Messaging Layer Security
Internet-Draft
Intended status: Informational
Expires: 23 April 2026

K. Kohbrok
Phoenix R&D

20 October 2025

Leaf Operation Intents
draft-kohbrok-mls-leaf-operation-intents-01

Abstract

The Messaging Layer Security (MLS) protocol defined in [RFC9420] is an asynchronous secure group messaging protocol, which allows group members to propose their removal from a group.

However, in some cases MLS clients can’t reliably use regular Remove or SelfRemove proposals to leave a group because they don’t have an up-to-date group state.

This document specifies a LeafOperationIntent, which does not need an up-to-date group state but which retains sufficient binding to the client’s current state to avoid replay attacks.

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://kkohbrok.github.io/draft-kohbrok-mls-leaf-operation-intents/draft-kohbrok-mls-leaf-operation-intents.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-kohbrok-mls-leaf-operation-intents/.

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

Source for this draft and an issue tracker can be found at https://github.com/kkohbrok/draft-kohbrok-mls-leaf-operation-intents.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Copyright Notice

Table of Contents

1. Introduction

To leave an MLS group, a member cannot create a commit, but rather has to propose its own removal. This can create difficulties, some of which have been solved by the introduction of the SelfRemove proposal, which may be included in external commits.

One drawback of Remove and SelfRemove proposals is that they (like other proposals) are bound to a specific group epoch. This means that authors of such proposals must have an up-to-date group state to send such a proposal and continue to keep track of that group state both to re-send the proposal if necessary.

This can be a problem if an application wants to cleanly leave a group and immediately delete the associated group state, e.g., to erase the associated metadata. The deletion of the group state makes it impossible for the client to re-send the proposal in case it’s not covered by the next commit. Similarly, the client might be offline at the time and the group state might not be up-to-date.

The LeafOperationIntent specified in this document allows a client to bind an intent to leave a group to the leaf’s current state. That intent can then be proposed by any party (e.g. an external sender) in any subsequent epoch, provided the leaf remains unchanged

As users often have more than one client that needs to be removed from the group as part of the user leaving, the intent allows expanding the leaf operation to associated leaves.

2. LeafOperationIntent

A LeafOperationIntent can be created by clients and distributed either to other group members or to one or more external senders.

```
HashReference LeafNodeRef;

MakeLeafNodeRef(value)
  = RefHash("MLS 1.0 LeafNode Reference", value)

enum {
  reserved(0),
  sender_removal_only(1),
  remove_associated_members(2),
  (255)
} RemovalMode

struct {
  opaque group_id<V>;
  uint32 sender_index;
  LeafNodeRef sender_leaf_ref;
  RemovalMode removal_mode;
} LeafOperationIntentTBS

struct {
  opaque group_id<V>;
  uint32 sender_index;
  LeafNodeRef sender_leaf_ref;
  RemovalMode removal_mode;
  /* SignWithLabel(., "LeafOperationIntentTBS", LeafOperationIntentTBS) */
  opaque signature<V>;
} LeafOperationIntent

struct {
  LeafOperationIntent intent;
} LeafOperationProposal
```

RefHash and SignWithLabel are as defined in [RFC9420].

* group_id: The ID of the MLS group in which context the LeafOperationIntent was sent

* sender_index: The index of the sender’s leaf in the group

* sender_leaf_ref: Hash computed over the LeafNode of the sending client using the MakeLeafNodeRef function

* removal_mode: Indicates whether only the sender should be removed, or whether additionally any other, associated members should be removed as well.

* signature: A signature over all fields except the signature itself using the sender’s leaf signature key

The purpose of the removal_mode is to allow the sender to signal that other members associated with the sender should be removed as part of this operation. This can be useful if the sender is part of a group of associated devices, e.g., multiple devices belonging to the same user, to facilitate the leaving of the entire user as opposed to just the sending client.

2.1. LeafOperationIntent WireFormat

A LeafOperationIntent is an MLS WireFormat and extends the select statement in the definition of MLSMessage as follows:

```
struct {
    ProtocolVersion version = mls10;
    WireFormat wire_format;
    select (MLSMessage.wire_format) {
      ...
      case mls_leaf_operation_intent:
        LeafOperationIntent leaf_operation_intent;
    };
} MLSMessage;
```

2.2. Creating and proposing a LeafOperationIntent

A group member creates a LeafOperationIntent by populating the group_id, sender_index and sender_leaf_ref according to the current state of the group and the sender’s leaf.

If the sender wants to signal the removal of any associated members, it can set the removal_mode accordingly.

Finally the sender creates the signature by calling SignWithLabel on the LeafOperationIntentTBS populated as described above with "LeafOperationIntentTBS" as label.

2.3. Processing a LeafOperationIntent

Recipients of a LeafOperationIntent MUST perform the following steps:

* Verify that the group_id matches the group in which the proposal was sent

* Verify that the sender_leaf_ref is the LeafRef of the leaf at the sender_index

* Verify the signature over the intent using the signature public key in the leaf at the sender_index

If any of the validation steps fail, the recipient MUST consider the proposal invalid.

If the validation was successful, the recipient MAY create a LeafOperationProposal containing the intent and either commit it directly or send it to the group.

2.4. Processing a LeafOperationProposal

Recipients of a LeafOperationProposal MUST perform the checks listed in Section 2.3 on the intent contained in the proposal.

If any of the validation steps fail, the recipient MUST consider the proposal invalid.

After that, the proposal MUST be validated and processed as if it were set of Remove proposals originating from the sender of the intent (not the sender of the proposal).

The set of Remove proposals consists of one Remove proposal targeting the intent’s sender_index.

Additionally, if removal_mode is remove_associated_members, the recipient MUST check with the authentication service (AS, see [RFC9750]) whether any other members of the group are associated with the sender (see Section 2.5). If there are any such members, the set of Remove proposals additionally contains one Remove proposal per associated targeting that member’s leaf index.

External commits may include one or more LeafOperationProposals. Any Removes validated as described above MUST thus be considered valid in this context.

2.5. Additional AS role

When using LeafOperationIntents, the AS gains the additional role of having to identify other members in a group that are _associated_ with the sender of a LeafOperationIntent. The sole purpose of association in this document is to determine whether other clients should be removed when a given client communicates its intent to leave a group. In most cases, association is determined by Credentials of the individual group members.

A set of clients could, for example, be considered associated if all clients belong to the same user.

3. Security Considerations

In contrast to proposals, LeafOperationIntents are not bound to an epoch and thus remain valid as long as the creator’s leaf doesn’t change its state.

Each LeafOperationIntent can thus be proposed and committed to until the sender is either removed from the group or updates its own leaf.

This allows scenarios, where, for example, members get added to or removed from a group in the time between the creation and the proposal of the intent.

If a tighter bound to the epoch, i.e. the current group state is required, clients should use regular Remove or SelfRemove proposals instead.

Epoch independence incurs a certain risk of replay attacks. The bound of the intent to the hash of the sender’s LeafNode limits that risk significantly. However, a replay is possible, for example, if the sender’s leaf still contains the LeafNode from a KeyPackage. In that case, if the sender is later added again with the same KeyPackage, the intent can be replayed.

4. IANA Considerations

4.1. MLS Proposal Types

This document requests the addition of a new Proposal Type under the heading of "Messaging Layer Security".

The leaf_operation_intent MLS Proposal Type is used to allow members or external senders to convey the intent of a leaf owner to perform an operation on their leaf.

* Value: TBD
* Name: leaf_operation_intent
* Recommended: Y
* External: Y
* Path Required: Y
* Reference: RFCXXXX

4.2. MLS WireFormats

This document requests the addition of a new Wire Format under the heading of "Messaging Layer Security".

The mls_leaf_operation_intent MLS WireFormat allows parties to send MLSMessages containing a LeafOperationIntent.

* Value: TBD

* Name: mls_leaf_operation_intent

* Recommended: Y

* Reference: RFCXXXX

5. Normative References

[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>.

[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>.

Acknowledgments

TODO acknowledge.

Contributors

Raphael Robert
Phoenix R&D
Email: ietf@raphaelrobert.com

Author’s Address

Konrad Kohbrok
Phoenix R&D
Email: konrad@ratchet.ing