More MLS extensions

draft-ietf-mls-extensions
draft-mahy-mls-group-anchors
The Leaving Problem?
Policy knobs?

Rohan Mahy, IETF 116, 30-Mar-2023

Content Advertisement
(in draft-ietf-mls-extensions already)

Content Advertisement MLS Extensions

- Express media types that are supported In LeafNodes in groups and in KeyPackages
  - `accepted_media_types`
- Express required types In GroupContext
  - `required_media_types`
- Express the media type inside an application message
- These use the extensive IANA media types registry names (straightforward registration, free vendor namespace)
  - Examples:
    text/plain
    text/html
    application/foo-im-protocol+json
    application/vnd.examplecorp.instantmessage
    multipart/alternative
    text/markdown

Application Framing

- Application Data is assumed to use Application Framing if `required_media_types` is in the `GroupContext` extensions for the group
- Application Framing contains the `media_type`, then the rest of the `application_content`

```
struct {
    MediaType media_type;
    opaque<V> application_content;
} ApplicationFraming;
```

- If the `media_type` is a zero-length vector, the first media type is assumed

Group Trust Anchors
draft-mahy-mls-group-anchors

Group Anchors Problem

- Operating system/application has a large list of trusted certificates
- To validate an identity Credential in another domain, want to be more specific to avoid one federated domain impersonating identities in another.
- atlanta.at is federating with biloxi.be and chicago.ch. Some domains want clients to restrict identity/credential validation to either a **specific root cert** or a **specific intermediary** rooted in a specific root cert for each of these domains.
- Even if intermediaryCA.atlanta.at is compromised, atlanta.at can't impersonate users in biloxi.be or chicago.ch without detection by clients from the other domains.

Group Anchors Mechanism

- Proposed a GroupContext extension with a list of domain/certchain pairs.
  Validate biloxi.be with this cert chain, atlanta.at with that cert chain, etc.

- Suggestion from Marta: cert chains are big. should be able to send a hash/ref of the root cert
  - Could not find a spec to reference, otherwise would have included it.
    Note: still need the actual certificate if it is an intermediate cert
- Comment from Richard: could be used also for distribution if there is a clear way to authorize the new keys.
  - Publish something on the domain (in DNS, .well-known, etc.), or
  - sign the information with a WebPKI certificate for the domain

The Leaving Problem

“The issue of inability for a client to remove itself from the group by itself seems unsolvable. I would like to see recommendations in the document for clients wishing to exclude themselves in situations when other members for some reasons don't cooperate in this process.”
— Valery Smyslov (via AD Francesca Palombini in her COMMENT on MLS Arch)

Proposal: Leave Proposal

- Problem: External Joins ignore pending proposals. In busy group can result in long delay / many retries before a client is removed. External Joiner can't verify Remove proposals.

- Proposal: Define new **Leave Proposal**, signed by the leaving member
  - Clients sending a Commit with pending Leave Proposals must include them. Includes External Joiners (must get Leave Proposal(s) with GroupInfo).
  - new Leave Proposal can be included in External Commit, and external committer can verify the signature.
  - Clients about to send an application message, send Commit if any Leave proposal is pending.

- Thoughts from WG?

Policy Knobs
(inspired by Section 6 of draft-ietf-mls-architecture)

Policies from Section 6 (Operational Reqs)

An Authentication Service, described fully in Section 3, defines the types of credentials which may be used in a deployment and provides methods for:

1. Issuing new credentials with a relevant credential lifetime
2. Validating a credential against a reference identifier
3. Validating whether or not two credentials represent the same client, and
4. Optionally revoking credentials which are no longer authorized.

A Delivery Service, described fully in Section 4, provides methods for:

1. Delivering messages for a group to all members in the group
2. Delivering Welcome messages to new members of a group.
3. Uploading new KeyPackages for a user's own clients.
4. Downloading KeyPackages for specific clients. Typically, KeyPackages are used once and consumed.

Additional services may or may not be required depending on the application design:

1. If assisted joining is desired (meaning that the ratchet tree is not provided in Welcome messages), there must be a method to download the ratchet tree corresponding to a group.

If assisted joining is desired and the Delivery Service is not able to compute the ratchet tree itself (because some proposals or commits are sent encrypted), there must be a method for group members to publish the updated ratchet tree after each commit.

If external joiners are allowed, there must be a method to publish a serialized GroupInfo object (with an external_pub extension) that corresponds to a specific group and epoch, and keep that object in sync with the state of the group.

If an application chooses not to allow assisted or external joining, it may instead provide a method for external users to solicit group members (or a designated service) to add them to a group.

If the application uses external PSKs, or uses resumption PSKs that all members of a group may not have access to, there must be a method for distributing these PSKs to group members.

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:

1. The maximum total lifetime that is acceptable for a KeyPackage.
2. How long to store the resumption secret for past epochs of a group.
3. The degree of tolerance that's allowed for out-of-order message delivery:
4. How long to keep unused nonce and key pairs for a sender
5. A maximum number of unused key pairs to keep.
6. A maximum number of steps that clients will move a secret tree ratchet forward in response to a single message before rejecting it.
7. Whether to buffer messages that aren't able to be understood yet due to other messages not arriving first, and if so, how many and for how long. For example, Commit messages that arrive before a proposal they reference, or application messages that arrive before the Commit starting an epoch.
8. MLS provides the following locations where an application may store arbitrary data. The format and intention of any data in these locations must align for two deployments to interoperate:
    - Application data, sent as the payload of an encrypted message.
    - Additional authenticated data, sent unencrypted in an otherwise encrypted message.
9. Group IDs, as decided by group creators and used to uniquely identify a group.
10. The application_id extension of a LeafNode.

MLS requires the following policies to be defined, which restrict the set of acceptable behavior in a group. These policies must be consistent between deployments for them to interoperate:

1. A policy on which ciphersuites are acceptable.
2. A policy on any mandatory or forbidden MLS extensions.
3. A policy on when to send proposals and commits in plaintext instead of encrypted.
4. A policy for which proposals are valid to have in a commit, including but not limited to:
    1. When a member is allowed to add or remove other members of the group.
    2. When, and under what circumstances, a reinitialization proposal is allowed.
    3. When proposals from external senders are allowed and how to authorize those proposals.
    4. When external joiners are allowed and how to authorize those external commits.
    5. Which other proposal types are allowed.
5. A policy of when members should commit pending proposals in a group.
6. A policy of how to protect and share the GroupInfo objects needed for external joins.
7. A policy for when two credentials represent the same client. Note that many credentials may be issued authenticating the same identity but for different signature keys, because each credential corresponds to a different device (client) owned by the same application user. However, one device may control many signature keys but should still only be considered a single client.
8. A policy on how long to allow a member to stay in a group without updating its leaf keys before removing them.
9. Finally, there are some additional application-defined behaviors that are partially an individual application's decision but may overlap with interoperability:
    1. If there's any policy on how or when to pad messages.
    2. If there is any policy for when to send a reinitialization proposal.
    3. How often clients should update their leaf keys.
    4. Whether to prefer sending full commits or partial/empty commits.
    5. Whether there should be a required_capabilities extension in groups.

Machine Readable Policies in GroupContext?

- Do we want a machine readable to communicate some of these policies?

- Do we want to be able to provably agree on any of these parameters?

Thank You

Backup: How do clients figure out the format?

- `required_media_types` is not in the GroupContext
  - Application Data (probably) does not using Application Framing. No change in message size.
- `required_media_type` is in the GroupContext. Assume Application Framing
  - if `media_type` is empty string
    - `application_content` is the first `media_type` in `required_media_types`. Message is 3-4 bytes longer.
  - if `media_type` is present
    - `application_content` is the specified `media_type`.
    - specified `media_type` is multipart/alternative.
      - multipart message includes at least one supported type for every member
      - each member that receives the message uses the first sent `media_type` they support

How does client know what formats are OK?

- For MLS this is covered in Section 2.3 of draft-ietf-mls-extensions (content advertisement) and related to Sections 7.2, 11.1, 12.1.7 of draft-ietf-mls-protocol (MLS core protocol)
- In brief:
  - ***supported*** media types are listed **for each member** of the group. are updated periodically in long-lived groups (after client upgrade very likely)
  - ***supported*** media types are advertised **in KeyPackages** (used to add clients). clients update these periodically and very likely after upgrade.
  - creator can list ***required*** media types for a group. All clients need to have support for these.
  - the required media types can be ***updated*** with a GroupContextExtensions Proposal, as long as the resulting clients