Matrix and MLS

Water, Oil and Mayonnaise

Nico

Mayonnaise, vegan

1 part unflavoured unsweetened soy milk

2.25ish parts canola oil, more oil = thicker

0.1 part vinegar or lemon juice (a teaspoon or two)

salt

put everything in a container slightly wider than your immersion blender, and then blend. move the thing around to "whip" the oil

Thank you for coming to my talk!

Emulsion

An emulsion is a mixture of two or more liquids that are normally immiscible (unmixable or unblendable) owing to liquid-liquid phase separation.

WTF?

Well, Matrix and MLS are kinda incompatible, but let me explain.

Self insert

- I'm Nico
- matrix:u/deepbluev7:neko.dev
- Nheko developer
- Part of the Matrix Governing Board
- Working for Famedly to bring Matrix to the healthcare system
- Current project, figure out MLS+Matrix for BWI
- I REMOVED REPLY FALLBACKS! ME! I DID IT! FINALLY!

The core of Matrix

Fancy state resolution:

- Servers can send state independently
- Conflicts are figured out after the fact
  - Higher power level wins!
- PARTITION TOLERANCE! (Byzantine faults?!?)
- Decentralized system with no central control or dependency
  - Let's just ignore room version 12 and policy servers for now
- State "bloat"

Nice APIs!

Directory and history sharing service?

The core of Matrix

Encryption via sender keys/double ratchet:

- 2 layers of encryption
- membership is controlled by servers
    - You don't want to trust servers
    - Verification doesn't tell you, who should be in a room!
- No central order might cause UTDs
    - Someone might be added to a group at the same time you send a message
    - Can't tell, why the message is UTD
- State is unencrypted.
    - Different key schedules
- Quite a bit of unencrypted metadata
    - Server side permissions!
- Technically no perfect forward secrecy with online key backup, but eh.
- Linear complexity -> N devices, N encryptions (once every 100 messages)

The core of MLS

- IETF standard for encryption
  - No message format or server APIs (see MIMI?)
- BSI approved, which matters for some customers...
- Ideally logarithmic complexities when encrypting
  - Can be worse in some cases!
- Normal messages and handshake messages (for changing encryption epoch and members)
  - No ordering requirement for normal messages, but central order for handshakes.
- You share ONE trusted set of users in a room (usually)
  - Needs to be centrally ordered currently
- Allows deriving keys for different use cases
- A shared group context, that you can put arbitrary data into
  - Encrypted state for free?!?

Less important differences

Encryption targets per sender or for the whole group is a choice

We don't care about the amount of layers, if they work well together (but do they?)

Matrix clobbers state, MLS has proposals, which are changes to one central state

The blender?

Decentralized state vs central synchronisation

Centralization requirement for one shared context & membership, kinda.

State resolution complicates perfect forward secrecy

- If you want to encrypt to one shared set of users and throw away keys ASAP, you can't wait indefinitely for parallel modifications of membership

What do we care about?

What if we have infinite funding and possibilities?

- Secure, private messaging where everybody can control their own data
- We don't want old devices to compromise all future history, when broken into
    - But how long of a window can we accept? 0s? 5m? 7d? 30d?
- Does the server need to know, who is in a room?
- Best privacy?
- Do we need decentralization for state changes or just for messages?
- Do we want resets to arbitrarily old state?
- Can we make a slider between more theoretical security and more decentralization with reasonable security?
- What permissions does the server need to enforce?

Radical removal

Remove:

- Senders from events
- Most state events
  - Especially membership events!
- Power levels (permissions)

We'll fix that in post!

Step one: Build a centralized MLS on top of Matrix

1. Define one core server to order handshake messages
2. Only that server can send commits
3. Every other server can send messages
4. Any member can invite others into the group
5. No kicks, or permissions. People will just not be able to decrypt stuff when removed.
6. State events just get put into the group context

Obviously not ideal, but really nice metadata protection.

Membership exposed only ephemerally, not to every server. (Sealed sender?)

Step two: Break MLS secrecy to add decentralization

- Client side state resolution
- Clients keep old commit events around to resolve conflicts
- Define some kind of ordering between servers, so that the higher PL server wins in a state conflict
- Changes, that get state resolved away need to be sent again.
- Somehow force servers to not go back on their word to make power transitions possible
    - Alternatively creators will have the highest PL forever

Step three: Fix unbounded forward secrecy problem?

- We don't want an old device to decrypt all the future (if it is compromised for example)
- What if we just kick devices, that don't add new entropy for 30 days?
    - If we have a cheap way to add them back and still give them history, that might be fine for most people?
- Throw away old secrets ASAP when you hear back from a different server
    - Provides forward progress, no infinitely old keys anymore apart from for dead servers
    - The new DMLS proposal does this with puncturable random functions (or so)

We have decentralization back? But still no permissions!

Step four: Permissions without exposing sender and users?

MLS allows us to derive keys and encrypt to specific subsets of users.

What if we give them some secret, they can use to sign something to then do certain priviledged actions?

Would just expose that a user has the power to do a thing, not who they are or who is admin in a group

Early stages, but maybe this could simplify state resolution? You can remove power by rotating secrets?

Open questions

- Can we even make that work? A lot of theories, but we still need to evalutate things more.
- What are the security tradeoffs?
- Can we improve state resolution this way?
- What is the usability like?
- Decouple directory service and history service to provide more anonymity?

Other cool use cases

We can derivce keys:

- Use it to bootstrap group call encryption?
- Use it for authentication against other services as a group member?

We have no sender:

- Account portability?

THE END