Network Working Group
Internet-Draft
Intended status: Informational
Expires: 4 September 2025

F. Kiefer
K. Bhargavan
Cryspen
R. L. Barnes
Cisco
J. Alwen
M. Mularczyk
AWS Wickr

3 March 2025

Light MLS
draft-kiefer-mls-light-02

Abstract

The Messaging Layer Security (MLS) protocol provides efficient asynchronous group key establishment for large groups with up to thousands of clients. In MLS, any member can commit a change to the group, and consequently, all members must download, validate, and maintain the full group state, which can incur a significant communication and computational costs, especially when joining a group. This document defines an MLS extension to support "light clients" that don’t undertake these costs. A light client cannot commit changes to the group, and only has partial authentication information for the other members of the group, but is otherwise able to participate in the group. In exchange for these limitations, a light client can participate in an MLS group with significantly lower requirements in terms of download, memory, and processing.

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://example.com/LATEST. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-kiefer-mls-light/.

Discussion of this document takes place on the WG Working Group mailing list (mailto:WG@example.com), which is archived at https://example.com/WG.

Source for this draft and an issue tracker can be found at https://github.com/USER/REPO.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Messaging Layer Security protocol [RFC9420] enables continuous group authenticated key exchange among a group of clients. The design of MLS requires all members to download, validate, and maintain the full MLS tree, including validating the credentials and signatures of all members. The size of the MLS tree is linear in the size of the group. Consequently, the MLS design results in a performance bottleneck for new members seeking to join a large group, and significant storage and memory requirements once the member has joined.

This document defines an extension to MLS to allow for "light clients" -- clients that do not download, validate, or maintain the entire ratchet tree for the group. On the one hand, this "lightness" allows a light client to participate in the group with much significantly lower communication and computation complexity. On the other hand, without the full ratchet tree, the light client cannot create Commit messages to put changes to the group into effect. Light clients also only have authentication information for the parts of the tree they download, not the whole group.

This document does not change the core logic of MLS, including: The structure of the ratchet tree and its associated algorithms, the key schedule, the secret tree, and application message protection. The messages sent and received by normal clients in the course of an MLS session are likewise unchanged. With proper modifications to the MLS Delivery Service, standard MLS clients can participate in a group with light clients without any modification.

The only modifications this document makes are to the local state stored at light clients, namely the component of MLS that manages, synchronizes, and authenticates the public group state. We also defines some "annotations" that need to be appended to group messages so that they can be processed by light clients. Light clients effectively run normal MLS algorithms, but with just-in-time delivery of exactly the subset of the public group state needed by a given algorithm. We achieve lightness due to the fact that aside from initial tree validation and sending commits, a client only needs log-scale information.

In summary, Light MLS allows light clients to obtain greater efficiency, at the cost of lowering the authentication guarantees they receive and losing the ability to make Commits. We discuss a few scenarios that motivate this trade-off in Section 3. The difference in authentication properties, in particular, is discussed in detail in Section 13.

2. Terminology

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.

Tree slice: A tree slice is the direct path from a leaf node to the root, together with the tree hashes on the co-path.

Membership proof: A tree slice that proves that a given leaf node is part of a ratchet tree with a given tree hash.

Light client: An MLS client that does not download, validate, and maintain a copy of the group’s ratchet tree. A light client does not store any public data about the group’s ratchet tree, only the HPKE decryption keys associated to nodes on the client’s direct path.

Full client: A normal MLS client, in possession of the full MLS ratchet tree for the group.

Sender-authenticated message: A signed MLS message such as Welcome or PublicMessage, together with a membership proof that proves the sender’s membership in the group.

Annotated Welcome: A Welcome message together with information that a light client needs to process it.

Annotated Commit: A Commit message (as a PublicMessage or PrivateMessage) together with information that a light client needs to process it.

As in MLS, message structures are defined using the TLS presentation syntax [RFC8446]. Unlike most MLS messages, however, these structures are not encapsulated in a signed or MAC'ed structure. So it may be more convenient for applications to encode these structures in application-specific encodings.

3. Use Cases

Light MLS is intended to support use cases where MLS groups are very large, from thousands to millions of participants. Application-level constraints arising from these use cases align well with the trade-offs introduced by Light MLS. It is usually acceptable (even desirable) that only certain members are able to send Commit messages. And in such large groups, clients often do not care about authenticating all members of the group.

The following subsections outline two concrete use cases.

3.1. Large Meetings / Webinars

MLS can be used to establish end-to-end encryption keys in real-time conferencing scenarios. In such scenarios, a client joins the MLS group when they are admitted to a meeting. With normal MLS, the client is required to download and validate the entire ratchet tree before being able to derive media encryption keys. In meetings with a thousand or more participants, this process can take enough time that it introduces a noticeable delay in joining the meeting. If a client joins as a light client, then they can download a log-sized AnnotatedWelcome message and immediately obtain the media encryption keys.

Such a client will not have authenticated the group when they join the meeting. However, applications often do not display identity information in such setting anyway. In "webinar" settings, it is common attendee identities to be visible only to panelists and administrators, not to other attendees. Light MLS allows MLS to align with this privacy property. If attendees join as light clients, they can be provided with membership proofs for attendees whom they are authorized to see, and not for others. Even in settings where attendees can all see each others' identities, user interface constraints usually cause only a small fraction of the attendee list to be visible at one time, so it is natural to load the tree dynamically as the client needs access to the authenticated identities of specific other clients.

The constraint on clients sending Commits is also natural here. In such large gatherings, there are usually administrators who are authorized to see the identities of all participants and control who is in the group, and conversely, there are certain actions that are not available to non-admin participants. So it makes sense for the administrators to use full clients that are able to make Commits to implement the actions they are authorized to take, and for more limited clients to be unable to make commits.

3.2. Broadcast sessions

Internet streaming platforms frequently host broadcasts with large numbers of viewers, but the entities providing these broadcasts might like to ensure that the streaming platform cannot see the streamed content. For example, the Media over QUIC Transport protocol, designed to support streaming use cases, states as a goal ensuring that "the media content itself remains opaque and private" from the relays involved in its distribution [I-D.ietf-moq-transport].

In such settings, the light client / full client roles align with the viewer / owner roles, respectively. Viewers do not care about the identities of other viewers (at most, they care that the stream comes from an authentic source) and they aren’t authorized to do anything in MLS besides join the group. Viewers are also typically in more constrained situations than the source of a stream. So the reduced resource requirements are well worth the loss of full-group authentication and the ability to Commit.

4. Protocol Overview

A light client does not receive or validate a full copy of the ratchet tree for a group, but still possesses the group’s secrets, including receiving updated secrets as the group evolves. When MLS messages are sent to a light client, they need dto be accompanied by annotations that provide the light client with just enough of information about the ratchet tree to process the message. These annotations can be computed by any party with knowledge of the group’s ratchet tree, including the committer and sometimes the DS.

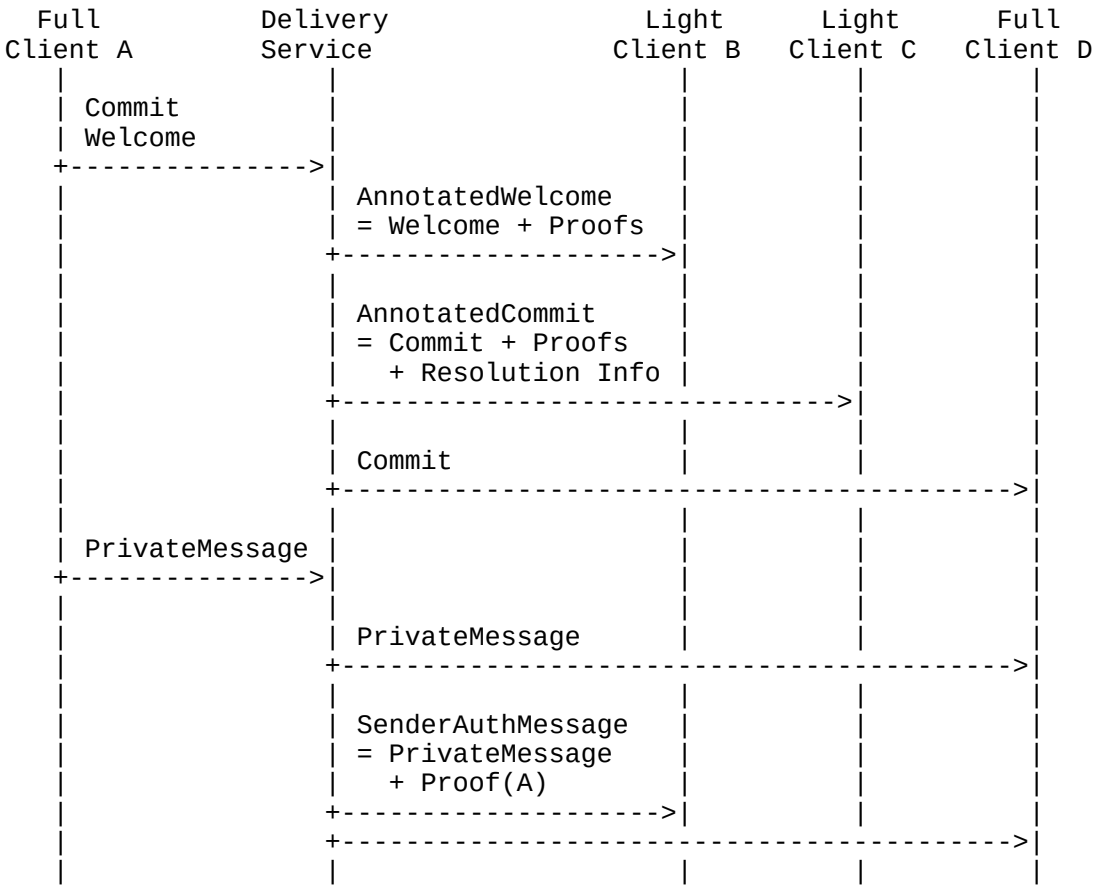

Figure 1: Overview of Light MLS

Figure 1 illustrates the main changes introduced by Light MLS:

1. When a light client is added to the group, they are provided an AnnotatedWelcome message, which comprises a normal Welcome message plus membership proofs for the sender and joiner.

2. From each Commit that is generated in the group, an individual AnnotatedCommit is generated for each light client. An AnntatedCommit comprises a normal MLS Commit message, together with membership proofs and the information that the light client needs in order to process the update path in the Commit.

3. When messages are sent in the group, e.g., carrying application data, they are extended with a membership proofs so that light clients can authenticate the sender’s membership in the group.

In this example, we have shown the required annotations being added by the DS. This allows full clients to behave as they would in normal MLS, but requires that the DS maintain a copy of the group's ratchet tree. It is also possible for committers to generate the required annotated messages. This document does not define who generates annotated messages from the base MLS messages, or how this entity learns which clients are light or full clients.

Light clients still need to be provided with access to any proposals sent in a group outside of Commits. Light clients cannot process proposals that modify the structure of the tree, in particular Add, Update, or Remove proposals. They can, however, verify that these proposals were included in a given Commit. And they need to see proposals such as PreSharedKey or GroupContextExtensions so that they can update their state appropriately.

Depending on how Light MLS is deployed, a client might need to inform the DS or other members of its status (light or full), so that the proper annotations can be generated when it is light. It is harmless for a full client to receive an AnnotatedCommit; the annotations can simply be ignored.

5. Upgrading and Downgrading

A light client can upgrade to being a full client at any time by downloading the full ratchet tree; a full client can downgrade by deleting its local copy of the ratchet tree. Before a light client uses a copy of the ratchet tree to upgrade to being a full client, it MUST verify the integrity of the ratchet tree in the same way it would when joining as a full client, following the steps in Section 12.4.3.1 of [RFC9420].

6. Membership Proofs and Partial Trees

Although light clients do not have a copy of the group's ratchet tree, they still agree on the root tree hash of the ratchet tree, via the MLS key schedule as usual. This fact, together with the Merkle-tree-like structure of the MLS tree hash, allows a light client to verify the legitimacy of partial information about the ratchet tree. In particular, for any leaf in the tree, anyone in possession of the public data of the ratchet tree can construct a "membership proof" that proves that a leaf node with specific contents is located at a specific leaf index in the tree.

A membership proof for a leaf node comprises:

* The number of leaves in the tree.

* The leaf index of the member’s leaf.

* The values of the nodes from the leaf node to the root of the tree, including both the leaf node and the root.

* The tree hash values for the nodes on the copath of the leaf node.

```
struct {
    opaque hash_value<V>;
} CopathHash;

struct {
  uint32 leaf_index;
  uint32 n_leaves;
  optional<Node> direct_path_nodes<V>;
  CopathHash copath_hashes<V>;
} MembershipProof;
```

From these values, the root tree hash of the ratchet tree can be recomputed, following the same recursive algorithm specified in Section 7.8 of [RFC9420]. The selection of nodes and subtree hashes provides the precise collection of inputs required by the algorithm.

A membership proof is said to be valid relative to a given tree hash if the tree hash recomputed in this way is equal to the given tree hash.

Two membership proofs are said to reference the same tree if their n_leaves fields are equal, and they produce identical root tree hashes.

7. Sender-Authenticated Messages

For several types of message, MLS authenticates that a message was created by the member at a specific leaf node of the group’s ratchet tree by signing the message with the private key corresponding to the signature_key in the leaf node. Full clients verify these messages by looking up the required signature verification key in their local copy of the ratchet tree.

Since light clients do not store the group’s ratchet tree, they cannot perform this lookup. A SenderAuthenticatedMessage presents a message along with a membership proof for the sender of a message, which provides the required leaf node and a proof of its inclusion in the tree.

```
struct {
    T message;
    MembershipProof sender_membership_proof;
} SenderAuthenticatedMessage<T>;
```

Before using the sender_membership_proof to verify the included message, a client processing a SenderAuthenticatedMessage MUST verify that the proof is valid relative to the group's tree hash for the epoch in which the message was sent. For a PublicMessage or PrivateMessage, this is the tree hash for the epoch indicated in the FramedContent. For a GroupInfo or Welcome, it is the tree hash in the object itself.

8. Joining via Annotated Welcome

An AnnotatedWelcome message provides a client joining a group with membership proofs for the sender and the joiner (i.e., the recipient of the message).

```
struct {
    SenderAuthenticatedMessage<Welcome> welcome;
    MembershipProof joiner_membership_proof;
} AnnotatedWelcome;
```

The fields in the AnnotatedWelcome have the following semantics:

welcome: A Welcome message, together with a membership proof for the sender relative to the ratchet tree specified in the Welcome.

joiner_membership_proof: A proof of the receipient's membership in the ratchet tree specified in the Welcome.

An AnnotatedWelcome can be generated by any party that knows the group's ratchet tree and the indices of the sender and joiner in the tree.

A light client processes an AnnotatedWelcome in the following way:

1. Verify that the sender_membership_proof and joiner_membership_proof reference the same tree.

2. Join the group using the procedure defined in Section 12.4.3.1 of [RFC9420], with the following modifications:

* When verifying the signature on the GroupInfo object, the signature public key is taken from the LeafNode in the sender_membership_proof tree slice. The signer field of the group_info object MUST be equal to the leaf_index field of the sender_membership_proof.

* The "Verify the integrity of the ratchet tree" step is replaced with a check that the tree_hash in the GroupInfo matches the root tree hash produced by the membership proofs.

* The my_leaf value is taken from the the leaf_index field of the joiner_membership_proof, instead of found by searching the tree.

9. Joining via External Commit

A light client cannot join via an external Commit, because light clients cannot generate commits. A client could, however, join as a full client via an external commit, then transition to being a light client by deleting its copy of the tree. This would still require the client to download and validate the tree, but would save the client the effort of maintaining their copy of the tree.

10. Annotated Commit

There are two main challenges for a light client processing a Commit. First, the light client cannot compute the resolution of the committer’s copath, so they cannot determine which of the HPKECiphertext objects in the UpdatePath they should decrypt to obtain a path secret. Second, the light client cannot compute the updated tree hash, since they don’t have the full tree. An AnnotatedCommit provides these pieces of information, along with proof that the sender and receiver are both still in the group after the Commit.

```
struct {
    MLSMessage commit;
    optional<MembershipProof> sender_membership_proof;

    opaque tree_hash_after<V>;
    optional<uint32> resolution_index;

    MembershipProof sender_membership_proof_after;
    MembershipProof receiver_membership_proof_after;
} AnnotatedCommit;
```

The fields in the AnnotatedCommit have the following semantics:

commit: An MLSMessage containing PrivateMessage or PublicMessage with content_type commit.

sender_membership_proof: A membership proof for the sender of the Commit relative to the tree for the epoch in which the Commit is sent. This field MUST be present if the sender_type for the Commit is member, and otherwise MUST be absent.

tree_hash_after: The tree hash of the group's ratchet tree after the Commit has been applied.

resolution_index: The recipient can compute which entry in the UpdatePath in the Commit it should use based on the sender index in the Commit. This index specifies which HPKECiphertext in the UpdatePathNode to use. This field MUST be included if and only if the Commit has a path field populated.

sender_membership_proof_after: A membership proof for the sender of the Commit relative to the tree after the Commit has been applied.

receiver_membership_proof_after: A membership proof for the receiver of the Commit relative to the tree after the Commit has been applied.

An AnnotatedCommit can be generated by any party that knows the group's ratchet tree (both before and after the Commit) and the indices of the sender and joiner in the tree. It is safe for the recipient to accept the tree_hash supplied by an unauthenticated party because the tree hash is authenticated by the confirmation_tag in the Commit.

A light client processes an AnnotatedCommit in the following way:

1. Verify that the sender_membership_proof in the commit field, if present, is valid relative to the group's current tree hash.

2. Verify that the sender_membership_proof_after and receiver_membership_proof_after reference the same tree, and that they are valid relative to tree_hash_after.

3. Process the Commit using the procedure defined in Section 12.4.2 of [RFC9420], with the following modifications:

* When validating a FramedContent with sender_type set to member, the sender_membership_proof field MUST be present, and the signature public key is taken from the LeafNode in the sender_membership_proof tree slice. The leaf_index field of the Sender object MUST be equal to the leaf_index field of the sender_membership_proof.

  - If the sender_type is set to new_member_commit (the only other valid value), then the signature public key is looked up in the included Add proposal, as normal. In this case, there is no further validation of the leaf_index field of of the sender_membership_proof.

* The proposal list validation step is omitted, because a light client doesn’t have sufficient information to check all of the validation rules.

* When applying proposals, only the proposals that do not modify the tree are applied, in particular, PreSharedKey and GroupContextExtensions proposals.

* Likewise, the creation of the new ratchet tree is omitted.

* In processing the path value, if present, replace the path node decryption steps with the following steps:

  - Use the leaf_index field of the sender_membership_proof to identify the lowest common ancestor between the committer. This is the node where the new path_secret will be inserted into the tree.

  - Determine the index update_path_index of the lowest common ancestor among the non-blank nodes in the committer’s direct path, as provided in the sender_membership_proof_after field.

  - From the entry at index update_path_index of the nodes vector in the UpdatePath, select the HPKECiphertext at index resolution_index from the encrypted_path_secret.

  - Identify the next non-blank node in the recipient’s direct path under the lowest common ancestor, using the direct path provided in the recipient_membership_proof_after field. Retrieve the private HPKE decryption key for this node.

  - Decrypt the encrypted path secret as normal, using the tree hash in the tree_hash_after field in the provisional GroupContext.

- Derive the remaining path secrets corresponding to the non-blank nodes in the recipient's new direct path, as provided in the recipient_membership_proof_after field.

- Define the commit_secret to be path_secret[n+1], as normal.

11. Application Messages

MLS clients can exchange messages by sending application data within the PrivateMessage framing. In a group where light clients are present, these messages should be further encapsulated in a SenderAuthenticatedMessage, so that light clients have the membership proof necessary to verify the sender's membership, the public key necessary to verify the message signature, and the credential necessary to verify the sender's identity.

As noted above, this can be accomplished either by the sender creating a SenderAuthenticatedMessage, or by the DS adding the relevant membership proof while the message is in transit.

Note that encapsulating a message as a SenderAuthenticatedMessage leaks information about the sender to the DS, including the sender's index in the tree and the sender's LeafNode. See Section 13.1 for more discussion of metadata privacy.

12. Operational Considerations

The major operational challenge in deploying Light MLS is ensuring that light clients receive the proper annotations to Welcome and Commit messages. As discussed in Section 4, this is up to the application. Since full clients don't need the annotations, applications will be more robust if they send annotations in a way that they can be cleanly ignored by full clients.

Light MLS substantially reduces the amount of data required to join an MLS group, since it replaces the linear-scale ratchet tree with two log-scale membership proofs. Light MLS does not address the potentially linear scaling of Commit messages; in fact, it makes Commits slightly bigger. There are other approaches to reducing Commit sizes, e.g., the SplitCommit approach in [I-D.mularczyk-mls-splitmls]. These approaches can be cleanly integrated with Light MLS via the AnnotatedCommit structure. Table 1 summarizes the scaling of the amount of data that a client needs to download in order to perform various MLS operations. Sending a Commit requires linear-scale work in any case.

Operation	RFC MLS	Light MLS	Split Commits	Light + Split
Join	O(N)	O(log N)	O(N)	O(log N)
Process Commit	O(N)	O(N)	O(log N)	O(log N)

Table 1: Download scaling under protocol variations

13. Security Considerations

The MLS protocol in [RFC9420] has a number of security analyses attached. To describe the security of Light MLS and how it relates to the security of full MLS we summarize the following main high-level guarantees of MLS as follows:

* *Membership Agreement*: If a client B has a local group state for group G in epoch N, and it receives (and accepts) an application message from a sender A for group G in epoch N, then A must be a member of G in epoch N at B, and if A is honest, then A and B agree on the full membership of the group G in epoch N.

* *Member Identity Authentication*: If a client B has a local group state for group G in epoch N, and B believes that A is a member of G in epoch N, and that A is linked to a user identity U, then either the signature key of U' s credential is compromised, or A belongs to U.

* *Group Key Secrecy*: If B has a local group state for group G in epoch N with group key K (init secret), then K can only be known to members of G in epoch N. That is, if the attacker knows K, then one of the signature or decryption keys corresponding to one of the leaves of the tree stored at B for G in epoch N must be compromised. To obtain these properties, each member in MLS verifies a number of signatures and MACs, and seeks to preserve the TreeKEM Tree Invariants:

* *Public Key Tree Invariant*: At each node of the tree at a member B, the public key, if set, was set by one of the members currently underneath that node

* *Path Secret Invariant*: At each node, the path secret stored at a member B, if set, was created by one of the members currently underneath that node

As a corollary of Group Key Secrecy, we also obtain authentication and confidentiality guarantees for application messages sent and received within a group.

To verify the security guarantees provided to light clients, a new security analysis was needed. We have analyzed the security of the protocol using two verification tools ProVerif and F*. The security analysis, and design of the security mechanisms, are inspired by work from Alwen et al. [AHKM22].

Light MLS preserves the invariants above and thereby all the security goals of MLS continue to hold at full members. However, a light member may not know the identities of all other members in the group, and it may only discover these identities on-demand. Consequently, the Member Identity Authentication guarantee is weaker on light clients. Furthermore, since light members do not store the MLS tree, membership agreement only holds for the hash of the MLS tree:

* *Light Membership Agreement*: If a light client B has a local group state for group G in epoch N, and it receives (and accepts) an application message from a sender A for group G in epoch N, then A must be a member of G in epoch N at B, and if A is honest, then A and B agree on the GroupContext of the group G in epoch N.

* *Light Member Identity Authentication*: If a light client B has a local group state for group G in epoch N, and B has verified A’s membership proof in G, and A is linked to a user identity U, then either the signature key of U’s credential is compromised, or A belongs to U.

* *Light Group Key Secrecy*: If a light client B has a local group state for group G in epoch N with group key K (init secret), and if the tree hash at B corresponds to a full tree, then K can only be known to members at the leaves of this tree. That is, if the attacker knows K, then the signature or decryption keys at one of the leaves must have been compromised.

Note that the Light Membership Agreement property holds irrespective of whether B has verified a membership proof from A. The membership proofs in this protocol are thus more about providing precise source authentication within the group, rather than simply proving membership in the group. Simply knowing the group’s symmetric secrets suffices for the latter.

Another technical caveat is that since light members do not have the full tree, they cannot validate the uniqueness of all HPKE and signature keys in the tree, as required by RFC MLS. The exact security implications of removing this uniqueness check is not clear

but is not expected to be significant. In a group where full clients are honest, there is no practical difference, since a full client will verify that all of the required uniqness properties hold before issuing a Commit. The main risk is that a malicious full client could cause a light client to accept a tree hash representing a tree with duplicate keys.

13.1. Metadata Privacy

The protocol described in this document assumes that the DS is trusted to know information about the group's ratchet tree. The scenario described in Section 4 assumes that the DS is maintaining a view of the ratchet tree and distributing appropriate portions of it to clients. In fact, if the DS is to generate membership proofs to accompany PrivateMessage messages, then it will need to know the index of the sender in the tree, information that is normally encrypted as part of the SenderData.

It is possible to operate this protocol in a more restrictive mode, where Commits are sent as PrivateMessage objects and the committer generates the required annotations for any light clients in the group. However, because there is no confidentiality protection for the annotations, they will leak information to the DS about the ratchet tree.

Fixing this leakage would require changes to logic at the committer and light clients. The annotations attached to a Welcome message could be sent as GroupInfo extensions; effectively a partial version of the ratchet_tree extension. The annotations attached to a Commit could be moved inside the PrivateMessage content, and the receiver signature validation logic updated to use the public key in the attached membership proof to validate the message signature.

Thus, while a more metadata-private mode could be added to this protocol, it has been omitted for now in the interest of avoiding changes to full endpoints.

14. IANA Considerations

This document makes no request of IANA.

15. References

15.1. 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>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

[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>.

15.2. Informative References

[AHKM22] Alwen, J., Hartmann, D., Kiltz, E., and M. Mularczyk, "Server-Aided Continuous Group Key Agreement", ACM, Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security pp. 69-82, DOI 10.1145/3548606.3560632, November 2022, <https://doi.org/10.1145/3548606.3560632>.

[I-D.ietf-moq-transport] Curley, L., Pugin, K., Nandakumar, S., Vasiliev, V., and I. Swett, "Media over QUIC Transport", Work in Progress, Internet-Draft, draft-ietf-moq-transport-09, 1 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-moq-transport-09>.

[I-D.mularczyk-mls-splitmls] "*** BROKEN REFERENCE ***".

Appendix A. Known Issues

* To realize the completely optimized performance profile discussed on Section 12, we should define a version of AnnotatedCommit that sends a SplitCommit instead of a normal Commit.

* There is no signaling within the group of whether any members are light clients, and if so which ones. This was omitted because it didn’t seem clearly necessary, but it could be useful. For example, if a client could include a LeafNode extension that declares that it is a light client, then a committer could use

this signal to proactively generate AnnotatedCommits for the members. An approach like this might be necessary if we wanted to enable cases where annotations were confidential to the group.

* There are no WireFormat values registered for the new messages defined here that are likely to be sent on the wire: AnnotatedCommit, AnnotatedWelcome, or SenderAuthenticatedMessage<PrivateMessage>. It's not clear that these WireFormat values would be needed or useful, though, since the annotations added in these messages are effectively outside the bounds of MLS. They're more like how the delivery of the ratchet tree is unspecified in RFC MLS.

Acknowledgments

TODO acknowledge.

Authors' Addresses

Franziskus Kiefer
Cryspen
Email: franziskuskiefer@gmail.com

Karthikeyan Bhargavan
Cryspen
Email: karthik.bhargavan@gmail.com

Richard L. Barnes
Cisco
Email: rlb@ipv.sx

Jol Alwen
AWS Wickr
Email: alwenjo@amazon.com

Marta Mularczyk
AWS Wickr
Email: mulmarta@amazon.ch