Messaging Layer Security
Internet-Draft
Intended status: Informational
Expires: 19 April 2026

R. Mahy
Rohan Mahy Consulting Services

16 October 2025

Semi-Private Messages in the Messaging Layer Security (MLS) Protocol
draft-mahy-mls-semiprivatemessage-06

Abstract

This document defines a SemiPrivateMessage for the Messaging Layer Security (MLS) protocol. It allows members to share otherwise private commits and proposals with a designated list of external receivers rather than send these handshakes in a PublicMessage.

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://rohanmahy.github.io/mls-semiprivatemessage/draft-mahy-mls-semiprivatemessage.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-mahy-mls-semiprivatemessage/.

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/rohanmahy/mls-semiprivatemessage.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 19 April 2026.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction

This document defines two extensions of MLS [RFC9420]. The first is the SemiPrivateMessage wire format, which allows an otherwise PrivateMessage to be shared with a predefined list of external receivers. It is restricted for use only with commits or proposals. The second is the external_receivers GroupContext extension that contains the list of external receivers and allows members to agree on that list.

SemiPrivateMessages are expected to be useful in federated environments where messages routinely cross multiple administrative domains, but the MLS Distribution Service needs to see the content of commits and proposals where group members would otherwise send handshakes using PublicMessage.

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.

This document uses terminology extensively from MLS [RFC9420] and the Safe Extensions framework, defined in Section 2 of [I-D.ietf-mls-extensions].

Whenever a hash function is mentioned, it refers to the hash function defined in the cipher suite in use for the relevant MLS group.

3. Syntax and Usage

The external_receivers GroupContext extension is used for all members to agree on the list of external receivers in the current epoch. Its construction mirrors the syntax of the external_senders extension in [RFC9420].

```
struct {
  HPKEPublicKey external_receiver_public_key;
  Credential credential;
} ExternalReceiver;
```

The mls_semiprivate_message wire format is advertised in the supported_wire_formats list in LeafNode.capabilities.extensions, (defined in Section 5 of [I-D.ietf-mls-extensions]). For SemiPrivateMessage to be used in a group, mls_semiprivate_message needs to be in the required_wire_formats list in the GroupContext.extension_types of that group, and there needs to be at least one entry in the external_receivers GroupContext extension.

SemiPrivateMessage substantially reuses the construction of PrivateMessage, but like a Welcome message also contains information (key_and_nonces) necessary to identify the sender leaf node and decrypt the SemiPrivateMessage struct’s ciphertext. Note that the encrypted_sender_data cannot be decrypted by an external receiver, but the sender_leaf_index is included with key_and_nonces and is verified in another step. key_and_nonces is encrypted once for each external receiver in the external_receivers extension.

3.1. Encryption of a SemiPrivateMessage

As with a PrivateMessage, the sending client chooses an unused generation in its own handshake ratchet and derives a key and nonce. It also generates a fresh random four-byte reuse_guard. The snippet below shows the syntax and encryption and decryption construction of keys_and_nonces into encrypted_keys_and_nonces for each external receiver.

```
struct {
  opaque key<V>;
  opaque nonce<V>;
  opaque reuse_guard[4];
  uint32 sender_leaf_index;
} PerMessageKeyAndNonces;

partial_context_hash = hash(sender_leaf_index || nonce)

struct {
  opaque group_id<V>;
  uint64 epoch;
  opaque partial_context_hash<V>;
} SemiPrivateMessageContext;

PerMessageKeyAndNonces key_and_nonces;
SemiPrivateMessageContext semi_private_message_context;

encrypted_key_and_nonces = EncryptWithLabel(
  external_receiver_public_key,
  "SemiPrivateMessageReceiver",
  semi_private_message_context,   /* context */
  keys_and_nonces)

key_and_nonces = DecryptWithLabel(
  external_receiver_private_key,
  "SemiPrivateMessageReceiver",
  semi_private_message_context,  /* context */
  encrypted_keys_and_nonces.kem_output,
  encrypted_keys_and_nonces.ciphertext)
```

The KeyForExternalReceiver structure contains a hash of the ExternalReceiver as a reference and the encrypted_key_and_nonces.

```
ExternalReceiverRef = hash(ExternalReceiver)

struct {
  ExternalReceiverRef external_receiver_ref;
  HPKECiphertext encrypted_keys_and_nonces;
} KeyForExternalReceiver;
```

The SemiPrivateMessage struct extends the PrivateMessage struct, adding the keys_for_external_receivers list, the partial_context_hash needed for its decryption context, and the hash of the FramedContentTBS to insure that the sender cannot encrypt content to the external receivers that is different from the other members, without detection.

The SemiPrivateContentAAD struct likewise extends the PrivateContentAAD struct, adding the keys_for_external_receivers list, the partial_context_hash and the framed_content_tbs_hash.

The SemiPrivateMessageContent struct is the same as PrivateMessageContent except application messages are not included.

framed_content_tbs_hash = hash(FramedContentTBS)

```
struct {
    opaque group_id<V>;
    uint64 epoch;
    ContentType content_type;
    opaque authenticated_data<V>;
    opaque partial_context_hash<V>;
    KeyForExternalReceiver keys_for_external_receivers<V>;
    opaque framed_content_tbs_hash<V>;
    opaque encrypted_sender_data<V>;
    opaque ciphertext<V>;
} SemiPrivateMessage;

struct {
    select (SemiPrivateMessage.content_type) {
        case proposal:
          Proposal proposal;
        case commit:
          Commit commit;
    };
    FramedContentAuthData auth;
    opaque padding[length_of_padding];
} SemiPrivateMessageContent;

struct {
    opaque group_id<V>;
    uint64 epoch;
    ContentType content_type;
    opaque authenticated_data<V>;
    opaque partial_context_hash<V>;
    KeyForExternalReceiver keys_for_external_receivers<V>;
    opaque framed_content_tbs_hash<V>;
} SemiPrivateContentAAD;

struct {
    ProtocolVersion version = mls10;
    WireFormat wire_format;
    select (MLSMessage.wire_format) {
        case mls_public_message:
            PublicMessage public_message;
```

```
            case mls_private_message:
                PrivateMessage private_message;
            ...
            case mls_semiprivate_message_:
                SemiPrivateMessage semiprivate_message;
        };
    } MLSMessage;
```

Encryption of the ciphertext uses the cipher suite's AEAD algorithm using the key, nonce xored with the reuse_guard, the SemiPrivateMessageContent as the plaintext, and the SemiPrivateContentAAD as the authenticated data.

Encryption of the encrypted_sender_data proceeds in the same way for SemiPrivateMessage as for PrivateMessage.

3.2. Decryption of SemiPrivateMessage as a member

When receiving a SemiPrivateMessage, a member receiver derives the sender_data_key and sender_data_nonce and decrypts the encrypted_sender_data, just as for a PrivateMessage.

The receiver uses the SenderData to lookup the key and nonce for the correct generation in the (non-blank) sender's handshake ratchet. The receiver verifies the partial_context_hash.

After xoring the nonce with the reuse_guard, the member decrypts the ciphertext. It verifies the padding consists of the appropriate number of zero bytes, and verifies that the framed_content_tbs_hash is correct. Finally, it verifies that the signature in the FramedContentAuthData is valid.

3.3. Decryption of SemiPrivateMessage as an external receiver

When receiving a SemiPrivateMessage, an external receiver looks in the keys_for_external_receivers field for its external_receiver_ref. It calculates the semi_private_message_context and uses HPKE to decrypt the encrypted_keys_and_nonces. Using the nonce and sender_leaf_node it verifies the partial_context_hash.

After xoring the nonce with the reuse_guard, the member decrypts the ciphertext. It verifies the padding consists of the appropriate number of zero bytes, and verifies that the framed_content_tbs_hash is correct. If the external receiver has a copy of the GroupContext, it verifies that the signature in the FramedContentAuthData is valid.

4. Security Considerations

These two extensions provide a privacy improvement over sending handshake messages using PublicMessage. The handshake is shared with a specific list of receivers, and that list is visible as part of the GroupContext.

TODO More Security.

5. IANA Considerations

5.1. SemiPrivateMessage Wire Format

* Value: TBD1 (to be assigned by IANA)
* Name: mls_semiprivate_message
* Recommended: Y
* Reference: RFC XXXX

5.2. External Receivers Extension Type

The external_receivers extension contains a list of external receivers targeted in a SemiPrivateMessage.

* Value: TBD2 (to be assigned by IANA)
* Name: external_receivers
* Message(s): GC. This extension may appear in GroupContext objects.
* Recommended: Y
* Reference: RFC XXXX

5.3. SemiPrivateMessageReceiver Public Key Encryption Label

* Label: "SemiPrivateMessageReceiver"
* Recommended: Y
* Reference: RFC XXXX

6. Normative References

[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>.

[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>.

Appendix A. Change log

A.1. Changes from draft-mahy-mls-semiprivatemessage-05 to -06

* fix a typo in the change log
* refresh almost expired document

A.2. Changes from draft-mahy-mls-semiprivatemessage-04 to -05

* remove the "safe extension" wire format
* use the supported/required_wire_formats extensions in mls-extensions
* register SemiPrivateMessageReceiver Public Key Encryption Label

A.3. Changes from draft-mahy-mls-semiprivatemessage-03 to -04

* corrected a typo in SemiPrivateMessageContent

A.4. Changes from draft-mahy-mls-semiprivatemessage-02 to -03

* do not attempt to decrypt SenderData for external receivers; instead also encrypt the sender_leaf_index and reuse_guard.

* make the encrypted_key_and_nonces context include the group_id, epoch, and a the hash of the sender_leaf_index and nonce. include that partial_context_hash in the AAD.

* add a hash of the FramedContentTBS to the AAD to make sure the content encrypted to the external receiver is the same as that sent to members.

* add explicit instructions about encryption and decryption.

Author’s Address

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