Network Working Group
Internet-Draft
Intended status: Informational
Expires: 23 April 2026

J. Alwen
K. Kohbrok
R. Robert
Phoenix R&D

20 October 2025

MLS Virtual Clients
draft-kohbrok-mls-virtual-clients-04

Abstract

This document describes a method that allows multiple MLS clients to emulate a virtual MLS client. A virtual client allows multiple emulator clients to jointly participate in an MLS group under a single leaf. Depending on the design of the application, virtual clients can help hide metadata and improve performance.

Table of Contents

1. Introduction 2
2. Terminology 3
3. Applications 3
3.1. Virtual clients for performance 4
3.2. Metadata hiding 4
4. Limitations 5
4.1. External remove proposals 5
4.2. External joins 5
5. Client emulation 6
5.1. Delivery Service 6
5.2. Generating Virtual Client Secrets 6
5.3. Creating LeafNodes and UpdatePaths 8
5.4. Adding emulator clients 9
5.5. Virtual client actions 9
5.5.1. Creating and uploading KeyPackages 10
5.5.2. Externally joining groups with the virtual client . . 11
5.6. Sending application messages 11
6. Security considerations 11
7. Privacy considerations 11
8. Performance considerations 12
8.1. Smaller Trees 12
8.2. Fewer blanks 12
9. Emulation costs 12
10. Normative References 12
Authors' Addresses 13

1. Introduction

The MLS protocol facilitates communication between clients, where in an MLS group, each client is represented by the leaf to which it holds the private key material. In this document, we propose the notion of a virtual client that is jointly emulated by a group of emulator clients, where each emulator client holds the key material necessary to act as the virtual client.

The use of a virtual client allows multiple distinct clients to be represented by a single leaf in an MLS group. This pattern of shared group membership provides a new way for applications to structure groups, can improve performance and help hide group metadata. The effect of the use of virtual clients depends largely on how it is applied (see Section 3).

We discuss technical challenges and propose a concrete scheme that allows a group of clients to emulate a virtual client that can participate in one or more MLS groups.

2. Terminology

* Virtual Client: A client for which the secret key material is held by one or more emulator clients, each of which can act on behalf of the virtual client.

* Emulator Client: A client that collaborates with other emulator clients in emulating a virtual client.

* Emulation group: Group used by emulator clients to coordinate emulation of a virtual client.

* Higher-level group: A group that is not an emulation group and that may contain one or more virtual clients.

* Simple multi-client: A simple alternative to the concept of virtual clients, where entities that are represented by more than one client (e.g. a user with multiple devices) are implemented by including all of the entities' clients in all groups the entity is participating in.

TODO: Terminology is up for debate. We've sometimes called this "user trees", but since there are other use cases, we should choose a more neutral name. For now, it's virtual client emulation.

3. Applications

Virtual clients generally allow multiple emulator clients to share membership in an MLS group, where the virtual client is represented as a single leaf. This is in contrast to the simple multi-client scenario as defined above.

Depending on the application, the use of virtual clients can have different effects. However, in all cases, virtual client emulation introduces a small amount of overhead for the emulator clients and certain limitations (see Section 4).

3.1. Virtual clients for performance

If a group of emulator clients emulate a virtual client in more than one group, the overhead caused by the emulation process can be outweighed by two performance benefits.

On the one hand, the use of virtual clients makes the higher-level groups (in which the virtual client is a member) smaller. Instead of one leaf for each emulator client, it only has a single leaf for the virtual client. As the complexity of most MLS operations depends on the number of group members, this increases performance for all members of that group.

At the same time, the virtual client emulation process (see Section 5) allows emulator clients to carry the benefit of a single operation in the emulation group to all virtual clients emulated in that group.

3.2. Metadata hiding

Virtual clients can be used to hide the emulator clients from other members of higher-level groups. For example, removing group members of the emulator group will only be visible in the higher-level group as a regular group update. Similarly, when an emulator client wants to send a message in a higher-level group, recipients will see the virtual client as the sender and won’t be able to discern which emulator client sent the message, or indeed the fact that the sender is a virtual client at all.

Hiding emulator clients behind their virtual client(s) can, for example, hide the number of devices a human user has, or which device the user is sending messages from.

As hiding of emulator clients by design obfuscates the membership in higher-level groups, it also means that other higher-level group members can’t identify the actual senders and recipients of messages. From the point of view of other group members, the "end" of the end-to-end encryption and authentication provided by MLS ends with the virtual client. The relevance of this fact largely depends on the security goals of the application and the design of the authentication service.

If the virtual client is used to hide the emulator clients, the delivery service and other higher-level group members also lose the ability to enforce policies to evict stale clients. For example, an emulator client could become stale (i.e. inactive), while another keeps sending updates. From the point of view of the higher-level group, the virtual client would remain active.

4. Limitations

The use of virtual clients comes with a few limitations when compared to MLS, where all emulator clients are themselves members of the higher-level groups.

4.1. External remove proposals

In some cases, it is desirable for an external sender (e.g. the messaging provider of a user) to be able to propose the removal of an individual (non-virtual) client from a group without requiring another client of the same user to be online. Doing so would allow another client to commit to said remove proposal and thus remove the client in question from the group.

This is not possible when using virtual clients. Here, the non-virtual client would be the emulator client of a virtual client in a higher-level group. While the server could propose the removal of the client from the emulation group, this would not effectively remove the client’s access to the higher-level groups through the virtual client.

For such a removal to take place, another emulator client would have to be online to update the key material of the virtual client (in addition to the removal in the emulation group).

Another possibility would be for emulator clients to provision KeyPackages for which only a subset of emulator clients have access to. The external sender could then propose the removal of the virtual client, coupled with the immediate addition of a new one using one of the KeyPackages.

4.2. External joins

In a simple multi-client scenario, new (emulator) clients are able to join via external commit without influencing the operation of any other emulator client and without requiring another emulator client to be online.

When using virtual clients and a client wishes to externally join the emulator group, it will not have immediate access to the secrets of the virtual clients associated with that group.

This can be remedied via one of the following options:

* Another emulator client could provide it with the necessary secrets

* The new emulator client could have the virtual client rejoin all higher-level groups

While the first option has the benefit of not requiring an external commit in any higher-level groups (thus reducing overhead), it either requires another emulator client to be online to share the necessary secrets directly, or a way for the new emulator client to retrieve the necessary without the help of another client. The latter can be achieved, for example, by encrypting the relevant secrets such that the new client can retrieve and decrypt them.

The second option on the other hand additionally requires the new emulator client to re-upload all KeyPackages of the virtual client, thus further increasing the difficulty of coordinating actions between emulation group and higher-level groups.

5. Client emulation

To ensure that all emulator clients can act through the virtual client, they have to coordinate some of its actions.

5.1. Delivery Service

Client emulation requires that any message sent by an emulator client on behalf of a virtual client be delivered not just to the rest of the supergroup to which the the message is sent, but also to all other clients in the emulator group.

5.2. Generating Virtual Client Secrets

Generally, secrets for virtual client operations are derived from the emulation group. To that end, emulator clients derive an epoch_base_secret with every new epoch of that group.

```
emulator_epoch_secret = SafeExportSecret(XXX)
```

TODO: Replace XXX with the component ID.

The emulator_epoch_secret is in turn used to derive two further secrets, after which it is deleted.

```
epoch_id =
  DeriveSecret(emulator_epoch_secret, "Epoch ID")
epoch_base_secret =
  DeriveSecret(emulator_epoch_secret, "Base Secret")
epoch_encryption_key =
  DeriveSecret(emulator_epoch_secret, "Encryption Key")
```

The epoch_base_secret is then used to key an instance of the PPRF defined in [I-D.ietf-mls-extensions] using a tree with 2^32 leaves.

Secrets are derived from the PPRF as follows:

VirtualClientSecret(Input) = tree_node_[LeafNode(Input)]_secret

Emulator client MUST store both the (punctured) epoch_base_secret and the epoch_id until no key material derived from it is actively used anymore. This is required for the addition of new clients to the emulator group as described in Section Section 5.4.

When deriving a secret for a virtual client, e.g. for use in a KeyPackage or LeafNode update, the deriving client samples a random octet string random and hashes it with its leaf index in the emulation group using the hash function of the emulation group’s ciphersuite.

```
struct {
  u32 leaf_index;
  opaque random<V>;
} HashInput
```

pprf_input = Hash(HashInput)

TODO: We could also hash in the specific operation to further separate domains.

The pprf_input is then used to derive an operation_secret.

operation_secret = VirtualClientSecret(pprf_input)

Given an epoch_id, random and the leaf_index of the emulator client performing the virtual client operation, other emulator clients can derive the operation_secret and use it to perform the same operation.

Depending on the operation, the acting emulator client will have to derive one or more secrets from the operation_secret.

There are four types of MLS-related secrets that can be derived from an operation_secret.

* signature_key_secret: Used to derive the signature key in a virtual client’s leaf

* init_key_secret: Used to derive the init_key HPKE key in a KeyPackage

* encryption_key_secret: Used to derive the encryption_key HPKE key in the LeafNode of a virtual client

* path_generation_secret: Used to generate path_secrets for the UpdatePath of a virtual client

```
signature_key_secret =
  DeriveSecret(epoch_base_secret, "Signature Key")

encryption_key_secret =
  DeriveSecret(epoch_base_secret, "Encryption Key")

init_key_secret =
  DeriveSecret(epoch_base_secret, "Init Key")

path_generation_secret =
  DeriveSecret(epoch_base_secret, "Path Generation")
```

From these secrets, the deriving client can generate the corresponding keypair by using the secret as the randomness required in the key generation process.

5.3. Creating LeafNodes and UpdatePaths

When creating a LeafNode, either for a Commit with path, an Update proposal or a KeyPackage, the creating emulator client MUST derive the necessary secrets from the current epoch of the emulator group as described in Section Section 5.2.

Similarly, if an emulator client generates an Commit with an update path, it MUST use path_generation_secret as the path_secret for the first parent_node instead of generating it randomly.

To signal to other emulator clients which epoch to use to derive the necessary secrets to recreate the key material, the emulator client includes an DerivationInfoExtension in the LeafNode.

```
struct {
  opaque epoch_id<V>;
  opaque ciphertext<V>;
} DerivationInfoExtension

struct {
  uint32 leaf_index;
  opaque random<V>;
} EpochInfoTBE
```

The ciphertext is the serialized EpochInfoTBE encrypted under the epoch’s epoch_encryption_key with the epoch_id as AAD using the AEAD scheme of the emulation group’s ciphersuite.

When other emulator clients receive an Update (i.e. either an Update proposal or a Commit with an UpdatePath) in group that the virtual client is a member in it uses the epoch_id to determine the epoch of the emulator group from which to derive the secrets necessary to re-create the key material of the LeafNode and a potential UpdatePath.

5.4. Adding emulator clients

There are two ways of adding new clients to the emulation group. Either new clients get sent the secret key material of all groups that the virtual client is currently in, or it joins into all of the virtual client’s groups, either via a regular or an external commit.

TODO: Specify protocol

5.5. Virtual client actions

There are two occasions where emulator clients need to communicate directly to operate the virtual client. In both cases, the acting emulator client sends a Commit to the emulation group before taking an action with the virtual client.

The commit serves two purposes: First, the agreement on message ordering facilitated by the DS prevents concurrent conflicting actions by two or more emulator clients. Second, the acting emulator client can attach additional information to the commit using the SafeAAD mechanism described in Section 4.9 of [I-D.ietf-mls-extensions].

```
enum {
  reserved(0),
  key_package_upload(1),
  external_join(2),
  255,
} ActionType;

struct {
  ActionType action_type;
  select (VirtualClientAction.action_type) {
    case key_package_upload:
      KeyPackageUpload key_package_upload;
    case external_join:
      ExternalJoin external_join;
  };
} VirtualClientAction;
```

5.5.1. Creating and uploading KeyPackages

When creating a KeyPackage, the creating emulator client derives the init_secret as described in Section 5.2.

Before uploading one or more KeyPackages for a virtual client, the uploading emulator client MUST create a KeyPackageUpload message and send it to the emulator group as described in Section 5.5.

The recipients can use the leaf_index of the sender, as well as the random and epoch_id to derive the init_key for each KeyPackageRef. If the recipients receive a Welcome, they can then check which init_key to use based on the KeyPackageRef.

```
struct {
  KeyPackageRef key_package_ref<V>;
  opaque random<V>;
} KeyPackageInfo

struct {
  opaque epoch_id<V>;
  KeyPackageInfo key_package_info<V>;
} KeyPackageUpload
```

After successfully sending the message, the sender MUST then upload the corresponding KeyPackages.

The key_package_refs allow emulator clients to identify which KeyPackage to use and how to derive it when the virtual client receives a Welcome message.

5.5.2. Externally joining groups with the virtual client

Before an emulator client uses an external commit to join a group with the virtual client, it MUST send an ExternalJoin message to the emulation group as described in Section 5.5.

```
struct {
  opaque group_id<V>;
} ExternalJoin
```

The sender MUST then use an external join to join the group with GroupID group_id. When creating the commit to join the group externally, it MUST generate the LeafNode and path as described in Section 5.3.

5.6. Sending application messages

There are two issues when emulator clients send application messages through a virtual client: Nonce-reuse and key-reuse. Both issues ares solved as specified in Section 4.4 of [I-D.draft-mcmillion-mls-subgroups].

6. Security considerations

TODO: Detail security considerations once the protocol has evolved a little more. Starting points:

Some of the performance benefits of this scheme depend on the fact that one can update once in the emulation group and “re-use” the new randomness for updates in multiple higher-level groups. At that point, clients only really recover when they update the emulation group, i.e. re-using somewhat old randomness of the emulation group won’t provide real PCS in higher-level groups.

7. Privacy considerations

TODO: Specify the metadata hiding properties of the protocol. The details depend on how we solve some of the problems described throughout this document. However, using a virtual client should mask add/remove activity in the underlying emulation group. If it actually hides the identity of the members may depend on the details of the AS, as well as how we solve the application messages problem.

8. Performance considerations

There are several use cases, where a specific group of clients represents a higher-level entity such as a user, or a part of an organization. If that group of clients shares membership in a large number of groups, where its sole purpose is to represent the higher-level entity, then instead emulating a virtual client can yield a number of performance benefits, especially if this strategy is employed across an implementation. Generally, the more emulator clients are hidden behind a single virtual client and the more clients are replaced by virtual clients, the higher the potential performance benefits.

8.1. Smaller Trees

As a general rule, groups where one or more sets of clients are replaced by virtual clients have fewer members, which leads to cheaper MLS operations where the cost depends on the group size, e.g., commits with a path, the download size of the group state for new members, etc. This increase in performance can offset performance penalties, for example, when using a PQ-secure cipher suite, or if the application requires high update frequencies (deniability).

8.2. Fewer blanks

Blanks are typically created in the process of client removals. With virtual clients, the removal of an emulator client will not cause the leaf of the virtual client (or indeed any node in the virtual client' s direct path) to be blanked, except if it is the last remaining emulator client. As a result, fluctuation in emulator clients does not necessarily lead to blanks in the group of the corresponding virtual clients, resulting in fewer overall blanks and better performance for all group members.

9. Emulation costs

From a performance standpoint, using virtual clients only makes sense if the performance benefits from smaller trees and fewer blanks outweigh the performance overhead incurred by emulating the virtual client in the first place.

10. Normative References

[I-D.draft-mcmillion-mls-subgroups]
McMillion, B., "MLS Subgroups", Work in Progress, Internet-Draft, draft-mcmillion-mls-subgroups-00, 19 March 2024, <https://datatracker.ietf.org/doc/html/draft-mcmillion-mls-subgroups-00>.

[I-D.ietf-mls-extensions]
Robert, R., "The Messaging Layer Security (MLS) Extensions", Work in Progress, Internet-Draft, draft-ietf-mls-extensions-08, 21 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-mls-extensions-08>.

Authors' Addresses

J. Alwen
Email: alwenjo@amazon.com

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

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