More Instant Messaging Interoperability (mimi)
Internet-Draft
Intended status: Informational
Expires: 8 January 2026
7 July 2025
H. Sarvaiya
E. W Burger
Virginia Tech

Detecting the Presence of a Malicious Hub in MIMI Protocol
draft-burger-mimi-audit-layer-00

Abstract

This document defines a Merkle-tree-based approach that can act as an audit-layer detection mechanism to identify a malicious hub, responsible for interoperable group communication between various messaging platforms. The proposed approach is based on the MIMI protocol, which uses a central hub for timestamping and broadcasting messages to clients operating on different platforms. Even though all MLS ciphertexts are end-to-end encrypted, they are routed through the hub, making it a lucrative attack surface for message reordering attacks. To detect such attacks, the proposed approach suggests creating Merkle proofs of messages and timestamps on the client-side, which can subsequently be broadcast to other clients for verification with local Merkle proofs. The broadcast messages are encrypted too and are sent probabilistically to avoid being dropped by the hub. If any of the proofs do not match, an alert is broadcast to the room, indicating a malicious hub. The approach has minimal communication overhead for practical purposes.

About This Document

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

Discussion of this document takes place on the mimi Working Group mailing list (mailto:mimi@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/mimi/. Subscribe at https://www.ietf.org/mailman/listinfo/mimi/.

Status of This Memo

This Internet-Draft will expire on 8 January 2026.

Copyright Notice

Table of Contents

1. Introduction

The MIMI architecture [I-D-mimi-arch] [I-D-mimi-proto] uses a hub as the center of each room, which is also responsible for routing and timestamping all messages. It employs MLS [RFC9420] for end-to-end encryption (E2EE) and security. If a hub is compromised, it can take advantage of the trust vested in it by the MIMI protocol. This can affect the integrity of the messages being routed through the hub, including, but not limited to message dropping, message reordering, targeted censorship attacks, etc [I-D-mimi-content] [E2EE-Attacks].

With this detailed example below, we will demonstrate few of the possible attacks that a compromised hub can execute. We are considering a group chat between Alice, Bob and Charlie.

Message reordering attack:

Seen by Alice:

08:00 Alice: _How to fix the system?_

08:02 Bob: _Press X, it should work!_

08:03 Alice: _I pressed X, the system crashed!_

08:04 Charlie: _Do not Press X, the system will crash!_

08:08 Charlie: _I told you not to press X, but you went with Bob’s suggestion._

Seen by Charlie:

08:00 Alice: _How to fix the system?_

08:04 Charlie: _Do not Press X, the system will crash!_

08:06 Bob: _Press X, it should work!_

08:07 Alice: _I pressed X, the system crashed!_

08:08 Charlie: _I told you not to press X, but you went with Bob’s suggestion._

Now, in the scenario presented above, Charlie sees a different version of the chat than Alice, which creates a misunderstanding between Alice and Charlie. Charlie thinks that Alice decided to go ahead with Bob’s suggestion and ignored his suggestion even though it was sent first, whereas Charlie’s message arrived after Alice had already acted upon Bob’s suggestion. Here, the hub reordered message timestamps for Charlie, undermining the integrity of the group chat.

Shown below is another example, where the hub specifically dropped or censored messages from Charlie, such that Alice never sees any messages from Charlie. This is a targeted censorship attack from the hub. Also, from the context of the messages, Charlie assumes that Alice provided an update in response to his message, whereas Alice never saw Charlie’s message and was updating Bob, due to which the malicious hub also avoided being detected.

Seen by Alice:

08:00 Alice: _How to fix the system?_

08:02 Bob: _Press X, it should work!_

08:07 Alice: _I pressed X, the system crashed!_

Seen by Charlie:

08:00 Alice: _How to fix the system?_

08:02 Bob: _Press X, it should work!_

08:04 Charlie: _Do not Press X, the system will crash!_

08:05 Charlie: _Any update Alice?_

08:07 Alice: _I pressed X, the system crashed!_

As per [I-D-mimi-content] [E2EE-Attacks], message reordering, and traffic analysis attacks are practically possible, even with E2EE being in place. The draft MIMI specification does not offer any solution for detecting or countering a malicious hub. The protocol described in this document introduces a client-driven audit layer to detect malicious hub behavior without the need to modify the hub, and the need to add new trusted servers, with practically minimal resource overhead. Following the proposed protocol, each client maintains a timestamp-ordered list of messages from the client perspective, while continuously maintaining a Merkle root for the list. With a random probability 'alpha' the client broadcasts the Merkle root to the group with a regular encrypted group message. On receipt of this message, each client individually verifies the Merkle root against their local list of ordered messages. If the Merkle root does not match with even one of the clients, the client raises an alarm, indicating a malicious hub.

2. Threat Model

Based on the examples and discussion in the previous section, we propose a formal threat model for a MIMI hub, as a critical point of security failure in the MIMI protocol. The threat model will be defined by the following three components:

1. Timestamp Authority of the hub enables it to back-date or falsify timestamps, leading to a timeline of messages that can be misleading.

2. Re-ordering via delay is another property of a malicious hub that causes downstream clients to see an arbitrary permutation of actual messages, due to the hub selectively delaying certain messages.

3. Isolated views indicate that clients are only dependent on the hub for receiving and sending messages, and have no other way of cross-checking the order of the messages received.

The Theat Model represented in Figure 1, represents a message reordering attack where, the messages sent by Client B and Client C get reordered by the malicious hub Server A, such that Client B sees a different order of messages than Client C and Client A.

```
ClientB   ServerB      ClientC      ServerC       ServerA        ClientA
 |            |            |            |             |              |
 |    M_b     |            |            |             |              |
 |----------->|            |            |             |              |
 |            |              /submit M_b              |              |
 |            |-------------------------------------->|              |
 |            |            |   200 OK   |             |              |
 |            |<--------------------------------------|              |
 |  Accepted  |            |            |             |              |
 |<-----------|            |            |             |              |
 |            |            |    M_c     |             |              |
 |            |            |----------->|             |              |
 |            |            |            | /submit M_c |              |
 |            |            |            |------------>|              |
 |            |            |            |    200 OK   |              |
 |            |            |            |<------------|              |
 |            |            |  Accepted  |             |              |
 |            |            |<-----------|             |              |
 |            |            |            +----------------------------+
 |            |            |            |Malicious reordering: M_c is|
 |            |            |            |delivered first to ClientA  |
 |            |            |            +----------------------------+
 |            |            |            |             |     M_c      |
 |            |            |            |             |------------->|
 |            |            |            |             |     M_b      |
 |            |            |            |             |------------->|
 |            |            |   /notify M_c            |              |
 |            |<--------------------------------------|              |
 |     M_c    |            |            |             |              |
 |<-----------|            |            |             |              |
 |            |            |            | /notify M_b |              |
 |            |            |            |<------------|              |
 |            |            |     M_b    |             |              |
+----------+  |            |<-----------|             |              |
|M_b -> M_c|  |            |            |             |              |
+----------+  |      +----------+       |             |              |
 |            |      |M_c -> M_b|       |             |              |
 |            |      +----------+       |             |    +----------+
 |            |            |            |             |    |M_c -> M_b|
 |            |            |            |             |    +----------+
 |            |            |            |             |              |
```

Figure 1: Threat Model

3. 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.

4. Protocol Design

The proposed mechanism is designed based on Merkle proof generation. Merkle proofs are generated using Merkle trees where each leaf is represented as a hash function of a message and its timestamp. The proof is calculated using the leaf nodes and if the message order is tampered with in any way, the proof generated will vary. This will help identify if the hub is malicious as it handles all the message routing and distribution mechanisms. The proposed mechanism is broadly divided into three components, which are described below:

4.1. Merkle Tree Construction and Proof Generation

All clients maintain a list of messages with their corresponding timestamps assigned by the central hub. The list of messages M = [(d_i, t_i)], where d_i is the message plaintext and t_i is the timestamp, stores messages in the ascending order of the timestamps. The hash generated for each message in the list and its corresponding timestamp forms a leaf node.

We use SHA-256 as the hash function H(.), for its collision resistance, wherein, each leaf node l_i can be denoted as a hash function of (d_i, t_i), i.e. l_i = H(d_i || t_i) . To construct a Merkle tree, a batch of messages are taken from the maintained message list, with s and e being the starting and ending message indices respectively. The parent node at each level of the tree is calculated by combining a pair of leaf nodes l_k and l_k+1 spanning from l_s to l_e. The calculation for parent nodes represented by n_j = H(l_k || l_k+1), continues recursively at each level until the single root node or the Merkle root R is generated, as shown in Algorithm 1. In case of even number of leaf nodes, the pairwise combination of leaf nodes works perfectly. In case of odd number of leaf nodes, the last node is duplicated to form a pair.

The generated Merkle proof is a tuple represented by (R, t_max, s, e), where t_max represents the maximum timestamp from message s to e. The generated Merkle proof will be broadcast randomly by the client to all other clients via the MIMI hub. To avoid detection by the MIMI hub, we will discuss the random sampling of the proof in the next section.

```
Algorithm 1: ComputeAndBroadcastMerkleProof(Lm, s, e)

Require: List of sorted messages Lm = M[1,...,m],
indices 1 <= s <= e <= m
Ensure: Proof tuple (R, t_max, s, e)

procedure ComputeMerkleProof(M, s, e)
    leaves := []
    t_max := 0
    for i := s to e do
        (d_i, t_i) := M[i]
        l_i := H(d_i || t_i)
        leaves.append(l_i)
        t_max := max(t_max, t_i)
    end for
    R := BuildMerkleTreeRoot(leaves)
    return (R, t_max, s, e)
end procedure

procedure BroadcastProof(alpha)
    if Bernoulli(alpha) then
        Broadcast(R, t_max, s, e)
    end if
end procedure
```

4.2. Probabilistic Proof Sampling and Broadcast

The Merkle proofs requested by a client are embedded with a probability alpha within the MLS PrivateMessage frames to prevent a malicious hub from predicting and selectively suppressing messages containing proofs. Following this, the MLS PrivateMessage containing the proof is broadcast to all the other clients, as per Algorithm 1. To test the ideal sampling rate, we varied the sampling rates and outlined the results in the evaluations section.

Given n number of clients, the probability of each requesting a Merkle proof independently is alpha. The Proof-request Probability per Message is given by:

P_proof = 1 - (1 - alpha)^n

The probability of a malicious hub attacking each message is given by beta. The probability of detection of a malicious hub in a single message is defined as:

P^(1)_detect = beta[1 - (1 - alpha)^n]

Assuming independence across messages, the escape probability for T messages (i.e., probability that messages go undetected) is given by:

P^(T)_escape = [1 - beta(1 - (1 - alpha)^n)]^T

The detection probability by message T is given by:

P^(T)_detect = 1 - [1 - beta(1 - (1 - alpha)^n)]^T

4.3. Client Verification and Malicious Hub Detection

When a client randomly broadcasts a Merkle proof to the other clients via the MIMI hub, the proof received by all the clients is verified locally against a maintained list of messages M specifically for message indices s to e. Each client computes a Merkle root and verifies it against the received Merkle root; the timestamp t_max is verified as well. Any discrepancy found will cause a client to raise an alert, and a proof mismatch is broadcast to all clients indicating the presence of a malicious hub, as described in Algorithm 2.

Algorithm 2: ProofVerificationAndBroadcast

```
procedure OnProof(R, t_max, s, e)
    compute leaves l_i = H(d_i || t_i) for i in [s, e]
    R_hat := ComputeRootFromLocalMessages(s, e)
    if R_hat != R or t_max != max_{i in [s,e]} t_i then
        RaiseAlert(ProofMismatch)
        Broadcast(ProofMismatch)
    end if
end procedure
```

5. Security Considerations

A compromised client is one of our primary security considerations that could undermine the efficacy of the proposed detection mechanism. Since a client is independent or autonomous in terms of Merkle proof generation, if compromised, it has the ability to generate fake proofs, which can lead to false alarms by other clients. Another possibility is that the compromised client generates a false alarm. Even if the received Merkle proof matches the locally generated proof, the client might raise a false alarm, disrupting communication or causing overheads to re-establish secure communication. To potentially mitigate the generation of fake proofs, cryptographically signed Merkle proofs could be used to verify the origin of the proof, potentially preventing proof spoofing.

A compromised client can also pull off a Denial of Service (DoS) attack by flooding the group with proofs or mismatched alerts. This may lead to higher communication overheads for continuous Merkle proof computations and verification against multiple proofs available on the group. This may also cause alert fatigue and confusion, generating irrelevant alerts due to a compromised client rather than a compromised hub. Dealing with this issue requires setting policies for handling alerts, including but not limited to limiting the rate of proof generation and broadcast, aggregation of alerts, and explicit alert handling to detect the presence of a malicious client.

Similar to post quantum decryption consideration for MLS, Merkle proof generation by the proposed mechanism can also be affected by the security of the underlying algorithm for proof generation. Another consideration would be the collision resistance property of the algorithm used for proof generation, which can potentially undermine the integrity of the generated proofs. Choosing and optimizing the available hash functions to suit the security and overhead requirements for Merkle proof generation is key in maintaining proof integrity. Algorithmic agility is another key component to rapidly switch between cryptographic algorithms as a response to emerging threats to the integrity of the proofs and the overall communication integrity.

Another point of possible intrusion could be the follower servers. A compromised follower server may do internal message reordering or dropping, acting as a malicious hub for its clients. In such a case, the MIMI hub will be the one held responsible, whereas the malicious follower server will bypass detection. To mitigate this situation, it might be a good idea to apply the audit-layer detection on the follower servers internally, while keeping the communication overhead minimal.

6. IANA Considerations

This document has no IANA actions.

7. References

7.1. Normative References

[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>.

[RFC9420] Barnes, R., Beurdouche, J., Omari, K., and R. Robert, "The Messaging Layer Security (MLS) Protocol", RFC 9420, October 2023, <https://www.rfc-editor.org/info/rfc9420>.

7.2. Informative References

[E2EE-Attacks]
, , Namavari, A., Nassi, B., Agarwal, R., and T. Ristenpart, "Injection Attacks Against End-to-End Encrypted Applications", Proceedings of the IEEE Symposium on Security and Privacy (S&P) 2024, pp. 2648-2665, 2024, <https://doi.org/10.1109/SP58266.2024.00174>.

[I-D-mimi-arch]
Barnes, R., "An Architecture for More Instant Messaging Interoperability (MIMI)", Work in Progress, Internet-Draft, draft-ietf-mimi-arch-01, November 2024, <https://datatracker.ietf.org/doc/draft-ietf-mimi-arch/01/>.

[I-D-mimi-content]
Mahy, R., "Message Content for More Instant Messaging Interoperability (MIMI)", Work in Progress, Internet-Draft, draft-ietf-mimi-content-07, February 2025, <https://datatracker.ietf.org/doc/draft-ietf-mimi-content/>.

[I-D-mimi-proto]
Barnes, R., Hodgson, M., Kohbrok, K., Mahy, R., Ralston, T., and , "More Instant Messaging Interoperability (MIMI) using HTTPS and MLS", Work in Progress, Internet-Draft, draft-ietf-mimi-protocol-03, March 2025, <https://datatracker.ietf.org/doc/draft-ietf-mimi-protocol/03/>.

Acknowledgments

We gratefully acknowledge the valuable feedback and constructive discussions received within the working group, in individual conversations, and during the MIMI interim meetings.

Authors' Addresses

Harditya Sarvaiya
Virginia Tech
United States of America
Email: harditya@vt.edu

Eric W Burger
Virginia Tech
United States of America
Email: ewburger@vt.edu