Messaging Layer Security
Internet-Draft
Intended status: Informational
Expires: 18 September 2025

K. Kohbrok
Phoenix R&D

17 March 2025

Decentralized Messaging Layer Security
draft-kohbrok-mls-decentralized-mls-00

Abstract

Messaging Layer Security provides strong end-to-end security guarantees for group messaging including Forward Secrecy (FS) and Post-Compromise Security (PCS). However, MLS requires a Delivery Service (DS) to facilitate agreement between group members on the order of Commit messages. In decentralized settings the only way to implement a functional DS is to require group members to retain key material so they can process commits out-of-order. Retaining key material this way is in violation of the MLS deletion schedule and significantly reduces the FS of the protocol. This draft specifies Decentralized MLS, based on the the Fork-Resilient Continuous Group Key Agreement protocol FREEK proposed by Alwen et al. [FRCGKA]. In essence, DMLS extends MLS such that key material can be retained to process Commits out-of-order with minimal losses to FS, thus allowing safer deployment in decentralized environments.

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://phnx-im.github.io/dmls-spec/draft-kohbrok-mls-decentralized-mls.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-kohbrok-mls-decentralized-mls/.

Discussion of this document takes place on the Messaging Layer Security 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/phnx-im/dmls-spec.

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

TODO: Introduction

Open Questions: - Why do removed members need to receive their commit confirmation from their removers? They should be able to process the removing commit without the init secret in the first place. - Is it safe to use the commit secret regularly in the key schedule _and_ to derive the commit confirmation or do we need an additional derivation step to ensure domain separation? - Why would we need to generate a fresh signature key pair with each update? - Do we need an additional membership tag?

2. Epoch identifiers

In MLS, each epoch is identified by a 64 bit unsigned integer, with the epoch increasing by one with each commit. The integer identifies epochs uniquely as long as there is only one chain of Commits. However, in a decentralized context there can be conflicting commits. For example, if two group member send a commit at the same time with different subsets of group members receiving a different commit first. After processing the newly arrived Commit, all group members would be in the same epoch, but in different group states. For subsequently arriving messages, it is unclear from the integer designating the epoch, which state the message belongs to. In such scenarios it is important that epochs are uniquely identifiable.

When using DMLS, epochs are represented as byte strings of length KDF.Nh (thus depending on the group’s ciphersuite). The byte string identifying an epoch is derived from the epoch’s epoch_secret and can thus be learned when creating or processing the commit that initiates that epoch.

pseudocode epoch = DeriveSecret(epoch_secret, "epoch")

3. DMLS key schedule

DMLS conceptually modifies the MLS key schedule by enabling the creation of multiple init_secrets, where each init secret can be used to initialize a subsequent epoch.

The individual init_secrets are derived through a puncturable pseudorandom function (PPRF, Section 4) keyed by the base_init_secret.

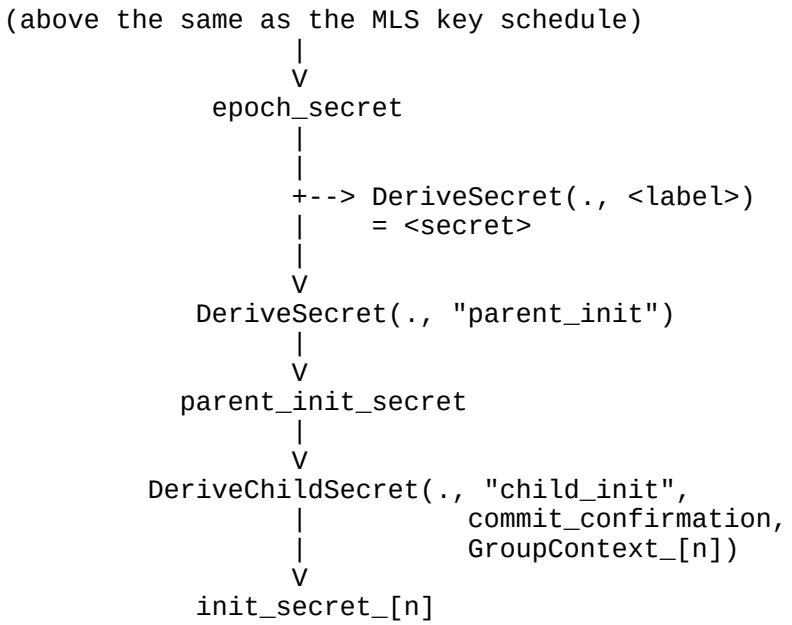

Figure 1: The DMLS Key Schedule

```
commit_confirmation = DeriveSecret(path_secret[n], "conf")

DeriveChildSecret(prf_key, label, input_secret, context) =
  DeriveFSSecret(prf_key, ExpandWithLabel(input_secret, label, context, KDF.Nh))
```

4. Puncturable pseudorandom function

A PPRF allows the derivation of keys in a forward secure way. In particular, a PRF that was evaluated with a given key and input can't be evaluated with those same parameters again. Storing the original input key thus doesn't harm the forward secrecy of (deleted) output keys.

The MLS Secret Tree as defined in [RFC9420] already represents a PPRF an needs to be modified only slightly for the purpose of this document.

In the context of MLS, the Secret Tree has as many leaves as the group has members. To derive child init secrets, the same tree is used but with KDF.Nh leaves.

The function DeriveFSSecret(secret, input) thus follows these steps:

* Check if secret and input are of length KDF.Nh

* With secret as the root node secret and input as the leaf index, derive the direct path nodes and the copath nodes as defined in Section 9 of [RFC9420]

* With leaf_node_secret as the resulting secret compute the final output using DeriveSecret(leaf_node_secret, "pprf")

5. State consolidation

The changes to MLS outlined above allow the processing of multiple commits for each epoch with improved forward secrecy. However, once a fork was created, they do not help in returning group members to a single, agreed-upon group state.

There is no one semantic to consolidate forks that is optimal for all given applications. Even more so if the application uses extensions that store additional data in the group's context.

This section thus details a simple state-consolitation procedure that focuses on consolidating group membership and key material only. To keep it simple, our approach makes a few assumptions.

TODO: The following is work in progress. While the assumptions won't disappear entirely, they will become weaker in future iterations.

* One group forked at epoch e_i

* Only one partition, half of the group on one side, the other on the other side

* Both sides of the partition have a server that helps them agree on commit ordering (imagine a netsplit in a federated approach with only two servers involved)

* The two servers involved coordinate in allowing commits/preventing forks

* Neither clients nor servers keep around old messages, but do keep around old group states s.t. they can process old messages

* No insider attacks, i.e. no insider is trying to create a fork and then make the others accept the new state

* No access control: All parties can add and remove members as they please

* KeyPackages are accessible: All parties can access KeyPackages of all other parties

Open Questions: - How to determine which fork wins, i.e. which fork is merged into which? Options: - Arbitrary: The fork with the alphanumerically highest transcript hash wins - Is it mandatory that the consolitation algorithm is deterministic s.t. everyone can compute it and arrive at the same result?

Timeline: - Alice, Alan, Bob and Betty are in a group - Alice and Alan on server A, Bob and Betty on server B - There is a disconnect between servers A and B - All group members keep sending messages - Alice adds Albert to the group - Betty removes Bob from the group - The servers reconnect - All parties receive and process all messages from the other side - All parties now have two forks of the same group - One party (say Alice) starts the consolitation procedure (see below)

5.1. Consolidation algorithm

* Given two forks, choose the one with the alphanumerically larger transcript hash

* On the losing fork, create a commit with a proposal that indicates that this fork is closed

* On the winning fork, create a commit that consolidates the group membership of both forks:

  - If a member was removed in the losing fork but not on the winning fork, remove it in the commit

  - If a member was added in the losing fork, but not on the winning fork, add it in the commit

Open Questions: - What if a member was removed in both forks and then re-added in the winning fork? Should it still be removed in the consolidating commit? Resolving this is tricky if not everyone has the message traces for both forks. - What if there is no overlap in members? In that case, there is no way to consolidate, because no one can create a commit in the other fork. - Maybe that’s okay? In that case, there’s not much to consolidate. Should the losing fork be closed? Or should both forks live on? Might depend on the application.

6. Security Considerations

TODO Security

* Note that while the PPRF gives you FS for init secrets, you also need to keep the secret state of old RatchetTree states around, thus damaging their FS

7. IANA Considerations

This document has no IANA actions.

8. References

8.1. Normative References

[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>.

8.2. Informative References

[FRCGKA] Alwen, J., Mularczyk, M., and Y. Tselekounis, "Fork-Resilient Continuous Group Key Agreement", 22 February 2024, <https://eprint.iacr.org/2023/394.pdf>.

Acknowledgments

TODO acknowledge.

Author’s Address

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