Linearized Matrix for MIMI

IETF 117 - July 24 & 26, 2023

Travis Ralston | Matthew Hodgson
travisr@matrix.org | matthew@matrix.org
What’s changed since IETF116?

- We have a standalone self-contained draft of Linearized Matrix for MIMI, with zero external dependencies on the Matrix spec!
- Lots of useful feedback from jdrosen and others - thank you!
- We have a few independent interoperable implementations of the draft, tested at the hackathon:
  - Eigenserver (original TS implementation) from Travis (Matrix.org)
  - Synapse (making an existing Matrix server talk LM too) from Patrick (Element)
  - Android Messages (experimental showing an existing stack) from Google
- Other independent implementation interest from Cloudflare, imMail and more.
Dual-stack Synapse <-> Eigenserver interop
Android Messages interop
Where could Linearized Matrix fit in?

MIMI message contents (maybe Matrix events?)

MLS

Linearized Matrix Transport

Simple (~1000 LOC), pragmatic, proven messaging transport with MLS on top. LM uses group management.
Architecture

For a given room:

Provider A (Hub)

Provider B

Gatekeeper

Provider C

Linearized Matrix
Architecture

**Room versions** define the specific edition of each algorithm the room is using to auth and distribute messages. Only I.1 is specified as a legal room version (for now).

Changes to the auth rules, or any other ACL mechanism, require a new room version. Rooms “upgrade” by creating an all-new room, referencing the old one.

A new name for “room versions” is needed.
Events

Currently JSON objects due to transport, but can change shape as needed.

**State events** track the metadata for the room, such as *user* membership, name, join rules, power levels (access control), etc.

MLS commits (and possibly proposals) would be sent as *room events*, or “non-state” events.
Events

**Content hashes** protect the event from modification as it transits between servers. **Reference hashes** cover content hashes and the event itself, and act as the event ID to eliminate malicious ID collisions.

**Signatures** guarantee that the originating server did actually create the event, implicitly covering the reference hash.

Content hashes and signatures are not needed if events are authenticated solely by MLS by calling out to the MLS layer.
Auth rules/ACLs

The core of Linearized Matrix. Uses `m.room.power_levels` state event (and others) as variables in the rules.

Operate on each and every event that pass through the server.

Group membership managed by server, but could call out to MLS.

They define:

- Who can invite, kick, ban, etc other users.
- Whether users can join without an invite (eg: public rooms)
- Whether users can send events of a given type
  - Including who can change the power levels/join rules
Transport

Defined transport is inspired by Matrix’s 2014-era design, to be replaced with gRPC in an upcoming draft.

Currently HTTPS + JSON. Used as a baseline for testing interoperability/feasibility at the hackathon.

Possible alternatives include websocket or anything more RESTful.
MLS

Linearized Matrix layers MLS on top for encryption (also supports Double Ratchet for pragmatism). Represents the Matrix team’s experimental work in supporting MLS out of band from MIMI.

Currently uses a mostly server-side model. Membership is tracked by the server, but user messages are (obviously) end-to-end encrypted.

Linearized Matrix’s auth rules can be applied to a pure client-side model, partial server-side model, or full server-side model as needed.
Upcoming changes

Simplify data structure for event content hashes.
Replace `prev_events` array with singular `prev_event` field.
Remove `auth_events` entirely.

Consider calling out to MLS groups to cryptographically define group membership (merging with I-D.robert-mimi-delivery-service?)
Consider merging LM access controls into I-D.mahy-mimi-group-chat (e.g. multiple levels of moderation)
Appendix:
What if LM was policy control for MIMI-DS?
Where do we see Linearized Matrix fitting in?

- **MIMI message contents (maybe Matrix events?)**
- **MLS extensions for policy management:** Linearized Matrix
- **MLS**

---

- **MLS Delivery Service Function**
- **MIMI Transport (derived from LM?)**
What does the Linearized Matrix do?

- Primarily a description of a room model
  - Architecture constraints for a transport (hub server, no owner)
  - Policy and permissions representative of all kinds of messaging providers
  - Model has been built since ~2014, serving interoperability for Matrix

- HTTPS+JSON transport also included, but should be broken out
  - We’re working on a new draft which uses MIMI-DS and gRPC.

- MLS semantics exist if layering MLS on top of LM
  - Suggestion is to layer something on top of MLS instead
Architecture

Rooms are MLS groups which contain events.

Rooms have a hub server which fans out messages to other servers (hub and spoke model).

Rooms do not have an owning server responsible for enforcing access control rules; all servers may enforce them.

All hub servers must use the same access control mechanisms, called auth rules. This lets different providers participate as hubs while being consistent on auth behaviour (and supports hub transfers in future).
Architecture
(for a given room)

Matrix

Matrix clients

Matrix Homeservers

Hub server
(for the conversation)

Linearized Matrix

DMA interoperaing providers

Network view for a given conversation
Overall connectivity
Core export: Policies

The legal membership transitions and permissions model are described as **auth rules**. “State events” configure the auth rules for the room.

In a LM-over-MLS model, MLS is responsible for persisting membership. LM’s auth rules define how a user/client transitions between different membership states.

Most actions are expressed via **power levels**. The rules themselves are versioned as **room versions**.
Power levels

Integer values associated with actions and users. Higher power gives more ability to do something.

“Roles” can be created by assigning actions to particular levels. For example, being able to kick users at 25, delete messages at 50, and promote other admins at 100.

Users must have at least the power for an action to do something. Assuming permission to change power levels, users can only promote up to their own level.
Why not RBAC?

Role-Based Access Control provides the user a lot of flexibility, but can become complicated. Not all messaging providers support RBAC either, making them hard to interoperate with.

Power levels have been proven to work with most messaging providers, with the exception of Discord-style RBAC.

Limiting the protocol’s supported roles helps a lot with these problems. Infinitely available and custom roles are significantly harder to support.
Room versions

Room versions describe the specific policies in use by a group/room.

Changing the policies would mean a Relnit (and new group ID), ensuring that all servers in the room know what the “current” policies are.

Experimental or customized room versions are possible through namespacing. In future, what makes up a room version would be the responsibility of a registry of some kind.