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

R. Robert
K. Kohbrok
Phoenix R&D

2 March 2026

Fewer signatures in MLS
draft-kohbrok-mls-fewer-signatures-00

Abstract

This draft specifies modified versions of MLS KeyPackage messages, as well as MLS PublicMessages and PrivateMessages holding Commits or Update Proposals that require one less signature than their original counterparts.

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-single-signature-keypackages/draft-kohbrok-mls-fewer-signatures.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-kohbrok-mls-fewer-signatures/.

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-single-signature-keypackages.

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/.

Copyright Notice

Table of Contents

1. Introduction

Both MLS KeyPackage messages, as well as PublicMessages and PrivateMessages holding Commits and Update Proposals can be safely sent with one fewer signature than specified in [RFC9420], although the latter two only if certain conditions are met.

Regular MLS KeyPackages require two signatures: One over the LeafNode and one over the KeyPackage around it. This draft introduces a LeafNode component that contains a hash over the KeyPackage fields surrounding the LeafNode. As a consequence, verifying the LeafNode also verifies the KeyPackage.

For Commits with an UpdatePath or Update Proposals (sent as PublicMessage or PrivateMessage) the issue is similar: One signature covers the LeafNode and one signature covers the majority of the struct that ends up being sent over the wire. This draft proposes new types of PublicMessage and PrivateMessage with only one signature, although the signature can only be omitted for Commits that contain an UpdatePath and for Commits and Update Proposals if the LeafNode doesn’t change the sender’s signature public key.

Saving a signature can result in a significant decrease in computational or bandwidth cost, especially in the context of PQ-secure signature schemes such as ML-DSA, where signatures are multiple orders of magnitude larger than those of most non-PQ signature schemes.

2. New MLSMessage variants

This document specifies three new entries for the IANA WireFormat registry, which results in the following changes to the MLSMessage struct as defined in [RFC9420].

```
struct {
  ...
  select (MLSMessage.wire_format) {
    ...
    case mls_os_key_package:
      OneSignatureKeyPackage key_package;
    case mls_os_private_message:
      OSPrivateMessage private_message;
    case mls_os_public_message:
      OSPublicMessage public_message;
  };
} MLSMessage;
```

See Section 3 for the definition of OneSignatureKeyPackage and Section 4 for the definitions of OSPrivateMessage and OSPublicMessage.

3. One Signature KeyPackages

A OneSignatureKeyPackage functions like a regular KeyPackage, except that it’s partitioned into two components: The OuterKeyPackage and the LeafNode. The OuterKeyPackage contains all fields of a regular KeyPackage except the LeafNode and the signature.

```
struct {
  ProtocolVersion version;
  CipherSuite cipher_suite;
  HPKEPublicKey init_key;
  Extension extensions<V>;
} OuterKeyPackage

struct {
  OuterKeyPackage outer_key_package;
  LeafNode leaf_node;
} OneSignatureKeyPackage
```

3.1. Creating a OneSignatureKeyPackage

A OneSignatureKeyPackage is created like a regular KeyPackage with the following exceptions.

* The signature around the outer KeyPackage is omitted

* An app_data_dictionary extension is added to the LeafNode (if not already present)

* An OuterKeyPackageHash component is included in the app_data_dictionary (see Section 3.2)

The original purpose of the signature over the KeyPackage is now served by the signature over the LeafNode, which by inclusion of the OuterKeyPackageHash provides authenticity for both the LeafNode itself _and_ the OuterKeyPackage around it.

3.2. OuterKeyPackage hash component

```
struct {
  opaque outer_key_package_hash;
} OuterKeyPackageHash
```

The OuterKeyPackageHash can be created by hashing the TLS-serialized outer_key_package of a OneSignatureKeyPackage using the hash function of its ciphersuite.

A OuterKeyPackageHash is only valid if two conditions are met.

* The leaf_node_source of the LeafNode is KeyPackage

* If the LeafNode is verified in the context of a OneSignatureKeyPackage, the outer_key_package_hash is the hash of the outer_key_package of that OneSignatureKeyPackage.

3.3. Processing a OneSignatureKeyPackage

Recipients of a OneSignatureKeyPackage process is like a regular KeyPackage with two exceptions

* There is no signature over the outer KeyPackage to verify

* The app_data_dictionary extension in the leaf_node must contain a valid OuterKeyPackageHash as defined in Section 3.2 under the component_id TBD.

4. One Signature Commits and Update Proposals

MLS PublicMessages and PrivateMessages carrying a Commit with UpdatePath or an Update Proposal can also be created with only one signature as long as the signature key of the sender is not changed by the respective operation. The resulting structs are called OSPublicMessage and OSPrivateMessage respectively.

The principle behind saving the signature is the same as for OneSignatureKeyPackage. The signature over the whole struct is omitted and instead a hash over the otherwise signed part of the struct (minus the LeafNode) is placed as a component in the LeafNode of the UpdatePath or the Update Proposal.

4.1. Changes in framing

The core change in framing an OSPublicMessage or OSPrivateMessage as compared to its regular counterparts is that the FramedContentAuthData is replaced by the OSFramedContentAuthData, where the latter lacks the signature that is part of the former. Other framing structs are changed as a result in that they contain an OSFramedContentAuthData struct instead of a FramedContentAuthData struct.

```
   struct {
     select (FramedContent.content_type) {
       case commit:
         /*
           MAC(confirmation_key,
             GroupContext.confirmed_transcript_hash)
         */
         MAC confirmation_tag;
       case application:
       case proposal:
         struct{};
     };
   } OSFramedContentAuthData;

   struct {
     WireFormat wire_format;
     FramedContent content;
     OSFramedContentAuthData auth;
   } OSAuthenticatedContent;

   struct {
     select (PrivateMessage.content_type) {
       case application:
         opaque application_data<V>;

         case proposal:
           Proposal proposal;

         case commit:
           Commit commit;
     };

     OSFramedContentAuthData auth;
     opaque padding[length_of_padding];
   } OSPrivateMessageContent;

   struct {
     FramedContent content;
     OSFramedContentAuthData auth;
     select (PublicMessage.content.sender.sender_type) {
         case member:
             MAC membership_tag;
         case external:
         case new_member_commit:
         case new_member_proposal:
             struct{};
     };
   } OSPublicMessage;
```

4.2. Changes in membership tag and transcript hash computation

The structs involved in membership tag and transcript hash computation also change slightly.

```
struct {
  FramedContentTBS content_tbs;
  OSFramedContentAuthData auth;
} OSAuthenticatedContentTBM;
```

For OSPublicMessages, the membership_tag is computed over OSAuthenticatedContentTBM instead of the regular AuthenticatedContentTBM.

```
struct {
  WireFormat wire_format;
  FramedContent content; /* with content_type == commit */
} OSConfirmedTranscriptHashInput;
```

Due to the changes in framing, the transcript hash for OSPublicMessages and OSPrivateMessages is computed over OSConfirmedTranscriptHashInput instead of the regular ConfirmedTranscriptHash. Since the LeafNode is within FramedContent and its signature covers what the original signature would have covered, this does not affect transcript coverage.

4.3. Outer update hash component

The OuterUpdateHash component ensures that the signature over the LeafNode covers whatever the omitted signature would have covered.

```
struct {
  opaque outer_update_hash;
} OuterUpdateHash

struct {
  opaque group_id<V>;
    uint64 epoch;
    Sender sender;
    opaque authenticated_data<V>;

    ContentType content_type;
    select (FramedContent.content_type) {
      case proposal:
        ProposalType proposal_type;
      case commit:
        ProposalOrRef proposals<V>;
        UpdatePathNode nodes<V>;
      case application:
        struct {};
    };
} OuterFramedContent

struct {
  ProtocolVersion version = mls10;
  WireFormat wire_format;
  OuterFramedContent content;
  GroupContext context;
} OSFramedContentTBH
```

The outer_update_hash MUST be a hash over the message's TLS-serialized OSFramedContentTBH using the hash function of the group's ciphersuite.

OSFramedContentTBH is the same as the FramedContentTBS struct defined in [RFC9420], except that it always contains a GroupContext (because Commits and Update Proposals only have member or new_member_commit as sender_type) and that it contains an OuterFramedContent instead of a regular FramedContent.

OuterFramedContent in turn is the same as FramedContent except that, depending on content_type, it either contains only content relevant to a Commit with an UpdatePath or an Update Proposal. For Commits, the proposals MUST be the same as in the message's Commit and the nodes MUST be equal to the nodes in the Commit's UpdatePath. For Update Proposals, the proposal_type MUST be update. The proposal type is the only relevant content, as the Update Proposal otherwise only consists of the (omitted) LeafNode.

In both cases, the actual LeafNode is omitted to prevent a circular dependency when computing the outer_update_hash for inclusion in said LeafNode.

4.4. Creating an OSPublicMessage or OSPrivateMessage

Clients create an OSPublicMessage or OSPrivateMessage just like their non-OS counterpart with three exceptions:

* The signature in FramedContentData is omitted by using OSFramedContentData and the other modified structs introduced in Section 4.1 instead of their non-OS counterparts.

* An app_data_dictionary is included in the LeafNode (either in the UpdatePath or the Update Proposal) and within it an OuterUpdateHash component (see Section 4.3).

* The membership tag and transcript hash are computed as specified in Section 4.2.

4.5. Processing an OSPublicMessage or OSPrivateMessage

OSPublicMessages and OSPrivateMessages are processed like their regular counterparts with the following exceptions.

* One of the following MUST be true

  - content_type = commit and the Commit contained within MUST have an UpdatePath

  - content_type = proposal and proposal_type = update

* The signature public key in the LeafNode MUST be the same as the sender’s current LeafNode. This MUST also be true for Commits with sender_type = new_member_commit that contain a Remove Proposal targeting the sender’s original leaf.

* The LeafNode (either in the UpdatePath or in the Update Proposal) MUST contain an app_data_dictionary extension with a valid OuterUpdateHash component as specified in Section 4.3.

* Membership tag and transcript hash are computed as specified in Section 4.2.

The second check for signature public key equality is necessary to ensure that the authentication properties of one-signature message variants are equivalent to their counterparts defined in RFC9420. Clients that want to change their signature public key MUST use the normal (external) Commits and Update Proposals to ensure that the new signature public key is signed by the old one.

5. Security Considerations

Security considerations around the one signature variants are the same as those of their regular MLS counterparts, except their content should not be trusted until the signature of the LeafNode was verified and the OuterKeyPackageHash or OuterUpdateHash component was validated.

6. IANA Considerations

6.1. Component Types

This document requests the addition of two new Component Types under the heading of "Messaging Layer Security".

6.1.1. OuterKeyPackageHash

The OuterKeyPackageHash component contains a hash over the outer parts of a OneSignatureKeyPackage.

* Value: TBD (suggested value 0x0009)
* outer_key_package_hash
* Where: LN
* Recommended: Y
* Reference: TBD

6.1.2. OuterUpdateHash

The OuterUpdateHash component contains a hash over the parts of an OSPublicMessage or OSPrivateMessage that would otherwise be covered by a signature.

* Value: TBD (suggested value 0x000C)
* outer_update_hash
* Where: LN

* Recommended: Y

* Reference: TBD

6.2. WireFormat

This document requests the addition of three new WireFormats under the heading of "Messaging Layer Security".

6.2.1. MLSOneSignatureKeyPackage

The mls_os_key_package allows saving the creation and verification of a signature that is necessary when creating a regular KeyPackage.

* Value: TBD

* Name: mls_os_key_package

* Recommended: Y

* Reference: TBD

6.2.2. MLSOneSignaturePrivateMessage

The mls_os_private_message allows saving the creation and verification of a signature that is necessary when creating a regular PrivateMessage that either contains a Commit with an UpdatePath or an Update Proposal.

* Value: TBD

* Name: mls_os_private_message

* Recommended: Y

* Reference: TBD

6.2.3. MLSOneSignaturePublicMessage

The mls_os_public_message allows saving the creation and verification of a signature that is necessary when creating a regular PublicMessage that either contains a Commit with an UpdatePath or an Update Proposal.

* Value: TBD

* Name: mls_os_public_message

* Recommended: Y

* Reference: TBD

7. 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>.

Acknowledgments

TODO acknowledge.

Authors' Addresses

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

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