying to Edited messages #67

- What Message ID do you put in the InReplyTo for a Reply or Reaction when a message has been edited (possibly multiple times)?
  - My opinion: use the Message ID of the version of the message to which you replied/reacted (might not be the first instance or most recent instance)
- Do we need to add text/guidance about this?

Next Steps

- Close these open issues
- Immediately start a WGLC
- Adopt mimi-message-status?
- (discussion of mimi-av-metadata is at the end)

https://github.com/ietf-wg-mimi/mimi-protocol/

MIMI Protocol

IETF 123, Madrid

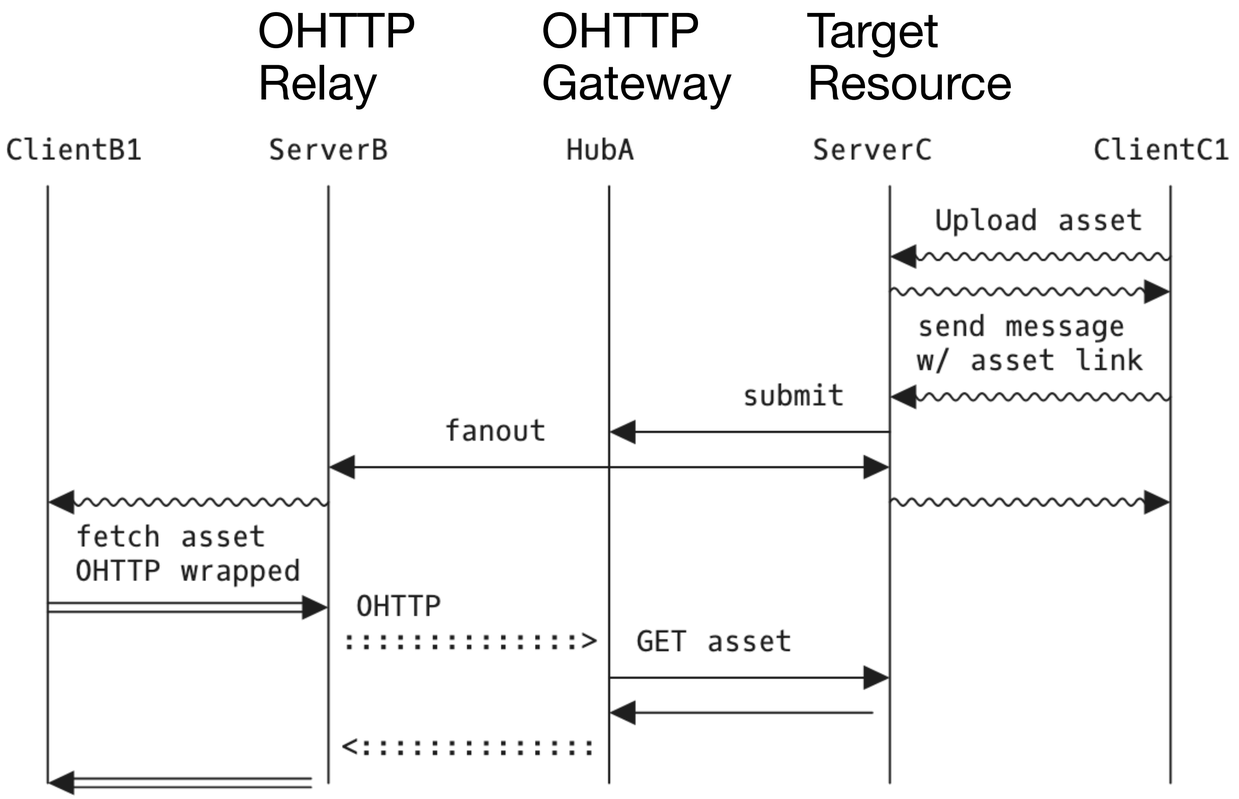
Rohan Mahy — rohan.ietf@gmail.com
24-July-2025

MIMI protocol changes since IETF122

- Merged proxyDownload endpoint #111 (see next slide)
- Make KeyMaterial requests safer from provider tampering #112
- Change franking_integrity to use signature #118
- Provide opportunity for hub to replace invalidated pending proposals immediately #120
- Fixes
  - Fix some text related to salts
  - Fix name of key in SignWithLabel in GroupInfoResponse #113
  - Disable autopublish in the CI #126

Download proxy service

- Clients currently upload “assets” (images/videos/ attachments) somewhere, for use by clients on the same provider
- If we did nothing, a client from another provider would just download this asset directly from the sender’s provider. This would work but would reveal a lot of private information about the downloading client
- With this PR, the client has 3 choices:
  - direct HTTP download
  - `downloadProxy` endpoint on Hub prevents sender's
    local provider from seeing IP addresses of room members.
  - Oblivious HTTP (OHTTP) RFC 9458



Download proxy service

- What information does each entity have in each option?

	downloading provider		hub provider	asset provider	
	client IP address; time of request	specific client (auth) target URL; infer which of its users in same room	target URL; downloading provider; time of access*	target URL; client IP addresses, cookies; time of access; infer social graph	target URL; hub provider; time of access*
direct download				X	
downloadProxy	X	X	X		X
OHTTP	X		X		X

*can be cached by the provider

Download proxy policy

a.com → storage.a.com
hub.net → assets.hub.net
b.org → assets.b-org.hub.net

- What do providers need to implement?
  - Make `downloadProxy` endpoint mandatory to implement on the Hub
  - Make OHTTP Gateway mandatory to implement/should deploy on the Hub
  - OHTTP Relay should implement/deploy on local providers.
  - Clients can decide to fetch attachments directly, only use `downloadProxy` endpoint, or use OHTTP (RECOMMENDED)

- Attachment policy in a room has a specific host for each provider represented in the room. Clients check for sender Alice an attachment is only coming from `storage.a.com` and for sender Bob an attachment is only coming from `assets.b-org.hub.net`
  - For both `downloadProxy` and OHTTP, hub MUST check that target is an allowed storage domain from one of its rooms. Prevents endpoint from becoming an open proxy

Safer KPs #112

- Make sure *requester's provider* can't tamper with client's KeyMaterialRequest
- Include client's credential and sign with corresponding key

```
struct {
    Protocol protocol;
    IdentifierUri requestingUser;
    IdentifierUri targetUser;
    IdentifierUri roomId;
    select (protocol) {
        case mls10:
            CipherSuite acceptableCiphersuites<V>;
            RequiredCapabilities requiredCapabilities;
            SignaturePublicKey requesterSignatureKey;
            Credential requesterCredential;
            /* SignWithLabel(requesterSignatureKey,                */
            /*   "KeyMaterialRequestTBS", KeyMaterialRequestTBS) */
            opaque key_material_request_signature<V>;
    };
} KeyMaterialRequest;

struct {
    Protocol protocol;
    IdentifierUri requestingUser;
    IdentifierUri targetUser;
    IdentifierUri roomId;
    select (protocol) {
        case mls10:
            CipherSuite acceptableCiphersuites<V>;
            RequiredCapabilities requiredCapabilities;
            SignaturePublicKey requesterSignatureKey;
            Credential requesterCredential;
    };
} KeyMaterialRequestTBS;

key_material_request_signature = SignWithLabel(requesterSignatureKey
                      "KeyMaterialRequestTBS", KeyMaterialRequestTBS)
```

Franking integrity calculation #118

- Replaced the a franking integrity HMAC with a franking integrity signature. Covers the same information (`server_frank` and `context`)
- Requires a `franking_signature_key` in the GroupContext (in a new application component).

When a valid `franking_agent` application component Section 10.2 is present in the GroupContext, the Hub can frank messages, and sign the frank with the Hub's `franking_private_key` corresponding to the `franking_signature_key` in the FrankingAgentData struct.

When the Hub receives an acceptable application message with the `frank_aad` Safe AAD component and a valid sender identity, it calculates a server frank for the message, and a `franking_integrity_signature` as follows:

```
context = sender_uri || room_uri || accepted_timestamp
server_frank = HMAC_SHA256(hub_key, franking_tag || context )
franking_integrity_signature =
  Signature.Sign(franking_private_key, server_frank || context)
```

PRs

- Include valid proposals with a GroupInfo #127
  - pending SelfRemove and AppDataUpdate proposals are valid in External Commits
  - Need a way to get them
  - Include with GroupInfo

```
+ struct {
+     MLSMessage proposal;
+     uint64 hub_accepted_time;
+ } PendingProposal;
+
  struct {
    GroupInfo groupInfo;   /* without embedded ratchet_tree */
    RatchetTreeOption ratchetTreeOption;
+   /* A list of valid pending proposals accepted by the hub */
+   /* during the current epoch                              */
+   PendingProposal pending_proposals<V>;
  } GroupInfoRatchetTreeTBE;
```

```
+ If successful, the response body contains the GroupInfo, a way to get the
+ `ratchet_tree`, and a list of any pending proposals for the epoch, all
+ encrypted with the `groupInfoPublcKey` passed in the request.
```

Include hub’s invalidated pending proposals immediately #120

- Provide opportunity for hub to replace invalidated pending proposals immediately
  - removals
  - policy enforcement)

- Hub generates some proposals, cues those up
- Someone comes along and does an external Commit
- Unfortunately, that external Commit can't include the hub's proposals
- So the Hub regenerates the proposals in the new epoch and distributes them stapled to the external commit.

- Might be irrelevant if implemented: draft-mahy-mls-external-commit-pp

```
        MLSMessage message;
        select (message.wire_format) {
          case application:
             optional Frank frank;
          case welcome:
             RatchetTreeOption ratchetTreeOption;
          case proposal:
             /* a list of additional proposals, each represented */
             /* as either PublicMessage or SemiPrivateMessage    */
             MLSMessage moreProposals<V>;
          case commit:
-            struct {};
+            /* If this was an external commit, and any pending      */
+            /* proposals were invalidated, staple the new epoch's   */
+            /* replacement proposals (from the hub) to the commit   */
+            /* commit */
+            MLSMessage externalProposals<V>;
        };
    };
  } FanoutMessage;
```

Issues

just discussed (PR #120)

To Do — Just work

mimi-content Issue #61

Will move to architecture, No action for now

To Do — Just work

To Do

chore

To Do

To Do

discussed earlier (download proxy)

see next slide

- **Minimize time that policy that policy enforcing proposals can be invalid.** has-PR needs-WG-discussion
  #128 · rohanmahy opened 2 weeks ago
- **add to security/privacy considerations a bunch of evil scenarios related to collusion and actively manipulating timing**
  #124 · rohanmahy opened 2 weeks ago
- **Do we need franking context to use a struct or separators?**
  #122 · rohanmahy opened 3 weeks ago
- **Timestamp integrity for history**
  #114 · rohanmahy opened on Mar 19
- **Need MIMI Threat model** ready-for-PR WG reviewed
  #93 · rohanmahy opened on Dec 4, 2024
- **Add description of flows with pseudonyms (opt-in privacy enhancement)** WG reviewed
  #84 · rohanmahy opened on Aug 13, 2024
- **Make more ASCII art flow changes** chore WG reviewed
  #68 · rohanmahy opened on Mar 4, 2024
- **Figure out versioning** ready-for-PR WG reviewed
  #52 · rohan-wire opened on Feb 19, 2024
- **Do we need knocks?** consensus-required WG reviewed
  #38 · turt2live opened on Nov 9, 2023
- **Support external media requests** has-PR WG reviewed
  #24 · turt2live opened on Nov 9, 2023
- **Tracking arbitary state?** consensus-required WG reviewed
  #23 · turt2live opened on Nov 9, 2023

Arbitrary State (Issue #23)

- The MLS AppDataUpdate and app_data_dictionary extensions in MLS extensions allows for applications to efficiently put arbitrary state in the MLS GroupContext or convey it in KeyPackages, LeafNodes, and GroupInfos.
- AppEphemeral allows members to send some arbitrary state to the group as well (ex: include a join code with a new joiner).
- This seems to satisfy most of the request in Issue #23
- What about for state that is “members-only”? We could encrypt it and leave the key distribution to members to another mechanism.
- Suggest closing the issue and open a new one “Share encrypted state with only members”

What's Next?

- Propose a versioning scheme
- Propose a knock scheme
- Can we move more detailed extensions for minimal metadata to other documents?
- Can we have this ready for WGLC by Montreal?

https://github.com/ietf-wg-mimi/mimi-room-policy/

MIMI Room Policy

draft-ietf-mimi-room-policy

Rohan Mahy: rohan.ietf@gmail.com

IETF 123, 24-July–2025

Changes and Issues

- No Changes since IETF 122
- Issues

Timo {

- **Add self_role_changes array for own-role transitions**
  #10 · timokoesters opened on May 14
- **Is canAddParticipant redundant?**
  #9 · timokoesters opened on Apr 16

already discussed with mimi-protocol downloadProxy

- **Download and link policies**
  #8 · rohanmahy opened on Mar 16

To Do — Just work

- **Note that subrooms are on the same hub as the parent room**
  #6 · rohanmahy opened on Jan 29

see next slide

- **Requirements and information associated with bans?**
  #5 · rohanmahy opened on Jan 29

should kick remove from participant list?

- **What are the semantics of kick?**
  #3 · rohanmahy opened on Jan 29

Advanced role changes #10 (Timo)

- Want room where Admins can
  - Make Regular users → Admins
  - Demote themselves Admin → Regular
  - Cannot demote other Admins
- Proposal
  - split role changes into two lists:
    - authorized_role_changes
    - self_role_changes
- Is there enough interest to incorporate this?

- Room has these roles
  - outsider (0)
  - banned (1)
  - regular (2)
  - admin (3)
  - owner (4)

Remove some capabilities #9 (Timo)

- Some actions require both a capability and appropriate authorized_role_change.
- Why?
  - authorized_role_change was developed as a secondary check.
  - It works.
  - It could potentially be more granular. ex: canRemoveParticipant but not canKick

- Could we get rid of some these caps?
  - canChangeUserRole
  - canChangeOwnRole canRemoveParticipant
  - canRemoveSelf
  - canBan
  - canUnban
  - etc..

Extra information for Bans? #5

- What information should be included

  Timo suggests:

  - Time the ban expires (if ever)
  - who initiated the ban
  - the reason for the ban

  optionally the (list of) old role(s) could be stored.

  This may be different from a suspension that allows a user to still view messages, or something similar.

Backup Slides

Role-Based Access Control — role changes

- With some capabilities Holder may change the role of other participants:
  - canChangeUserRole
  - canChangeOwnRole
  - canAddParticipant
  - canAddOwnClient
  - canAddSelf
  - canUseJoinCode
  - canRemoveParticipant
  - canRemoveOwnClient
  - canRemoveSelf
  - canKick
  - canChangeUserRole
  - canChangeOwnRole
  - canBan
  - canUnban
- Role transitions
  - adding, moves from role 0 to a specific role; removing moves to role 0
  - banning moves to role 1; unbanning moves from role 1 (if role 1 exists)
  - authorized_role_changes = [(0,[1,2,3]), (1,[0,2,3]), (2,[0,1,3]), (3,[0,1,2])]

Role-Based Access Control — external senders

- **New**: Hub/Anti-spam external senders can be authorized via one or more roles (ex: `policy_enforcer`) in the participant list.
- They are authenticated via MLS `external_senders` credential, but authorized via RBAC mechanism.
  - Configuration can give the policy enforcer the capabilities to delete clients/users but not add them.
  - Constraints also can prevent policy enforcer from becoming an *active* participant

Preauthorized users

- **Used to give a role to users based on identity properties**
  - **When is it consulted?**
    - When attempted joiner is *neither* in Participant List *nor* has join code
    - When requester is in Participant List and tries to change its own role (ex: become a moderator or admin after being added as a regular user)
  - How do you determine if user is authorized?
    - List of claims per preauth entry. The client must match all claims (AND semantics) to be authorized for the role in an entry.
    - Client gets the first matching role. For example, entries that are banned go first, then the most powerful roles to least powerful.