Workgroup: Network Working Group
Internet-Draft: draft-ietf-mls-extensions-00
Published: 25 November 2022
Intended Status: Informational
Expires: 29 May 2023
Authors: R. Robert
Phoenix R&D

The Messaging Layer Security (MLS) Extensions

Abstract

This document describes extensions to the Messaging Layer Security (MLS) protocol.

Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at https://github.com/mlswg/mls-extensions.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes extensions to [mls-protocol] that are not part of the main protocol specification. The protocol specification includes a set of core extensions that are likely to be useful to many applications. The extensions described in this document are intended to be used by applications that need to extend the MLS protocol.

1.1. Change Log

RFC EDITOR PLEASE DELETE THIS SECTION.

draft-00

- *Initial adoption of draft-robert-mls-protocol-00 as a WG item.

- *Add Targeted Messages extension (*)

2. Extensions

2.1. AppAck

Type: Proposal

2.1.1. Description

An AppAck proposal is used to acknowledge receipt of application messages. Though this information implies no change to the group, it is structured as a Proposal message so that it is included in the group's transcript by being included in Commit messages.

```
struct {
    uint32 sender;
    uint32 first_generation;
    uint32 last_generation;
} MessageRange;

struct {
    MessageRange received_ranges<V>;
} AppAck;
```

An AppAck proposal represents a set of messages received by the sender in the current epoch. Messages are represented by the sender and generation values in the MLSCiphertext for the message. Each MessageRange represents receipt of a span of messages whose generation values form a continuous range from first_generation to last_generation, inclusive.

AppAck proposals are sent as a guard against the Delivery Service dropping application messages. The sequential nature of the generation field provides a degree of loss detection, since gaps in the generation sequence indicate dropped messages. AppAck completes this story by addressing the scenario where the Delivery Service drops all messages after a certain point, so that a later generation is never observed. Obviously, there is a risk that AppAck messages could be suppressed as well, but their inclusion in the transcript means that if they are suppressed then the group cannot advance at all.

The schedule on which sending AppAck proposals are sent is up to the application, and determines which cases of loss/suppression are detected. For example:

* The application might have the committer include an AppAck proposal whenever a Commit is sent, so that other members could know when one of their messages did not reach the committer.

* The application could have a client send an AppAck whenever an application message is sent, covering all messages received since its last AppAck. This would provide a complete view of any losses experienced by active members.

* The application could simply have clients send AppAck proposals on a timer, so that all participants' state would be known.

An application using AppAck proposals to guard against loss/ suppression of application messages also needs to ensure that AppAck messages and the Commits that reference them are not dropped. One way to do this is to always encrypt Proposal and Commit messages, to make it more difficult for the Delivery Service to recognize which messages contain AppAcks. The application can also have clients enforce an AppAck schedule, reporting loss if an AppAck is not received at the expected time.

2.2. Targeted messages

2.2.1. Description

MLS application messages make sending encrypted messages to all group members easy and efficient. Sometimes application protocols mandate that messages are only sent to specific group members, either for privacy or for efficiency reasons.

Targeted messages are a way to achieve this without having to create a new group with the sender and the specific recipients - which might not be possible or desired. Instead, targeted messages define the format and encryption of a message that is sent from a member of an existing group to another member of that group.

The goal is to provide a one-shot messaging mechanism that provides confidentiality and authentication.

Targeted Messages reuse mechanisms from [mls-protocol], in particular [hpke].

2.2.2. Format

This extensions introduces a new message type to the MLS protocol, TargetedMessage in WireFormat and MLSMessage:

```
enum {
  ...
  mls_targeted_message(6),
  ...
  (255)
} WireFormat;

struct {
    ProtocolVersion version = mls10;
    WireFormat wire_format;
    select (MLSMessage.wire_format) {
        ...
        case mls_targeted_message:
            TargetedMessage targeted_message;
    }
} MLSMessage;
```

The TargetedMessage message type is defined as follows:

```
struct {
  opaque group_id<V>;
  uint64 epoch;
  uint32 recipient_leaf_index;
  opaque authenticated_data<V>;
  opaque encrypted_sender_auth_data<V>;
  opaque hpke_ciphertext<V>;
} TargetedMessage;

enum {
  hpke_auth_psk(0),
  signature_hpke_psk(1),
} TargetedMessageAuthScheme;

struct {
  uint32 sender_leaf_index;
  TargetedMessageAuthScheme authentication_scheme;
  select (authentication_scheme) {
    case HPKEAuthPsk:
    case SignatureHPKEPsk:
      opaque signature<V>;
  }
  opaque kem_output<V>;
} TargetedMessageSenderAuthData;

struct {
  opaque group_id<V>;
  uint64 epoch;
  uint32 recipient_leaf_index;
  opaque authenticated_data<V>;
  TargetedMessageSenderAuthData sender_auth_data;
} TargetedMessageTBM;

struct {
  opaque group_id<V>;
  uint64 epoch;
  uint32 recipient_leaf_index;
  opaque authenticated_data<V>;
  uint32 sender_leaf_index;
  TargetedMessageAuthScheme authentication_scheme;
  opaque kem_output<V>;
  opaque hpke_ciphertext<V>;
} TargetedMessageTBS;

struct {
  opaque group_id<V>;
  uint64 epoch;
  opaque label<V> = "MLS 1.0 targeted message psk";
} PSKId;
```

Note that TargetedMessageTBS is only used with the TargetedMessageAuthScheme.SignatureHPKEPsk authentication mode.

2.2.3. Encryption

Targeted messages use HPKE to encrypt the message content between two leaves. The HPKE keys of the LeafNode are used to that effect, namely the encryption_key field.

In addition, TargetedMessageSenderAuthData is encrypted in a similar way to MLSSenderData as described in section 7.3.2 in [mls-protocol]. The TargetedMessageSenderAuthData.sender_leaf_index field is the leaf index of the sender. The TargetedMessageSenderAuthData.authentication_scheme field is the authentication scheme used to authenticate the sender. The TargetedMessageSenderAuthData.signature field is the signature of the TargetedMessageTBS structure. The TargetedMessageSenderAuthData.kem_output field is the KEM output of the HPKE encryption.

The key and nonce provided to the AEAD are computed as the KDF of the first KDF.Nh bytes of the hpke_ciphertext generated in the following section. If the length of the hpke_ciphertext is less than KDF.Nh, the whole hpke_ciphertext is used. In pseudocode, the key and nonce are derived as:

``` sender_auth_data_secret = MLS-Exporter("targeted message sender auth data", "", KDF.Nh)

ciphertext_sample = hpke_ciphertext[0..KDF.Nh-1]

sender_data_key = ExpandWithLabel(sender_auth_data_secret, "key", ciphertext_sample, AEAD.Nk) sender_data_nonce = ExpandWithLabel(sender_auth_data_secret, "nonce", ciphertext_sample, AEAD.Nn) ```

The Additional Authenticated Data (AAD) for the SenderAuthData ciphertext is the first three fields of TargetedMessage:

```
struct {
  opaque group_id<V>;
  uint64 epoch;
  uint32 recipient_leaf_index;
} SenderAuthDataAAD;
```
```

2.2.3.1. Padding

The TargetedMessage structure does not include a padding field. It is the responsibility of the sender to add padding to the message as used in the next section.

2.2.4. Authentication

For ciphersuites that support it, HPKE mode_auth_psk is used for authentication. For other ciphersuites, HPKE mode_psk is used along with a signature. The authentication scheme is indicated by the authentication_scheme field in TargetedMessageContent. See Section 2.2.5 for more information.

For the PSK part of the authentication, clients export a dedicated secret:

```
targeted_message_psk = MLS-Exporter("targeted message psk", "",
KDF.Nh)
```

Th functions SealAuth and OpenAuth are defined in [hpke]. Other functions are defined in [mls-protocol].

2.2.4.1. Authentication with HPKE

The sender MUST set the authentication scheme to TargetedMessageAuthScheme.HPKEAuthPsk.

The sender then computes the following:

```
(kem_output, hpke_ciphertext) =
SealAuthPSK(receiver_node_public_key, group_context,
targeted_message_tbm, message, targeted_message_psk, psk_id,
sender_node_private_key)
```

The recipient computes the following:

```
message = OpenAuthPSK(kem_output, receiver_node_private_key,
group_context, targeted_message_tbm, hpke_ciphertext,
targeted_message_psk, psk_id, sender_node_public_key)
```

2.2.4.2. Authentication with signatures

The sender MUST set the authentication scheme to TargetedMessageAuthScheme.SignatureHPKEPsk. The signature is done using the signature_key of the sender's LeafNode and the corresponding signature scheme used in the group.

The sender then computes the following:

``` (kem_output, hpke_ciphertext) = SealPSK(receiver_node_public_key, group_context, targeted_message_tbm, message, targeted_message_psk, epoch)

signature = SignWithLabel(., "TargetedMessageTBS", targeted_message_tbs) ```

The recipient computes the following:

message = OpenPSK(kem_output, receiver_node_private_key, group_context, targeted_message_tbm, hpke_ciphertext, targeted_message_psk, epoch)

The recipient MUST verify the message authentication:

VerifyWithLabel.verify(sender_leaf_node.signature_key, "TargetedMessageTBS", targeted_message_tbs, signature)

#### 2.2.5. Guidance on authentication schemes

If the group's ciphersuite does not support HPKE mode_auth_psk, implementations MUST choose TargetedMessageAuthScheme.SignatureHPKEPsk.

If the group's ciphersuite does support HPKE mode_auth_psk, implementations CAN choose TargetedMessageAuthScheme.HPKEAuthPsk if better efficiency and/or repudiability is desired. Implementations SHOULD consult [hpke-security-considerations] beforehand.

#### 2.2.6. Security considerations

In addition to the sender authentication, Targeted Messages are authenticated by using a preshared key (PSK) between the sender and the recipient. The PSK is exported from the group key schedule using the label "targeted message psk". This ensures that the PSK is only valid for a specific group and epoch, and the Forward Secrecy and Post-Compromise Security guarantees of the group key schedule apply to the targeted messages as well. The PSK also ensures that an attacker needs access to the private group state in addition to the HPKE/signature's private keys. This improves confidentiality guarantees against passive attackers and authentication guarantees against active attackers.

## 3. IANA Considerations

This document requests the addition of various new values under the heading of "Messaging Layer Security". Each registration is organized under the relevant registry Type.
```

RFC EDITOR: Please replace XXXX throughout with the RFC number assigned to this document

3.1. MLS Extension Types

3.1.1. targeted_messages_capability MLS Extension

The targeted_messages_capability MLS Extension Type is used in the capabilities field of LeafNodes to indicate the support for the Targeted Messages Extension. The extension does not carry any payload.

Template:

*Value: 0x0006

*Name: targeted_messages_capability

*Message(s): LN: This extension may appear in LeafNode objects

*Recommended: Y

*Reference: RFC XXXX

3.1.2. targeted_messages MLS Extension

The targeted_messages MLS Extension Type is used inside GroupContext objects. It indicates that the group supports the Targeted Messages Extension.

Template:

*Value: 0x0007

*Name: targeted_messages

*Message(s): GC: This extension may appear in GroupContext objects

*Recommended: Y

*Reference: RFC XXXX

3.2. MLS Proposal Types

3.2.1. AppAck Proposal

Template:

*Value: 0x0008

*Name: app_ack

*Recommended: Y

*Path Required: Y

*Reference: [RFC XXXX]

4. Informative References

[hpke] "Hybrid Public Key Encryption", n.d., <https://www.rfc-editor.org/rfc/rfc9180.html](https://www.rfc-editor.org/rfc/rfc9180.html>.

[hpke-security-considerations] "HPKE Security Considerations", n.d., <https://www.rfc-editor.org/rfc/rfc9180.html#name-key-compromise-impersonatio](https://www.rfc-editor.org/rfc/rfc9180.html#name-key-compromise-impersonatio>.

[mls-protocol] "The Messaging Layer Security (MLS) Protocol", n.d., <https://datatracker.ietf.org/doc/draft-ietf-mls-protocol/](https://datatracker.ietf.org/doc/draft-ietf-mls-protocol/>.

Author's Address

Raphael Robert
Phoenix R&D

Email: ietf@raphaelrobert.com