Network Working Group
Internet-Draft
Intended status: Informational
Expires: 8 January 2026

R. Robert
K. Kohbrok
Phoenix R&D

7 July 2025

MIMI Attachments
draft-robert-mimi-attachments-04

Abstract

This document describes MIMI Attachments.

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 8 January 2026.

Table of Contents

1. Introduction

Attachments are known from email, where they are used to attach files to an email message. Attachments are also used in instant messaging, where they are used to attach files, images, videos, etc.

The main difference between attachments and other messages is that attachments are bigger in size and their content is typically saved in files on the client side. Since downloading attachments can be expensive for instant messaging clients, it is common to download attachments only on demand, e.g. when the user clicks on the attachment.

draft-mimi-content defines various message content formats that can be used in either MLS application messages or attachments. This document describes how attachments are used in MIMI.

1.1. Change Log

draft-01

* Version bump to prevent expiration

draft-02

* Version bump to prevent expiration

draft-03

* Version bump to prevent expiration

draft-04

* Version bump to prevent expiration

2. Access control & quotas

Attachments are bound to a specific MLS group. Only members of the group can access the attachment. The Delivery Service enforces this access control.

The Delivery Service can keep track of the size of attachments for each group, its total number, who uploaded them, etc. This information can be used to enforce quotas on the size of attachments, the number of attachments, etc.

3. Distribution

Since attachments are not directly distributed as messages, they are distributed over the MIMI Delivery Service.

3.1. Uploading attachments

When a client wants to send an attachment, it first uploads the attachment to the Delivery Service. The Delivery Service returns a token that can be used to download the attachment. The client then sends a message that contains the URL and cryptographic key material to the intended recipients.

TODO: This probably become part of _draft-robert-mimi-delivery-service_.

The client first encrypts the attachment as descibed in Section 3.6 and then uploads it to the DS using the following message:

```
struct {
  opaque group_id<V>;
  opaque content<V>;
} UploadAttachmentRequest;
```

Upon succes, the Delivery Service returns the following message:

```
struct {
  opaque token<V>;
} UploadAttachmentResponse;
```

3.2. Sending attachments

Once clients have uploaded an attachment to the Delivery Service, they can send messages that contain a reference to the attachment. The client sets the contentType of the body as follows:

```
body.contentType = "message/external-body; access-type=token;" +
  "token=<token>; nonce=<nonce>; hash=<hash>";
```

The token is the token that was received from the Delivery Service as part of the UploadAttachmentResponse message. The nonce is a random byte sequence that is used in the key derivation descibed in Section 3.6. The hash is the hash of the plaintext attachment content, as defined in Section 3.6. The hash acts as a commitment hash for the attachment content.

TODO: This needs to be better aligned with draft-ietf-mimi-content.

3.3. Receiving attachments

When a client receives a message that contains a reference to an attachment, it first downloads the attachment from the Delivery Service using the following message:

```
struct {
  opaque group_id<V>;
  opaque token<V>;
} DownloadAttachmentRequest;
```

Upon succes, the Delivery Service returns the following message:

```
struct {
  opaque content<V>;
} DownloadAttachmentResponse;
```

The client then decrypts the attachment using the nonce for the key derivation described in Section 3.6.

3.4. Deleting attachments

Attachments can either be deleted by members of the group (according to a predefined policy), or by the Delivery Service after a predefined time.

3.5. Access control

Attachments are bound to a specific MLS group. Only members of the group can access the attachment. The Delivery Service enforces this access control.

Attachments can however be shared between groups by either re-uploading the attachment to the Delivery Service or by asking the Delivery Service to copy an attachment to another group. Copying an attachment to another group is only possible if both groups are

located on the same Delivery Service, and the user has access to both groups. It is an optimization to avoid re-uploading the same attachment multiple times, but otherwise follows the same rules as uploading a new attachment.

3.6. Encryption

Attachments are encrypted using a nonce/key pair that is exported from the MLS group exporter and combined with a randomly generated byte sequence on the sender's side, and context data.

```
struct {
   opaque attachment_hash;
   uint64 sender_index;
} AttachmentContext;
```

Since attachments are specific to an MLS group, we reuse components the cipher suite of the group:

* KDF

* Hash

* AEAD

* ExpandWithLabel

The sender encrypts the attachment as follows:

* Compute the hash of the attachment data:

```
attachment_hash = Hash(plaintext)
```

* Generate a random byte sequence nonce of length KDF.Nh bytes.

* Set the context to the following values: AttachmentContext.hash to the previously computed attachment_hash and AttachmentContext.sender_index to the leaf index of the sender in the group.

* Compute a secret attachment_exporter_secret from the MLS group exporter using the previously computed context and extracts attachment_secret from it and the nonce:

```
   attachment_exporter_secret =
      MLS-Exporter("MIMI attachment", context, KDF.Nh)

   attachment_secret =
      KDF.Extract(attachment_exporter_secret, nonce)
```

* Compute the nonce/key pair:

```
   attachment_nonce =
      ExpandWithLabel(attachment_secret, "nonce", "", AEAD.Nn)

   attachment_key =
      ExpandWithLabel(attachment_secret, "key", "", AEAD.Nk)
```

* Encrypt the attachment using the key/nonce pair.

Authors' Addresses

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

Konrad Kohbrok
Phoenix R&D
Email: konrad.kohbrok@datashrine.de