More Instant Messaging Interoperability
Internet-Draft
Intended status: Informational
Expires: 21 June 2026

R. Mahy
Rohan Mahy Consulting Services

18 December 2025

Room Policy for the More Instant Messaging Interoperability (MIMI) Protocol
draft-ietf-mimi-room-policy-03

Abstract

This document describes a set of concrete room policies for the More Instant Messaging Interoperability (MIMI) Working Group. It describes several independent properties and policy attributes which can be combined to model a wide range of chat and multimedia conference types.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://ietf-wg-mimi.github.io/mimi-room-policy/draft-ietf-mimi-room-policy.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-mimi-room-policy/.

Discussion of this document takes place on the More Instant Messaging Interoperability Working Group mailing list (mailto:mimi@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/mimi/. Subscribe at https://www.ietf.org/mailman/listinfo/mimi/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-wg-mimi/mimi-room-policy.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The MIMI architecture [I-D.ietf-mimi-arch] describes how each room has an associated policy. Providers offer a "policy envelope" of supported and allowed policy settings, from which the creator of a room selects a specific room policy. The room policy might further allow individual participants to make specific choices (for example, allowing but not requiring read-message notifications), while constraining other choices (for example, prohibiting self-deleting messages). Individual users can examine the room policy to determine if it is consistent with policies they accept either before or immediately on joining a room. Section 4.4 of [I-D.ietf-mimi-arch]

Making rooms interoperable across existing clients is challenging, as rooms and clients can support different policies and capabilities across vendors and providers. Our goal is to balance the policy and authorization goals of the room with the policy and authorization goals of the end user, so we can support a broad range of vendors and providers.

Each room is owned by one provider at a time. The owning provider controls the range of acceptable policies. The user responsible for the room can further choose among the acceptable policies. Users (regardless if on other providers) can either accept the policies of the room or not.

However we want to make it as easy as possible for clients from other providers to comply with the room policy primitives without enumerating specific features or requiring all clients implementations to present an identical user experience. An important aspect of this is the system of configurable Role-based access control with granular capabilities per role (described in Section 3). Each user in the participant list (defined in [I-D.ietf-mimi-protocol]) has exactly one role. By virtue of a user's credential, a user might also be _preauthorized_ with a specific role as described in Section 4.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Room ID: An identifier which uniquely identifies a room.

User ID: An internal identifier which uniquely identifies a user.

Nickname: The identifier by which a user is referred inside a room. Depending on the context it may be a display name, handle, pseudonym, or temporary identifier. The nickname in one room need not correlate with the nickname for the same user in a different room.

Client ID: An internal identifier which uniquely identifies one client/device instance of one user account.

Persistent vs. Temporary rooms: A temporary room is no longer joinable once the last participant exits whereas a persistent room is not destroyed when the last participant exist. As MLS has no notion of a group with no members, a persistent room could consist of a sequence of distinct MLS groups, zero or one of which would exist at a time.

2.1. Moderation Terms

Knock: To request entry into a room.

Ban: To remove a user from a room such that the user is not allowed to re-enter the room (until and unless the ban has been removed). It is distinct from merely removing a user from a room.

Kick: To temporarily remove a participant’s clients from a room. The user is allowed to re-enter the room at any time.

Voice (noun): The privilege to send messages in a room.

Revoke Voice: To remove the permission to send messages in a room.

Grant Voice: To grant the permission to send messages in a room.

3. Role-Based Access Control

Most instant messaging systems have a concept of room membership being managed by a set of moderators or administrators, or collectively managed by existing members. In some cases, rooms are completely open to new joiners unless they have been banned in some way. In an enterprise context, it is also common (but not required) for users from a particular domain, group, or workgroup to be pre-authorized to add themselves to various types of rooms. All these variations of room access are managed in MIMI using roles, capabilities (Section 8), and preauthorization (Section 4).

The Role-Based Access Control component contains a list of all the roles in the room, and the capabilities associated with them. It contains a role_index, which is used to refer to the role elsewhere. (Note that role indexes might not be contiguous.) The role_index zero is reserved to refer to a participant that does not (yet) or no longer appears (or will no longer appear) in the participant list.

The component also contains a role_name (a human-readable text string name for the role), and a role_description (another string, which can have zero length).

Each Role also can contain constraints on the minimum and maximum number of participants, and the minimum and maximum number of active participants. If the minimum number is zero, there is no minimum number of participants for that particular role. If there is no maximum number of participants for a particular role, that parameter is absent.

> If the maximum number of active participants is zero, then no participants are allowed to have clients in the room's MLS group.

The authorized_role_changes field is used to provide fine-grained control about which transitions are allowed when adding and removing participants and when moving participants to new roles, including banning/unbanning, and promoting/demoting to or from roles with moderator or administrator privileges. A more detailed discussion is in the description of the specific capabilities in the next section.

> This design results in each participant only having a single role at a time, with a single list of capabilities and an explicit list of allowed role transitions. It makes the authorization process for a verifier consistent regardless of the complexity of the set of authorization rules.

Some examples are provided in Appendix A.

RoleData is the format of the data field inside the ComponentData struct for the Role-Based Access Control component in the app_data_dictionary GroupContext extension defined in [I-D.ietf-mls-extensions].

```
/* See MIMI Capability Types IANA registry */
uint16 CapabilityType;

struct {
   uint32 from_role_index;
   uint32 target_role_indexes<V>;
} SingleSourceRoleChangeTargets;

struct {
  uint32 role_index;
  opaque role_name<V>;
  opaque role_description<V>;
  CapabilityType role_capabilities<V>;
  uint32 minimum_participants_constraint;
  optional uint32 maximum_participants_constraint;
  uint32 minimum_active_participants_constraint;
  optional uint32 maximum_active_participants_constraint;
  SingleSourceRoleChangeTargets authorized_role_changes<V>;
} Role;

struct {
  Role roles<V>;
} RoleData;

RoleData roles_list;
RoleData RoleUpdate;
```

RoleUpdate (which has the same format as RoleData) is the format of the update field inside the AppDataUpdate struct in an AppDataUpdate Proposal for the Role-Based Access Control component. If the contents of the update field are valid and if the proposer is authorized to generate such an update, the value of the update field completely replaces the value of the data field.

> Note that in the MIMI environment, changing the definitions of roles is anticipated to be very rare over the lifetime of a room (for example changing a room which has grown dramatically from cooperatively managed by all participants to explicitly moderated or administered).

Changing Role definitions is sufficiently disruptive, that an update to this component is not valid if it appear in the same commit as any Participant List change.

4. Preauthorized Users

Preauthorized users are MIMI users and external senders that have authorization to adopt a role in a room by virtue of certain credential claims or properties, as opposed to being individually enumerated in the participant list. For example, a room for employee benefits might be available to join with the regular participant role to all full-time employees with a residence in a specific country; while anyone working in the human resources department might be able to join the same room as a moderator. This data structure is consulted in two situations: for external joins (external commits) and external proposals when the requester does not already appear in the participant list; and separately when an existing participant explicitly tries to change its _own_ role.

> Only consulting Preauthorized users in these cases prevents several attacks. For example, it prevents an explicitly banned user from rejoining a group based on a preauthorization.

PreAuthData is the format of the data field inside the ComponentData struct for the Preauthorized Participants component in the application_data GroupContext extension.

The individual PreAuthRoleEntry rules in PreAuthData are consulted one at a time. A PreAuthRoleEntry matches for a requester when every Claim.claim_id has a corresponding claim in the requester’s MLS Credential which exactly matches the corresponding claim_value. When the rules in a Preauthorized users struct match multiple roles, the requesting client receives the first role which matches its claims.

> *TODO*: refactor Claims

```
struct {
  /* MLS Credential Type of the "claim"  */
  CredentialType credential_type;
  /* the binary representation of an X.509 OID, a JWT claim name  */
  /* string, or the CBOR map claim key in a CWT (an int or tstr)  */
  opaque id<V>;
} ClaimId;

struct {
  ClaimId claim_id;
  opaque claim_value<V>;
} Claim;

struct {
  /* when all claims in the claimset are satisfied, the claimset */
  */ is satisfied */
  Claim claimset<V>;
  Role target_role;
} PreAuthRoleEntry;

struct {
  PreAuthRoleEntry preauthorized_entries<V>;
} PreAuthData;

PreAuthData preauth_list;
PreAuthData PreAuthUpdate;
```

PreAuthUpdate (which has the same format as PreAuthData) is the format of the update field inside the AppDataUpdate struct in an AppDataUpdate Proposal for the Preauthorized Participants component. If the contents of the update field are valid and if the proposer is authorized to generate such an update, the value of the update field completely replaces the value of the data field.

> As with the definition of roles, in MIMI it is not expected that the definition of Preauthorized users would change frequently. Instead the claims in the underlying credentials would be modified without modifying the preauthorization policy.

Changing Preauthorized user definitions is sufficiently disruptive, that an update to this component is not valid if it appears in the same commit as any Participant List change, except for user removals.

Because the Preauthorized users component usually authorizes non-members, it is also a natural choice for providing concrete authorization for policy enforcing systems incorporated into or which run in coordination with the MIMI Hub provider or specific MLS Distribution Services. For example, a preauthorized role could allow

the Hub to remove participants and to ban them, but not to add any users or devices. This unifies the authorization model for members and non-members.

5. Base Room policy component syntax

The following format is an MLS component which expresses top-level policy constraints, including global rules related to how membership is interpreted. The rest of the rules about membership (the bulk) are expressed using roles Section 3, capabilities Section 8, and preauthorization Section 4.

Rooms with fixed_membership set to true (fixed-membership rooms) have the list of participants specified when they are created. While clients of existing participants can be added, other users cannot be added, so none of its non-zero, non-banned roles can contain the canAddParticipant capability. Ordinary users cannot leave or be removed, however a user can remove all its clients from the associated MLS group. The most common case of a fixed-membership room is a 1:1 conversation. This room membership style is used to implement Direct Message (DM) and Group DM features. Only a single fixed-membership room can exist for any unique set of participants.

In rooms with parent_dependent set to true (a parent-dependent room), the list of participants of the room MUST be a strict subset of the participants of the parent room. If a user leaves or is removed from the parent room, that user is automatically removed from any parent-dependent rooms of that parent. A parent-dependent room is always hosted on the same Hub as the parent room.

If parent-dependent is true, the parent_room MUST be set with the room ID of the parent. Otherwise the field is zero-length.

> Note: A room can be both fixed_membership and parent_dependent, for example, for room used for a multi-media call of clients in a Group DM.

If multi_device is true (the default), the MLS group may contain multiple clients per user. If false only a single client can be an MLS member at one time.

When max_clients has a value, the room’s associated MLS group MUST NOT have more clients than the provided value. Likewise when max_users has a value, the room MUST NOT have more non-banned entries in the participant list than that value.

```
enum {
  false(0),
  true(1)
} bool;

struct {
    bool fixed_membership;
    bool parent_dependant;
    Uri parent_room<V>;
    bool multi_device;
    optional uint32 max_clients;
    optional uint32 max_users;
    bool pseudonyms_allowed;
    bool persistent_room;
    bool discoverable;
    Component policy_component_ids<V>;
} BaseRoomPolicy;

BaseRoomPolicy BaseRoomData;
BaseRoomPolicy BaseRoomUpdate;
```

If pseudonyms_allowed is true, clients in the MLS group are free to use pseudonymous identifiers in their MLS credentials. Otherwise the policy of the room is that "real" long-term identifiers are required in MLS credentials in the room’s corresponding MLS group.

If persistent_room is false, the room will be automatically inaccsessible when the corresponding MLS group is destroyed (when there are no clients in the group). If persistent_room is true, the room policy will remain and a client whose user has appropriate authorization can create a new MLS group for the same room.

If discoverable is true, the room is searchable in some way. Presumably this means that if discoverable is false, the only way to join the room in a client user interface is to be added by an administrator or to use a joining link.

Finally, the Component IDs of the other policy components that are relevant to this room are listed in the policy_component_ids vector, including the roles_list (from Section 3) and preauth_list components (from Section 4), if present. This extensibility mechanism allows for future addition or replacement of new room policies.

6. Other MIMI policy components

6.1. Status Notifications component

Delivery and Read notifications are a very popular feature of instant messaging systems, but also can leak private information such as the online status of participants. Such status notifications can also consume a large amount of resources, especially in large rooms.

```
enum {
  optional(0),
  required(1),
  forbidden(2)
} Optionality;

struct {
  Optionality delivery_notifications;
  Optionality read_receipts;
} StatusNotificationPolicy;

StatusNotificationPolicy StatusNotificationPolicyData;
StatusNotificationPolicy StatusNotificationPolicyUpdate;
```

The delivery_notifications value can be set to "forbidden", "optional", or "required". If the value is set to "optional", the client uses its local configuration to determine if it should send delivery notifications in the room.

The read_receipts value can be set to "forbidden", "optional", or "required". If the value is set to "optional", the client uses its local configuration to determine if it should send read receipts in the room.

The format for delivery notifications and read receipts is described in [I-D.mahy-mimi-message-status].

6.2. Join Link policies component

Inside the JoinLinkPolicy are several fields that describe the behavior of new join links.

If the on_request field is true, a maximum of one joining link will be persisted in the room policy; the client will need to fetch a joining link out-of-band or generate a valid one for itself before a new one can be generated. If present, the URI in link_requests can be used by the client to request an invite code.

If the on_request field is false, multiple joining links can be generated and persisted. If links can be generated for multiple users, multiuser is true. The expiration field represents the duration in seconds that a new link can be valid after creation.

```
struct {
  bool on_request;
  Uri join_link;
  bool multiuser;
  uint32 expiration;
} JoinLinkPolicy;

JoinLinkPolicy JoinLinkPolicyData;
JoinLinkPolicy JoinLinkPolicyUpdate;
```

The active join links in a room are persisted separately in a JoinLinks Component.

```
struct {
  opaque join_link;
} JoinLink;

JoinLink JoinLinksData<V>;

struct {
  uint32 removedIndices<V>;
  JoinLink added_links<V>;
} JoinLinksUpdate;
```

6.3. Link Preview policy component

Link preview policy is concerned with the safe rendering of explicit or implicit hyperlinks in the text of an instant message.

The autodetect_hyperlinks_in_text setting indicates if a message composer is expected to detect hyperlinks from text which resembles links (ex: http://example.com). The value of autodetect_hyperlinks_in_text MUST NOT be mandatory. The send_link_previews setting indicates if the sender of a message including a link preview (a desirable feature, but a malicious sender could generate a preview inconsistent with the actual link content) is mandatory, optional, or forbidden.

The automatic_link_previews setting indicates if the receiver of a message generating link previews (a desirable feature, but a potential privacy concern) is mandatory, optional, or forbidden. The link_preview_proxy_use setting indicates if using a specialized link preview proxy is mandatory, optional, or forbidden when link previews are generated.

The link_preview_proxy setting MUST include the URI of a link preview proxy if link_preview_proxy_use is mandatory or optional.

```
struct {
  Optionality autodetect_hyperlinks_in_text;
  Optionality send_link_previews;
  Optionality automatic_link_previews;
  Optionality link_preview_proxy_use;
  select (link_preview_proxy_use) {
    case mandatory:
      Uri link_preview_proxy<V>;
    case optional:
      Uri link_preview_proxy<V>;
    case forbidden:
      struct {};
  }
} LinkPreviewPolicy;

LinkPreviewPolicy LinkPreviewPolicyData;
LinkPreviewPolicy LinkPreviewPolicyUpdate;
```

6.4. Asset policies component

Assets refer to attached files, images, audio files, and video files.

The asset_upload_location could be unspecified, indicating any location; localProvider, indicating that each client uploads assets to its local provider; or hub, indicating that all clients upload assets to the hub. Using localProvider is RECOMMENDED.

The asset_upload_domains is a list of asset_upload_destinations per provider domain name. If the asset_upload_location is hub, only one provider matching the hub domain is present in asset_upload_domains.

Unless asset_upload_location is unspecified, the clients verify that assets in the host part of the url of any MIMI content [I-D.ietf-mimi-content] ExternalPart corresponds to an entry in asset_upload_domains. If the asset_upload_location is localProvider the asset_upload_domains list is from the provider domain name that exactly matches the sender URI domain name.

download_privacy describes the mechanisms acceptable for downloading assets. The allowed_download_types and forbidden_download_types specify the download mechanisms which are are allowed and forbidden, respectively. The default_download_type is the default or suggested download mechanism. direct refers to client direct download as described in Section 5.10.1 of [I-D.ietf-mimi-protocol]. hubProxy refers to client download through a proxy on the hub as described in Section 5.10.2 of [I-D.ietf-mimi-protocol]. ohttp refers to client download through Oblivious HTTP [RFC9458] through the hub as described in Section 5.10.3 of [I-D.ietf-mimi-protocol].

```
   enum {
     unspecified(0),
     localProvider(1),
     hub(2),
     (255)
   } AssetUploadLocation;

   struct {
     opaque domain<V>;
   } DomainName;

   struct {
     DomainName provider;
     DomainName asset_upload_destinations<V>;
   } ProviderAssetUploadDomains;

   enum {
     direct(0),
     hubProxy(1),
     ohttp(2),
     (255)
   } DownloadPrivacyType;

   struct {
     DownloadPrivacyType allowed_download_types<V>;
     DownloadPrivacyType forbidden_download_types<V>;
     DownloadPrivacyType default_download_type;
   } DownloadPrivacy;

   struct {
     AssetUploadLocation asset_upload_location;
     ProviderAssetUploadDomains upload_domains<V>;
     DownloadPrivacy download_privacy;
     uint64 max_image;
     uint64 max_audio;
     uint64 max_video;
     uint64 max_attachment;
     MediaType forbidden_media_types<V>;
     optional<MediaType> permitted_media_types<V>;
   } AssetPolicy;

   AssetPolicy AssetPolicyData;
   AssetPolicy AssetPolicyUpdate;
```

The max_image, max_audio, max_video, and max_attachment fields indication the maximum size in bytes of the corresponding assets that will be accepted. These amounts could be further limited at the client according to local policy or at the upload location based on various forms of authorization and quotas.

The following paragraph refers to fields that use the MediaType struct defined in Section 6.2.2 of [I-D.ietf-mls-extensions]. forbidden_media_types is a list of media types (type and subtype) that are not allowed at all in the room. If present, permitted_media_types is a list of media types that are permitted. When it is present, media types MUST be one of the entries in the permitted_media_types list, and MUST NOT be in the forbidden_media_types list. If a media type with no parameters (for example, text/markdown) is present in one of these lists, that entry matches all media types of that type and subtype that contain additional parameters.

6.5. Logging policy component

Some messaging systems (for example in the health care or financial services sectors) often require mandatory logging of calls and messages. The goal of these policies is to make detection of such policies automatic, to allow clients to make appropriate local policy decisions when such policies exist.

Inside the LoggingPolicy, the logging field can be forbidden, optional, or required. If logging is forbidden then the other fields are empty. If logging is required, the list of logging_clients needs to contain at least one logging URI. Each provider should have no more than one logging client at a time in a room. The machine_readable_policy and human_readable_policy fields optionally contain pointers to the owning provider’s machine readable and human readable logging policies, respectively. If logging is optional and there is at least one logging_client then logging is active for the room.

```
struct {
  Optionality logging;
  select (logging) {
    case mandatory:
      Uri logging_clients<V>;
      Uri machine_readable_policy;
      Uri human_readable_policy;
    case optional:
      Uri logging_clients<V>;
      Uri machine_readable_policy;
      Uri human_readable_policy;
    case forbidden:
      struct {};
  }
} LoggingPolicy;

LoggingPolicy LoggingPolicyData;
LoggingPolicy LoggingPolicyUpdate;
```

6.6. Chat history policy component

One of the most requested features of instant messaging systems is that new joiners can view some or all of the message history before joining. While useful, it has serious implications to the privacy of existing members, and substantially weakens forward secrecy (FS) (See Section 8.2.2 of [RFC9750]).

Inside the HistoryPolicy, if history_sharing is forbidden, this means that clients (including bots) are expected to not to share chat history with new joiners, in which case roles_that_can_share is empty, automatically_share is false, and max_time_period is zero. Otherwise roles_that_can_share is a list of roles that are authorized to share history (for example, only admins and owners can share). The role index zero (non-participant) and one (banned) cannot be used in the who_can_share list, nor can any role where max_active_participants is zero. If automatically_share is true, clients can share history with new joiners without user initiation. The history that is shared is limited to max_time_period seconds worth of history.

```
struct {
  Optionality history_sharing;
  select (history_sharing) {
    case mandatory:
      uint32 roles_that_can_share<V>;
      bool automatically_share;
      uint32 max_time_period;
    case optional:
      uint32 roles_that_can_share<V>;
      bool automatically_share;
      uint32 max_time_period;
    case forbidden:
      struct {};
  }
} HistoryPolicy;

HistoryPolicy HistoryPolicyData;
HistoryPolicy HistoryPolicyUpdate;
```

6.7. Chat bot policy component

There are several types of chat bot in instant messaging systems, some of which only interact with the local client.

Inside the BotPolicy there is a list of allowed_bots, each of which has several fields. The name, description, and homepage are merely descriptive.

If local_client_bot is true, the bot would not act as a participant; it would have access to the contents of the room only with another client operated by a (presumably human) user.

The bot_role_index indicates the role index in which the bot operates; this controls the capabilities of the bot. A bot_role_index of zero indicates that the bot is not a active participant in the room. A bot with local_client_bot set to true has a bot_role_index of 0.

If can_target_message_in_group is true it indicates that the chat bot can send an MLS targeted message (see Section 2.2 of [I-D.ietf-mls-extensions]) or use a different conversation or out-of-band channel to send a message to specific individual users in the room.

If per_user_content is true, the chat bot is allowed to send messages with distinct content to each member. (For example a poker bot could deal a different hand to each user in a chat).

Users could set policies to reject or leave groups with bots rights that are inconsistent with the user’s privacy goals.

```
struct {
  opaque name<V>;
  opaque description<V>;
  Uri homepage;
  bool local_client_bot;
  uint32 bot_role_index;
  bool can_target_message_in_group;
  bool per_user_content;
} Bot;

struct {
  Bot allowed_bots<V>;
} BotPolicy;

BotPolicy BotPolicyData;
BotPolicy BotPolicyUpdate;
```

6.8. Message expiration policy component

Many instant messaging systems have an automatically expiring messages feature.

If expiring messages are required, optional, or forbidden is controlled by the expiring_messages field.

When expiring_messages are required or optional, the min_expiration_duration indicates the shortest acceptable expiration duration in seconds. The max_expiration_duration indicates the longest acceptable duration in seconds. The default_expiration_duration optionally indicates a preferred duration in seconds.

When expiring_messages is forbidden, both the min_expiration_duration and the max_expiration_duration are set to zero, and the default_expiration_duration is not present.

```
struct {
  Optionality expiring_messages;
  select (expiring_messages) {
    case mandatory:
      uint32 min_expiration_duration;
      uint32 max_expiration_duration;
      optional uint32 default_expiration_duration;
    case optional:
      uint32 min_expiration_duration;
      uint32 max_expiration_duration;
      optional uint32 default_expiration_duration;
    case forbidden:
      struct {};
  }
} MessageExpiration;

MessageExpiration MessageExpirationData;
MessageExpiration MessageExpirationUpdate;
```

7. Operational policy component

Section 7 of the [RFC9750] defines a set of operational policy considerations that influence interoperability of MLS clients. MIMI explicitly address a handful of the issues in the document by taking a position on ordering (Proposals referenced in a Commit need to be received before the Commit; the Commit entering a new epoch needs to be received before any other messages in that epoch), privacy of handshake messages (handshakes can be a PublicMessage or SemiPrivateMessage), and GroupInfo storage (committers need to provide a valid GroupInfo to the Hub). The rest of these issues are described here. Just because a topic is listed does not mean that a room needs to take a position; nor different rooms on a Hub need to have different policies for these items.

7.1. Some MLS-related policy that could be tied to a room

* any mandatory or forbidden MLS extensions.

* which proposals are valid to have in a commit, including but not limited to:

  - when, and under what circumstances, a reinitialization proposal is allowed.

  - when proposals from external senders are allowed and how to authorize those proposals.

  - when external joiners are allowed and how to authorize those external commits.

  - which other proposal types are allowed.

* when members should commit pending proposals in a group.

* when two credentials represent the same client.

* how long to allow a member to stay in a group without updating its leaf keys before removing them.

* When and how to pad messages.

* When to send a reinitialization proposal.

* How often clients should update their leaf keys.

* Whether to prefer sending full commits or partial/empty commits.

* Whether there should be a required_capabilities extension in groups.

* minimum and maximum lifetime of KeyPackages

* if last resort KeyPackages are allowed

* how long to store resumption PSK (how much time and how many epochs)

* minimum and maximum number past epochs to keep

* how long to keep unused nonce and key pairs for a sender

* maximum number of unused key pairs to keep

* maximum number of steps that clients will move a secret tree ratchet forward in response to a single message before rejecting it

* tolerance to out of order app messages

* tolerance to out of order handshake messages

* handshakes may be which of PublicMessage, PrivateMessage, or SemiPrivateMessage.

* if external joiners are allowed

* if external proposals are allowed

  - if so, who can submit

  - which member(s) are responsible for submitting pending proposals

* how a joiner gets access to the ratchet_tree

The structure below describes provides a way to describe many of these parameters.

```
struct {
  ProtocolVersion versions<V>;
  CipherSuite cipher_suites<V>;
  ExtensionType extensions<V>;
  ProposalType proposals<V>;
  CredentialType credentials<V>;
  WireFormats wire_formats<V>;
  ComponentID component_ids<V>;
  ComponentID safe_aad_types<V>;
  MediaType media_types<V>;
  ContentType content_types<V>;
} ExtendedCapabilities;

enum {
  unspecified(0),
  immediate_commit(1),
  random_delay(2),
  (255)
} PendingProposalStrategy;

struct {
  PendingProposalStrategy pending_proposal_strategy;
  select (pending_proposal_strategy) {
    case unspecified:
      struct {};
    case immediate_commit:
      struct {};
    case random_delay:
      uint64 minimum_delay_ms;
      uint64 maximum_delay_ms;
    case extension:
      ComponentID id_of_strategy_params;
  }
} PendingProposalPolicy;

struct {
```

```
    uint64 minimum_time;
    uint64 default_time;
    uint64 maximum_time;
  } MinDefaultMaxTime;

  struct {
    uint8  epoch_tolerance;
    uint16 pad_to_size;
    uint32 max_generations_skipahead;
  } AppMessagePolicy;

  struct {
    ExtendedCapabilities mandatory_capabilities;
    ExtendedCapabilities default_capabilities;
    ExtendedCapabilities forbidden_capabilities;
    WireFormats handshake_formats<V>;
    bool external_proposal_allowed;
    bool external_commit_allowed;
    PendingProposalPolicy pending_proposal_policy;
    MinDefaultMaxTime LeafNode_update_time;
    AppMessagePolicy app_message_policy;
    unit64 max_kp_lifetime;
    uint64 max_credential_lifetime;
    uint64 resumption_psk_lifetime;
    MinDefaultMaxTime sender_nonce_keypair_lifetime;
    uint32 max_keypairs;
    MinDefaultMaxTime buffer_incoming_message_time;
    uint32 max_buffered_messages;
  } OperationalParameters;

  OperationalParameters OperationalParametersData;
  OperationalParameters OperationalParametersUpdate;
```

7.2. Not relevant to MIMI (between client and its provider)

* how many KPs to keep active

* how group IDs are constructed

7.3. Areas for future works

How to protect and share the GroupInfo objects needed for external joins.

If an application wishes to detect and possibly discipline members that send malformed commits with the intention of corrupting a group’s state, there must be a method for reporting and validating malformed commits.

MLS requires the following parameters to be defined, which must be the same for two implementations to interoperate:

Application-level identifiers of public key material (specifically the application_id extension as defined in Section 5.3.3 of [RFC9420]).

8. Role Capabilities

As described in Section 3, each role has a list of capabilities, which in rare cases could be empty. When we say that the holder of a capability can take some action, we mean that whatever entity is taking the action (a participant, a potential future participant, or an external party) has a specific entry in the Participant List struct and a corresponding role--or is preauthorized to take action with a specific role via the Preauthorized Users struct--and that the role_capabilities list contains the relevant capability.

Unless otherwise specified, capabilities apply both to sending a set of consistent MLS proposals that could be committed by any member of the corresponding MLS group, and to sending an MLS commit containing a set of consistent MLS proposals.

8.1. Membership Capabilities

The membership capabilities below allow authorized holders to update the Participant list, or change the active participants (by removing and adding MLS clients corresponding to those participants), or both.

8.1.1. Adding

* canAddParticipant - the holder of this capability can add another user, that is not already in the participant list, to the participant list. (This capability does not apply to the holder adding itself.) The authorized_role_changes list in the holder’s role is consulted to authorize the added user’s target role. The authorized_role_changes list MUST have an entry where the authorized_role_changes.from_role_index equals zero, and that entry’s target_role_indexes list includes the target role. The proposed action is only authorized if the action respects both the maximum_participants_constraint (if present) and maximum_active_participants_constraint (if present) for the added user’s target role. When the participant list addition for the target role is authorized, the holder is also authorized to add any MLS clients matching the added user to the room’s MLS group .

* canAddOwnClient - a holder of this capability that is in the participant list, can add its own client (via an external commit or external proposal); and can add other clients that share the same user identity (via Add proposals) if the holder's client is already a member of the corresponding MLS group.

* canOpenJoin - when this capability appears on role zero, any user who is not already in the participant list can add itself externally, with certain conditions. The authorized_role_changes list MUST have an entry with from_role_index equal to zero. The holder can add itself with any non-zero target_role_indexes from that entry, if the action respects both the maximum_participants_constraint (if present) and maximum_active_participants_constraint (if present) for the added user's target role. canOpenJoin MUST NOT appear in any non-zero role.

* canJoinIfPreauthorized - when this capability appears on a non-zero role, a client that is not already in the participant list can externally join as that target role if authorized for that role as the first matching role in the Preauthorized users mechanism. The authorized_role_changes list is not consulted for this capability. The action MUST respect both the maximum_participants_constraint (if present) and maximum_active_participants_constraint (if present) for the added user's target role.

* canUseJoinCode - the holder of this capability can externally join a room using a join code for that room, provided the join code is valid, the join code refers to a valid target role, and both the maximum_participants_constraint (if present) and maximum_active_participants_constraint (if present) constraints are respected.

8.1.2. Removing

* canRemoveParticipant - the holder of this capability can propose a) the removal of another user (excluding itself) from the participant list, and b) removal of all of that user's clients, as a single action. There MUST NOT be any clients of the removed user in the MLS group after the corresponding commit. A proposer holding this capability consults its role's authorized_role_changes entries for an entry where from_role_index matches the target user's current role; if the target_role_indexes for that entry contains zero, and the minimum_participants_constraint and minimum_active_participants_constraint are satisfied, the proposal is authorized.

* canRemoveOwnClient - the holder of this capability can propose to remove its own client using an MLS Remove or SelfRemove proposal without changing the Participant list. Due to restrictions in MLS which insure the consistency of the group, this action cannot be committed by the leaving user. If the minimum_active_participants_constraint is satisfied, the proposal is authorized.

* canRemoveSelf - the holder of this capability can propose to remove itself from (i.e. leave) the participant list; it MUST simultaneously propose to remove all of its remaining clients from the corresponding MLS group. Due to restrictions in MLS which insure the consistency of the group, this action cannot be committed by the leaving user. A proposer holding this capability consults its role's authorized_role_changes entries for an entry where from_role_index matches its current role; if the target_role_indexes for that entry contains zero, and the minimum_participants_constraint and minimum_active_participants_constraint are satisfied, the proposal is authorized.

* canKick - the holder of this capability can propose removal of another participant's clients, without changing the Participant List. If the minimum_active_participants_constraint is satisfied, the proposal is authorized.

8.1.3. Role Changes

* canChangeUserRole - the holder of this capability is authorized to change the role of another participant (but not itself), according to the holder's authorized_role_changes list, from a role represented by an entry where the target's current role matches from_role_index to any of the non-zero target_role_indexes in the same element of authorized_role_changes. The minimum_participants_constraint and minimum_active_participants_constraint for the target user's current role, and the maximum_participants_constraint (if present) and maximum_active_participants_constraint (if present) for the target user's target role must also be satisfied.

* canChangeOwnRole - the holder of this capability is authorized to change its own role to the first non-zero role it matches in the Preauthorized users component (see Section 4). The authorized_role_changes list is _not_ consulted. The minimum_participants_constraint and minimum_active_participants_constraint for the holder's original role, and the maximum_participants_constraint (if present) and maximum_active_participants_constraint (if present) for the holder's target role must also be satisfied.

* canBan - the holder of this capability can propose to "ban" another user. Specifically, a successful ban changes the target user's role to a special "banned" role (if it exists), and removes all the banned user's clients. The "banned" role always has role_index = 1 and role_name = "banned" (without quotes).

  A "banned" role does not have to exist in a room, but to use the canBan and canUnban capabilities, the role needs to exist exactly as described above. While holding canChangeUserRole and canKick capabilities would allow the same action, it could potentially allow the holder other actions which might be undesirable in some contexts, such as kicking clients without banning.

  A proposer holding this capability consults its role's authorized_role_changes entries for an entry where from_role_index matches the target user's current role; if the target_role_indexes for that entry contains the role_index 1; that role_name = "banned" for the role with role_index = 1, and the minimum_participants_constraint and minimum_active_participants_constraint are satisfied, the proposal is authorized.

* canUnban - the holder of this capability can propose to "unban" another user. Specifically, a successful unban changes the target user's role from role_index = 1 to another non-zero role_index allowed by the holder's authorized_role_changes list. Adding clients for that unbanned user is _not_ authorized by this capability. The authorization of this capability is identical to the canChangeUserRole capability, except that the from_role_index for the unbanned user MUST be 1, and the role_name of role 1 MUST be "banned".

8.2. Adjust metadata

The holder of each of the following capabilities is authorized to update the Room metadata defined in [I-D.ietf-mimi-protocol], changing the relevant field:

* canChangeRoomName

* canChangeRoomDescription

* canChangeRoomAvatar

* canChangeRoomSubject

* canChangeRoomMood

8.3. Message Capabilities

The capabilities below refer to functionality related to the instant messages, for example sent using the MIMI content format [I-D.ietf-mimi-content].

* canSendMessage - the holder can send instant messages to the room. Setting specific message fields may require additional capabilities.

* canReceiveMessage - the holder can receive instant messages from the room.

* canCopyMessage - the holder can copy content from a received instant message.

* canReportAbuse - the holder can report a franked instant message as abusive.

* canReplyToMessage - the holder can send a message replying to another message.

* canReactToMessage - the holder can send a reaction, replying to another message, and using the "reaction" disposition.

* canEditReaction - the holder can replace its own previous reaction with another reaction

* canDeleteOwnReaction - the holder can retract (unlike) its own previous reaction.

* canDeleteOtherReaction - the holder can delete the reaction of another user’s previous reaction

* canEditOwnMessage - the holder can edit the content of one of its own previously sent messages

* canDeleteOwnMessage - the holder can retract one of its own previously sent messages

* canDeleteOtherMessage - the holder can retract messages for other users.

* canStartTopic - the holder can set the topic for a message

* canReplyInTopic - the holder can send a message replying to a previous message, using the same topic as the original sender.

* canEditOwnTopic - the holder can change the topic of a previously sent message

* canEditOtherTopic - the holder can change the topic of a message previously sent by another user.

* canSendLink - the holder can send an inline link

* canSendLinkPreview - the holder can send an inline link with an associated preview.

* canFollowLink - the holder can open a sent inline link.

* canCopyLink - the holder can copy the URL of a sent inline link.

The Hub can enforce whether a member can send a message. It can also withhold fanout of application messages to clients of a user. The other capabilities in this section can only be enforced by other clients.

8.4. Asset Capabilities

* canUploadAttachment - the holder can upload a file with the "attachent" disposition.

* canDownloadAttachment - the holder can download a file with the "attachent" disposition.

* canUploadImage - the holder can upload a file with the media type of "image" and the disposition of "render"

* canDownloadImage - the holder can download a file with the media type of "image" and the disposition of "render"

* canUploadVideo - the holder can upload a file with the media type of "video" and the disposition of "render"

* canDownloadVideo - the holder can download a file with the media type of "video" and the disposition of "render"

* canUploadAudio - the holder can upload a file with the media type of "audio" and the disposition of "render"

* canDownloadAudio - the holder can download a file with the media type of "audio" and the disposition of "render"

8.5. Real-time media

The MIMI Working has not yet defined requirements for real-time media, however the capabilities below are widely representative of the permissions that would be required.

* canStartCall - the holder can initiate a new real-time call/ conference

* canJoinCall - the holder can join an existing real-time call/ conference

* canSendAudio - the holder is authorized to contribute audio in a call/conference.

* canReceiveAudio - the holder is authorized to receive audio in a call/conference.

* canSendVideo - the holder is authorized to contribute video in a call/conference.

* canReceiveVideo - the holder is authorized to receive video in a call/conference.

* canShareScreen - the holder is authorized to contribute screen sharing in a call/conference

* canViewSharedScreen - the holder is authorized to receive screen sharing in a call/conference

8.6. Disruptive Policy Changes

* canChangeRoomMembershipStyle - the holder is authorized to modify the base room membership style.

* canChangeRoleDefinitions - the holder is authorized to make changes to the definitions of the Roles component.

* canChangePreauthorizedUserList - the holder is authorized to make changes to the Preauthorized Users component.

* canDestroyRoom - the holder is authorized to completely destroy the room.

* canSendMLSReinitProposal - the holder is authorized to send an MLS ReInit proposal.

8.7. Reserved Capabilities

The following capability names are reserved for possible future use

* canCreateJoinCode

* canDeleteJoinCode

* canKnock

* canAcceptKnock

* canCreateSubgroup

* canSendDirectMessage

* canTargetMessage

* canChangeOwnName

* canChangeOwnPresence

* canChangeOwnMood

* canChangeOwnAvatar

* canCreateRoom

* canChangeMlsOperationalPolicies

* canChangeOtherPolicyAttribute

* MLS specific

  - update - update policy

  - PSK - psk policy

  - external proposal - general operational policy rules

- external commit - general operational policy rules

9. Security Considerations

This entire document focuses on authorization policy. TODO More Security

10. IANA Considerations

RFC EDITOR: Please replace XXXX throughout with the RFC number assigned to this document.

10.1. New MLS application components

This document registers the following MLS Component Types per Section 7.5 of [I-D.ietf-mls-extensions].

10.1.1. mls_operational_policy MLS Component Type

* Value: TBD0 (suggested value 0x0024)
* Name: mls_operational_policy
* Where: GC
* Recommended: Y
* Reference: Section 7 of RFCXXXX

10.1.2. roles_list MLS Component Type

* Value: TBD1 (suggested value 0x0025)
* Name: roles_list
* Where: GC
* Recommended: Y
* Reference: Section 3 of RFCXXXX

10.1.3. preauth_list MLS Component Type

* Value: TBD2 (suggested value 0x0026)
* Name: preauth_list
* Where: GC

* Recommended: Y
* Reference: Section 4 of RFCXXXX

10.1.4. base_room_policy MLS Component Type

* Value: TBD3 (suggested value 0x0027)
* Name: base_room_policy
* Where: GC
* Recommended: Y
* Reference: Section 4 of RFCXXXX

10.1.5. status_notification_policy MLS Component Type

* Value: TBD4 (suggested value 0x0028)
* Name: status_notification_policy
* Where: GC
* Recommended: Y
* Reference: Section 6.1 of RFCXXXX

10.1.6. join_link_policy MLS Component Type

* Value: TBD5 (suggested value 0x0029)
* Name: join_link_policy
* Where: GC
* Recommended: Y
* Reference: Section 6.2 of RFCXXXX

10.1.7. join_links MLS Component Type

* Value: TBD6 (suggested value 0x002A)
* Name: join_links
* Where: GC

* Recommended: Y
* Reference: Section 6.2 of RFCXXXX

10.1.8. link_preview_policy MLS Component Type

* Value: TBD7 (suggested value 0x002B)
* Name: link_preview_policy
* Where: GC
* Recommended: Y
* Reference: Section 6.3 of RFCXXXX

10.1.9. asset_policy MLS Component Type

* Value: TBD8 (suggested value 0x002C)
* Name: asset_policy
* Where: GC
* Recommended: Y
* Reference: Section 6.4 of RFCXXXX

10.1.10. logging_policy MLS Component Type

* Value: TBD9 (suggested value 0x002D)
* Name: logging_policy
* Where: GC
* Recommended: Y
* Reference: Section 6.5 of RFCXXXX

10.1.11. chat_history_policy MLS Component Type

* Value: TBD10 (suggested value 0x002E)
* Name: chat_history_policy
* Where: GC

* Recommended: Y

* Reference: Section 6.6 of RFCXXXX

10.1.12. bot_policy MLS Component Type

* Value: TBD11 (suggested value 0x002F)

* Name: bot_policy

* Where: GC

* Recommended: Y

* Reference: Section 6.7 of RFCXXXX

10.1.13. Message expiration policy component

* Value: TBD12 (suggested value 0x0030)

* Name: message_expiration_policy

* Where: GC

* Recommended: Y

* Reference: Section 6.8 of RFCXXXX

10.2. New MIMI Role Capabilities registry

This document requests the creation of a new IANA "MIMI Role Capabilities" registry. The registry should be under the heading of "More Instant Messaging Interoperability (MIMI)". Assignments to this registry in the range 0x0000 to 0xF000 are via Specification Required policy [RFC8126] using the MIMI Designated Experts. Assignments in the range 0xF000 to 0xFFFF are for private use.

Template:

* Value: The numeric value of the role capability

* Name: The name of the role capability

* Reference: The document where this role capability is defined

Value	Name	Reference
0x0000	canAddParticipant	RFCXXXX
0x0001	canRemoveParticipant	RFCXXXX
0x0002	canAddOwnClient	RFCXXXX
0x0003	canRemoveOwnClient	RFCXXXX
0x0004	canOpenJoin	RFCXXXX
0x0005	canJoinIfPreauthorized	RFCXXXX
0x0006	canRemoveSelf	RFCXXXX
0x0007	canCreateJoinCode (reserved)	RFCXXXX
0x0008	canDeleteJoinCode (reserved)	RFCXXXX
0x0009	canUseJoinCode	RFCXXXX
0x000a	canBan	RFCXXXX
0x000b	canUnBan	RFCXXXX
0x000c	canKick	RFCXXXX
0x000d	canKnock (reserved)	RFCXXXX
0x000e	canAcceptKnock (reserved)	RFCXXXX
0x000f	canChangeUserRole	RFCXXXX
0x0010	canChangeOwnRole	RFCXXXX
0x0011	canCreateSubgroup (reserved)	RFCXXXX
0x0100	canSendMessage	RFCXXXX
0x0101	canReceiveMessage	RFCXXXX
0x0102	canCopyMessage	RFCXXXX
0x0103	canReportAbuse	RFCXXXX
0x0104	canReplyToMessage	RFCXXXX

0x0105	canReactToMessage	RFCXXXX
0x0106	canEditReaction	RFCXXXX
0x0107	canDeleteOwnReaction	RFCXXXX
0x0108	canDeleteOtherReaction	RFCXXXX
0x0109	canEditOwnMessage	RFCXXXX
0x010a	canDeleteOwnMessage	RFCXXXX
0x010b	canDeleteOtherMessage	RFCXXXX
0x010c	canStartTopic	RFCXXXX
0x010d	canReplyInTopic	RFCXXXX
0x010e	canEditOwnTopic	RFCXXXX
0x010f	canEditOtherTopic	RFCXXXX
0x0110	canSendDirectMessage (reserved)	RFCXXXX
0x0111	canTargetMessage (reserved)	RFCXXXX
0x0200	canUploadImage	RFCXXXX
0x0201	canUploadAudio	RFCXXXX
0x0202	canUploadVideo	RFCXXXX
0x0203	canUploadAttachment	RFCXXXX
0x0204	canDownloadImage	RFCXXXX
0x0205	canDownloadAudio	RFCXXXX
0x0206	canDownloadVideo	RFCXXXX
0x0207	canDownloadAttachment	RFCXXXX
0x0208	canSendLink	RFCXXXX
0x0209	canSendLinkPreview	RFCXXXX
0x020a	canFollowLink	RFCXXXX

0x020b	canCopyLink	RFCXXXX
0x0300	canChangeRoomName	RFCXXXX
0x0301	canChangeRoomDescription	RFCXXXX
0x0302	canChangeRoomAvatar	RFCXXXX
0x0303	canChangeRoomSubject	RFCXXXX
0x0304	canChangeRoomMood	RFCXXXX
0x0380	canChangeOwnName (reserved)	RFCXXXX
0x0381	canChangeOwnPresence (reserved)	RFCXXXX
0x0382	canChangeOwnMood (reserved)	RFCXXXX
0x0383	canChangeOwnAvatar (reserved)	RFCXXXX
0x0400	canStartCall	RFCXXXX
0x0401	canJoinCall	RFCXXXX
0x0402	canSendAudio	RFCXXXX
0x0403	canReceiveAudio	RFCXXXX
0x0404	canSendVideo	RFCXXXX
0x0405	canReceiveVideo	RFCXXXX
0x0406	canShareScreen	RFCXXXX
0x0407	canViewSharedScreen	RFCXXXX
0x0500	canCreateRoom (reserved)	RFCXXXX
0x0501	canDestroyRoom	RFCXXXX
0x0502	canChangeRoomMembershipStyle	RFCXXXX
0x0503	canChangeRoleDefinitions	RFCXXXX
0x0504	canChangePreauthorizedUserList	RFCXXXX
0x0505	canChangeOtherPolicyAttribute (reserved)	RFCXXXX

0x0600	canChangeMlsOperationalPolicies (reserved)	RFCXXXX
0x0601	canSendMLSReinitProposal	RFCXXXX
0x0602	canSendMLSUpdateProposal (reserved)	RFCXXXX
0x0603	canSendMLSPSKProposal (reserved)	RFCXXXX
0x0604	canSendMLSExternalProposal (reserved)	RFCXXXX
0x0605	canSendMLSExternalCommit (reserved)	RFCXXXX

Table 1

11. References

11.1. Normative References

[I-D.ietf-mimi-arch]
Barnes, R., "An Architecture for More Instant Messaging Interoperability (MIMI)", Work in Progress, Internet-Draft, draft-ietf-mimi-arch-02, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-mimi-arch-02>.

[I-D.ietf-mimi-content]
Mahy, R., "More Instant Messaging Interoperability (MIMI) message content", Work in Progress, Internet-Draft, draft-ietf-mimi-content-07, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-mimi-content-07>.

[I-D.ietf-mimi-protocol]
Barnes, R., Hodgson, M., Kohbrok, K., Mahy, R., Ralston, T., and R. Robert, "More Instant Messaging Interoperability (MIMI) using HTTPS and MLS", Work in Progress, Internet-Draft, draft-ietf-mimi-protocol-05, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-mimi-protocol-05>.

[I-D.ietf-mls-extensions]
Robert, R., "The Messaging Layer Security (MLS) Extensions", Work in Progress, Internet-Draft, draft-ietf-mls-extensions-08, 21 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-mls-extensions-08>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC9420] Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., and K. Cohn-Gordon, "The Messaging Layer Security (MLS) Protocol", RFC 9420, DOI 10.17487/RFC9420, July 2023, <https://www.rfc-editor.org/rfc/rfc9420>.

11.2. Informative References

[I-D.mahy-mimi-message-status] Mahy, R., "A Message Status format for the More Instant Messaging Interoperability (MIMI) content format", Work in Progress, Internet-Draft, draft-mahy-mimi-message-status-00, 4 July 2025, <https://datatracker.ietf.org/doc/html/draft-mahy-mimi-message-status-00>.

[RFC9458] Thomson, M. and C. A. Wood, "Oblivious HTTP", RFC 9458, DOI 10.17487/RFC9458, January 2024, <https://www.rfc-editor.org/rfc/rfc9458>.

[RFC9750] Beurdouche, B., Rescorla, E., Omara, E., Inguva, S., and A. Duric, "The Messaging Layer Security (MLS) Architecture", RFC 9750, DOI 10.17487/RFC9750, April 2025, <https://www.rfc-editor.org/rfc/rfc9750>.

Appendix A. Role examples

A.1. Cooperatively administered room

This is an example set of role policies, which is suitable for friends and family rooms and small groups of peers in a workgroup or club.

* no_role
  - role_index = 0

- no capabilities
- constraints
  - minimum_participants_constraint = 0
  - maximum_participants_constraint = null
  - minimum_active_participants_constraint = 0
  - maximum_active_participants_constraint = 0
  - authorized_role_changes = []

* banned
  - role_index = 1
  - no capabilities
  - constraints
    - minimum_participants_constraint = 0
    - maximum_participants_constraint = null
    - minimum_active_participants_constraint = 0
    - maximum_active_participants_constraint = 0
    - authorized_role_changes = []
* ordinary_user
  - role_index = 2
  - authorized capabilities
    - canAddParticipant
    - canRemoveParticipant
    - canAddOwnClient
    - canRemoveOwnClient
    - canRemoveSelf

- canSendMessage
- canReceiveMessage
- canCopyMessage
- canReportAbuse
- canReplyToMessage
- canReactToMessage
- canDeleteOwnReaction
- canEditOwnMessage
- canDeleteOwnMessage
- canStartTopic
- canReplyInTopic
- canEditOwnTopic
- canUploadImage
- canUploadVideo
- canUploadAudio
- canUploadAttachment
- canDownloadImage
- canDownloadVideo
- canDownloadAudio
- canDownloadAttachment
- canSendLink
- canSendLinkPreview
- canFollowLink
- canCopyLink

- o canChangeRoomName
- o canChangeRoomAvatar
- o canChangeRoomSubject
- o canChangeRoomMood
- o canChangeOwnName
- o canChangeOwnPresence
- o canChangeOwnMood
- o canChangeOwnAvatar

- constraints
  - o minimum_participants_constraint = 0
  - o maximum_participants_constraint = null
  - o minimum_active_participants_constraint = 0
  - o maximum_active_participants_constraint = null
  - o authorized_role_changes = [(0,[2]), (2,[0])]

* group_admin
  - role_index = 3
  - authorized capabilities
    - o (include all the capabilities authorized for an ordinary_user)
    - o canBan
    - o canUnBan
    - o canKick
    - o canRevokeVoice
    - o canGrantVoice
    - o canChangeUserRole

        - canDeleteOtherMessage
        - canEditOtherTopic
        - canChangeRoomDescription
    - constraints
        - minimum_participants_constraint = 1
        - maximum_participants_constraint = null
        - minimum_active_participants_constraint = 0
        - maximum_active_participants_constraint = null
        - authorized_role_changes = [(0,[1,2,3]), (1,[0,2,3]), (2,[0,1,3]), (3,[0,1,2])]

* super_admin
    - role_index = 4
    - authorized capabilities
        - (include all the capabilities authorized for a group_admin)
        - canChangeRoomMembershipStyle
        - canChangePreauthorizedUserList
        - canDestroyRoom
    - constraints
        - minimum_participants_constraint = 0
        - maximum_participants_constraint = null
        - minimum_active_participants_constraint = 0
        - maximum_active_participants_constraint = null
        - authorized_role_changes = [(0,[1,2,3,4]), (1,[0,2,3,4]), (2,[0,1,3,4]), (3,[0,1,2,4]), (4,[0,1,2,3])]

* policy_enforcer

- role_index = 5
- capabilities
  - (does not include any other capabilities)
  - canRemoveParticipant
  - canChangeUserRole
  - canBan
  - canUnban
  - canChangeRoleDefinitions
  - canChangePreauthorizedUserList
  - canChangeMlsOperationalPolicies
  - canDestroyRoom
  - canSendMLSReinitProposal
- constraints
  - minimum_participants_constraint = 1
  - maximum_participants_constraint = 2
  - minimum_active_participants_constraint = 0
  - maximum_active_participants_constraint = 0
  - authorized_role_changes = [(0,[1]), (1,[0]), (2,[0,1]), (3,[0,1]), (4,[0,1])]
- Notes: can remove a banned user from the list (cleanup) but not restore them

A.2. Strictly administered room

This is an example set of role policies, which is suitable for friends and family rooms and small groups of peers in a workgroup or club.

* no_role

- role_index = 0
- authorized capabilities
  - canUseJoinCode
- constraints
  - minimum_participants_constraint = 0
  - maximum_participants_constraint = null
  - minimum_active_participants_constraint = 0
  - maximum_active_participants_constraint = 0
  - authorized_role_changes = [(0,[2])]

* banned
  - role_index = 1
  - no capabilities
  - constraints
    - minimum_participants_constraint = 0
    - maximum_participants_constraint = null
    - minimum_active_participants_constraint = 0
    - maximum_active_participants_constraint = 0
    - authorized_role_changes = []
* ordinary_user
  - role_index = 2
  - authorized capabilities
    - canAddOwnClient
    - canJoinIfPreauthorized
    - canRemoveOwnClient

- canRemoveSelf
- canChangeOwnRole
- canSendMessage
- canReceiveMessage
- canCopyMessage
- canReportAbuse
- canReactToMessage
- canDeleteOwnReaction
- canEditOwnMessage
- canDeleteOwnMessage
- canStartTopic
- canReplyInTopic
- canUploadImage
- canUploadVideo
- canUploadAudio
- canUploadAttachment
- canDownloadImage
- canDownloadVideo
- canDownloadAudio
- canDownloadAttachment
- canSendLink
- canSendLinkPreview
- canFollowLink
- canCopyLink

        - canChangeOwnName
        - canChangeOwnPresence
        - canChangeOwnMood
        - canChangeOwnAvatar
    - constraints
        - minimum_participants_constraint = 0
        - maximum_participants_constraint = null
        - minimum_active_participants_constraint = 0
        - maximum_active_participants_constraint = null
        - authorized_role_changes = [(0,[2]), (2,[0])]
* group_admin
    - role_index = 3
    - authorized capabilities
        - (include all the capabilities authorized for an ordinary_user)
        - canAddParticipant
        - canRemoveParticipant
        - canBan
        - canUnBan
        - canKick
        - canChangeUserRole
        - canCreateJoinCode - reserved for future use
        - canDeleteJoinCode - reserved for future use
        - canDeleteOtherReaction
        - canDeleteOtherMessage

- o canEditOwnTopic
- o canEditOtherTopic
- o canChangeRoomDescription
- o canChangeRoomName
- o canChangeRoomAvatar
- o canChangeRoomSubject
- o canChangeRoomMood

- constraints
  - o minimum_participants_constraint = 1
  - o maximum_participants_constraint = null
  - o minimum_active_participants_constraint = 0
  - o maximum_active_participants_constraint = null
  - o authorized_role_changes = [(0,[1,2,3]), (1,[0,2,3]), (2,[0,1,3]), (3,[0,1,2])]

* super_admin
  - role_index = 4
  - authorized capabilities
    - o (include all the capabilities authorized for a group_admin)
    - o canChangeRoomMembershipStyle
    - o canChangeRoleDefinitions
    - o canChangePreauthorizedUserList
    - o canDestroyRoom
    - o canSendMLSReinitProposal
  - constraints
    - o minimum_participants_constraint = 0

- maximum_participants_constraint = null
- minimum_active_participants_constraint = 0
- maximum_active_participants_constraint = null
- authorized_role_changes = [(0,[1,2,3,4]), (1,[0,2,3,4]), (2,[0,1,3,4]), (3,[0,1,2,4]), (4,[0,1,2,3])]

* policy_enforcer
  - role_index = 5
  - capabilities
    - (does not include any other capabilities)
    - canRemoveParticipant
    - canChangeUserRole
    - canBan
    - canUnban
    - canChangeRoleDefinitions
    - canChangePreauthorizedUserList
    - canChangeMlsOperationalPolicies
    - canDestroyRoom
    - canSendMLSReinitProposal
  - constraints
    - minimum_participants_constraint = 1
    - maximum_participants_constraint = 2
    - minimum_active_participants_constraint = 0
    - maximum_active_participants_constraint = 0
    - authorized_role_changes = [(0,[1]), (1,[0]), (2,[0,1]), (3,[0,1]), (4,[0,1])]

- Notes: can remove a banned user from the list (cleanup) but not restore them

A.3. Moderated room

* no_role
  - role_index = 0
  - authorized capabilities
    - canUseJoinCode
  - constraints
    - minimum_participants_constraint = 0
    - maximum_participants_constraint = null
    - minimum_active_participants_constraint = 0
    - maximum_active_participants_constraint = null
    - authorized_role_changes = [(0,[2,3])]
* banned
  - role_index = 1
  - no capabilities
  - constraints
    - minimum_participants_constraint = 0
    - maximum_participants_constraint = null
    - minimum_active_participants_constraint = 0
    - maximum_active_participants_constraint = 0
    - authorized_role_changes = []
* guest
  - role_index = 2
  - authorized capabilities

- canRemoveSelf
- canReceiveMessage
- canCopyMessage
- canReactToMessage
- canDeleteOwnReaction
- canFollowLink
- canCopyLink
- canDownloadImage
- canDownloadVideo
- canDownloadAudio

- constraints
  - minimum_participants_constraint = 0
  - maximum_participants_constraint = null
  - minimum_active_participants_constraint = 0
  - maximum_active_participants_constraint = null
  - authorized_role_changes = [(0,[2]), (2,[0])]

* attendee
  - role_index = 3
  - authorized capabilities
    - (include all the capabilities authorized for a guest)
    - canAddOwnClient
    - canJoinIfPreauthorized
    - canRemoveOwnClient
    - canChangeOwnRole

        - canReportAbuse
        - canReplyInTopic
        - canDownloadAttachment
        - canChangeOwnName
        - canChangeOwnPresence
        - canChangeOwnAvatar
    - constraints
        - minimum_participants_constraint = 0
        - maximum_participants_constraint = null
        - minimum_active_participants_constraint = 0
        - maximum_active_participants_constraint = null
        - authorized_role_changes = [(0,[3]), (3,[0])]
* speaker
    - role_index = 4
    - authorized capabilities
        - (include all the capabilities authorized for a speaker)
        - canSendMessage
        - canEditOwnMessage
        - canDeleteOwnMessage
        - canStartTopic
        - canUploadImage
        - canUploadVideo
        - canUploadAudio
        - canUploadAttachment

- canSendLink
- canSendLinkPreview

- constraints
  - minimum_participants_constraint = 0
  - maximum_participants_constraint = null
  - minimum_active_participants_constraint = 0
  - maximum_active_participants_constraint = null
  - authorized_role_changes = [(0,[4]), (4,[0])]

* moderator
  - role_index = 5
  - authorized capabilities
    - (include all the capabilities authorized for an ordinary_user)
    - canAddParticipant
    - canRemoveParticipant
    - canBan
    - canUnBan
    - canKick
    - canChangeUserRole
    - canCreateJoinCode - reserved for future use
    - canDeleteJoinCode - reserved for future use
    - canDeleteOtherReaction
    - canDeleteOtherMessage
    - canEditOwnTopic
    - canEditOtherTopic

        - canChangeRoomName
        - canChangeRoomAvatar
        - canChangeRoomSubject
        - canChangeRoomMood
    - constraints
        - minimum_participants_constraint = 1
        - maximum_participants_constraint = null
        - minimum_active_participants_constraint = 0
        - maximum_active_participants_constraint = null
        - authorized_role_changes = [(0,[1,2,3,4,5]), (1,[0,2,3,4,5]), (2,[0,1,3,4,5]), (3,[0,1,2,4,5]), (4,[0,1,2,3,5]), (5,[0,1,2,3,4])]

* super_admin
    - role_index = 6
    - authorized capabilities
        - (include all the capabilities authorized for a moderator)
        - canChangeRoomDescription
        - canChangeRoomMembershipStyle
        - canChangeRoleDefinitions
        - canChangePreauthorizedUserList
        - canDestroyRoom
        - canSendMLSReinitProposal
    - constraints
        - minimum_participants_constraint = 0
        - maximum_participants_constraint = null

- minimum_active_participants_constraint = 0
- maximum_active_participants_constraint = null
- authorized_role_changes = [(0,[1,2,3,4,5,6]), (1,[0,2,3,4,5,6]), (2,[0,1,3,4,5,6]), (3,[0,1,2,4,5,6]), (4,[0,1,2,3,5,6]), (5,[0,1,2,3,4,6]), (6,[0,1,2,3,4,5])]

* policy_enforcer
  - role_index = 7
  - capabilities
    - (does not include any other capabilities)
    - canRemoveParticipant
    - canChangeUserRole
    - canBan
    - canUnban
    - canChangeRoleDefinitions
    - canChangePreauthorizedUserList
    - canChangeMlsOperationalPolicies
    - canDestroyRoom
    - canSendMLSReinitProposal
  - constraints
    - minimum_participants_constraint = 1
    - maximum_participants_constraint = 2
    - minimum_active_participants_constraint = 0
    - maximum_active_participants_constraint = 0
    - authorized_role_changes = [(0,[1]), (1,[0]), (2,[0,1]), (3,[0,1]), (4,[0,1]), (5, [0,1]), (6, [0,1])]

- Notes: can remove a banned user from the list (cleanup) but not restore them

A.4. Multi-organization administered room

In this example room policy, Alice from organization A is a super admin. There are per organization user and admin roles for orgs A, B, and C. Organizational admins can only move users to and from their org user role, their org admin role, the no_role; and can ban (but not unban) their own org users. The non-host orgs do not have the canChangeOwnRole and canJoinIfPreauthorized, and are limited to 3 admins per org.

* no_role
  - role_index = 0
  - no capabilities
  - constraints
    - minimum_participants_constraint = 0
    - maximum_participants_constraint = null
    - minimum_active_participants_constraint = 0
    - maximum_active_participants_constraint = 0
    - authorized_role_changes = []
* banned
  - role_index = 1
  - no capabilities
  - constraints
    - minimum_participants_constraint = 0
    - maximum_participants_constraint = null
    - minimum_active_participants_constraint = 0
    - maximum_active_participants_constraint = 0
    - authorized_role_changes = []

* org_a_user
  - role_index = 2
  - authorized capabilities
    - (all capabilities of org_b_user)
    - canChangeOwnRole
    - canJoinIfPreauthorized
  - constraints
    - minimum_participants_constraint = 0
    - maximum_participants_constraint = null
    - minimum_active_participants_constraint = 0
    - maximum_active_participants_constraint = null
    - authorized_role_changes = [(0,[2]), (2,[0])]
* org_b_user
  - role_index = 3
  - authorized capabilities
    - canRemoveSelf
    - canAddOwnClient
    - canRemoveOwnClient
    - canSendMessage
    - canReceiveMessage
    - canCopyMessage
    - canReportAbuse
    - canReplyInTopic
    - canReactToMessage

        - canDeleteOwnReaction
        - canEditOwnMessage
        - canSendLink
        - canSendLinkPreview
        - canFollowLink
        - canCopyLink
        - canDownloadImage
        - canDownloadVideo
        - canDownloadAudio
        - canDownloadAttachment
        - canChangeOwnName
        - canChangeOwnPresence
        - canChangeOwnAvatar
    - constraints
        - minimum_participants_constraint = 0
        - maximum_participants_constraint = null
        - minimum_active_participants_constraint = 0
        - maximum_active_participants_constraint = null
        - authorized_role_changes = [(0,[3]), (3,[0])]
* org_c_user
    - role_index = 4
    - authorized capabilities
        - (same capabilities as org_b_user)
    - constraints

        - minimum_participants_constraint = 0
        - maximum_participants_constraint = null
        - minimum_active_participants_constraint = 0
        - maximum_active_participants_constraint = null
        - authorized_role_changes = [(0,[4]), (4,[0])]

    * org_a_admin
        - role_index = 5
        - authorized capabilities
            - (all capabilities of org_b_admin)
            - canChangeOwnRole
            - canJoinIfPreauthorized
        - constraints
            - minimum_participants_constraint = 0
            - maximum_participants_constraint = null
            - minimum_active_participants_constraint = 0
            - maximum_active_participants_constraint = null
            - authorized_role_changes = [(0,[2,5]), (2,[0,1,5]), (5,[0,1,2])]

    * org_b_admin
        - role_index = 6
        - authorized capabilities
            - (all capabilities of org_b_user)
            - canDeleteOwnMessage
            - canStartTopic
            - canUploadImage

- o canUploadVideo
- o canUploadAudio
- o canUploadAttachment
- o canAddParticipant
- o canRemoveParticipant
- o canBan
- o canKick
- o canChangeUserRole

- constraints
  - o minimum_participants_constraint = 1
  - o maximum_participants_constraint = 3
  - o minimum_active_participants_constraint = 1
  - o maximum_active_participants_constraint = null
  - o authorized_role_changes = [(0,[3,6]), (3,[0,1,6]), (6,[0,1,3])]

* org_c_admin
  - role_index = 7
  - authorized capabilities
    - o (all capabilities of org_b_admin)
  - constraints
    - o minimum_participants_constraint = 1
    - o maximum_participants_constraint = 3
    - o minimum_active_participants_constraint = 1
    - o maximum_active_participants_constraint = null

- authorized_role_changes = [(0,[4,7]), (4,[0,1,7]), (7,[0,1,4])]

* super_admin
  - role_index = 8
  - authorized capabilities
    - (include all the capabilities authorized for org_a_admin)
    - canUnBan
    - canDeleteOtherReaction
    - canDeleteOtherMessage
    - canEditOwnTopic
    - canEditOtherTopic
    - canChangeRoomDescription
    - canChangeRoomName
    - canChangeRoomAvatar
    - canChangeRoomSubject
    - canChangeRoomMood
    - canChangeRoomMembershipStyle
    - canChangeRoleDefinitions
    - canChangePreauthorizedUserList
    - canDestroyRoom
    - canSendMLSReinitProposal
  - constraints
    - minimum_participants_constraint = 1
    - maximum_participants_constraint = null
    - minimum_active_participants_constraint = 1

- o maximum_active_participants_constraint = null
- o authorized_role_changes = [(0,[1,2,3,4,5,6,7,8]), (1,[0,2,3,4,5,6,7,8]), (2,[0,1,5,8]), (3,[0,1,6]), (4,[0,1,7]), (5,[0,1,2,8]), (6,[0,1,3]), (7,[0,1,4]), (8,[0,1,2,5])]

* policy_enforcer
  - role_index = 9
  - capabilities
    - o (does not include any other capabilities)
    - o canRemoveParticipant
    - o canChangeUserRole
    - o canBan
    - o canUnban
    - o canChangeRoleDefinitions
    - o canChangePreauthorizedUserList
    - o canChangeMlsOperationalPolicies
    - o canDestroyRoom
    - o canSendMLSReinitProposal
  - constraints
    - o minimum_participants_constraint = 1
    - o maximum_participants_constraint = 2
    - o minimum_active_participants_constraint = 0
    - o maximum_active_participants_constraint = 0
    - o authorized_role_changes = [(0,[1]), (1,[0]), (3,[0,1]), (4,[0,1]), (5,[0,1]), (6,[0,1]), (7,[0,1]), (8,[0,1])]
  - Notes: can remove a banned user from the list (cleanup) but not restore them

Appendix B. Complete TLS Presentation Language Syntax

```
   enum {
     false(0),
     true(1)
   } bool;

   struct {
     /* a valid Uniform Resource Identifier (URI) */
     opaque uri<V>;
   } Uri;

   struct {
     opaque domain<V>;
   } DomainName;

   enum {
     optional(0),
     required(1),
     forbidden(2)
   } Optionality;


   /* See MIMI Capability Types IANA registry */
   uint16 CapabilityType;

   struct {
      uint32 from_role_index;
      uint32 target_role_indexes<V>;
   } SingleSourceRoleChangeTargets;

   struct {
     uint32 role_index;
     opaque role_name<V>;
     opaque role_description<V>;
     CapabilityType role_capabilities<V>;
     uint32 minimum_participants_constraint;
     optional uint32 maximum_participants_constraint;
     uint32 minimum_active_participants_constraint;
     optional uint32 maximum_active_participants_constraint;
     SingleSourceRoleChangeTargets authorized_role_changes<V>;
   } Role;

   struct {
     Role roles<V>;
   } RoleData;

   RoleData roles_list;
```

```
   RoleData RoleUpdate;


   struct {
     /* MLS Credential Type of the "claim"  */
     CredentialType credential_type;
     /* the binary representation of an X.509 OID, a JWT claim name  */
     /* string, or the CBOR map claim key in a CWT (an int or tstr)  */
     opaque id<V>;
   } ClaimId;

   struct {
     ClaimId claim_id;
     opaque claim_value<V>;
   } Claim;

   struct {
     /* when all claims in the claimset are satisfied, the claimset */
     */ is satisfied */
     Claim claimset<V>;
     Role target_role;
   } PreAuthRoleEntry;

   struct {
     PreAuthRoleEntry preauthorized_entries<V>;
   } PreAuthData;

   PreAuthData preauth_list;
   PreAuthData PreAuthUpdate;


   struct {
       bool fixed_membership;
       bool parent_dependant;
       Uri parent_room<V>;
       bool multi_device;
       optional uint32 max_clients;
       optional uint32 max_users;
       bool pseudonyms_allowed;
       bool persistent_room;
       bool discoverable;
       Component policy_component_ids<V>;
   } BaseRoomPolicy;

   BaseRoomPolicy BaseRoomData;
   BaseRoomPolicy BaseRoomUpdate;
```

```
   struct {
     Optionality delivery_notifications;
     Optionality read_receipts;
   } StatusNotificationPolicy;

   StatusNotificationPolicy StatusNotificationPolicyData;
   StatusNotificationPolicy StatusNotificationPolicyUpdate;


   struct {
     bool on_request;
     Uri join_link;
     bool multiuser;
     uint32 expiration;
   } JoinLinkPolicy;

   JoinLinkPolicy JoinLinkPolicyData;
   JoinLinkPolicy JoinLinkPolicyUpdate;

   struct {
     opaque join_link;
   } JoinLink;

   JoinLink JoinLinksData<V>;

   struct {
     uint32 removedIndices<V>;
     JoinLink added_links<V>;
   } JoinLinksUpdate;


   struct {
     Optionality autodetect_hyperlinks_in_text;
     Optionality send_link_previews;
     Optionality automatic_link_previews;
     Optionality link_preview_proxy_use;
     select (link_preview_proxy_use) {
       case mandatory:
         Uri link_preview_proxy<V>;
       case optional:
         Uri link_preview_proxy<V>;
       case forbidden:
         struct {};
     }
   } LinkPreviewPolicy;

   LinkPreviewPolicy LinkPreviewPolicyData;
   LinkPreviewPolicy LinkPreviewPolicyUpdate;
```

```
   enum {
     unspecified(0),
     localProvider(1),
     hub(2),
     (255)
   } AssetUploadLocation;

   struct {
     DomainName provider;
     DomainName asset_upload_destinations<V>;
   } ProviderAssetUploadDomains;

   enum {
     direct(0),
     hubProxy(1),
     ohttp(2),
     (255)
   } DownloadPrivacyType;

   struct {
     DownloadPrivacyType allowed_download_types<V>;
     DownloadPrivacyType forbidden_download_types<V>;
     DownloadPrivacyType default_download_type;
   } DownloadPrivacy;

   struct {
     AssetUploadLocation asset_upload_location;
     ProviderAssetUploadDomains upload_domains<V>;
     DownloadPrivacy download_privacy;
     uint64 max_image;
     uint64 max_audio;
     uint64 max_video;
     uint64 max_attachment;
     MediaType forbidden_media_types<V>;
     optional<MediaType> permitted_media_types<V>;
   } AssetPolicy;

   AssetPolicy AssetPolicyData;
   AssetPolicy AssetPolicyUpdate;


   struct {
     Optionality logging;
     select (logging) {
       case mandatory:
         Uri logging_clients<V>;
         Uri machine_readable_policy;
         Uri human_readable_policy;
```

```
      case optional:
        Uri logging_clients<V>;
        Uri machine_readable_policy;
        Uri human_readable_policy;
      case forbidden:
        struct {};
    }
  } LoggingPolicy;

  LoggingPolicy LoggingPolicyData;
  LoggingPolicy LoggingPolicyUpdate;


  struct {
    Optionality history_sharing;
    select (history_sharing) {
      case mandatory:
        uint32 roles_that_can_share<V>;
        bool automatically_share;
        uint32 max_time_period;
      case optional:
        uint32 roles_that_can_share<V>;
        bool automatically_share;
        uint32 max_time_period;
      case forbidden:
        struct {};
    }
  } HistoryPolicy;

  HistoryPolicy HistoryPolicyData;
  HistoryPolicy HistoryPolicyUpdate;


  struct {
    opaque name<V>;
    opaque description<V>;
    Uri homepage;
    bool local_client_bot;
    uint32 bot_role_index;
    bool can_target_message_in_group;
    bool per_user_content;
  } Bot;

  struct {
    Bot allowed_bots<V>;
  } BotPolicy;

  BotPolicy BotPolicyData;
```

```
   BotPolicy BotPolicyUpdate;


   struct {
     Optionality expiring_messages;
     select (expiring_messages) {
       case mandatory:
         uint32 min_expiration_duration;
         uint32 max_expiration_duration;
         optional uint32 default_expiration_duration;
       case optional:
         uint32 min_expiration_duration;
         uint32 max_expiration_duration;
         optional uint32 default_expiration_duration;
       case forbidden:
         struct {};
     }
   } MessageExpiration;

   MessageExpiration MessageExpirationData;
   MessageExpiration MessageExpirationUpdate;


   struct {
     ProtocolVersion versions<V>;
     CipherSuite cipher_suites<V>;
     ExtensionType extensions<V>;
     ProposalType proposals<V>;
     CredentialType credentials<V>;
     WireFormats wire_formats<V>;
     ComponentID component_ids<V>;
     ComponentID safe_aad_types<V>;
     MediaType media_types<V>;
     ContentType content_types<V>;
   } ExtendedCapabilities;

   enum {
     unspecified(0),
     immediate_commit(1),
     random_delay(2),
     (255)
   } PendingProposalStrategy;

   struct {
     PendingProposalStrategy pending_proposal_strategy;
     select (pending_proposal_strategy) {
       case unspecified:
         struct {};
```

```
      case immediate_commit:
        struct {};
      case random_delay:
        uint64 minimum_delay_ms;
        uint64 maximum_delay_ms;
      case extension:
        ComponentID id_of_strategy_params;
    }
  } PendingProposalPolicy;

  struct {
    uint64 minimum_time;
    uint64 default_time;
    uint64 maximum_time;
  } MinDefaultMaxTime;

  struct {
    uint8  epoch_tolerance;
    uint16 pad_to_size;
    uint32 max_generations_skipahead;
  } AppMessagePolicy;

  struct {
    ExtendedCapabilities mandatory_capabilities;
    ExtendedCapabilities default_capabilities;
    ExtendedCapabilities forbidden_capabilities;
    WireFormats handshake_formats<V>;
    bool external_proposal_allowed;
    bool external_commit_allowed;
    PendingProposalPolicy pending_proposal_policy;
    MinDefaultMaxTime LeafNode_update_time;
    AppMessagePolicy app_message_policy;
    unit64 max_kp_lifetime;
    uint64 max_credential_lifetime;
    uint64 resumption_psk_lifetime;
    MinDefaultMaxTime sender_nonce_keypair_lifetime;
    uint32 max_keypairs;
    MinDefaultMaxTime buffer_incoming_message_time;
    uint32 max_buffered_messages;
  } OperationalParameters;

  OperationalParameters OperationalParametersData;
  OperationalParameters OperationalParametersUpdate;
```

Acknowledgments

TODO acknowledge.

Author’s Address

Rohan Mahy
Rohan Mahy Consulting Services
Email: rohan.ietf@gmail.com