Stream: Internet Engineering Task Force (IETF)
RFC: 9750
Category: Informational
Published: April 2025
ISSN: 2070-1721
Authors: B. Beurdouche E. Rescorla E. Omara S. Inguva A. Duric
Inria & Mozilla

RFC 9750
The Messaging Layer Security (MLS) Architecture

Abstract

The Messaging Layer Security (MLS) protocol (RFC 9420) provides a group key agreement protocol for messaging applications. MLS is designed to protect against eavesdropping, tampering, and message forgery, and to provide forward secrecy (FS) and post-compromise security (PCS).

This document describes the architecture for using MLS in a general secure group messaging infrastructure and defines the security goals for MLS. It provides guidance on building a group messaging system and discusses security and privacy trade-offs offered by multiple security mechanisms that are part of the MLS protocol (e.g., frequency of public encryption key rotation). The document also provides guidance for parts of the infrastructure that are not standardized by MLS and are instead left to the application.

While the recommendations of this document are not mandatory to follow in order to interoperate at the protocol level, they affect the overall security guarantees that are achieved by a messaging application. This is especially true in the case of active adversaries that are able to compromise clients, the Delivery Service (DS), or the Authentication Service (AS).

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9750.

Copyright Notice

Table of Contents

1. Introduction

End-to-end security is used in the vast majority of instant messaging systems and is also deployed in systems for other purposes such as calling and conferencing. In this context, "end-to-end" captures the notion that users of the system enjoy some level of security -- with the precise level depending on the system design -- even in the face of malicious actions by the operator of the messaging system.

Messaging Layer Security (MLS) specifies an architecture (this document) and a protocol [RFC9420] for providing end-to-end security in this setting. MLS is not intended as a full instant messaging protocol but rather is intended to be embedded in concrete protocols, such as the Extensible Messaging and Presence Protocol (XMPP) [RFC6120]. Implementations of the MLS protocol will interoperate at the cryptographic level, though they may have incompatibilities in terms of how protected messages are delivered, contents of protected messages, and identity/authentication infrastructures. The MLS protocol has been designed to provide the same security guarantees to all users, for all group sizes, including groups of only two clients.

2. General Setting

2.1. Protocol Overview

MLS provides a way for *clients* to form *groups* within which they can communicate securely. For example, a set of users might use clients on their phones or laptops to join a group and communicate with each other. A group may be as small as two clients (e.g., for simple person-to-person messaging) or as large as hundreds of thousands. A client that is part of a group is a *member* of that group. As groups change membership and group or member properties, they advance from one *epoch* to another and the cryptographic state of the group evolves.

The group is represented as a tree, which represents the members as the leaves of a tree. It is used to efficiently encrypt to subsets of the members. Each member has a state called a *LeafNode* object holding the client's identity, credentials, and capabilities.

Various messages are used in the evolution from epoch to epoch. A *Proposal* message proposes a change to be made in the next epoch, such as adding or removing a member. A *Commit* message initiates a new epoch by instructing members of the group to implement a collection of proposals. Proposals and Commits are collectively called *handshake messages*. A *KeyPackage* provides keys that can be used to add the client to a group, including a public encryption key and a signature key (both stored in the KeyPackage's `LeafNode` object). A *Welcome* message provides a new member to the group with the information to initialize their state for the epoch in which they were added.

Of course most (but not all) applications use MLS to send encrypted group messages. An *application message* is an MLS message with an arbitrary application payload.

Finally, a *PublicMessage* contains an integrity-protected MLS handshake message, while a *PrivateMessage* contains a confidential, integrity-protected handshake or application message.

For a more detailed explanation of these terms, please consult the MLS protocol specification [RFC9420].

2.2. Abstract Services

MLS is designed to operate within the context of a messaging service, which may be a single service provider, a federated system, or some kind of peer-to-peer system. The service needs to provide two services that facilitate client communication using MLS:

- An Authentication Service (AS), which is responsible for attesting to bindings between application-meaningful identifiers and the public key material used for authentication in the MLS protocol. The AS must also be able to generate credentials that encode these bindings and validate credentials provided by MLS clients.
- A Delivery Service (DS), which can receive and distribute messages between group members. In the case of group messaging, the DS may also be responsible for acting as a "broadcaster" where the sender sends a single message which is then forwarded to each

recipient in the group by the DS. The DS is also responsible for storing and delivering initial public key material required by MLS clients in order to proceed with the group secret key establishment that is part of the MLS protocol.

For presentation purposes, this document treats the AS and DS as conventional network services. However, MLS does not require a specific implementation for the AS or DS. These services may reside on the same server or different servers, they may be distributed between server and client components, and they may even involve some action by users. For example:

- Several secure messaging services today provide a centralized DS and rely on manual comparison of clients' public keys as the AS.
- MLS clients connected to a peer-to-peer network could instantiate a decentralized DS by transmitting MLS messages over that network.
- In an MLS group using a Public Key Infrastructure (PKI) for authentication, the AS would comprise the certificate issuance and validation processes, both of which involve logic inside MLS clients as well as various existing PKI roles (e.g., Certification Authorities).

It is important to note that the AS can be completely abstract in the case of a service provider which allows MLS clients to generate, distribute, and validate credentials themselves. As with the AS, the DS can be completely abstract if users are able to distribute credentials and messages without relying on a central DS (as in a peer-to-peer system). Note, though, that in such scenarios, clients will need to implement logic that assures the delivery properties required of the DS (see Section 5.2).

Figure 1 shows the relationship of these concepts, with three clients and one group, and clients 2 and 3 being part of the group and client 1 not being part of any group.

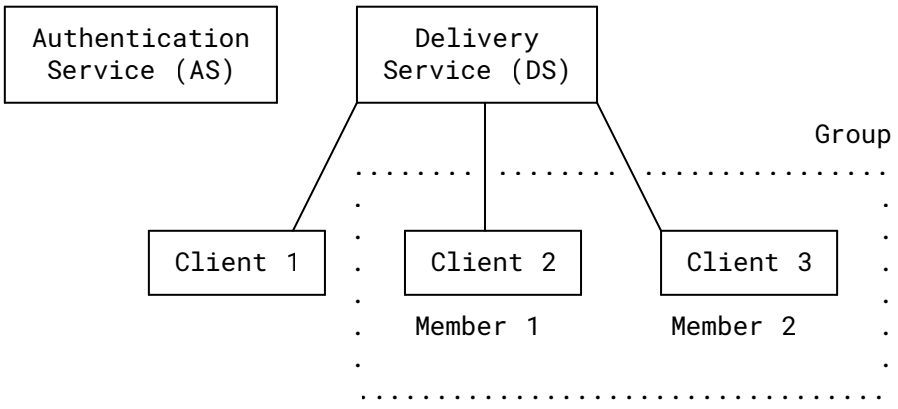

Figure 1: A Simplified Messaging System

3. Overview of Operation

Figure 2 shows the formation of an example group consisting of Alice, Bob, and Charlie, with Alice driving the creation of the group.

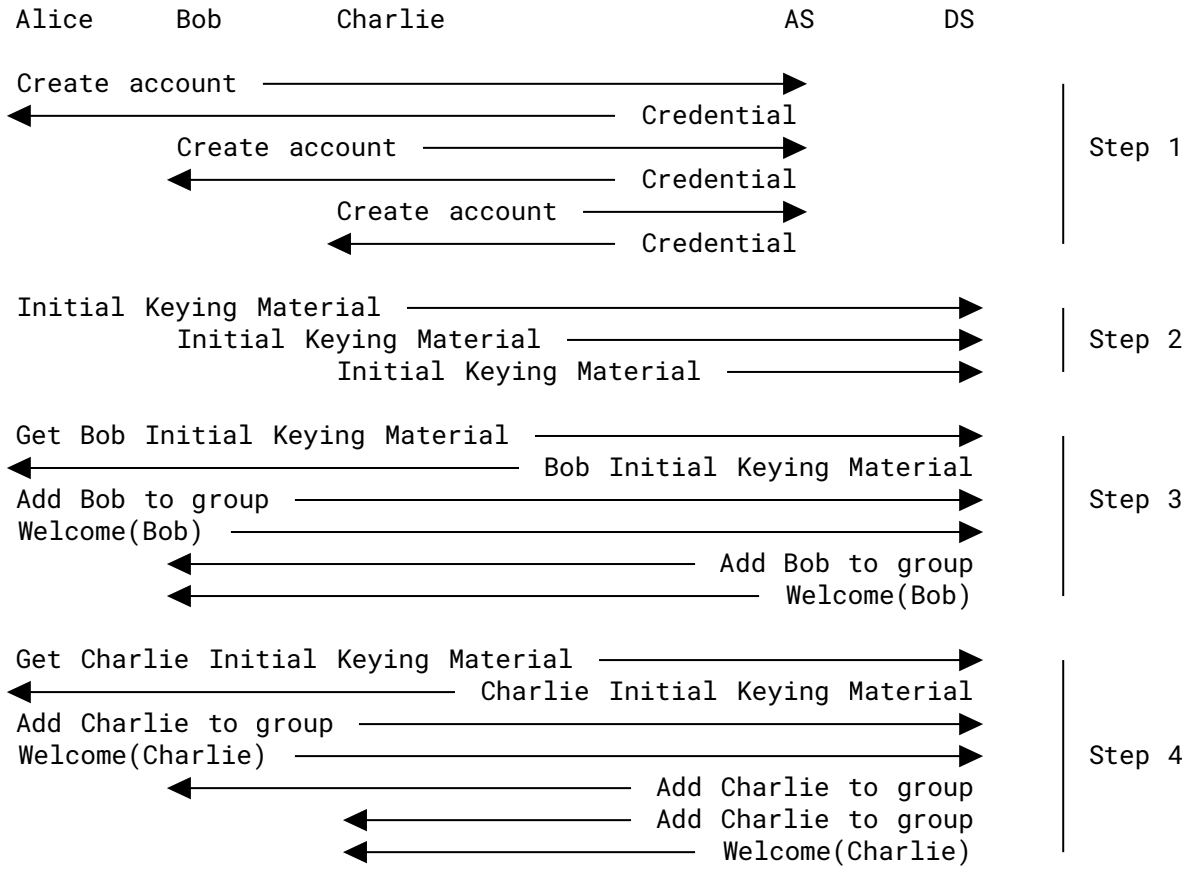

Figure 2: Group Formation Example

This process proceeds as follows.

3.1. Step 1: Account Creation

Alice, Bob, and Charlie create accounts with a service provider and obtain credentials from the AS. This is a one-time setup phase.

3.2. Step 2: Initial Keying Material

Alice, Bob, and Charlie authenticate to the DS and store some initial keying material which is used to send encrypted messages to them for the first time. This keying material is authenticated with their long-term credentials. Although in principle this keying material can be reused for multiple senders, in order to provide forward secrecy it is better for this material to be regularly refreshed so that each sender can use a new key and delete older keys.

3.3. Step 3: Adding Bob to the Group

When Alice wants to create a group including Bob, she first uses the DS to look up his initial keying material. She then generates two messages:

- A message to the entire group (which at this point is just her and Bob) that adds Bob to the group.
- A Welcome message just to Bob encrypted with his initial keying material that includes the secret keying information necessary to join the group.

She sends both of these messages to the DS, which is responsible for sending them to the appropriate people. Note that the security of MLS does not depend on the DS forwarding the Welcome message only to Bob, as it is encrypted for him; it is simply not necessary for other group members to receive it.

3.4. Step 4: Adding Charlie to the Group

If Alice then wants to add Charlie to the group, she follows a similar procedure as with Bob. She first uses the DS to look up his initial keying material and then generates two messages:

- A message to the entire group (consisting of her, Bob, and Charlie) adding Charlie to the group.
- A Welcome message just to Charlie encrypted with his initial keying material that includes the secret keying information necessary to join the group.

At the completion of this process, we have a group with Alice, Bob, and Charlie, which means that they share a single encryption key which can be used to send messages or to key other protocols.

3.5. Other Group Operations

Once the group has been created, clients can perform other actions, such as:

- sending a message to everyone in the group
- receiving a message from someone in the group
- adding one or more clients to an existing group
- removing one or more members from an existing group
- updating their own key material

- leaving a group (by asking to be removed)

Importantly, MLS does not itself enforce any access control on group operations. For instance, any member of the group can send a message to add a new member or to evict an existing member. This is in contrast to some designs in which there is a single group controller who can modify the group. MLS-using applications are responsible for setting their own access control policies. For instance, if only the group administrator is allowed to change group members, then it is the responsibility of the application to inform members of this policy and who the administrator is.

3.6. Proposals and Commits

The general pattern for any change in the group state (e.g., to add or remove a user) is that it consists of two messages:

Proposal: This message describes the change to be made (e.g., add Bob to the group) but does not effect a change.

Commit: This message changes the group state to include the changes described in a set of proposals.

The simplest pattern is for a client to just send a Commit which contains one or more Proposals. For instance, Alice could send a Commit with the Proposal Add(Bob) embedded to add Bob to the group. However, there are situations in which one client might send a Proposal and another might send the corresponding Commit. For instance, Bob might wish to remove himself from the group and send a Remove proposal to do so (see Section 12.1.3 of [RFC9420]). Because Bob cannot send the Commit, an existing member must do so. Commits can apply to multiple valid Proposals, in which case all the listed changes are applied.

It is also possible for a Commit to apply to an empty set of Proposals, in which case it just updates the cryptographic state of the group without changing its membership.

3.7. Users, Clients, and Groups

While it's natural to think of a messaging system as consisting of groups of users, possibly using different devices, in MLS the basic unit of operation is not the user but rather the "client". Formally, a client is a set of cryptographic objects composed of public values such as a name (an identity), a public encryption key, and a public signature key. As usual, a user demonstrates ownership of the client by demonstrating knowledge of the associated secret values.

In some messaging systems, clients belonging to the same user must all share the same signature key pair, but MLS does not assume this; instead, a user may have multiple clients with the same identity and different keys. In this case, each client will have its own cryptographic state, and it is up to the application to determine how to present this situation to users. For instance, it may render messages to and from a given user identically regardless of which client they are associated with, or it may choose to distinguish them. It is also possible to have multiple clients associated with the same user share state, as described in Section 8.2.4.

When a client is part of a group, it is called a member. A group in MLS is defined as the set of clients that have knowledge of the shared group secret established in the group key establishment phase. Note that until a client has been added to the group and contributed to the group secret in a manner verifiable by other members of the group, other members cannot assume that the client is a member of the group; for instance, the newly added member might not have received the Welcome message or been unable to decrypt it for some reason.

4. Authentication Service

The Authentication Service (AS) has to provide three services:

1. Issue credentials to clients that attest to bindings between identities and signature key pairs.
2. Enable a client to verify that a credential presented by another client is valid with respect to a reference identifier.
3. Enable a group member to verify that a credential represents the same client as another credential.

A member with a valid credential authenticates its MLS messages by signing them with the private key corresponding to the public key bound by its credential.

The AS is considered an abstract layer by the MLS specification; part of this service could be, for instance, running on the members' devices, while another part is a separate entity entirely. The following examples illustrate the breadth of this concept:

- A PKI could be used as an AS [RFC5280]. The issuance function would be provided by the certificate authorities in the PKI, and the verification function would correspond to certificate verification by clients.
- Several current messaging applications rely on users verifying each other's key fingerprints for authentication. In this scenario, the issuance function is simply the generation of a key pair (i.e., a credential is just an identifier and public key, with no information to assist in verification). The verification function is the application function that enables users to verify keys.
- In a system based on end-user Key Transparency (KT) [KT], the issuance function would correspond to the insertion of a key in a KT log under a user's identity. The verification function would correspond to verifying a key's inclusion in the log for a claimed identity, together with the KT log's mechanisms for a user to monitor and control which keys are associated with their identity.

By the nature of its role in MLS authentication, the AS is invested with a large amount of trust and the compromise of the AS could allow an adversary to, among other things, impersonate group members. We discuss security considerations regarding the compromise of the different AS functions in detail in Section 8.4.3.

The association between members' identities and their signature keys is fairly flexible in MLS. As noted above, there is no requirement that all clients belonging to a given user have the same signature key (in fact, having duplicate signature keys in a group is forbidden). A member can

also rotate the signature key they use within a group. These mechanisms allow clients to use different signature keys in different contexts and at different points in time, providing unlinkability and post-compromise security benefits. Some security trade-offs related to this flexibility are discussed in Section 8.

In many applications, there are multiple MLS clients that represent a single entity, such as a human user with a mobile and desktop version of an application. Often, the same set of clients is represented in exactly the same list of groups. In applications where this is the intended situation, other clients can check that a user is consistently represented by the same set of clients. This would make it more difficult for a malicious AS to issue fake credentials for a particular user because clients would expect the credential to appear in all groups of which the user is a member. If a client credential does not appear in all groups after some relatively short period of time, clients have an indication that the credential might have been created without the user's knowledge. Due to the asynchronous nature of MLS, however, there may be transient inconsistencies in a user's client set, so correlating users' clients across groups is more of a detection mechanism than a prevention mechanism.

5. Delivery Service

The Delivery Service (DS) plays two major roles in MLS:

- As a directory service, providing the initial keying material for clients to use. This allows a client to establish a shared key and send encrypted messages to other clients even if they're offline.
- Routing MLS messages among clients.

While MLS depends on correct behavior by the AS in order to provide endpoint authentication and hence confidentiality of the group key, these properties do not depend on correct behavior by the DS; even a malicious DS cannot add itself to groups or recover the group key. However, depending precisely on how MLS is used, the DS may be able to determine group membership or prevent changes to the group from taking place (e.g., by blocking group change messages).

5.1. Key Storage and Retrieval

Upon joining the system, each client stores its initial cryptographic key material with the DS. This key material, called a KeyPackage, advertises the functional abilities of the client (e.g., supported protocol versions, supported extensions, etc.) and the following cryptographic information:

- A credential from the AS attesting to the binding between the identity and the client's signature key.
- The client's asymmetric encryption public key.

All the parameters in the KeyPackage are signed with the signature private key corresponding to the credential. As noted in Section 3.7, users may own multiple clients, each with their own keying material. Each KeyPackage is specific to an MLS version and cipher suite, but a client may

want to offer support for multiple protocol versions and cipher suites. As such, there may be multiple KeyPackages stored by each user for a mix of protocol versions, cipher suites, and end-user devices.

When a client wishes to establish a group or add clients to a group, it first contacts the DS to request KeyPackages for each of the other clients, authenticates the KeyPackages using the signature keys, includes the KeyPackages in Add proposals, and encrypts the information needed to join the group (the *GroupInfo* object) with an ephemeral key; it then separately encrypts the ephemeral key with the public encryption key (`init_key`) from each KeyPackage. When a client requests a KeyPackage in order to add a user to a group, the DS should provide the minimum number of KeyPackages necessary to satisfy the request. For example, if the request specifies the MLS version, the DS might provide one KeyPackage per supported cipher suite, even if it has multiple such KeyPackages to enable the corresponding client to be added to multiple groups before needing to upload more fresh KeyPackages.

In order to avoid replay attacks and provide forward secrecy for messages sent using the initial keying material, KeyPackages are intended to be used only once, and `init_key` is intended to be deleted by the client after decryption of the Welcome message. The DS is responsible for ensuring that each KeyPackage is only used to add its client to a single group, with the possible exception of a "last resort" KeyPackage that is specially designated by the client to be used multiple times. Clients are responsible for providing new KeyPackages as necessary in order to minimize the chance that the "last resort" KeyPackage will be used.

> **Recommendation:** Ensure that "last resort" KeyPackages don't get used by provisioning enough standard KeyPackages.
>
> **Recommendation:** Rotate "last resort" KeyPackages as soon as possible after being used or if they have been stored for a prolonged period of time. Overall, avoid reusing "last resort" KeyPackages as much as possible.
>
> **Recommendation:** Ensure that the client for which a "last resort" KeyPackage has been used is updating leaf keys as early as possible.
>
> **Recommendation:** Ensure that clients delete the private component of their `init_key` after processing a Welcome message, or after the rotation of the "last resort" KeyPackage.

Overall, it needs to be noted that key packages need to be updated when signature keys are changed.

5.2. Delivery of Messages

The main responsibility of the DS is to ensure delivery of messages. Some MLS messages need only be delivered to specific clients (e.g., a Welcome message initializing a new member's state), while others need to be delivered to all the members of a group. The DS may enable the latter delivery pattern via unicast channels (sometimes known as "client fanout"), broadcast channels ("server fanout"), or a mix of both.

For the most part, MLS does not require the DS to deliver messages in any particular order. Applications can set policies that control their tolerance for out-of-order messages (see Section 7), and messages that arrive significantly out of order can be dropped without otherwise affecting the protocol. There are two exceptions to this. First, Proposal messages should all arrive before the Commit that references them. Second, because an MLS group has a linear history of epochs, the members of the group must agree on the order in which changes are applied. Concretely, the group must agree on a single MLS Commit message that ends each epoch and begins the next one.

In practice, there's a realistic risk of two members generating Commit messages at the same time, based on the same epoch, and both attempting to send them to the group at the same time. The extent to which this is a problem, and the appropriate solution, depend on the design of the DS. Per the CAP theorem [CAPBR], there are two general classes of distributed systems that the DS might fall into:

- Consistent and Partition-tolerant, or Strongly Consistent, systems, which can provide a globally consistent view of data but have the inconvenience of clients needing to handle rejected messages.
- Available and Partition-tolerant, or Eventually Consistent, systems, which continue working despite network issues but may return different views of data to different users.

Strategies for sequencing messages in strongly and eventually consistent systems are described in the next two subsections. Most DSs will use the strongly consistent paradigm, but this remains a choice that can be handled in coordination with the client and advertised in the KeyPackages.

However, note that a malicious DS could also reorder messages or provide an inconsistent view to different users. The "generation" counter in MLS messages provides per-sender loss detection and ordering that cannot be manipulated by the DS, but this does not provide complete protection against partitioning. A DS can cause a partition in the group by partitioning key exchange messages; this can be detected only by out-of-band comparison (e.g., confirming that all clients have the same `epoch_authenticator` value). A mechanism for more robust protections is discussed in [EXTENSIONS].

Other forms of DS misbehavior are still possible that are not easy to detect. For instance, a DS can simply refuse to relay messages to and from a given client. Without some sort of side information, other clients cannot generally detect this form of Denial-of-Service (DoS) attack.

5.2.1. Strongly Consistent

With this approach, the DS ensures that some types of incoming messages have a linear order and all members agree on that order. The Delivery Service is trusted to break ties when two members send a Commit message at the same time.

As an example, there could be an "ordering server" DS that broadcasts all messages received to all users and ensures that all clients see messages in the same order. This would allow clients to only apply the first valid Commit for an epoch and ignore subsequent Commits. Clients that send a Commit would then wait to apply it until it is broadcast back to them by the Delivery Service, assuming that they do not receive another Commit first.

Alternatively, the DS can rely on the `epoch` and `content_type` fields of an MLSMessage to provide an order only to handshake messages, and possibly even filter or reject redundant Commit messages proactively to prevent them from being broadcast. There is some risk associated with filtering; this is discussed further in Section 5.3.

5.2.2. Eventually Consistent

With this approach, the DS is built in a way that may be significantly more available or performant than a strongly consistent system, but where it offers weaker consistency guarantees. Messages may arrive to different clients in different orders and with varying amounts of latency, which means clients are responsible for reconciliation.

This type of DS might arise, for example, when group members are sending each message to each other member individually or when a distributed peer-to-peer network is used to broadcast messages.

Upon receiving a Commit from the DS, clients can either:

1. Pause sending new messages for a short amount of time to account for a reasonable degree of network latency and see if any other Commits are received for the same epoch. If multiple Commits are received, the clients can use a deterministic tie-breaking policy to decide which to accept, and then resume sending messages as normal.
2. Accept the Commit immediately but keep a copy of the previous group state for a short period of time. If another Commit for a past epoch is received, clients use a deterministic tie-breaking policy to decide if they should continue using the Commit they originally accepted or revert and use the later one. Note that any copies of previous or forked group states must be deleted within a reasonable amount of time to ensure that the protocol provides forward secrecy.

If the Commit references an unknown proposal, group members may need to solicit the DS or other group members individually for the contents of the proposal.

5.2.3. Welcome Messages

Whenever a commit adds new members to a group, MLS requires the committer to send a Welcome message to the new members. Applications should ensure that Welcome messages are coupled with the tie-breaking logic for commits (see Sections 5.2.1 and 5.2.2). That is, when multiple commits are sent for the same epoch, applications need to ensure that only Welcome messages corresponding to the commit that "succeeded" are processed by new members.

This is particularly important when groups are being reinitialized. When a group is reinitialized, it is restarted with a different protocol version and/or cipher suite but identical membership. Whenever an authorized member sends and commits a ReInit proposal, this immediately freezes the existing group and triggers the creation of a new group with a new `group_id`.

Ideally, the new group would be created by the same member that committed the `ReInit` proposal (including sending Welcome messages for the new group to all of the previous group's members). However, this operation is not always atomic, so it's possible for a member to go

offline after committing a ReInit proposal but before creating the new group. If this happens, it's necessary for another member to continue the reinitialization by creating the new group and sending out Welcome messages.

This has the potential to create a race condition, where multiple members try to continue the reinitialization at the same time, and members receive multiple Welcome messages for each attempt at reinitializing the same group. Ensuring that all members agree on which reinitialization attempt is "correct" is key to prevent this from causing forks.

5.3. Invalid Commits

Situations can arise where a malicious or buggy client sends a Commit that is not accepted by all members of the group, and the DS is not able to detect this and reject the Commit. For example, a buggy client might send an encrypted Commit with an invalid set of proposals, or a malicious client might send a malformed Commit of the form described in Section 16.12 of [RFC9420].

In situations where the DS is attempting to filter redundant Commits, the DS might update its internal state under the assumption that a Commit has succeeded and thus end up in a state inconsistent with the members of the group. For example, the DS might think that the current epoch is now `n+1` and reject any commits from other epochs, while the members think the epoch is `n`, and as a result, the group is stuck -- no member can send a Commit that the DS will accept.

Such "desynchronization" problems can arise even when the DS takes no stance on which Commit is "correct" for an epoch. The DS can enable clients to choose between Commits, for example by providing Commits in the order received and allowing clients to reject any Commits that violate their view of the group's policies. As such, all honest and correctly implemented clients will arrive at the same "first valid Commit" and choose to process it. Malicious or buggy clients that process a different Commit will end up in a forked view of the group.

When these desynchronizations happen, the application may choose to take action to restore the functionality of the group. These actions themselves can have security implications. For example, a client developer might have a client automatically rejoin a group, using an external join, when it processes an invalid Commit. In this operation, however, the client trusts that the GroupInfo provided by the DS faithfully represents the state of the group, and not, say, an earlier state containing a compromised leaf node. In addition, the DS may be able to trigger this condition by deliberately sending the victim an invalid Commit. In certain scenarios, this trust can enable the DS or a malicious insider to undermine the post-compromise security guarantees provided by MLS.

Actions to recover from desynchronization can also have availability and DoS implications. For example, if a recovery mechanism relies on external joins, a malicious member that deliberately posts an invalid Commit could also post a corrupted GroupInfo object in order to prevent victims from rejoining the group. Thus, careful analysis of security implications should be made for any system for recovering from desynchronization.

6. Functional Requirements

MLS is designed as a large-scale group messaging protocol and hence aims to provide both performance and security (e.g., integrity and confidentiality) to its users. Messaging systems that implement MLS provide support for conversations involving two or more members, and aim to scale to groups with tens of thousands of members, typically including many users using multiple devices.

6.1. Membership Changes

MLS aims to provide agreement on group membership, meaning that all group members have agreed on the list of current group members.

Some applications may wish to enforce Access Control Lists (ACLs) to limit addition or removal of group members to privileged clients or users. Others may wish to require authorization from the current group members or a subset thereof. Such policies can be implemented at the application layer, on top of MLS. Regardless, MLS does not allow for or support addition or removal of group members without informing all other members.

Membership of an MLS group is managed at the level of individual clients. In most cases, a client corresponds to a specific device used by a user. If a user has multiple devices, the user will generally be represented in a group by multiple clients (although applications could choose to have devices share keying material). If an application wishes to implement operations at the level of users, it is up to the application to track which clients belong to a given user and ensure that they are added/removed consistently.

MLS provides two mechanisms for changing the membership of a group. The primary mechanism is for an authorized member of the group to send a Commit that adds or removes other members. A secondary mechanism is an "external join": A member of the group publishes certain information about the group, which a new member can use to construct an "external" Commit message that adds the new member to the group. (There is no similarly unilateral way for a member to leave the group; they must be removed by a remaining member.)

With both mechanisms, changes to the membership are initiated from inside the group. When members perform changes directly, this is clearly the case. External joins are authorized indirectly, in the sense that a member publishing a GroupInfo object authorizes anyone to join who has access to the GroupInfo object, subject to whatever access control policies the application applies for external joins.

Both types of joins are done via a Commit message, which could be blocked by the DS or rejected by clients if the join is not authorized. The former approach requires that Commits be visible to the DS; the latter approach requires that clients all share a consistent policy. In the unfortunate event that an unauthorized member is able to join, MLS enables any member to remove them.

Application setup may also determine other criteria for membership validity. For example, per-device signature keys can be signed by an identity key recognized by other participants. If a certificate chain is used to authenticate device signature keys, then revocation by the owner adds an alternative mechanism to prompt membership removal.

An MLS group's secrets change on every change of membership, so each client only has access to the secrets used by the group while they are a member. Messages sent before a client joins or after they are removed are protected with keys that are not accessible to the client. Compromise of a member removed from a group does not affect the security of messages sent after their removal. Messages sent during the client's membership are also secure as long as the client has properly implemented the MLS deletion schedule, which calls for the secrets used to encrypt or decrypt a message to be deleted after use, along with any secrets that could be used to derive them.

6.2. Parallel Groups

Any user or client may have membership in several groups simultaneously. The set of members of any group may or may not overlap with the members of another group. MLS guarantees that the FS and PCS goals within a given group are maintained and not weakened by user membership in multiple groups. However, actions in other groups likewise do not strengthen the FS and PCS guarantees within a given group, e.g., key updates within a given group following a device compromise do not provide PCS healing in other groups; each group must be updated separately to achieve these security objectives. This also applies to future groups that a member has yet to join, which are likewise unaffected by updates performed in current groups.

Applications can strengthen connectivity among parallel groups by requiring periodic key updates from a user across all groups in which they have membership.

MLS provides a pre-shared key (PSK) mechanism that can be used to link healing properties among parallel groups. For example, suppose a common member M of two groups A and B has performed a key update in group A but not in group B. The key update provides PCS with regard to M in group A. If a PSK is exported from group A and injected into group B, then some of these PCS properties carry over to group B, since the PSK and secrets derived from it are only known to the new, updated version of M, not to the old, possibly compromised version of M.

6.3. Asynchronous Usage

No operation in MLS requires two distinct clients or members to be online simultaneously. In particular, members participating in conversations protected using MLS can update the group's keys, add or remove new members, and send messages without waiting for another user's reply.

Messaging systems that implement MLS have to provide a transport layer for delivering messages asynchronously and reliably.

6.4. Access Control

Because all clients within a group (members) have access to the shared cryptographic material, the MLS protocol allows each member of the messaging group to perform operations. However, every service/infrastructure has control over policies applied to its own clients. Applications managing MLS clients can be configured to allow for specific group operations. On the one hand, an application could decide that a group administrator will be the only member to perform Add and Remove operations. On the other hand, in many settings such as open discussion forums, joining can be allowed for anyone.

While MLS application messages are always encrypted, MLS handshake messages can be sent either encrypted (in an MLS PrivateMessage) or unencrypted (in an MLS PublicMessage). Applications may be designed such that intermediaries need to see handshake messages, for example to enforce policy on which commits are allowed, or to provide MLS ratchet tree data in a central location. If handshake messages are unencrypted, it is especially important that they be sent over a channel with strong transport encryption (see Section 8) in order to prevent external attackers from monitoring the status of the group. Applications that use unencrypted handshake messages may take additional steps to reduce the amount of metadata that is exposed to the intermediary. Everything else being equal, using encrypted handshake messages provides stronger privacy properties than using unencrypted handshake messages, as it prevents intermediaries from learning about the structure of the group.

If handshake messages are encrypted, any access control policies must be applied at the client, so the application must ensure that the access control policies are consistent across all clients to make sure that they remain in sync. If two different policies were applied, the clients might not accept or reject a group operation and end up in different cryptographic states, breaking their ability to communicate.

> **Recommendation:** Avoid using inconsistent access control policies, especially when using encrypted group operations.

MLS allows actors outside the group to influence the group in two ways: External signers can submit proposals for changes to the group, and new joiners can use an external join to add themselves to the group. The `external_senders` extension ensures that all members agree on which signers are allowed to send proposals, but any other policies must be assured to be consistent, as noted above.

> **Recommendation:** Have an explicit group policy setting the conditions under which external joins are allowed.

6.5. Handling Authentication Failures

Within an MLS group, every member is authenticated to every other member by means of credentials issued and verified by the AS. MLS does not prescribe what actions, if any, an application should take in the event that a group member presents an invalid credential. For example, an application may require such a member to be immediately evicted or may allow

some grace period for the problem to be remediated. To avoid operational problems, it is important for all clients in a group to have a consistent view of which credentials in a group are valid, and how to respond to invalid credentials.

> **Recommendation:** Have a uniform credential validation process to ensure that all group members evaluate other members' credentials in the same way.

> **Recommendation:** Have a uniform policy for how invalid credentials are handled.

In some authentication systems, it is possible for a previously valid credential to become invalid over time. For example, in a system based on X.509 certificates, credentials can expire or be revoked. The MLS update mechanisms allow a client to replace an old credential with a new one. This is best done before the old credential becomes invalid.

> **Recommendation:** Proactively rotate credentials, especially if a credential is about to become invalid.

6.6. Recovery After State Loss

Group members whose local MLS state is lost or corrupted can reinitialize their state by rejoining the group as a new member and removing the member representing their earlier state. An application can require that a client performing such a reinitialization prove its prior membership with a PSK that was exported from the previous state.

There are a few practical challenges to this approach. For example, the application will need to ensure that all members have the required PSK, including any new members that have joined the group since the epoch in which the PSK was issued. And of course, if the PSK is lost or corrupted along with the member's other state, then it cannot be used to recover.

Reinitializing in this way does not provide the member with access to group messages exchanged during the state loss window, but enables proof of prior membership in the group. Applications may choose various configurations for providing lost messages to valid group members that are able to prove prior membership.

6.7. Support for Multiple Devices

It is common for users within a group to own multiple devices. A new device can be added to a group and be considered as a new client by the protocol. This client will not gain access to the history even if it is owned by someone who owns another member of the group. MLS does not provide direct support for restoring history in this case, but applications can elect to provide such a mechanism outside of MLS. Such mechanisms, if used, may reduce the FS and PCS guarantees provided by MLS.

6.8. Extensibility

The MLS protocol provides several extension points where additional information can be provided. Extensions to KeyPackages allow clients to disclose additional information about their capabilities. Groups can also have extension data associated with them, and the group agreement properties of MLS will confirm that all members of the group agree on the content of these extensions.

6.9. Application Data Framing and Type Advertisements

Application messages carried by MLS are opaque to the protocol and can contain arbitrary data. Each application that uses MLS needs to define the format of its `application_data` and any mechanism necessary to determine the format of that content over the lifetime of an MLS group. In many applications, this means managing format migrations for groups with multiple members who may each be offline at unpredictable times.

> **Recommendation:** Use the content mechanism defined in [EXTENSIONS], unless the specific application defines another mechanism that more appropriately addresses the same requirements for that application of MLS.

The MLS framing for application messages also provides a field where clients can send information that is authenticated but not encrypted. Such information can be used by servers that handle the message, but group members are assured that it has not been tampered with.

6.10. Federation

The protocol aims to be compatible with federated environments. While this document does not specify all necessary mechanisms required for federation, multiple MLS implementations can interoperate to form federated systems if they use compatible authentication mechanisms, cipher suites, application content, and infrastructure functionalities. Federation is described in more detail in [FEDERATION].

6.11. Compatibility with Future Versions of MLS

It is important that multiple versions of MLS be able to coexist in the future. Thus, MLS offers a version negotiation mechanism; this mechanism prevents version downgrade attacks where an attacker would actively rewrite messages with a lower protocol version than the messages originally offered by the endpoints. When multiple versions of MLS are available, the negotiation protocol guarantees that the creator is able to select the best version out of those supported in common by the group.

In MLS 1.0, the creator of the group is responsible for selecting the best cipher suite supported across clients. Each client is able to verify availability of protocol version, cipher suites, and extensions at all times once it has at least received the first group operation message.

Each member of an MLS group advertises the protocol functionality they support. These capability advertisements can be updated over time, e.g., if client software is updated while the client is a member of a group. Thus, in addition to preventing downgrade attacks, the members of a group can also observe when it is safe to upgrade to a new cipher suite or protocol version.

7. Operational Requirements

MLS is a security layer that needs to be integrated with an application. A fully functional deployment of MLS will have to make a number of decisions about how MLS is configured and operated. Deployments that wish to interoperate will need to make compatible decisions. This section lists all of the dependencies of an MLS deployment that are external to the protocol specification, but would still need to be aligned within a given MLS deployment, or for two deployments to potentially interoperate.

The protocol has a built-in ability to negotiate protocol versions, cipher suites, extensions, credential types, and additional proposal types. For two deployments to interoperate, they must have overlapping support in each of these categories. The `required_capabilities` extension (Section 7.2 of [RFC9420]) can promote interoperability with a wider set of clients by ensuring that certain functionality continues to be supported by a group, even if the clients in the group aren't currently relying on it.

MLS relies on the following network services, which need to be compatible in order for two different deployments based on them to interoperate.

- An **Authentication Service**, described fully in Section 4, defines the types of credentials which may be used in a deployment and provides methods for:
  1. Issuing new credentials with a relevant credential lifetime,
  2. Validating a credential against a reference identifier,
  3. Validating whether or not two credentials represent the same client, and
  4. Optionally revoking credentials which are no longer authorized.
- A **Delivery Service**, described fully in Section 5, provides methods for:
  1. Delivering messages for a group to all members in the group.
  2. Delivering Welcome messages to new members of a group.
  3. Uploading new KeyPackages for a user's own clients.
  4. Downloading KeyPackages for specific clients. Typically, KeyPackages are used once and consumed.
- Additional services may or may not be required, depending on the application design:
  - In cases where group operations are not encrypted, the DS has the ability to observe and maintain a copy of the public group state. In particular, this is useful for either (1) clients that do not have the ability to send the full public state in a Welcome message when inviting a user or (2) clients that need to recover from losing their state. Such public state

can contain privacy-sensitive information such as group members' credentials and related public keys; hence, services need to carefully evaluate the privacy impact of storing this data on the DS.

- If external joiners are allowed, there must be a method for publishing a serialized `GroupInfo` object (with an `external_pub` extension) that corresponds to a specific group and epoch, and for keeping that object in sync with the state of the group.
- If an application chooses not to allow external joining, it may instead provide a method for external users to solicit group members (or a designated service) to add them to a group.
- If the application uses PSKs that members of a group may not have access to (e.g., to control entry into the group or to prove membership in the group in the past, as discussed in Section 6.6), there must be a method for distributing these PSKs to group members who might not have them -- for instance, if they joined the group after the PSK was generated.
- If an application wishes to detect and possibly discipline members that send malformed commits with the intention of corrupting a group's state, there must be a method for reporting and validating malformed commits.

MLS requires the following parameters to be defined, which must be the same for two implementations to interoperate:

- The maximum total lifetime that is acceptable for a KeyPackage.
- How long to store the resumption PSK for past epochs of a group.
- The degree of tolerance that's allowed for out-of-order message delivery:
  - How long to keep unused nonce and key pairs for a sender.
  - A maximum number of unused key pairs to keep.
  - A maximum number of steps that clients will move a secret tree ratchet forward in response to a single message before rejecting it.
  - Whether to buffer messages that aren't yet able to be understood due to other messages not arriving first, and, if so, how many and for how long -- for example, Commit messages that arrive before a proposal they reference or application messages that arrive before the Commit starting an epoch.

If implementations differ in these parameters, they will interoperate to some extent but may experience unexpected failures in certain situations, such as extensive message reordering.

MLS provides the following locations where an application may store arbitrary data. The format and intention of any data in these locations must align for two deployments to interoperate:

- Application data, sent as the payload of an encrypted message.
- Additional authenticated data, sent unencrypted in an otherwise encrypted message.
- Group IDs, as decided by group creators and used to uniquely identify a group.
- Application-level identifiers of public key material (specifically, the `application_id` extension as defined in Section 5.3.3 of [RFC9420]).

MLS requires the following policies to be defined, which restrict the set of acceptable behaviors in a group. These policies must be consistent between deployments for them to interoperate:

- A policy on which cipher suites are acceptable.
- A policy on any mandatory or forbidden MLS extensions.
- A policy on when to send proposals and commits in plaintext instead of encrypted.
- A policy for which proposals are valid to have in a commit, including but not limited to:
  - When a member is allowed to add or remove other members of the group.
  - When, and under what circumstances, a reinitialization proposal is allowed.
  - When proposals from external senders are allowed and how to authorize those proposals.
  - When external joiners are allowed and how to authorize those external commits.
  - Which other proposal types are allowed.
- A policy of when members should commit pending proposals in a group.
- A policy of how to protect and share the GroupInfo objects needed for external joins.
- A policy for when two credentials represent the same client, distinguishing the following two cases:
  - When there are multiple devices for a given user.
  - When a single device has multiple signature keys -- for instance, if the device has keys corresponding to multiple overlapping time periods.
- A policy on how long to allow a member to stay in a group without updating its leaf keys before removing them.

Finally, there are some additional application-defined behaviors that are partially an individual application's decision but may overlap with interoperability:

- When and how to pad messages.
- When to send a reinitialization proposal.
- How often clients should update their leaf keys.
- Whether to prefer sending full commits or partial/empty commits.
- Whether there should be a `required_capabilities` extension in groups.

8. Security and Privacy Considerations

MLS adopts the Internet threat model [RFC3552] and therefore assumes that the attacker has complete control of the network. It is intended to provide the security services described in Section 8.2 in the face of attackers who can:

- Monitor the entire network.
- Read unprotected messages.
- Generate, inject, and delete any message in the unprotected transport layer.

While MLS should be run over a secure transport such as QUIC [RFC9000] or TLS [RFC8446], the security guarantees of MLS do not depend on the transport. This departs from the usual design practice of trusting the transport because MLS is designed to provide security even in the face of compromised network elements, especially the DS.

Generally, MLS is designed under the assumption that the transport layer is present to keep metadata private from network observers, while the MLS protocol provides confidentiality, integrity, and authentication guarantees for the application data (which could pass through multiple systems). Additional properties such as partial anonymity or deniability could also be achieved in specific architecture designs.

In addition, these guarantees are intended to degrade gracefully in the presence of compromise of the transport security links as well as of both clients and elements of the messaging system, as described in the remainder of this section.

8.1. Assumptions on Transport Security Links

As discussed above, MLS provides the highest level of security when its messages are delivered over an encrypted transport, thus preventing attackers from selectively interfering with MLS communications as well as protecting the already limited amount of metadata. Very little information is contained in the unencrypted header of the MLS protocol message format for group operation messages, and application messages are always encrypted in MLS.

> **Recommendation:** Use transports that provide reliability and metadata confidentiality whenever possible, e.g., by transmitting MLS messages over a protocol such as TLS [RFC8446] or QUIC [RFC9000].

MLS avoids the need to send the full list of recipients to the server for dispatching messages because that list could potentially contain tens of thousands of recipients. Header metadata in MLS messages typically consists of an opaque `group_id`, a numerical value to determine the epoch of the group (the number of changes that have been made to the group), and whether the message is an application message, a proposal, or a commit.

Even though some of this metadata information does not consist of sensitive information, when correlated with other data a network observer might be able to reconstruct sensitive information. Using a secure channel to transfer this information will prevent a network attacker from accessing this MLS protocol metadata if it cannot compromise the secure channel.

8.1.1. Integrity and Authentication of Custom Metadata

MLS provides an authenticated "Additional Authenticated Data" (AAD) field for applications to make data available outside a PrivateMessage, while cryptographically binding it to the message.

> **Recommendation:** Use the "Additional Authenticated Data" field of the PrivateMessage instead of using other unauthenticated means of sending metadata throughout the infrastructure. If the data should be kept private, the infrastructure should use encrypted application messages instead.

8.1.2. Metadata Protection for Unencrypted Group Operations

Having no secure channel to exchange MLS messages can have a serious impact on privacy when transmitting unencrypted group operation messages. Observing the contents and signatures of the group operation messages may lead an adversary to extract information about the group membership.

> **Recommendation:** Never use the unencrypted mode for group operations without using a secure channel for the transport layer.

8.1.3. DoS Protection

In general, we do not consider DoS resistance to be the responsibility of the protocol. However, it should not be possible for anyone aside from the DS to perform a trivial DoS attack from which it is hard to recover. This can be achieved through the secure transport layer, which prevents selective attack on MLS communications by network attackers.

In the centralized setting, DoS protection can typically be performed by using tickets or cookies which identify users to a service for a certain number of connections. Such a system helps in preventing anonymous clients from sending arbitrary numbers of group operation messages to the DS or the MLS clients.

> **Recommendation:** Use credentials uncorrelated with specific users to help prevent DoS attacks, in a privacy-preserving manner. Note that the privacy of these mechanisms has to be adjusted in accordance with the privacy expected from secure transport links. (See more discussion in the next section.)

8.1.4. Message Suppression and Error Correction

As noted above, MLS is designed to provide some robustness in the face of tampering within the secure transport, e.g., tampering by the DS. The confidentiality and authenticity properties of MLS prevent the DS from reading or writing messages. MLS also provides a few tools for detecting message suppression, with the caveat that message suppression cannot always be distinguished from transport failure.

Each encrypted MLS message carries a per-sender incrementing "generation" number. If a group member observes a gap in the generation sequence for a sender, then they know that they have missed a message from that sender. MLS also provides a facility for group members to send authenticated acknowledgments of application messages received within a group.

As discussed in Section 5, the DS is trusted to select the single Commit message that is applied in each epoch from among the Commits sent by group members. Since only one Commit per epoch is meaningful, it's not useful for the DS to transmit multiple Commits to clients. The risk remains that the DS will use the ability maliciously.

8.2. Intended Security Guarantees

MLS aims to provide a number of security guarantees, covering authentication, as well as confidentiality guarantees to different degrees in different scenarios.

8.2.1. Message Secrecy and Authentication

MLS enforces the encryption of application messages and thus generally guarantees authentication and confidentiality of application messages sent in a group.

In particular, this means that only other members of a given group can decrypt the payload of a given application message, which includes information about the sender of the message.

Similarly, group members receiving a message from another group member can authenticate that group member as the sender of the message and verify the message's integrity.

Message content can be deniable if the signature keys are exchanged over a deniable channel prior to signing messages.

Depending on the group settings, handshake messages can be encrypted as well. If that is the case, the same security guarantees apply.

MLS optionally allows the addition of padding to messages, mitigating the amount of information leaked about the length of the plaintext to an observer on the network.

8.2.2. Forward Secrecy and Post-Compromise Security

MLS provides additional protection regarding secrecy of past messages and future messages. These cryptographic security properties are forward secrecy (FS) and post-compromise security (PCS).

FS means that access to all encrypted traffic history combined with access to all current keying material on clients will not defeat the secrecy properties of messages older than the oldest key of the compromised client. Note that this means that clients have to delete the appropriate keys as soon as they have been used with the expected message; otherwise, the secrecy of the messages and the security of MLS are considerably weakened.

PCS means that if a group member's state is compromised at some time t1 but the group member subsequently performs an update at some time t2, then all MLS guarantees apply to messages sent by the member after time t2 and to messages sent by other members after they have processed the update. For example, if an attacker learns all secrets known to Alice at time t1, including both Alice's long-term secret keys and all shared group keys, but Alice performs a key update at time t2, then the attacker is unable to violate any of the MLS security properties after the updates have been processed.

Both of these properties are satisfied even against compromised DSs and ASes in the case where some other mechanism for verifying keys is in use, such as Key Transparency [KT].

Confidentiality is mainly ensured on the client side. Because FS and PCS rely on the active deletion and replacement of keying material, any client which is persistently offline may still be holding old keying material and thus be a threat to both FS and PCS if it is later compromised.

MLS partially defends against this problem by active members including new keying material. However, not much can be done on the inactive side especially in the case where the client has not processed messages.

> **Recommendation:** Mandate key updates from clients that are not otherwise sending messages and evict clients that are idle for too long.

These recommendations will reduce the ability of idle compromised clients to decrypt a potentially long set of messages that might have been sent after the point of compromise.

The precise details of such mechanisms are a matter of local policy and beyond the scope of this document.

8.2.3. Non-Repudiation vs. Deniability

MLS provides strong authentication within a group, such that a group member cannot send a message that appears to be from another group member. Additionally, some services require that a recipient be able to prove to the service provider that a message was sent by a given client, in order to report abuse. MLS supports both of these use cases. In some deployments, these services are provided by mechanisms which allow the receiver to prove a message's origin to a third party. This is often called "non-repudiation".

Roughly speaking, "deniability" is the opposite of "non-repudiation", i.e., the property that it is impossible to prove to a third party that a message was sent by a given sender. MLS does not make any claims with regard to deniability. It may be possible to operate MLS in ways that provide certain deniability properties, but defining the specific requirements and resulting notions of deniability requires further analysis.

8.2.4. Associating a User's Clients

When a user has multiple devices, the base MLS protocol only describes how to operate each device as a distinct client in the MLS groups that the user is a member of. As a result, the other members of the group will be able to identify which of a user's devices sent each message and, therefore, which device the user was using at the time. Group members would also be able to detect when the user adds or removes authorized devices from their account. For some applications, this may be an unacceptable breach of the user's privacy.

This risk only arises when the leaf nodes for the clients in question provide data that can be used to correlate the clients. One way to mitigate this risk is by only doing client-level authentication within MLS. If user-level authentication is still desirable, the application would have to provide it through some other mechanism.

It is also possible to maintain user-level authentication while hiding information about the clients that a user owns. This can be done by having the clients share cryptographic state, so that they appear as a single client within the MLS group. Appearing as a single client has the privacy

benefits of no longer leaking which device was used to send a particular message and no longer leaking the user's authorized devices. However, the application would need to provide a synchronization mechanism so that the state of each client remains consistent across changes to the MLS group. Flaws in this synchronization mechanism may impair the ability of the user to recover from a compromise of one of their devices. In particular, state synchronization may make it easier for an attacker to use one compromised device to establish exclusive control of a user's account, locking them out entirely and preventing them from recovering.

8.3. Endpoint Compromise

The MLS protocol adopts a threat model which includes multiple forms of endpoint/client compromise. While adversaries are in a strong position if they have compromised an MLS client, there are still situations where security guarantees can be recovered thanks to the PCS properties achieved by the MLS protocol.

In this section we will explore the consequences and recommendations regarding the following compromise scenarios:

- The attacker has access to a symmetric encryption key.
- The attacker has access to an application ratchet secret.
- The attacker has access to the group secrets for one group.
- The attacker has access to a signature oracle for any group.
- The attacker has access to the signature key for one group.
- The attacker has access to all secrets of a user for all groups (full state compromise).

8.3.1. Compromise of Symmetric Keying Material

As described above, each MLS epoch creates a new group secret.

These group secrets are then used to create a per-sender ratchet secret, which in turn is used to create a per-sender Authenticated Encryption with Associated Data (AEAD) [RFC5116] key that is then used to encrypt MLS plaintext messages. Each time a message is sent, the ratchet secret is used to create a new ratchet secret and a new corresponding AEAD key. Because of the properties of the key derivation function, it is not possible to compute a ratchet secret from its corresponding AEAD key or compute ratchet secret n-1 from ratchet secret n.

Below, we consider the compromise of each of these pieces of keying material in turn, in ascending order of severity. While this is a limited kind of compromise, it can be realistic in cases of implementation vulnerabilities where only part of the memory leaks to the adversary.

8.3.1.1. Compromise of AEAD Keys

In some circumstances, adversaries may have access to specific AEAD keys and nonces which protect an application message or a group operation message. Compromise of these keys allows the attacker to decrypt the specific message encrypted with that key but no other; because the AEAD keys are derived from the ratchet secret, it cannot generate the next ratchet secret and hence not the next AEAD key.

In the case of an application message, an AEAD key compromise means that the encrypted application message will be leaked as well as the signature over that message. This means that the compromise has both confidentiality and privacy implications on the future AEAD encryptions of that chain. In the case of a group operation message, only the privacy is affected, as the signature is revealed, because the secrets themselves are protected by Hybrid Public Key Encryption (HPKE). Note that under that compromise scenario, authentication is not affected in either of these cases. As every member of the group can compute the AEAD keys for all the chains (they have access to the group secrets) in order to send and receive messages, the authentication provided by the AEAD encryption layer of the common framing mechanism is weak. Successful decryption of an AEAD encrypted message only guarantees that some member of the group -- or in this case an attacker who has compromised the AEAD keys -- sent the message.

Compromise of the AEAD keys allows the attacker to send an encrypted message using that key, but the attacker cannot send a message to a group that appears to be from any valid client because the attacker cannot forge the signature. This applies to all the forms of symmetric key compromise described in Section 8.3.1.

8.3.1.2. Compromise of Ratchet Secret Material

When a ratchet secret is compromised, the adversary can compute both the current AEAD keys for a given sender and any future keys for that sender in this epoch. Thus, it can decrypt current and future messages by the corresponding sender. However, because it does not have previous ratchet secrets, it cannot decrypt past messages as long as those secrets and keys have been deleted.

Because of its forward secrecy guarantees, MLS will also retain secrecy of all other AEAD keys generated for *other* MLS clients, outside this dedicated chain of AEAD keys and nonces, even within the epoch of the compromise. MLS provides post-compromise security against an active adaptive attacker across epochs for AEAD encryption, which means that as soon as the epoch is changed, if the attacker does not have access to more secret material they won't be able to access any protected messages from future epochs.

8.3.1.3. Compromise of the Group Secrets of a Single Group for One or More Group Epochs

An adversary who gains access to a set of group secrets -- as when a member of the group is compromised -- is significantly more powerful. In this section, we consider the case where the signature keys are not compromised. This can occur if the attacker has access to part of the memory containing the group secrets but not to the signature keys which might be stored in a secure enclave.

In this scenario, the adversary gains the ability to compute any number of ratchet secrets for the epoch and their corresponding AEAD encryption keys and thus can encrypt and decrypt all messages for the compromised epochs.

If the adversary is passive, it is expected from the PCS properties of the MLS protocol that as soon as the compromised party remediates the compromise and sends an honest Commit message, the next epochs will provide message secrecy.

If the adversary is active, the adversary can engage in the protocol itself and perform updates on behalf of the compromised party with no ability for an honest group to recover message secrecy. However, MLS provides PCS against active adaptive attackers through its Remove group operation. This means that as long as other members of the group are honest, the protocol will guarantee message secrecy for all messages exchanged in the epochs after the compromised party has been removed.

8.3.2. Compromise by an Active Adversary with the Ability to Sign Messages

If an active adversary has compromised an MLS client and can sign messages, two different scenarios emerge. In the strongest compromise scenario, the attacker has access to the signing key and can forge authenticated messages. In a weaker, yet realistic scenario, the attacker has compromised a client but the client signature keys are protected with dedicated hardware features which do not allow direct access to the value of the private key and instead provide a signature API.

When considering an active adaptive attacker with access to a signature oracle, the compromise scenario implies a significant impact on both the secrecy and authentication guarantees of the protocol, especially if the attacker also has access to the group secrets. In that case, both secrecy and authentication are broken. The attacker can generate any message, for the current and future epochs, until the compromise is remediated and the formerly compromised client sends an honest update.

Note that under this compromise scenario, the attacker can perform all operations which are available to a legitimate client even without access to the actual value of the signature key.

8.3.3. Compromise of Authentication with Access to a Signature Key

The difference between having access to the value of the signature key and only having access to a signing oracle is not about the ability of an active adaptive network attacker to perform different operations during the time of the compromise; the attacker can perform every operation available to a legitimate client in both cases.

There is a significant difference, however, in terms of recovery after a compromise.

Because of the PCS guarantees provided by the MLS protocol, when a previously compromised client recovers from compromise and performs an honest Commit, both secrecy and authentication of future messages can be recovered as long as the attacker doesn't otherwise get access to the key. Because the adversary doesn't have the signing key, they cannot authenticate messages on behalf of the compromised party, even if they still have control over some group keys by colluding with other members of the group.

This is in contrast with the case where the signature key is leaked. In that case, the compromised endpoint needs to refresh its credentials and invalidate the old credentials before the attacker will be unable to authenticate messages.

Beware that in both oracle and private key access, an active adaptive attacker can follow the protocol and request to update its own credential. This in turn induces a signature key rotation, which could provide the attacker with part or the full value of the private key, depending on the architecture of the service provider.

> **Recommendation:** Signature private keys should be compartmentalized from other secrets and preferably protected by a Hardware Security Module (HSM) or dedicated hardware features to allow recovery of the authentication for future messages after a compromise.

> **Recommendation:** When the credential type supports revocation, the users of a group should check for revoked keys.

8.3.4. Security Considerations in the Context of a Full State Compromise

In real-world compromise scenarios, it is often the case that adversaries target specific devices to obtain parts of the memory or even the ability to execute arbitrary code in the targeted device.

Also, recall that in this setting, the application will often retain the unencrypted messages. If so, the adversary does not have to break encryption at all to access sent and received messages. Messages may also be sent by using the application to instruct the protocol implementation.

> **Recommendation:** If messages are stored on the device, they should be protected using encryption at rest, and the keys used should be stored securely using dedicated mechanisms on the device.

> **Recommendation:** If the threat model of the system includes an adversary that can access the messages on the device without even needing to attack MLS, the application should delete plaintext and ciphertext messages as soon as practical after encryption or decryption.

Note that this document makes a clear distinction between the way signature keys and other group shared secrets must be handled. In particular, a large set of group secrets cannot necessarily be assumed to be protected by an HSM or secure enclave features. This is especially true because these keys are frequently used and changed with each message received by a client.

However, the signature private keys are mostly used by clients to send a message. They also provide strong authentication guarantees to other clients; hence, we consider that their protection by additional security mechanisms should be a priority.

Overall, there is no way to detect or prevent these compromises, as discussed in the previous sections: Performing separation of the application secret states can help recovery after compromise; this is the case for signature keys, but similar concerns exist for a client's encryption private keys.

> **Recommendation:** The secret keys used for public key encryption should be stored similarly to the way the signature keys are stored, as keys can be used to decrypt the group operation messages and contain the secret material used to compute all the group secrets.

Even if secure enclaves are not perfectly secure or are even completely broken, adopting additional protections for these keys can ease recovery of the secrecy and authentication guarantees after a compromise where, for instance, an attacker can sign messages without having access to the key. In certain contexts, the rotation of credentials might only be triggered by the AS through ACLs and hence be beyond the capabilities of the attacker.

8.4. Service Node Compromise

8.4.1. General Considerations

8.4.1.1. Privacy of the Network Connections

There are many scenarios leading to communication between the application on a device and the DS or the AS. In particular, when:

- The application connects to the AS to generate or validate a new credential before distributing it.
- The application fetches credentials at the DS prior to creating a messaging group (one-to-one or more than two clients).
- The application fetches service provider information or messages on the DS.
- The application sends service provider information or messages to the Delivery Service.

In all these cases, the application will often connect to the device via a secure transport which leaks information about the origin of the request, such as the IP address and -- depending on the protocol -- the MAC address of the device.

Similar concerns exist in the peer-to-peer use cases for MLS.

> **Recommendation:** In the case where privacy or anonymity is important, using adequate protection such as Multiplexed Application Substrate over QUIC Encryption (MASQUE) [MASQUE-PROXY], Tor [Tor], or a VPN can improve metadata protection.

More generally, using anonymous credentials in an MLS-based architecture might not be enough to provide strong privacy or anonymity properties.

8.4.1.2. Storage of Metadata and Encryption at Rest on the Servers

In the case where private data or metadata has to be persisted on the servers for functionality (mappings between identities and push tokens, group metadata, etc.), it should be stored encrypted at rest and only decrypted upon need during the execution. Honest service providers can rely on such "encryption at rest" mechanisms to be able to prevent access to the data when not using it.

> **Recommendation:** Store cryptographic material used for server-side decryption of sensitive metadata on the clients and only send it when needed. The server can use the secret to open and update encrypted data containers after which they can delete these keys until the next time they need it, in which case those can be provided by the client.

> **Recommendation:** Rely on group secrets exported from the MLS session for server-side encryption at rest and update the key after each removal from the group. Otherwise, rotate those keys on a regular basis.

8.4.2. Delivery Service Compromise

MLS is intended to provide strong guarantees in the face of compromise of the DS. Even a totally compromised DS should not be able to read messages or inject messages that will be acceptable to legitimate clients. It should also not be able to undetectably remove, reorder, or replay messages.

However, a malicious DS can mount a variety of DoS attacks on the system, including total DoS attacks (where it simply refuses to forward any messages) and partial DoS attacks (where it refuses to forward messages to and from specific clients). As noted in Section 5.2, these attacks are only partially detectable by clients without an out-of-band channel. Ultimately, failure of the DS to provide reasonable service must be dealt with as a customer service matter, not via technology.

Because the DS is responsible for providing the initial keying material to clients, it can provide stale keys. This does not inherently lead to compromise of the message stream, but does allow the DS to attack post-compromise security to a limited extent. This threat can be mitigated by having initial keys expire.

Initial keying material (KeyPackages) using the `basic` credential type is more vulnerable to replacement by a malicious or compromised DS, as there is no built-in cryptographic binding between the identity and the public key of the client.

> **Recommendation:** Prefer a credential type in KeyPackages which includes a strong cryptographic binding between the identity and its key (for example, the `x509` credential type). When using the `basic` credential type, take extra care to verify the identity (typically out of band).

8.4.2.1. Privacy of Delivery and Push Notifications

Push tokens provide an important mechanism that is often ignored from the standpoint of privacy considerations. In many modern messaging architectures, applications are using push notification mechanisms typically provided by OS vendors. This is to make sure that when messages are available at the DS (or via other mechanisms if the DS is not a central server), the recipient application on a device knows about it. Sometimes the push notification can contain the application message itself, which saves a round trip with the DS.

To "push" this information to the device, the service provider and the OS infrastructures use unique per-device, per-application identifiers called push tokens. This means that the push notification provider and the service provider have information on which devices receive information and at which point in time. Alternatively, non-mobile applications could use a WebSocket or persistent connection for notifications directly from the DS.

Even though the service provider and the push notification provider can't necessarily access the content (typically encrypted MLS messages), no technical mechanism in MLS prevents them from determining which devices are recipients of the same message.

For secure messaging systems, push notifications are often sent in real time, as it is not acceptable to create artificial delays for message retrieval.

> **Recommendation:** If real-time notifications are not necessary, one can delay notifications randomly across recipient devices using a mixnet or other techniques.

Note that with a legal request to ask the service provider for the push token associated with an identifier, it is easy to correlate the token with a second request to the company operating the push notification system to get information about the device, which is often linked with a real identity via a cloud account, a credit card, or other information.

> **Recommendation:** If stronger privacy guarantees are needed with regard to the push notification provider, the client can choose to periodically connect to the DS without the need of a dedicated push notification infrastructure.

Applications can also consider anonymous systems for server fanout (for example, [Loopix]).

8.4.3. Authentication Service Compromise

The Authentication Service design is left to the infrastructure designers. In most designs, a compromised AS is a serious matter, as the AS can serve incorrect or attacker-provided identities to clients.

- The attacker can link an identity to a credential.
- The attacker can generate new credentials.
- The attacker can sign new credentials.
- The attacker can publish or distribute credentials.

An attacker that can generate or sign new credentials may or may not have access to the underlying cryptographic material necessary to perform such operations. In that last case, it results in windows of time for which all emitted credentials might be compromised.

> **Recommendation:** Use HSMs to store the root signature keys to limit the ability of an adversary with no physical access to extract the top-level signature private key.

Note that historically some systems generate signature keys on the AS and distribute the private keys to clients along with their credential. This is a dangerous practice because it allows the AS or an attacker who has compromised the AS to silently impersonate the client.

8.4.3.1. Authentication Compromise: Ghost Users and Impersonation

One important property of MLS is that all members know which other members are in the group at all times. If all members of the group and the AS are honest, no parties other than the members of the current group can read and write messages protected by the protocol for that group.

This guarantee applies to the cryptographic identities of the members. Details about how to verify the identity of a client depend on the MLS credential type used. For example, cryptographic verification of credentials can be largely performed autonomously (e.g., without user interaction) by the clients themselves for the `x509` credential type.

In contrast, when MLS clients use the `basic` credential type, some other mechanism must be used to verify identities. For instance, the Authentication Service could operate some sort of directory server to provide keys, or users could verify keys via an out-of-band mechanism.

> **Recommendation:** Select the MLS credential type with the strongest security which is supported by all target members of an MLS group.

> **Recommendation:** Do not use the same signature key pair across groups. Update all keys for all groups on a regular basis. Do not preserve keys in different groups when suspecting a compromise.

If the AS is compromised, it could validate a signature key pair (or generate a new one) for an attacker. The attacker could then use this key pair to join a group as if it were another of the user's clients. Because a user can have many MLS clients running the MLS protocol, it possibly has many signature key pairs for multiple devices. These attacks could be very difficult to detect, especially in large groups where the UI might not reflect all the changes back to the users. If the application participates in a key transparency mechanism in which it is possible to determine every key for a given user, then this would allow for detection of surreptitiously created false credentials.

> **Recommendation:** Make sure that MLS clients reflect all the membership changes to the users as they happen. If a choice has to be made because the number of notifications is too high, the client should provide a log of state of the device so that the user can examine it.

> **Recommendation:** Provide a key transparency mechanism for the AS to allow public verification of the credentials authenticated by this service.

While the ways to handle MLS credentials are not defined by the protocol or the architecture documents, the MLS protocol has been designed with a mechanism that can be used to provide out-of-band authentication to users. The `authentication_secret` generated for each user at each epoch of the group is a one-time, per-client authentication secret which can be exchanged between users to prove their identities to each other. This can be done, for instance, using a QR code that can be scanned by the other parties.

> **Recommendation:** Provide one or more out-of-band authentication mechanisms to limit the impact of an AS compromise.

We note, again, that the AS may not be a centralized system and could be realized by many mechanisms such as establishing prior one-to-one deniable channels, gossiping, or using trust on first use (TOFU) for credentials used by the MLS protocol.

Another important consideration is the ease of redistributing new keys on client compromise, which helps recovering security faster in various cases.

8.4.3.2. Privacy of the Group Membership

Group membership is itself sensitive information, and MLS is designed to limit the amount of persistent metadata. However, large groups often require an infrastructure that provides server fanout. In the case of client fanout, the destination of a message is known by all clients; hence, the server usually does not need this information. However, servers may learn this information through traffic analysis. Unfortunately, in a server-side fanout model, the Delivery Service can learn that a given client is sending the same message to a set of other clients. In addition, there may be applications of MLS in which the group membership list is stored on some server associated with the DS.

While this knowledge is not a breach of the protocol's authentication or confidentiality guarantees, it is a serious issue for privacy.

Some infrastructures keep a mapping between keys used in the MLS protocol and user identities. An attacker with access to this information due to compromise or regulation can associate unencrypted group messages (e.g., Commits and Proposals) with the corresponding user identity.

> **Recommendation:** Use encrypted group operation messages to limit privacy risks whenever possible.

In certain cases, the adversary can access specific bindings between public keys and identities. If the signature keys are reused across groups, the adversary can get more information about the targeted user.

> **Recommendation:** Ensure that linking between public keys and identities only happens in expected scenarios.

8.5. Considerations for Attacks Outside of the Threat Model

Physical attacks on devices storing and executing MLS principals are not considered in depth in the threat model of the MLS protocol. While non-permanent, non-invasive attacks can sometimes be equivalent to software attacks, physical attacks are considered outside of the MLS threat model.

Compromise scenarios typically consist of a software adversary, which can maintain active adaptive compromise and arbitrarily change the behavior of the client or service.

On the other hand, security goals consider that honest clients will always run the protocol according to its specification. This relies on implementations of the protocol to securely implement the specification, which remains non-trivial.

> **Recommendation:** Additional steps should be taken to protect the device and the MLS clients from physical compromise. In such settings, HSMs and secure enclaves can be used to protect signature keys.

8.6. No Protection Against Replay by Insiders

MLS does not protect against one group member replaying a PrivateMessage sent by another group member within the same epoch that the message was originally sent. Similarly, MLS does not protect against the replay (by a group member or otherwise) of a PublicMessage within the same epoch that the message was originally sent. Applications for whom replay is an important risk should apply mitigations at the application layer, as discussed below.

In addition to the risks discussed in Section 8.3.1, an attacker with access to the ratchet secrets for an endpoint can replay PrivateMessage objects sent by other members of the group by taking the signed content of the message and re-encrypting it with a new generation of the original sender's ratchet. If the other members of the group interpret a message with a new generation as a fresh message, then this message will appear fresh. (This is possible because the message signature does not cover the `generation` field of the message.) Messages sent as PublicMessage objects similarly lack replay protections. There is no message counter comparable to the `generation` field in PrivateMessage.

Applications can detect replay by including a unique identifier for the message (e.g., a counter) in either the message payload or the `authenticated_data` field, both of which are included in the signatures for PublicMessage and PrivateMessage.

8.7. Cryptographic Analysis of the MLS Protocol

Various academic works have analyzed MLS and the different security guarantees it aims to provide. The security of large parts of the protocol has been analyzed by [BBN19] (for MLS Draft 7), [ACDT21] (for MLS Draft 11), and [AJM20] (for MLS Draft 12).

Individual components of various drafts of the MLS protocol have been analyzed in isolation and with differing adversarial models. For example, [BBR18], [ACDT19], [ACCKKMPPWY19], [AJM20], [ACJM20], [AHKM21], [CGWZ25], and [WPB25] analyze the ratcheting tree sub-protocol of MLS that facilitates key agreement; [WPBB22] analyzes the sub-protocol of MLS for group state agreement and authentication; and [BCK21] analyzes the key derivation paths in the ratchet tree and key schedule. Finally, [CHK21] analyzes the authentication and cross-group healing guarantees provided by MLS.

9. IANA Considerations

This document has no IANA actions.

10. References

10.1. Normative References

[RFC5116]

McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[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/info/rfc9420>.

10.2. Informative References

[ACCKKMPPWY19] Alwen, J., Capretto, M., Cueto, M., Kamath, C., Klein, K., Markov, I., Pascual-Perez, G., Pietrzak, K., Walter, M., and M. Yeo, "Keep the Dirt: Tainted TreeKEM, Adaptively and Actively Secure Continuous Group Key Agreement", Cryptology ePrint Archive, 2019, <https://eprint.iacr.org/2019/1489>.

[ACDT19] Alwen, J., Coretti, S., Dodis, Y., and Y. Tselekounis, "Security Analysis and Improvements for the IETF MLS Standard for Group Messaging", Cryptology ePrint Archive, 2019, <https://eprint.iacr.org/2019/1189.pdf>.

[ACDT21] Alwen, J., Coretti, S., Dodis, Y., and Y. Tselekounis, "Modular Design of Secure Group Messaging Protocols and the Security of MLS", Cryptology ePrint Archive, 2021, <https://eprint.iacr.org/2021/1083.pdf>.

[ACJM20] Alwen, J., Coretti, S., Jost, D., and M. Mularczyk, "Continuous Group Key Agreement with Active Security", Cryptology ePrint Archive, 2020, <https://eprint.iacr.org/2020/752.pdf>.

[AHKM21] Alwen, J., Hartmann, D., Kiltz, E., and M. Mularczyk, "Server-Aided Continuous Group Key Agreement", Cryptology ePrint Archive, 2021, <https://eprint.iacr.org/2021/1456.pdf>.

[AJM20] Alwen, J., Jost, D., and M. Mularczyk, "On The Insider Security of MLS", Cryptology ePrint Archive, 2020, <https://eprint.iacr.org/2020/1327.pdf>.

[BBN19] Bhargavan, K., Beurdouche, B., and P. Naldurg, "Formal Models and Verified Protocols for Group Messaging: Attacks and Proofs for IETF MLS", HAL ID hal-02425229, 2019, <https://inria.hal.science/hal-02425229/document>.

[BBR18] Bhargavan, K., Barnes, R., and E. Rescorla, "TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups - A protocol proposal for Messaging Layer Security (MLS)", HAL ID hal-02425247, 2018, <https://hal.inria.fr/hal-02425247/file/treekem+%281%29.pdf>.

[BCK21] Brzuska, C., Cornelissen, E., and K. Kohbrok, "Security Analysis of the MLS Key Distribution", Cryptology ePrint Archive, 2021, <https://eprint.iacr.org/2021/137.pdf>.

[CAPBR] Brewer, E. A., "Towards robust distributed systems (abstract)", Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing, p. 7, DOI 10.1145/343477.343502, July 2000, <https://dl.acm.org/doi/10.1145/343477.343502>.

[CGWZ25] Cremers, C., Günsay, E., Wesselkamp, V., and M. Zhao, "ETK: External-Operations TreeKEM and the Security of MLS in RFC 9420", 2025, <https://eprint.iacr.org/2025/229.pdf>.

[CHK21] Cremers, C., Hale, B., and K. Kohbrok, "The Complexities of Healing in Secure Group Messaging: Why Cross-Group Effects Matter", Proceedings of the 30th USENIX Security Symposium, August 2021, <https://www.usenix.org/system/files/sec21-cremers.pdf>.

[EXTENSIONS] Robert, R., "The Messaging Layer Security (MLS) Extensions", Work in Progress, Internet-Draft, draft-ietf-mls-extensions-06, 19 February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-mls-extensions-06>.

[FEDERATION] Omara, E. and R. Robert, "The Messaging Layer Security (MLS) Federation", Work in Progress, Internet-Draft, draft-ietf-mls-federation-03, 9 September 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-mls-federation-03>.

[KT] McMillion, B., "Key Transparency Architecture", Work in Progress, Internet-Draft, draft-ietf-keytrans-architecture-03, 25 February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-keytrans-architecture-03>.

[Loopix] Piotrowska, A. M., Hayes, J., Elahi, T., Meiser, S., and G. Danezis, "The Loopix Anonymity System", Proceedings of the 26th USENIX Security Symposium, August 2017.

[MASQUE-PROXY] Schinazi, D., "The MASQUE Proxy", Work in Progress, Internet-Draft, draft-schinazi-masque-proxy-05, 18 February 2025, <https://datatracker.ietf.org/doc/html/draft-schinazi-masque-proxy-05>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/info/rfc3552>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[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>.

[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/info/rfc8446>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[Tor] "The Tor Project", <https://torproject.org/>.

[WPB25] Wallez, T., Protzenko, J., and K. Bhargavan, "TreeKEM: A Modular Machine-Checked Symbolic Security Analysis of Group Key Agreement in Messaging Layer Security", 2025, <https://eprint.iacr.org/2025/410.pdf>.

[WPBB22] Wallez, T., Protzenko, J., Beurdouche, B., and K. Bhargavan, "TreeSync: Authenticated Group Management for Messaging Layer Security", Cryptology ePrint Archive, 2022, <https://eprint.iacr.org/2022/1732.pdf>.

Contributors

Richard Barnes
Cisco
Email: rlb@ipv.sx

Katriel Cohn-Gordon
Meta Platforms
Email: me@katriel.co.uk

Cas Cremers
CISPA Helmholtz Center for Information Security
Email: cremers@cispa.de

Britta Hale
Naval Postgraduate School
Email: britta.hale@nps.edu

Albert Kwon
Badge Inc.
Email: kwonalbert@badgeinc.com

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

Rohan Mahy
Wire
Email: rohan.mahy@wire.com

Brendan McMillion
Email: brendanmcmillion@gmail.com

Thyla van der Merwe
Email: tjvdmerwe@gmail.com

Jon Millican
Meta Platforms
Email: jmillican@meta.com

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

Authors' Addresses

Benjamin Beurdouche
Inria & Mozilla
Email: ietf@beurdouche.com

Eric Rescorla
Email: ekr@rtfm.com

Emad Omara
Email: emad.omara@gmail.com

Srinivas Inguva
Email: singuva@yahoo.com

Alan Duric
Email: alan@duric.net