Messaging Layer Security
Internet-Draft
Intended status: Informational
Expires: 3 September 2026

R. Mahy

M. Chenani
XMTP Labs

2 March 2026

Private External Message extensions for Messaging Layer Security (MLS)
draft-mahy-mls-private-external-01

Abstract

MLS groups that use private handshakes lose member privacy when sending external proposals. This document addresses this shortcoming by encrypting external proposals using an HPKE public key derived from the epoch secret. It also provides a mechanism to share this key and protect it from tampering by a malicious intermediary.

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-private-external/draft-mahy-mls-private-external.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-mahy-mls-private-external/.

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-private-external.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The MLS protocol [RFC9420] was designed to support both a model where the Distribution Service (DS) sees the contents of MLS handshake messages and often assumes a policy enforcement role, and a model where the DS is merely responsible for forwarding handshake messages and possibly enforcing ordering of messages. In the first model clients send every handshake as a PublicMessage (or a SemiPrivateMessage [I-D.mahy-mls-semiprivatemessage]), whereas in the second model the clients send in-group handshakes as a PrivateMessage. As of this writing there are non-trivial commercial deployments using both the PublicMessage model (ex: Cisco, Amazon, Ring Central, Wire) and the PrivateMessage model (ex: XMTP, Germ).

In the PrivateMessage model, group members enjoy substantially more privacy from the DS. In the PublicMessage model, the DS usually can provide (authorized) non-members with enough information that they can join a group via an external commit. Even in the PublicMessage model, some (usually large) groups use external proposals to join. In the PrivateMessage model, (authorized) non-members can also join using external proposals (or rarely using external commits if the GroupInfo is shared by an existing member), however the joiner is currently forced to send the proposal (or commit) as a PublicMessage and therefore reveal potentially private information such as their credential and capabilities to the DS.

This extension allows groups using PrivateMessage to maintain the privacy of external handshake messages by encrypting them to a public key derived from the group's epoch secret. It also provides a way to convey that public key safely to prevent active attacks.

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.

3. Mechanism

3.1. External Encryption Key Derivation

Groups using this extension derive a dedicated HPKE [RFC9180] key pair from the next epoch secret for encrypting external messages. When creating a provisional commit, the committer first computes the epoch secret that will result from processing the provisional commit, then derives the external encryption key from that epoch secret.

This ensures that removed members cannot decrypt external messages, as they do not have access to the next epoch secret.

3.1.1. Computing the Next Epoch Secret

When a member creates a provisional commit, they compute the next epoch secret before sending the commit, following the key schedule defined in [RFC9420], Section 8.

The next epoch secret is derived through the standard MLS key schedule: the commit_secret (from the commit's UpdatePath) and the current epoch's init_secret produce the joiner_secret, which is combined with any PSK secrets to produce the epoch_secret for the new epoch. This document refers to this value as epoch_secret_next.

3.1.2. Deriving the External Encryption Key

The external encryption key is then derived from the next epoch secret:

```
external_encryption_secret =
    ExpandWithLabel(epoch_secret_next, "external encryption", "", KDF.Nh)

(external_encryption_private_key, external_encryption_public_key) =
    KEM.DeriveKeyPair(external_encryption_secret)
```

Where:

* epoch_secret_next is the epoch secret computed for the next epoch
* ExpandWithLabel is defined in Section 8 of [RFC9420]
* KEM.DeriveKeyPair is defined in Section 4 of [RFC9180]
* KDF.Nh is the size of an output from KDF.Extract for the cipher suite, as defined in Section 8 of [RFC9420]

The epoch field in ExternalEncryptionInfo MUST be set to current_epoch + 1.

The public key is made available to external senders via the ExternalEncryptionInfo structure (Section 3.2). All group members in the new epoch can derive the same key pair from their shared next epoch secret.

3.2. Additional information shared in every commit

Groups participating in this mechanism include a root_private_signature_key component (see Section 4.6 of [I-D.ietf-mls-extensions]) in the GroupContext of type RootPrivateSignature, containing a unique random private signature key corresponding to the group's cipher suite. Whenever a commit removes a member from a group, this component MUST be replaced with a new unique random private signature key.

Members sending a commit need to calculate the future epoch_secret, external_encryption_secret, and external_encryption_public_key for the new epoch that would result if the commit is accepted. The commit sender includes one additional Additional Authentication Data (AAD) component (see Section 4.9 of [I-D.ietf-mls-extensions]) of type ExternalEncryptionInfo in every commit (including commits sent in a PrivateExternalMessage). The ExternalEncryptionInfo includes the external_encryption_public_key for the future epoch.

> Note: SafeSignWithLabel is not used, because there are two different component IDs represented.

```
struct {
    opaque root_private_signature_key<V>;
} RootPrivateSignature;

struct {
    ProtocolVersion version = mls10;
    opaque group_id<V>;
    uint64 epoch;
    CipherSuite ciphersuite;
    HPKEPublicKey external_encryption_public_key;
    SignaturePublicKey root_public_signature_key;
} ExternalEncryptionInfoTBS;

struct {
    CipherSuite ciphersuite;
    HPKEPublicKey external_encryption_public_key;
    SignaturePublicKey root_public_signature_key;
    /* SignWithLabel(root_private_signature_key, */
    /*    "ExternalEncryptionInfoTBS", ExternalEncryptionInfoTBS) */
    opaque external_encryption_signature<V>;
} ExternalEncryptionInfo;
```

The epoch field in ExternalEncryptionInfoTBS indicates the epoch for which the external encryption key is valid. Since the key is derived from the next epoch secret, this field MUST be set to current_epoch + 1, where current_epoch is the epoch number at the time the provisional commit is created. Once the commit is processed and the group advances to the new epoch, the epoch field will match the group's current epoch.

3.3. Sending an external proposal or external commit to the group

A non-member client that wishes to send a message to the group first obtains the ExternalEncryptionInfo from the group's most recent commit. Before using the external_encryption_public_key, the external sender MUST verify the external_encryption_signature by computing VerifyWithLabel using the embedded root_public_signature_key and the label "ExternalEncryptionInfoTBS" over the reconstructed ExternalEncryptionInfoTBS. If verification fails, the ExternalEncryptionInfo MUST be rejected.

The external sender then constructs a PublicMessage called external_message_plaintext. The sender_type in the inner PublicMessage MUST NOT be member, since this mechanism is for external senders only.

The PrivateExternalMessage wire format wraps that external_message_plaintext by encrypting it to the external_encryption_public_key.

The PrivateExternalMessageContext is an empty struct, serialized to a zero-length byte string:

```
struct {
} PrivateExternalMessageContext;

struct {
    opaque group_id<V>;
    uint64 epoch;
    ContentType content_type;
    opaque authenticated_data<V>;
    HPKECiphertext encrypted_public_message;
} PrivateExternalMessage;
```

The encryption and message construction are as follows:

```
encrypted_public_message = EncryptWithLabel(external_encryption_public_key,
    "PrivateExternalMessageContent", PrivateExternalMessageContext,
    external_message_plaintext)

PrivateExternalMessage.authenticated_data =
   external_message_plaintext.content.authenticated_data
```

The PrivateExternalMessage is sent as a new variant of MLSMessage:

```
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_private_external_message:
            PrivateExternalMessage private_external_message;
    };
} MLSMessage;
```

3.4. Decryption and verification by members

Members receiving a PrivateExternalMessage MUST verify that the group_id matches a known group and that the epoch field matches their current epoch. If either check fails, the message MUST be rejected.

To decrypt the message, members derive the external encryption key pair from their current epoch secret. Since the ExternalEncryptionInfo was created using the next epoch secret (which is now the members' current epoch secret), the derivation will produce the correct key:

```
external_encryption_secret =
    ExpandWithLabel(epoch_secret, "external encryption", "", KDF.Nh)

(external_encryption_private_key, external_encryption_public_key) =
    KEM.DeriveKeyPair(external_encryption_secret)

external_message_plaintext = DecryptWithLabel(
    external_encryption_private_key,
    "PrivateExternalMessageContent", PrivateExternalMessageContext,
    encrypted_public_message.kem_output,
    encrypted_public_message.ciphertext)
```

If decryption fails, the message MUST be rejected.

Members then verify that the following values in the PrivateExternalMessage match their corresponding field in the external_message_plaintext.content:

* group_id,

* epoch,

* content_type, and

* authenticated_data

If any of these checks fail, the message MUST be rejected.

Members MUST also verify that the sender_type in the decrypted external_message_plaintext is not member. Messages from group members MUST NOT be wrapped in a PrivateExternalMessage.

Finally, they process the external_message_plaintext as if it were a regular PublicMessage.

4. Security Considerations

An established MLS group which only exchanges handshakes using MLS PrivateMessage enjoys a high level of privacy for its members. The GroupContext and the ratchet tree, including the contents of the credentials in MLS leaf nodes is not visible to outsiders nor to the DS. However, during the process of joining, private information is often leaked to the DS. This mechanism focuses on improving the privacy for the external joining mechanisms.

There are three mechanisms for potential new members to join an MLS group: an existing member gets a KeyPackage (KP) for the new member and commits an Add proposal with the KP; the joiner sends an external proposal asking to join the group that needs to be committed by an existing member; or the joiner fetches the GroupInfo of the group (usually from the DS) and sends an external commit. In the base MLS protocol [RFC9420], an external join or external commit needs to be sent as an MLS PublicMessage, which greatly reduces the privacy of the group.

4.1. Security of External Proposals

External Add proposals in [RFC9420] are sent using an MLS PublicMessage, which is integrity protected but reveals the public signature key, MLS capabilities, MLS credential to the DS, and KeyPackageRef (used to correlate Welcome messages). If a public key representing the entire target MLS group is available, the external proposer can encrypt this information to all group members without revealing it to the DS. The external proposer needs a way to get this public key and not the key of an active attacker, and the DS and members need a reasonable authorization and rate limiting mechanisms to prevent from being overwhelmed by such encrypted requests.

The ExternalEncryptionInfo defined in Section 3.2 contains a per-group, per-epoch signature key shared by all members of the group. The ExternalEncryptionInfo could be posted in transparency ledger, shared as gossip, or additionally signed by a specific member. The specific mechanism can be tailored to a specific application as needed.

Application protocols above the MLS layer would also need to provide authorization. For example, in the MIMI protocol [I-D.ietf-mimi-protocol] this could be a join code. Other techniques such as using single or limited use pseudonymous tokens, privacy pass [RFC9576], or anonymous credit tokens [I-D.schlesinger-cfrg-act] are all reasonable options. The privacy of some of these techniques could also be reinforced by using Oblivious HTTP [RFC9458].

4.2. Use of Next Epoch Secret

This specification derives the external encryption key from the next epoch secret (the epoch that results from processing the commit) rather than the current epoch secret. This design choice is critical for maintaining post-compromise security.

If the external encryption key were derived from the current epoch secret, removed members would be able to decrypt external messages sent after their removal, because they possess the current epoch secret. By deriving the key from the next epoch secret, removed members do not have access to the keying material and cannot decrypt external messages.

This approach follows the same pattern as Welcome messages in [RFC9420], which are encrypted using keys from the new epoch rather than the current epoch.

4.3. Security of External Commits

External commits in [RFC9420] are sent as PublicMessage and reveal the joiner's credential, public signature key, capabilities, and UpdatePath (including HPKE public keys for every node on the joiner's direct path). This allows the DS to learn the identity of the joiner and correlate it with other group memberships.

When wrapped in a PrivateExternalMessage, the DS can only observe the group_id, epoch, content_type, and authenticated_data fields in the outer wrapper. The joiner's credential, signature key, capabilities, and UpdatePath are encrypted and visible only to group members.

However, some metadata leakage remains:

* The DS can observe that an external commit occurred (from the content_type field and the subsequent epoch change).

* The DS can observe the size of the encrypted message, which may reveal information about the joiner's credential size or the depth of the ratchet tree.

* The Welcome message sent back to the joiner is a separate message that the DS can observe and correlate with the external commit.

The ExternalEncryptionInfo signature prevents a malicious DS from substituting its own HPKE public key to perform an active attack. Without this signature, the DS could decrypt the external commit, inspect the joiner's credentials, then re-encrypt with the legitimate key and forward it, completely defeating the privacy goal.

Note that the external_pub key (used for the ExternalInit proposal within the commit) and the external_encryption_public_key (used for the PrivateExternalMessage encryption) serve different purposes. The former is part of the MLS key schedule for deriving the init_secret; the latter protects the confidentiality of the external commit message itself. Both are derived from the epoch secret but from different labeled expansions.

4.4. Security of KeyPackages

In the classical usage of MLS, a member of a group fetches a KeyPackage, commits an Add proposal containing that KeyPackage, then sends a Welcome to the new member. In order to forward a Welcome message to the correct recipient, the DS needs to be able to associate the KeyPackageRef with some resource that eventually delivers to the appropriate client.

As long as KeyPackages are exchanged securely out-of-band, this extension extends the privacy of the MLS GroupContext and ratchet tree to external joiners.

When the DS knows the relationship between a fetched KeyPackage and the requesting user or target group, the DS can then link an added member (via its KeyPackageRef) to the requesting user or target group. An appropriate privacy-preserving mechanism (e.g., via Oblivious HTTP [RFC9458]) can associate a KeyPackageRef with the target member, without a correlation to the requesting user or target group.

Applications SHOULD consider using such privacy-preserving mechanisms for KeyPackage retrieval when deploying this extension.

4.5. Security of Welcomes

Welcome messages in [RFC9420] are encrypted to the new member's KeyPackage and contain the GroupInfo and path_secret values needed to initialize the new member's state. The Welcome message itself does not reveal group contents to the DS. However, the DS can observe:

* That a Welcome message was sent (confirming a successful join).

* The KeyPackageRef in the Welcome, which allows the DS to correlate the Welcome with a previously fetched KeyPackage and identify the new member.

This extension does not modify the Welcome message format. Applications concerned about Welcome correlation SHOULD consider additional measures such as using pseudonymous KeyPackage distribution or Oblivious HTTP [RFC9458] for Welcome delivery.

5. IANA Considerations

IANA, please replace the value RFC XXXX with the name of this document.

5.1. MLS Wire Formats

This document requests the addition of a new entry to the "MLS Wire Formats" registry defined in Section 17.2 of [RFC9420]:

Value	Name	Recommended	Reference
TBD1	mls_private_external_message	Y	RFC XXXX

Table 1

5.2. MLS Signature Labels

This document requests the addition of a new entry to the "MLS Signature Labels" registry defined in Section 17.6 of [RFC9420]:

Label	Recommended	Reference
ExternalEncryptionInfoTBS	Y	RFC XXXX

Table 2

5.3. MLS Public Key Encryption Labels

This document requests the addition of a new entry to the "MLS Public Key Encryption Labels" registry defined in Section 17.7 of [RFC9420]:

Label	Recommended	Reference
PrivateExternalMessageContent	Y	RFC XXXX

Table 3

5.4. MLS Extension Types

This document requests the addition of a new entry to the "MLS Extension Types" registry defined in Section 17.3 of [RFC9420]. This extension type is used to identify the ExternalEncryptionInfo AAD component (see Section 4.9 of [I-D.ietf-mls-extensions]):

Value	Name	Message(s)	Recommended	Reference
TBD2	external_encryption_info	GI	Y	RFC XXXX

Table 4

5.5. MLS Component Types

This document registers two new MLS Component Types in the Specification Required range:

5.5.1. root_private_signature_key MLS Component Type

* Value: TBD3 (suggested value 0x000A)
* Name: root_private_signature_key
* Where: GC
* Recommended: Y
* Reference: RFC XXXX

5.5.2. external_encryption_public_key MLS Component Type

* Value: TBD4 (suggested value 0x000B)
* Name: external_encryption_public_key
* Where: AD
* Recommended: Y
* Reference: RFC XXXX

6. References

6.1. 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>.

[RFC9180] Barnes, R., Bhargavan, K., Lipp, B., and C. Wood, "Hybrid Public Key Encryption", RFC 9180, DOI 10.17487/RFC9180, February 2022, <https://www.rfc-editor.org/rfc/rfc9180>.

[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>.

6.2. Informative References

[I-D.ietf-mimi-protocol]
Barnes, R., Hodgson, M., Kohbrok, K., Mahy, R., Ralston, T., and R. Robert, "More Instant Messaging Interoperability (MIMI) using HTTPS and MLS", Work in Progress, Internet-Draft, draft-ietf-mimi-protocol-05, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-mimi-protocol-05>.

[I-D.mahy-mls-semiprivatemessage]
Mahy, R., "Semi-Private Messages in the Messaging Layer Security (MLS) Protocol", Work in Progress, Internet-Draft, draft-mahy-mls-semiprivatemessage-06, 16 October 2025, <https://datatracker.ietf.org/doc/html/draft-mahy-mls-semiprivatemessage-06>.

[I-D.schlesinger-cfrg-act]
Schlesinger, S. and J. Katz, "Anonymous Credit Tokens", Work in Progress, Internet-Draft, draft-schlesinger-cfrg-act-01, 13 February 2026, <https://datatracker.ietf.org/doc/html/draft-schlesinger-cfrg-act-01>.

[RFC9458] Thomson, M. and C. A. Wood, "Oblivious HTTP", RFC 9458, DOI 10.17487/RFC9458, January 2024, <https://www.rfc-editor.org/rfc/rfc9458>.

[RFC9576] Davidson, A., Iyengar, J., and C. A. Wood, "The Privacy Pass Architecture", RFC 9576, DOI 10.17487/RFC9576, June 2024, <https://www.rfc-editor.org/rfc/rfc9576>.

Authors' Addresses

Rohan Mahy
Email: rohan.ietf@gmail.com

Mojtaba Chenani
XMTP Labs
Email: chenani@outlook.com