Messaging Layer Security
Internet-Draft
Intended status: Informational
Expires: 13 March 2026

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

9 September 2025

Distributed MLS
draft-xue-distributed-mls-01

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). Still, 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 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.

This Internet-Draft will expire on 13 March 2026.

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 MLS adaptation for facilitating group messaging in such use cases by instantiating an MLS group per participant, such that each participant has a dedicated 'send' group within a communication superset of such groups. This allows each participant to locally and independently control the sequence of update processing and encrypt messages using MLS accordingingly. This draft further addresses how to incorporate randomness from other participant's 'send' groups to ensure post-compromise security (PCS) is maintained.

1.1. Terminology

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

Universe: A superset of MLS participants comprised of the owners of all Send Groups.

1.2. Protocol Overview

Within a universe U of distributed participants, we can resolve state conflict by assigning each member local state that only they control. In DMLS, we assign each member an MLS group to operate as a Send Group. The Send Group owner can export secrets from other groups owned by the Universe and import the epoch randomness through use of Proposal messages into their own Send Group. This enables each Send Group to include entropy 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.

1.3. 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, prehaps, 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 of MLS group state among members. This perhaps leads to inability of some authorized members to read other authorized members' messages, i.e., a loss of availability of the message-passing service provided by MLS. 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 that the full weight of the CAP theorem is thus levied directly on the MLS members in this case. However, use cases exist that benefit from, or even necessitate, MLS and its accompanying security guarantees for group message passing.

The DMLS operating constraints specified above allow honest members to form a distributed system that satisfies these requirements despite a decentralized DS.

2. Send Group Operation

An MLS Send Group operates in the following constrained 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 * Each group owner updates their contribution to the group with a full or empty commit. To incorporate fresh keying material inputs from another member, the group owner creates an exporter key from the other member's Send Group and imports that as a PSK Proposal.

To facilitate binding Send Groups together, we define the following exported values: * derived groupid: MLS-Exporter("derivedGroupId", leafNodePublicSigningKey, Length)

This is a unique value for each participant derived from the group's current epoch * exportPSK: 'MLS-Exporter("exporter-psk", "psk_id", KDF.Nh)'

3. Group Operations

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

3.1. INIT

Given a list of DMLS participants, initialize an DMLS context by (1) creating an MLS group, (2) adding all other participants (generating a set of Welcome messages and a GroupInfo message). It is the responsibility of a DMLS implementation to define the Universe of participants and the mechanism of generating the individual send groups. Two possible approaches are described below.

3.1.1. Over the wire definition

For example, U can be defined over the wire by inferring it from a newly created send group.

Assume Alice has keypackages for some other members M_i

Alice can construct a DMLS group * with a randomly generated groupId * constructing a commit adding all other members M_i

Alice can distribute the Welcome message with an Application Message that indicates * this is a Send Group for Alice * that defines a Universe U as the members of this group * with universe identifier equal to the groupId for Alice’s send group * and defines a common export key length

3.1.2. Application-directed definition

U can also be defined by the application layer, which provides each member: * keypackages for all other members in U * a random universe identifier for the DMLS group * common export key length

Keypackages can be reusable, e.g., marked as last-resort. Keypackages can also be single-use if the application layer retrieves $N-1$ (where $|U|=N$) unique keypackages from each member.

Alice can construct her view of a DMLS group: * by creating an MLS group with randomly generated groupId * and then constructing a Commit and Welcome message adding all other members in U

With this approach, an additional message is not required as common configuration items are provided by the application layer.

3.2. UPDATE

A member Alice of U can introduce new key material to the universe U by authoring a full or empty commit in Alice’s send group, which provides PCS with regard to the committer.

3.3. COMMIT

When Bob receives Alice’s DMLS update (as a full or empty commit in Alice’s send group), Bob can incorporate PCS from Alice’s commit by importing a PSK from Alice’s send group. Precisely, Bob: * Creates a PSK proposal in Bob’s send group using the exportPskId and exportPSK from the epoch of Alice’s send group after Alice’s DMLS update * Bob generates a commit covering the PSK proposal (for each send group in which he has observed a new DMLS update).

The psk_group_id for this PSK is more specifically defined as follows: psk_group_id = (opaque<8>) groupEpoch | groupId where epoch_bytes is the byte-vector representation of the epoch in which the exporter was generated, in network byte order. Since epoch is of type uint64, this results in a fixed 8-byte vector. groupId, which is of type opaque<V>, is then appended to epoch_bytes. When a exportPskId is received as part of an incoming PSK proposal, it can then be processed as follows: groupId = exportPskId[8..] epoch = (uint64) exportPskId[0..7]

Per [RFC9420], the psk_nonce must be a fresh random value of length KDF.Nh when the PSK proposal is generated. This ensures key separation between a PSK generated by, for example, (1) a PSK generated by Bob from Alice’s group and (2) a PSK generated by Charlie from Alice’s group.

3.4. PROTECT

A member Bob protects a ciphertext message and encrypting it to U by encrypting it as an application message in their send group. As in MLS, before encrypting an application message, Bob should incorporate any DMLS updates he has received.

Each of the 3 MLS configurations of commit are possible: * If Bob has observed no updates but wishes to issue an update, they can author an empty commit. If bob has observed DMLS updates, * Bob can incorporate those updates without a DMLS of his own with a partial commit covering PSK proposals from each updated send group. * Alternatively, Bob can incorporate his own new keys by covering those PSK proposals with a full commit.

3.5. 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 attempt to decrypt it with that send group.

4. DMLS Requirements

The application layer over MLS has the responsibility to define * The Universe U of members of this DMLS group * Mapping groupIds to members of U * Additional common rules, such as accepted cipher suites

(nothing inherently requires the send groups to agree on a cipher suite - each sender could choose their own as long as they agree on export key length)

The DMLS layer should recommend a policy for issuing DMLS updates.

5. 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.

6. 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.

7. Security Considerations

DMLS inherits and matches MLS in most security considerations with one notable change to PCS nuances. In MLS each group member can largely control when their updates 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.

8. IANA Considerations

This document has no IANA actions.

9. 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).

10. 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