Messaging Layer Security
Internet-Draft
Intended status: Informational
Expires: 23 April 2026

M. Xue
Germ Network, Inc.
J. W. Lukefahr
B. Hale
US Naval Postgraduate School

20 October 2025

Distributed MLS
draft-xue-distributed-mls-02

Abstract

The Messaging Layer Security (MLS) protocol enables a group of participants to negotiate a common cryptographic state for messaging, providing Forward Secrecy (FS) and Post-Compromise Security (PCS). There are some use cases where message ordering challenges may make it difficult for a group of participants to agree on a common state or use cases where reaching eventual consistency is impractical for the application. This document describes Distributed-MLS (DMLS), a composition protocol for using MLS sessions to protect messages among participants without negotiating a common group state.

About This Document

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

The latest revision of this draft can be found at https://germ-mark.github.io/distributed-mls-id/draft-xue-distributed-mls.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-xue-distributed-mls/.

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

Source for this draft and an issue tracker can be found at https://github.com/germ-mark/distributed-mls-id.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Copyright Notice

Table of Contents

1. Introduction

Participants operating in peer-to-peer or partitioned network topologies may find it impractical to access a centralized Delivery Service (DS), or reach consensus on message sequencing to arrive at a consistent Commit for each MLS epoch.

DMLS is an composition or 'super'-group for facilitating group messaging in such use cases that uses MLS as a 'sub' protocol. It instantiates an MLS session per participant, such that each participant acts as the 'owner' of its own group. These sub-groups also act as 'Send Groups', in which the owner sends all of their application messages and determines the ordering of Proposals and Commits. This allows each participant to locally and independently control the sequence of Update processing and encrypt messages using MLS accordingingly. A distributed group (DGroup) using DMLS then comprises the communication superset of such Send Groups. This draft further addresses how to incorporate randomness from other participants' 'Send Groups' to ensure post-compromise security (PCS) is maintained across the superset.

2. Terminology

Send Group: An MLS session where one designated sender (the group 'owner') authors all messages and other members use the group only to receive from the designated sender.

DMLS Session: A session between a set of communication participants, where each participant can broadcast messages by operating their Send Group.

DGroup: the set of communication participants of a DMLS Session.

DMembers: Members of the DGroup participating in the DMLS session.

3. Protocol Overview

Within a group of distributed DMembers, we can resolve state conflict by assigning each member local state that only they control. In a DMLS Session, we assign to each DMember ownership of an MLS group that they then operate as a Send Group. The Send Group owner can export secrets from other groups owned by DMembers and import such secrets as added randomness into their own Send Group through use of

epochal Proposal messages. The Send Group owner can also Update leaf nodes of other DMembers; if updating the leaf nodes of other DMembers within the owner's Send Group, the owner MUST use the public keys published by the respective DMember within their own Send Group. This enables each Send Group to include entropy and fresh public keys from other receive-only members of their Send Group, providing for both PCS and FS without the need to reach global consensus on ordering of Updates.

3.1. Meeting MLS Delivery Service Requirements

The MLS Architecture Guide specifies two requirements for an abstract Delivery Service related to message ordering. First, Proposal messages should all arrive before the Commit that references them. Second, members of an MLS group must agree on a single MLS Commit message that ends each epoch and begins the next one.

An honest centralized DS, in the form of a message queuing server or content distribution network, can guarantee these requirements to be met. By controlling the order of messages delivered to MLS participants, for example, it can guarantee that Commit messages always follow their associated Proposal messages. By filtering Commit messages based on some pre-determined criteria, it can ensure that only a single Commit message per epoch is delivered to participants.

A decentralized DS, on the other hand, can take the form of a message queuing server without specialized logic for handling MLS messages, a mesh network, or, perhaps, simply a local area network. These DS instantiations cannot offer any such guarantees.

The MLS Architecture Guide highlights the risk of two MLS members generating different Commits in the same epoch and then sending them at the same time. The impact of this risk is inconsistency or forking of MLS group state among members, which in turn risks authorized members being unable to read each other's messages. A decentralized DS offers no mitigation strategy for this risk, so the members themselves must agree on strategies, or in our terminology, operating constraints. We could say in such cases that the full weight of the CAP theorem is therefor levied directly on the MLS members. However, use cases exist that benefit from, or even necessitate, MLS and its accompanying security guarantees for distributed group communications.

The DMLS operating constraints specified above allow honest members to form a distributed system that satisfies these requirements despite a decentralized DS. Moreover, instead of mitigating or providing methods for resolving Commit collisions, it effectively

eliminates any risk of them occuring. It also, consequently removes the risk of insider state desyncronization attacks, as an insider (a DMember) can only control state in their own Send Group. The Send Group methodology ensures that a single owner controls the Send sequence in their own group, including both application messages and Commits. As a potential functional benefit in some use cases, DMLS further enables flexibility in receive-only modes, namely that any DMember can continue to receive messages sent from other groups, even if not sending information themselves.

Downsides of the DMLS design that may make it not suitable for all settings include increased overhead, namely due to the fact that within any Send Group, intermediate nodes along the parent path of any non-owner remain blank. While the owner path Updates when it Commits and other leaf nodes can be Updated as explained later, the parent path of the other leaf nodes is not filled in. Thus DMLS comes with functional trade-offs.

4. Protocol Definition

4.1. Send Group Operation

A DMLS Send Group operates in the following way: * The creator of the group, occupying leaf index 0, is the designated owner of the Send Group * Other members only accept messages from the owner * Members only accept messages as defined in Group Operations

4.1.1. Send Group Mutation

Under this configuration, only the Send Group owner can mutate the group. The owner can Commit to their Send Group to Update their leaf node and/or to incorporate new keys and entropy from other DMembers.

4.1.1.1. (DMLS Update) Broadcast new keys for creator

Alice can provide PCS for herself in her Send Group by authoring a (full or empty) Commit that Updates her own leaf node.

4.1.1.2. (DMLS Commit) Incorporate new key material from others

If Alice has received DMLS Updates from other members, Alice can incorporate them as follows:

If the latest DMLS Update Alice received from Bob in his Send Group is a Commit starting epoch k, and was not already incorporated into Alice’s Send Group, Alice can author a Commit that * imports a PSK from Bob’s Send Group, epoch k with the following parameters * psk_id: k || (Bob’s Send Group’s groupID) where k is a fixed width

8-byte encoding of the epoch in network byte order * psk: MLS-Exporter("exporter-psk", "psk_id", KDF.Nh) * replaces Bob’s leaf node in Alice’s Send Group with Bob’s new leaf note in Commit k

An MLS Commit in a Send Group can convey either a DMLS Update or Commit, or both.

4.1.2. Leaf Node Update

When Bob incorporates PCS from Alice’s Commit into his own Send Group by importing a PSK from Alice’s Send Group, it is also critical that the associated leaf node changes are also Updated in Bob’s Send Group.

Thus, when Bob creates a psk_id as defined above, it directly references a specific groupID and epoch from Alice’s Send Group, which itself corresponds to the current leaf node of Alice for that epoch. When Bob generates a Commit covering the PSK Proposal, Bob MUST also Update Alice’s leaf node within Bob’s own Send Group to match the leaf node of Alice for the given groupID and epoch. When other DMembers receiving messages in Bob’s Send Group receive the Commit, they MUST also Update their tree representations to reflect Alice’s leaf node corresponding to the groupID and epoch.

Alice’s leaf node placement in Bob’s own MLS tree MAY be different than in Alice’ MLS tree, and consequently the Alice’s Update within her own Send Group does not correspond to intermediate node Updates for Alice’s path in Bob’s Send Group. Only the leaf node is Updated.

4.2. DMLS Session Operations

Similar to MLS, DMLS provides a participant application programming interface (API) with the following functions:

4.2.1. INIT

An application constructs a DMLS Session by defining * the DGroup * a scheme for assigning a Send Group identifier for each DMember * allowed cipher suites, and an export key length. and distributing an initial keypackage for each DMember

(Nothing inherently requires the Send Groups to agree on a cipher suite - each sender could choose their own, suitable to their own data transmission protection requirements, as long as they agree on export key length. It is advisable that the application set minimum requirements for all Send Groups within the DGroup.)

The DMLS application SHOULD recommend a policy for issuing DMLS Updates.

Each DMember creates their Send Group by constructing a MLS group * with the assigned Send Group identifier * adding all other Dembers * distributing the resulting welcome message

4.2.2. UPDATE

As defined above under Send Group Mutation

4.2.3. COMMIT

As defined above under Send Group Mutation

4.2.4. PROTECT

A member Bob protects a ciphertext message and encrypts it to the DMembers by encrypting it as an application message in his Send Group, as in MLS. As in MLS, before encrypting an application message, Bob SHOULD incorporate any DMLS Updates of Bob’s own or PSK proposals corresponding to Updates in other DMember Send Groups that he has observed.

4.2.5. UNPROTECT

On receipt of an MLS message, a member can look up the corresponding Send Group by the MLS groupID in the message metadata and decrypt it with the keys associated with that Send Group.

5. Characteristics

Under DMLS, members can successfully encrypt messages at any time without waiting for in-flight handshake messages from other members. A DMLS Commit by Alice acknowledges to everyone else the newest DMLS Update Alice has received from each member. Alice can delete her k_th leaf node private key when all members have committed a newer leafNode from her.

An offline member that has not been issuing DMLS Commits may prevent Alice from deleting old private keys. As in MLS, applications SHOULD handle members that are offline for excessive periods by dropping them from the Send Group. For example, if Bob has been offline past an application’s threshold and not acknowleged Alice’s kth Update, Alice may choose to delete her k_th key anyway, allowing for the chance that she may stop receiving messages from Bob (if Bob later Commits Alice’s kth Update).

Alice can signal this in her next DMLS Update or Commit by removing Bob from her Send Group. This allows each member of the universe to independently excise offline members, and signal to everyone (including the removed member) that they are doing so. On receipt of Alice's removal of Bob, another DMember Charlie MUST also remove Bob from his Send Group, as Bob will not be able to process any future DMLS Commit that incorporates newer group state from Alice.

Reintroducing offline members is outside the scope of this draft, and could be done by initiating a new DGroup.

5.1. Tolerance to dropped messages

Analogously to MLS, where members must receive every Commit message and apply them in order to be able to compute the group's most recent state, withing a given Send Group, each DMember must receive every Commit from the Send Group owner. Recipients must apply Commits from each Send Group in order provided by the Send Group owner, aided by MLS message metadata.

The injection of PSK's across groups introduces an additional desirable Commit ordering consideration, although injection order is within the control of the Send Group owner. The format of the PSK ID helps DMembers order the application of Commits across Send Groups to succesfully import PSK's: * Alice issues a DMLS Update in the Commit starting epoch k of her Send Group. * Bob receives Alice's kth DMLS Update, and incorporates it in the j_th Commit of his Send Group * Charlie, on receipt of Bob's j_th Commit, can process it and understand it depends on a psk_id that he can parse as k_th Commit from Alice.

The dependency order of Commits forms a directed (acyclic) graph among pairs of (epoch, groupId) in a DMLS Session.

6. Wire Formats

DMLS uses standard wire formats as defined in [RFC9420]. An application using DMLS SHOULD define formats for any additional messages containing common configuration or operational parameters.

7. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

8. Security Considerations

DMLS inherits and matches MLS in most security considerations with one notable change to security control nuances. In MLS each group member can largely control when their Update will be introduced to the group state, with deconfliction only down to the DS. In contrast, in DMLS the Send Group owner controls when key Update material is included from each member; namely, every member Updates in their own Send Group and fresh keying material is then imported to other Send Groups through use of the exporter key and PSK Proposal option, with timing controlled by the respective Send Group owners. This means that while the PCS healing frequency of a given member in MLS is under their own control, in DMLS the PCS healing frequency and timeliness of PSK import is controlled by the Send Group owner. However, the Send Group owner is also the only member sending data in the Send Group. This means that there is a natural incentive to Update frequently and in a timely manner, and that PCS impacts on sending of data are not delayed from the point of original Update.

FS guarantees per Send Group follow similarly; the Send Group owner determines the frequency of key roll-over.

Notably, since the Send Group owner determines what is introduced as a PSK from other DMembers, it is not possible for an insider threat to disrupt the group state and cause desynchronization of the group view. This is unlike in MLS, where all contributing parties must behave honestly to avoid state disruption.

As in MLS, it is essential for PCS security that all members Update frequently. In MLS, if a member Bob does not receive an Update from another member, Alice, Bob’s state will become desynchronized from the rest of the MLS group, leading Bob to be unable to send messages that other group members (which have correctly processed Alice’s Updates) will be able to decrypt and also preventing Bob from decrypting messages sent by those members. Bob must obtain the missing Alice Update from the DS. In DMLS, however, the removal of desynchronization risk means that Bob will continue to be able to send messages to DMembers in Bob’s Send Group even if Bob has not yet observed Alice’s Update. This presents both a benefit and a risk, as a denial-of-service attacker that has compromised Alice’s state could prevent Bob from receiving Alice’s PCS Update, meaning that Bob continues to send messages accessible under Alice’s compromised state even after Alice has Updated. To prevent this, the application SHOULD specify minimal Update frequency and Send Group owners SHOULD remove members from which no Update has been observed for an excessive period.

9. IANA Considerations

This document has no IANA actions.

10. References

CAPBR: # Brewer, E., "Towards robust distributed systems (abstract)", ACM, Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing, DOI 10.1145/343477.343502, July 2000, https://doi.org/10.1145/343477.343502 (https://doi.org/10.1145/343477.343502).

11. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC9420] Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., and K. Cohn-Gordon, "The Messaging Layer Security (MLS) Protocol", RFC 9420, DOI 10.17487/RFC9420, July 2023, <https://www.rfc-editor.org/rfc/rfc9420>.

Acknowledgments

TODO acknowledge.

Authors' Addresses

Mark Xue
Germ Network, Inc.
Email: mark@germ.network

Joseph W. Lukefahr
US Naval Postgraduate School
Email: joseph.lukefahr@nps.edu

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