Workgroup: MIMI
Internet-Draft: draft-mahy-mimi-content-02
Published: 13 March 2023
Intended Status: Informational
Expires: 14 September 2023
Authors: R. Mahy
Wire

More Instant Messaging Interoperability (MIMI) message content

Abstract

This document describes content semantics common in Instant Messaging (IM) systems and describes an example profile suitable for instant messaging interoperability of messages end-to-end encrypted inside the MLS (Message Layer Security) Protocol.

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 14 September 2023.

Table of Contents

1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2219].

The terms MLS client, MLS group, and KeyPackage have the same meanings as in the MLS protocol [I-D.ietf-mls-protocol].

2. Introduction

MLS [I-D.ietf-mls-protocol] is a group key establishment protocol motivated by the desire for group chat with efficient end-to-end encryption. While one of the motivations of MLS is interoperable standards-based secure messaging, the MLS protocol does not define or prescribe any format for the encrypted "application messages" encoded by MLS. The development of MLS was strongly motivated by the needs of a number of Instant Messaging (IM) systems, which encrypt messages end-to-end using variations of the Double Ratchet protocol [DoubleRatchet].

End-to-end encrypted instant messaging was also a motivator for the Common Protocol for Instant Messaging (CPIM) [RFC3862], however the model used at the time assumed standalone encryption of each message using a protocol such as S/MIME [RFC8551] or PGP [RFC3156] to interoperate between IM protocols such as SIP [RFC3261] and XMPP [RFC6120]. For a variety of practical reasons, interoperable end-to-end encryption between IM systems was never deployed commercially.

There are now several instant messaging vendors implementing MLS, and the MIMI (More Instant Messaging Interoperability) Working Group is charted to standardize an extensible interoperable messaging format for common features to be conveyed "inside" MLS application messages. Most of these features can reuse the semantics of previously-defined URIs, message headers, and media types. This document represents a solution to one part of the MIMI problem outline [I-D.mahy-mimi-problem-outline].

This document assumes that MLS clients advertise media types they support and can determine what media types are required to join a specific MLS group using the content advertisement extensions in [I-D.ietf-mls-extensions]. It allows implementations to define MLS groups with different media type requirements and allows MLS clients to send extended or proprietary messages that would be interpreted by some members of the group while assuring that an interoperable end-to-end encrypted baseline is available to all members, even when the group spans multiple systems or vendors.

Below is a list of some features commonly found in IM group chat systems:

*plain text and rich text messaging
*mentions
*replies
*reactions

*edit or delete previously sent messages
*expiring messages
*delivery notifications
*read receipts
*shared files/audio/videos
*calling / conferencing
*message threading

3. Overview

3.1. Naming schemes

IM systems have a number of types of identifiers. These are described in detail in [I-D.mahy-mimi-identity]. A few of these used in this document are:

*handle identifier (external, friendly representation). This is the type of identifier described later as the senderUserUrl in the examples, which is analogous to the From header in email.
*client/device identifier (internal representation). This is the type of identifier described as the senderClientUrl in the examples.
*group or conversation or channel name (either internal or external representation). This is the type of identifier described as the MLS group URL in the examples.

This proposal relies on URIs for naming and identifiers. All the example use the im: URI scheme (defined in [RFC3862]), but any instant messaging scheme could be used.

3.2. Message Container

Most common instant messaging features are expressed as individual messages. A plain or rich text message is obviously a message, but a reaction (ex: like), a reply, editing a previous message, deleting an earlier message, and read receipts are all typically modeled as another message with different properties.

This document describes the semantics of a message container, which contains a message ID and timestamp and represents most of these previously mentioned messages. The container typically carries one or more body parts with the actual message content (for example, an emoji used in a reaction, a plain text or rich text message or reply, a link, or an inline image).

3.3. Message Status Report

This document also describes the semantics of a status report of other messages. The status report has a timestamp, but does not have a message ID of its own. Because some messaging systems deliver

messages in batches and allow a user to mark several messages read at a time, the report format allows a single report to convey the read/delivered status of multiple messages (by message ID) within the same MLS group at a time.

4. MIMI Content Container Message Semantics

Each MIMI Content message is a container format with three categories of information:

- the required message ID and timestamp fields,
- the message behavior fields (which can have default or empty values), and
- the body part(s) and associated parameters

To focus on the semantics of a MIMI Content message, we use C/C++ struct notation to describe its data fields. These fields are numbered in curly braces for reference in the text. We do not propose any specific syntax for the format, but two reasonable constraints are:

- we do not want to scan body parts to check for boundary marker collisions. This rules out using multipart MIME types.
- we do not want to base64 encode body parts with binary media types (ex: images). This rules out using JSON to carry the binary data.

4.1. Required Fields

Every MIMI content message has a message ID {1}. The message ID has a local part and a domain part. The domain part corresponds to the domain of the sender of the message. The local part must be unique among all messages sent in the domain. Using a UUID for the local part is RECOMMENDED.

```
struct MessageId {
    Octets localPart;
    String domain;
};

struct MimiContent {
    MessageId messageId;         // required value {1}
    double timestamp;            // seconds since 01-Jan-1970 {2}
    MessageId inReplyTo;         // {3}
    MessageId replaces;          // {4}
    MessageId threadId;          // {5}
    uint32 expires;              // 0 = does not expire {6}
    NestablePart body;           // {7}
};
```

Every MIMI content message has a timestamp {2}, represented as the number of (fractional) seconds since the start of the UNIX epoch (01-Jan-1970 00:00:00 UTC).

4.2. Message Behavior Fields

The inReplyTo {3} data field indicates that the current message is a related continuation of the message ID of another message sent in the same MLS group. For all three message behavior fields which take a message ID, if the field is empty (i.e. both the message ID localPart and the domain are zero length), the receiver assumes that the current message has not identified any special relationship with another previous message.

The replaces {4} data field indicates that the current message is a replacement or update to a previous message whose message ID is in the replaces data field. It is used to edit previously-sent messages, delete previously-sent messages, and adjust reactions to messages to which the client previously reacted.

The threadId {5} data field indicates that the current message is part of a logical thread of messages which begins with a message with the message ID specified in the threadId data field.

The expires {6} data field is a hint from the sender to the receiver that the message should be locally deleted and disregarded at a specific timestamp in the future. Indicate a message with no specific expiration time with the value zero. The data field is an unsigned integer number of seconds after the start of the UNIX epoch. Using an 32-bit unsigned integer allows expiration dates until the year 2106. Note that specifying an expiration time provides no assurance that the client actually honors or can honor the expiration time, nor that the end user didn't otherwise save the expiring message (ex: via a screenshot).

4.3. Message Bodies

Every MIMI content message has a body {7} which can have multiple, possibly nested parts. A body with zero parts is permitted when deleting or unliking {8}. When there is a single body, its IANA media type, subtype, and parameters are included in the contentType field {9}.

```
typedef std::monostate NullPart; // {8}

struct SinglePart {
    String contentType;   // An IANA media type {9}
    Octets content;       // The actual content
};

typedef std::vector<NestablePart> MultiParts;

enum PartSemantics { // {10}
    nullPart = 0,
    singlePart = 1, // the bodyParts is a single part
    chooseOne = 2,  // receiver picks exactly one part to process
    singleUnit = 3  // receiver processes all parts as single unit
    processAll = 4 // receiver processes all parts individually
};

enum Disposition {
    unspecified = 0,
    render = 1,
    reaction = 2,
    profile = 3,
    inline = 4,
    icon = 5,
    attachment = 6,
    session = 7
};

struct NestablePart {
    Disposition disposition;  // {11}
    String language;          // {12}
    uint16 partIndex;         // {13}
    PartSemantics partSemantics;
    std::variant<NullPart, SinglePart, MultiParts> part;
};
```

With some types of message content, there are multiple media types associated with the same message which need to be rendered together, for example a rich-text message with an inline image. With other messages, there are multiple choices available for the same content, for example a choice among multiple languages, or between two different image formats. The relationship semantics among the parts is specified as an enumeration {10}.

The nullPart part semantic is used when there is no body part—(U+2014)for deleting and unliking. The singlePart part semantic is used when there is a single body part.

The chooseOne part semantic is roughly analogous to the semantics of the multipart/alternative media type, except that the ordering of the nested body parts is merely a preference of the sender. The receiver can choose the body part among those provided according to its own policy.

The singleUnit part semantic is roughly analogous to the semantics of the multipart/related media type, in that all the nested body parts at this level are part of a single entity (for example, a rich text message with an inline image). If the receiver does not understand even one of the nested parts at this level, the receiver should not process any of them.

The processAll part semantic is roughly analogous to the semantics of the multipart/mixed media type. The receiver should process as many of the nested parts at this level as possible. For example, a rich text document with a link, and a preview image of the link target could be expressed using this semantic. Processing the preview image is not strictly necessary for the correct rendering of the rich text part.

The disposition {11} and language {12} of each part can be specified for any part, including for nested parts. The disposition represents the intended semantics of the body part or a set of nested parts. It is inspired by the values in the Content-Disposition MIME header [RFC2183]. The render and inline dispositions mean that the content should be rendered "inline" directly in the chat interface. The attachment disposition means that the content is intended to be downloaded by the receiver instead of being rendered immediately. The reaction disposition means that the content is a single reaction to another message, typically an emoji, but which could be an image, sound, or video. The disposition was originally published in [RFC9078], but was incorrectly placed in the Content Disposition Parameters IANA registry instead of in the Content Disposition Values registry. The session disposition means that the content is a description of a multimedia session, or a URI used to join one. The preview disposition means that the content is a sender-generated preview of something, such as the contents of a link. The value of the language data field is an empty string or a comma-separated list of one or more Language-tags as defined in [RFC2382].

Each part also has an part index {13}, which is a zero-indexed, depth-first integer. It is used to efficiently refer to a specific body part (for example, an inline image) within another part. See {Nested body examples} for an example of how the part index is calculated.

4.4. Derived Data Values

In addition to fields which are contained in a MIMI content message, there are also two fields which the implementation can definitely derive (the MLS group ID {14}, and the leaf index of the sender {15}). Many implementations could also determine one or more of: the senders client identifier URL {16}, the user identifier URL of the credential associated with the sender {17}, and the identifier URL for the MLS group {18}.

```
struct MessageDerivedValues {
    Octets mlsGroupId;       // value always available {14}
    uint32 senderLeafIndex;  // value always available {15}
    ImUrl senderClientUrl;   // {16}
    ImUrl senderUserUrl;     // "From" {17}
    ImUrl mlsGroupUrl;       // "To" {18}
};
```

5. Examples

In the following examples, we assume that an MLS group is already established and that either out-of-band or using the MLS protocol or MLS extensions that the following is known to every member of the group:

- The membership of the group (via MLS).
- The identity of any MLS client which sends an application message (via MLS).
- The MLS group ID (via MLS)
- The human readable name(s) of the MLS group, if any (out-of-band or extension).
- Which media types are mandatory to implement (MLS content advertisement extensions).
- For each member, the media types each supports (MLS content advertisement extensions).

Messages sent to an MLS group are delivered to every member of the group active during the epoch in which the message was sent.

5.1. Original Message

In this example, Alice Smith sends a rich-text (Markdown) [RFC7763] message to the Engineering Team MLS group. The following values are derived from the client:

- Sender leaf index: 4
- Sender client ID URL: im:3b52249d-68f9-45ce-8bf5-c799f3cad7ec/0003@example.com
- Sender user handle URL: im:%40alice-smith@example.com
- MLS group ID: 7u4NEqe1tbeBFa0aHdsTgRyD/XOHxD5meZpZS+7aJr8=

*The MLS group URL: im:#engineering_team@example.com
*The MLS group name: "Engineering Team"

Below are the relevant data fields set by the sender:

```
messageId = "28fd19857ad7@example.com";
timestamp = 1644387225.019;  // 2022-02-08T22:13:45-00:00
expires = 0;
body.partIndex = 0;
body.contentType = "text/markdown;charset=utf-8";
body.content = "Hi everyone, we just shipped release 2.0." +
               " __Good work__!";
```

5.2. Reply

A reply message looks similar, but contains the message ID of the original message in the inReplyTo data field. The derived MLS group ID, URL, and name do not change in this example. The derived senderClientId and senderLeafIndex are not especially relevant so all but the user handle URL will be omitted.

*Sender user handle URL: im:%40bob-jones@example.com

The data fields needed:

```
messageId = "e701beee59f9@example.com";
timestamp = 1644387237.492;   // 2022-02-08T22:13:57-00:00
inReplyTo: "28fd19857ad7@example.com";
expires = 0;
body.partIndex = 0;
body.contentType = "text/markdown;charset=utf-8";
body.content = "Right on! _Congratulations_ 'all!";
```

5.3. Reaction

A reaction, uses the Disposition token of reaction. It is modeled on the reaction Content-Disposition token defined in [RFC9078]. Both indicate that the intended disposition of the contents of the message is a reaction.

The content in the sample message is a single Unicode heart character (U+2665). Discovering the range of characters each implementation could render as a reaction can occur out-of-band and is not within the scope of this proposal. However, an implementation which receives a reaction character string it does not recognize could render the reaction as a reply, possibly prefixing with a localized string such as "Reaction: ". Note that a reaction could theoretically even be another media type (ex: image, audio, or video), although not currently implemented in major instant

messaging systems. Note that many systems allow mutiple independent reactions per sender.

*Sender user handle URL: im:cathy-washington@example.com

```
messageId = "1a771ca1d84f@example.com";
timestamp = 1644387237.728;   // 2022-02-08T22:13:57-00:00
inReplyTo: "28fd19857ad7@example.com";
expires = 0;
body.disposition = reaction;
body.partIndex = 0;
body.contentType = "text/plain;charset=utf-8";
body.content = "♥";
```

5.4. Mentions

In instant messaging systems and social media, a mention allows special formatting and behavior when a name, handle, or tag associated with a known group is encountered, often when prefixed with a commercial-at "@" character for mentions of users or a hash "#" character for groups or tags. A message which contains a mention may trigger distinct notifications on the IM client.

We can convey a mention by linking the user handle URI, or group URI in Markdown or HTML rich content. For example, a mention using Markdown is indicated below.

*Sender user handle URL: im:cathy-washington@example.com

```
messageId = "4dcab7711a77@example.com";
timestamp = 1644387243.008;   // 2022-02-08T22:14:03-00:00
expires = 0;
body.partIndex = 0;
body.contentType = "text/markdown;charset=utf-8";
body.content = "Kudos to [@Alice Smith](im:alice-smith@example.com)"
             + "for making the release happen!";
```

The same mention using HTML [W3C.CR-html52-20170808] is indicated below.

```
body.contentType = "text/html;charset=utf-8";
body.content = "<p>Kudos to <a href='im:alice-smith@example.com'>" +
               "@Alice Smith</a> for making the release happen!</p>"
```

5.5. Edit

Unlike with email messages, it is common in IM systems to allow the sender of a message to edit or delete the message after the fact. Typically the message is replaced in the user interface of the

receivers (even after the original message is read) but shows a visual indication that it has been edited.

The replaces data field includes the message ID of the message to edit/replace. The message included in the body is a replacement for the message with the replaced message ID.

Here Bob Jones corrects a typo in his original message:

*Sender user handle URL: im:%40bob-jones@example.com

```
messageId = "89d3472622a4@example.com";
timestamp = 1644387248.621;   // 2022-02-08T22:14:08-00:00
replaces: "e701beee59f9@example.com";
expires = 0;
body.partIndex = 0;
body.contentType = "text/markdown;charset=utf-8";
body.content = "Right on! _Congratulations_ y'all!";
```

5.6. Delete

In IM systems, a delete means that the author of a specific message has retracted the message, regardless if other users have read the message or not. Typically a placeholder remains in the user interface showing that a message was deleted. Replies which reference a deleted message typically hide the quoted portion and reflect that the original message was deleted.

If Bob deleted his message instead of modifying it, we would represent it using the replaces data field, and using an empty body (NullPart), as shown below.

```
messageId = "89d3472622a4@example.com";
timestamp = 1644387248.621;   // 2022-02-08T22:14:08-00:00
replaces: "e701beee59f9@example.com";
expires = 0;
body.partSemantics = nullPart;
body.part = NullPart;
```

5.7. Unlike

In most IM systems, not only is it possible to react to a message ("Like"), but it is possible to remove a previous reaction ("Unlike"). This can be accomplished by deleting the message which creates the original reaction

If Cathy removes her reaction, we would represent the removal using a replaces data field with an empty body, referring to the message which created the reaction, as shown below.

*Sender user handle URL: im:cathy-washington@example.com

```
messageId = "d052cace46f8@example.com";
timestamp = 1644387250.389;   // 2022-02-08T22:14:10-00:00
replaces: "1a771ca1d84f@example.com";
expires = 0;
body.disposition = reaction;
body.partIndex = 0;
body.partSemantics = nullPart;
body.part = NullPart;
```

5.8. Expiring

Expiring messages are designed to be deleted automatically by the receiving client at a certain time whether they have been read or not. As with manually deleted messages, there is no guarantee that an uncooperative client or a determined user will not save the content of the message, however most clients respect the convention.

The expires data field contains the timestamp when the message can be deleted. The semantics of the header are that the message is automatically deleted by the receiving clients at the indicated time without user interaction or network connectivity necessary.

*Sender user handle URL: im:alice-smith@example.com

```
messageId = "5c95a4dfddab@example.com";
timestamp = 1644389403.227;   // 2022-02-08T22:49:06-00:00
expires = 1644390004;         // ~10 minutes later
body.partIndex = 0;
body.contentType = "text/markdown;charset=utf-8";
body.content = "__*VPN GOING DOWN*__\n" +
    "I'm rebooting the VPN in ten minutes unless anyone objects."
```

5.9. Attachments

The message/external-body MIME Type is a convenient way to present a URL to download an attachment which should not be rendered inline. The disposition data field is set to attachment.

```
body.disposition = attachment;
body.contentType = "message/external-body; access-type=URL;" +
  "URL=\"https://example.com/storage/bigfile.m4v\"" +
  "size=708234961";
```

5.10. Conferencing

Joining a conference via URL is also possible. The link could be rendered to the user, requiring a click. Alternatively the disposition could be specified as session which could be processed differently by the client (for example, alerting the user or presenting a dialog box). Further discussion of calling and conferencing functionality is out-of-scope of this document.

```
body.disposition = session;
body.contentType = "message/external-body; access-type=URL;" +
  "URL=\"https://example.com/join/12345\"";
```

5.11. Threading

Clients participating in a thread populate the threadId with the message ID of the first message sent in the thread. The sort order for messages within a thread uses the timestamp field. If more than one message has the same timestamp, the lexically lowest message ID sorts earlier.

5.12. Delivery Reporting and Read Receipts

In instant messaging systems, read receipts typically generate a distinct indicator for each message. In some systems, the number of users in a group who have read the message is subtly displayed and the list of users who read the message is available on further inspection.

Of course, Internet mail has support for read receipts as well, but the existing message disposition notification mechanism defined for email in [RFC8098] is completely inappropriate in this context:

- notifications can be sent by intermediaries
- only one notification can be sent about a single message per recipient
- a human-readable version of the notification is expected
- each notification can refer to only one message
- it is extremely verbose

Instead we would like to be able to include status changes about multiple messages in each report, the ability to mark a message delivered, then read, then unread, then expired for example.

The proposed format below, application/mimi-message-status is sent by one member of an MLS group to the entire group and can refer to multiple messages in that group. The format contains its own timestamp, and a list of message ID / status pairs. As the status at the recipient changes, the status can be updated in a subsequent notification.

```
enum MessageStatus {
    unread = 0,
    delivered = 1,
    read = 2,
    expired = 3,
    deleted = 4,
    hidden = 5,
    error = 6
};

struct PerMessageStatus {
    MessageId messageId;
    MessageStatus status;
};

struct MessageStatusReport {
    double timestamp;
    // a vector of message statuses in the same MLS group
    std::vector<PerMessageStatus> statuses;
};
```

*Sender user handle URL: im:bob-jones@example.com

```
timestamp = 1644284703.227;
statuses[0].messageId = "4dcab7711a77@example.com";
statuses[0].status = read;
statuses[1].messageId = "285f75c46430@example.com";
statuses[1].status = read;
statuses[2].messageId = "c5e0cd6140e6@example.com";
statuses[2].status = unread;
statuses[3].messageId = "5c95a4dfddab@example.com";
statuses[3].status = expired;
```

6. Support for Specific Media Types

6.1. MIMI Required and Recommended media types

As the MIMI Content container is just a container, the plain text or rich text messages sent inside, and any image or other formats needs to be specified. Clients compliant with MIMI MUST be able to receive the following media types:

* application/mimi-content — (U+2014) the MIMI Content container format (described in this document)
* text/plain;charset=utf-8
* text/markdown;variant=GFM — (U+2014) Github Flavored Markdown [GFM])
* message/external-body [RFC4483]

Note that it is acceptable to render the contents of a received markdown document as plain text.

The following MIME types are RECOMMENDED:

- *text/markdown;variant=CommonMark CommonMark
- *text/html
- *application/mimi-message-status (described in this document)
- *image/jpeg
- *image/png

6.2. Use of proprietary media types

As most messaging systems are proprietary, standalone systems, it is useful to allow clients to send and receive proprietary formats among themselves. Using the functionality in the MIMI Content container, clients can send a message using the basic functionality described in this document AND a proprietary format for same-vendor clients simultaneously over the same group with end-to-end encryption. An example is given in the Appendix.

7. IANA Considerations

7.1. MIME subtype registration of application/mimi-message-status

This document proposes registration of a media subtype with IANA.

TBC

7.2. MIME subtype registration of application/mimi-content

This document proposes registration of a media subtype with IANA.

TBC

8. Security Considerations

TBC

9. Normative References

[GFM] GitHub, "GitHub Flavored Markdown Spec, Version 0.29-gfm", 6 March 2019, <https://github.github.com/gfm/>.

[I-D.ietf-mls-extensions] Robert, R., "The Messaging Layer Security (MLS) Extensions", Work in Progress, Internet-Draft, draft-ietf-mls-extensions-01, 13 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-mls-extensions-01>.

[I-D.mahy-mimi-problem-outline]
Mahy, R., "More Instant Messaging Interoperability (MIMI) problem outline", Work in Progress, Internet-Draft, draft-mahy-mimi-problem-outline-01, 24 October 2022, <https://datatracker.ietf.org/doc/html/draft-mahy-mimi-problem-outline-01>.

[RFC2219] Hamilton, M. and R. Wright, "Use of DNS Aliases for Network Services", BCP 17, RFC 2219, DOI 10.17487/RFC2219, October 1997, <https://www.rfc-editor.org/info/rfc2219>.

[RFC2382] Crawley, E., Ed., Berger, L., Berson, S., Baker, F., Borden, M., and J. Krawczyk, "A Framework for Integrated Services and RSVP over ATM", RFC 2382, DOI 10.17487/RFC2382, August 1998, <https://www.rfc-editor.org/info/rfc2382>.

[RFC3862] Klyne, G. and D. Atkins, "Common Presence and Instant Messaging (CPIM): Message Format", RFC 3862, DOI 10.17487/RFC3862, August 2004, <https://www.rfc-editor.org/info/rfc3862>.

[RFC4483] Burger, E., Ed., "A Mechanism for Content Indirection in Session Initiation Protocol (SIP) Messages", RFC 4483, DOI 10.17487/RFC4483, May 2006, <https://www.rfc-editor.org/info/rfc4483>.

[RFC7763] Leonard, S., "The text/markdown Media Type", RFC 7763, DOI 10.17487/RFC7763, March 2016, <https://www.rfc-editor.org/info/rfc7763>.

10. Informative References

[DoubleRatchet] Perrin, T. and M. Marlinspike, "The Double Ratchet Algorithm", 20 November 2016, <https://signal.org/docs/specifications/doubleratchet/>.

[I-D.ietf-mls-protocol]
Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., and K. Cohn-Gordon, "The Messaging Layer Security (MLS) Protocol", Work in Progress, Internet-Draft, draft-ietf-mls-protocol-18, 13 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-mls-protocol-18>.

[I-D.mahy-mimi-identity] Mahy, R., "More Instant Messaging Interoperability (MIMI) Identity Concepts", Work in Progress, Internet-Draft, draft-mahy-mimi-identity-01, 24

October 2022, <https://datatracker.ietf.org/doc/html/draft-mahy-mimi-identity-01>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/info/rfc2046>.

[RFC2183] Troost, R., Dorner, S., and K. Moore, Ed., "Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field", RFC 2183, DOI 10.17487/RFC2183, August 1997, <https://www.rfc-editor.org/info/rfc2183>.

[RFC3156] Elkins, M., Del Torto, D., Levien, R., and T. Roessler, "MIME Security with OpenPGP", RFC 3156, DOI 10.17487/RFC3156, August 2001, <https://www.rfc-editor.org/info/rfc3156>.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <https://www.rfc-editor.org/info/rfc3261>.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120, March 2011, <https://www.rfc-editor.org/info/rfc6120>.

[RFC8098] Hansen, T., Ed. and A. Melnikov, Ed., "Message Disposition Notification", STD 85, RFC 8098, DOI 10.17487/RFC8098, February 2017, <https://www.rfc-editor.org/info/rfc8098>.

[RFC8551] Schaad, J., Ramsdell, B., and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification", RFC 8551, DOI 10.17487/RFC8551, April 2019, <https://www.rfc-editor.org/info/rfc8551>.

[RFC9078] Crocker, D., Signes, R., and N. Freed, "Reaction: Indicating Summary Reaction to a Message", RFC 9078, DOI 10.17487/RFC9078, August 2021, <https://www.rfc-editor.org/info/rfc9078>.

[W3C.CR-html52-20170808] Faulkner, S., Eicholz, A., Leithead, T., Danilo, A., and S. Moon, "HTML 5.2", World Wide Web Consortium CR CR-html52-20170808, 8 August 2017, <https://www.w3.org/TR/2017/CR-html52-20170808>.

Appendix A. Multipart examples

A.1. Proprietary and Common formats sent as alternatives

TODO: Revise to use the built-in NestedPart data structure.

Example sending this profile and proprietary messaging protocol simultaneously.

```
Content-type: multipart/alternative; boundary=XcrSXMwuRwk9

--XcrSXMwuRwk9
Content-type: message/cpim

From: <im:alice-smith@example.com>
DateTime: 2022-02-08T22:13:45-00:00
Message-ID: <28fd19857ad7@example.com>

Content-Type: text/plain; charset=utf-8

Test Message
--XcrSXMwuRwk9
Content-type: application/vnd.examplevendor-fancy-im-message

<content of example vendor's fancy proprietary format>
--XcrSXMwuRwk9
```

A.2. Mulitple Reactions Example

This shows sending a reaction with multiple separate emojis.

TBC

A.3. Complicated Nested Example

This example shows separate English and French versions of HTML message with inline images. Each of the images is presented in alternate formats: an animated GIF, and a single PNG.

TBC

A.4. TLS Presentation Language multipart container format

TODO: Revise to use the built-in NestedPart data structure.

In a heterogenous group of IM clients, it is often desirable to send more than one media type as alternatives, such that IM clients have a choice of which media type to render. For example, imagine an IM group containing a set of clients which support a common video format and a subset which only support animated GIFs. The sender

could send a multipart/alternative [RFC2046] container containing both media types. Every client in the group chat could render something resembling the media sent.

Likewise it is often desirable to send more than one media type intended to be rendered together as in (for example a rich text document with embedded images), which can be represented using the multipart/mixed [RFC2046] media type.

Some implementors complain that the multipart types are unnatural to use inside a binary protocol which requires explicit lengths such as MLS [I-D.ietf-mls-protocol]. Concretely, an implementation has to scan through the entire content to construct a boundary token which is not contained in the content.

While the author does not care about the specific syntax used, for comparison purposes presents a multipart container format using the TLS presentation language syntax used by the MLS protocol.

Note that there is a minor semantic difference between multipart/alternative and the proposal below. In multipart/alternative, the parts are presented in preference order by the sender. The receiver is support to render the first type which it supports. This container includes an ordering flag. As well, even if the flag is ordered, it is up to the IETF community to decide if it is acceptable for the receiver to choose its "best" format to render among an ordered preference list provided by the sender, or if the receiver must respect the ordered preference of the sender.

```
struct {
    /* a valid "Language-tag" as defined in RFC 5646 */
    opaque language_tag<1..52>;
} LanguageTag;

struct {
  ContentType content_type;
  LanguageTag content_languages<V>;
  opaque<V> body;
} Part;

enum {
  reserved(0),
  multipart_container_v1(1),
  (255)
} MultipartVersion;

enum {
  reserved(0),
  mixed(1),
  alternative(2),
  (255)
} MultipartSemantics;

enum {
  reserved(0),
  unordered(1),
  ordered(2),
  (255)
} MultipartOrdering;

struct {
    uint8 container_version;
    uint16 number_of_parts;
    MultipartSemantics semantics;
    MultipartOrdering ordering;
    Part parts<V>;
} MultipartContainer;
```

Author's Address

Rohan Mahy
Wire

Email: rohan.mahy@wire.com