MOQ
Internet-Draft
Intended status: Informational
Expires: 1 January 2026

C. Jennings
S. Nandakumar
R. L. Barnes
Cisco

30 June 2025

End-to-end Security for Media over QUIC
draft-jennings-moq-e2ee-mls-03

Abstract

The Media over QUIC system allows relays to assist in the delivery of real-time media. While these relays are trusted to facilitate media delivery, they are not trusted to access the media content. The document describes an end-to-end security system that prevents relays from accessing media content. MLS is used to establish keys that are available only to legitimate participants in a session, which are then used to protect media data using SFrame.

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://suhashere.github.io/moq-e2ee-mls. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-jennings-moq-e2ee-mls/.

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

Source for this draft and an issue tracker can be found at https://github.com/suhasHere/moq-e2ee-mls.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Copyright Notice

Table of Contents

1. Introduction

The Media over QUIC (MOQ) system allows relays to assist in the delivery of real-time media. While these relays are trusted to facilitate media delivery, they are not trusted to access the media content. This distinction is especially important in the more flexible relay topology that MOQ envisions, where a client might enlist a local relay to assist in media distribution, having no prior

relationship with this relay.

The document describes an end-to-end security system that prevents relays from accessing media content. MOQ tracks are associated to "security groups" so that content in the track can only be accessed by clients that are part of the security group. Security groups also allow clients to authenticate one anothers' identities, so that any client can verify that there are no unauthorized parties in the security group.

To create these security groups, we rely on two widely deployed security technologies: MLS for group key exchange and SFrame for lightweight media encryption [MLS] [SFrame]. The security group maintains an MLS group that establishes the identities of group members and sets up shared secrets. The MLS messages required to keep MLS state up to date as clients join and leave are primarily distributed over MOQ. The only additional infrastructure required is a coordination service that can be provided either by a decentralized algorithm or a lightweight HTTPS service. Media is protected from relays simply by adding a layer of SFrame encryption, using keys derived from the MLS group.

2. Terminology

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.

This document makes extensive use of terminology from MOQT, MLS, and SFrame [MOQT] [MLS] [SFrame]. The few data structures we define use the TLS presentation syntax, defined in Section 3 of [RFC8446], with the optional extension defined in Section 2.1.1 of [MLS].

We introduce the following terms:

Security group: A logical grouping of clients that provides cryptographic security services to MOQ tracks.

Coordination service: A logical service that coordinates the operations required to maintain a security group.

3. Protocol Overview

In the Media over QUIC Transport (MOQT), media streams are arranged in tracks, each of which carries many individual media objects. This document adds a notion of "security groups", each of which is used to protect one or more MOQT tracks. (Each track has at most one security group.)

3.1. Security Groups

Logically, a security group is a set of clients with the following properties:

* Each client can authenticate the identity of every other client.

* Each client can encrypt media that can be decrypted by every other client.

A security group is represented on a client an MLS group and an SFrame context. The MLS group stores information about the other clients pariticipating in the security group, and information to facilitate updating the group’s secrets as clients join and leave. The SFrame context uses secrets produced by the MLS group to perform encryption and decryption operations.

To facilitate the MLS interactions required to maintain the security group, the group must have access to a Coordination Service (CS). This function is shown as a discrete role below, but could be provided either by a centralized service or by a decentralized algorithm. For example, MLS requires exactly one Commit message per epoch of the group; in this system, the CS decides which Commit will be sent to group members. The structure of the CS is not defined here, except for one notional design in Section 4.2.

A security group needs to be configured with three tracks:

CommitTrack: A track on which MLS Commit messages will be sent to the group.

RequestTrack: A track on which members attempting to join/leave will send requests for action by other members of the group.

WelcomeTrack: A track on which MLS Welcome messages will be sent to new joiners.

All members of the security group subscribe to the CommitTrack so that they can keep up to date on the state of the group. "Committers", members who might make Commits to add or remove other

members subscribe to the RequestTrack in order to learn what work needs to be done. New joiners subscribe to the WelcomeTrack temporarily, so that they can get the Welcome information they need to join the group, then they unsubscribe.

3.2. Lifecycle of a Security Group

Figure 1 summarizes the interactions involved in managing a security group.

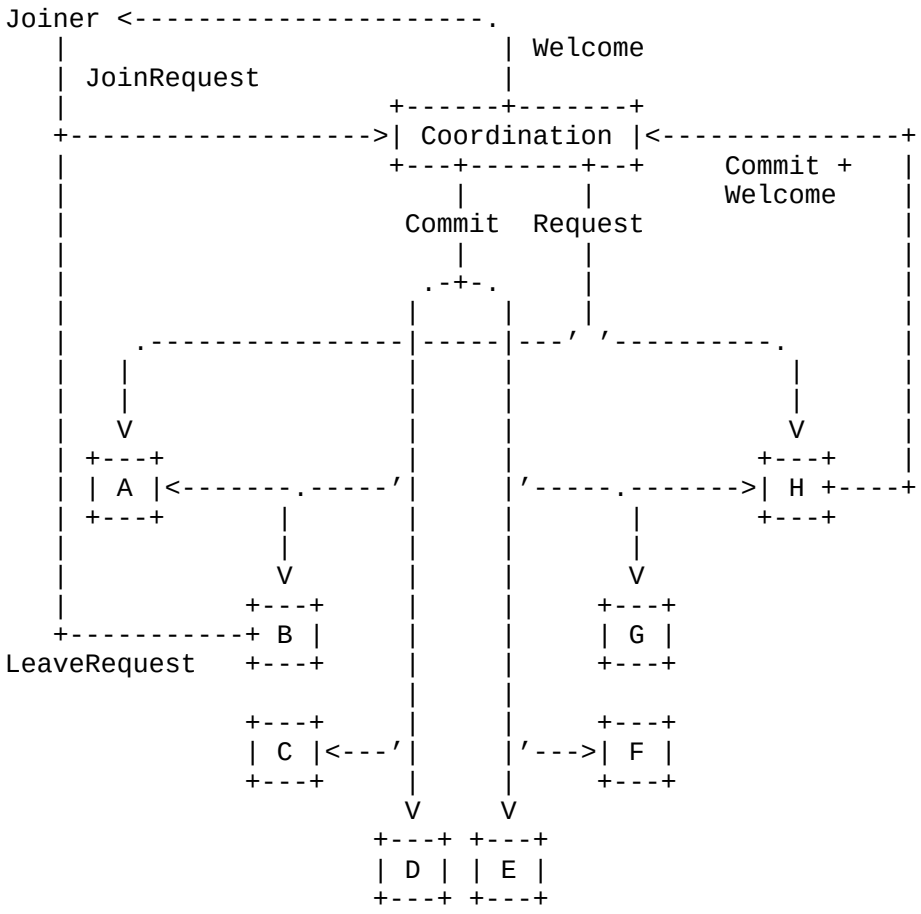

Figure 1: Overview of E2EE Coordination

The life of the security group begins when the first client seeks to join, say client A. Like any other joiner, A subscribes to the CommitTrack and the WelcomeTrack. A has been informed out of band that they will act as a committer, so A also commits to the RequestTrack.

When A sends a request to join the group, the CS informs A that the group does not yet exist. A then locally creates a one-member group.

```
# A joins
A->Relay: SUBSCRIBE(CommitTrack)
A->Relay: SUBSCRIBE(WelcomeTrack)
A->Relay: SUBSCRIBE(RequestTrack)
A -> CS: JoinRequest
CS -> A: Error(Group does not exist)
A: <Creates group, epoch=0>
A->Relay: UNSUBSCRIBE(WelcomeTrack)
```

When B joins, their JoinRequest is instead forwarded to the RequestTrack, and thus to A. A creates a Welcome message that adds B to the group, and a Commit message that updates current members of the group. A sends the Commit and the Welcome to the CS, who forwards the Welcome to the WelcomeTrack (and thus to B), and the Commit on the CommitTrack.

When B receives the Welcome, it can initiate its MLS state. When A receives their Commit back, they know that their Commit has been selected by the CS, and thus that it is safe to update to the next epoch, which includes B.

At the end of this process, A and B are now both members of the security group. They can authenticate each other, and they share secret keys that they can use to encrypt media tracks associated to this security group.

```
# B joins
B: SUBSCRIBE(CommitTrack)
B: SUBSCRIBE(WelcomeTrack)
B -> CS: JoinRequest
CS -> RequestTrack: JoinRequest
RequestTrack -> A: JoinRequest
A -> CS: Commit + Welcome
CS -> WelcomeTrack: Welcome
CS -> CommitTrack: Commit

WelcomeTrack -> B: Welcome
B: <Uses Welcome to initialize state, epoch=1>
B->Relay: UNSUBSCRIBE(WelcomeTrack)

CommitTrack -> A: Commit
A: <Processes Commit, epoch=1>
```

The join process for subsequent members unfolds in the same way: A request to join results in a Commit and Welcome from an existing member, which are distributed to the group and the joiner. The only difference is that now there are members of the group other than the joiner and the committer. These passive members simply apply the Commits as they arrive.

```
# C joins
C->Relay: SUBSCRIBE(CommitTrack)
C->Relay: SUBSCRIBE(WelcomeTrack)
C -> CS: JoinRequest
CS -> RequestTrack: JoinRequest
RequestTrack -> A: JoinRequest
A -> CS: Commit + Welcome
CS -> WelcomeTrack: Welcome
CS -> CommitTrack: Commit

WelcomeTrack -> C: Welcome
C: <Uses Welcome to initialize state, epoch=2>
C->Relay: UNSUBSCRIBE(WelcomeTrack)

CommitTrack -> A: Commit
A: <Processes Commit, epoch=2>

CommitTrack -> B: Commit
B: <Processes Commit, epoch=2>
```

A member leaving works similarly. The member sends a LeaveRequest to the CS, who forwards it on the RequestTrack. Another member generates a Commit that removes the leaving member; no Welcome is needed in this case.

```
# B Leaves
B->Relay: UNSUBSCRIBE(CommitTrack)
B -> CS: LeaveRequest
CS -> RequestTrack: LeaveRequest
RequestTrack -> A: LeaveRequest
A -> CS: Commit
CS -> CommitTrack: Commit

WelcomeTrack -> C: Commit
B: <Processes Commit, epoch=3>

CommitTrack -> A: Commit
A: <Processes Commit, epoch=3>
```

4. Group Management

A security group is managed using three tracks and a Coordination Service (CS).

All members of the group subscribe to the CommitTrack. Each object sent on this track contains an MLS PrivateMessage object, with content type commit. The MOQT group ID for this object MUST be the epoch number in which the Commit is sent. The MOQT object ID for this object MUST be zero. When a client receives an object on the CommitTrack, it applies the enclosed Commit to its local MLS state in order to advance to the next epoch.

Members of the group that might make Commits subscribe to the RequestTrack. Each object sent on this track contains a Request object in the format described in Section 4.1. The MOQT group ID and object ID for objects sent within this track are set by the CS.

Clients seeking to join the group subscribe to the WelcomeTrack. Each object sent in this track contains an MLS Welcome object. A client uses information in the Welcome to detect with one is intended for them, and ignores others. The MOQT group ID and object ID for objects sent within this track are set by the CS.

The CS must present two interfaces:

* A *request interface* that clients can use to ask to join and leave the group. If a request is accepted, it is forwarded on the RequestTrack.

* A *commit interface* that allows a group member to submit an MLS Commit that changes the state of the group, together with an optional MLS Welcome message that adds any new members. If a Commit+Welcome is accepted, the Commit is forwarded on the CommitTrack and the Welcome (if present) is forwarded on the WelcomeTrack.

4.1. Join and Leave Requests

A client requests to join a group by submitting an MLS KeyPackage object. This object provides the information that a group member needs to add the new client to the group.

A client requests to leave a group by submitting an MLS SelfRemove proposal that proposes the client's own removal [I-D.ietf-mls-extensions]. This proposal MUST be sent within an MLS PrivateMessage structure to allow other clients to verify its authenticity.

```
struct {
    RequestType request_type;
    switch (request_type) {
        case join:
            KeyPackage key_package;
        case leave:
            PrivateMessage remove;
    }
} Request;
```

4.2. HTTPS-based Coordination Service

This section describes a simple HTTPS-based CS. The CS has two HTTP endpoints, a request endpoint and a commit endpoint. The only state that the CS keeps is the current epoch, which is initally set to an unknown value.

The request endpoint accepts POST requests from clients. The body of the POST request MUST be a Request object in the format defined in Section 4.1. The response to a request indicates the disposition of the request using an HTTP status code:

200 (OK): The group exists, and the request has been accepted and forwarded on the RequestTrack for the group.

201 (Created): The group did not exist prior to this request. The CS has updated its internal epoch counter to 0, and the requestor should locally create a one-member group.

The CS MUST ensure that only one client receives a 201 response per group. If the CS receives initial requests from two clients simultaneously, it MUST return 201 to one of them and 200 to the other, and perform the corresponding actions.

The commit endpoint accepts POST requests from clients. The body of the POST request MUST be a CommitRequest object of the following form:

```
struct {
    PrivateMessage commit;
    optional<Welcome> welcome;
} CommitRequest;
```

The response to a request indicates the disposition of the request using an HTTP status code:

200 (OK): The group exists and the CS's internal epoch counter matches the epoch value in the PrivateMessage commit. The commit data has been forwarded on the CommitTrack and the welcome data, if present, has been forwarded on the WelcomeTrack. The CS has incremented its internal epoch counter by one.

409 (Conflict): The group exists and the CS's internal epoch counter does not match the epoch value in the PrivateMessage commit. No further action has been taken.

4.3. Optimizations for Large Groups

MOQT sessions are envisioned to include large numbers of clients. In such settings, MLS Commit and Welcome messages can become large, slowing down MLS operations and potentially causing gaps in media. Commit messages can be optimized from linear size to constant size at the expense of putting more processing in the CS [I-D.mularczyk-mls-splitcommit]. Welcome messages can be similarly optimized, but only by reducing the authentication guarantees that clients receive [I-D.kiefer-mls-partial].

5. Media Protection

In a track with an assiciated security group, each object contains an SFrame ciphertext, as specified in [SFrame]. A media sender applies SFrame encryption immediately before sending the object; the plaintext encapsulated in the ciphertext is the data that would have been sent in the object if a security group were not associated.

SFrame parameters are set using the MLS-based key management framewrok described in Section 5.2 of [RFC9605]. This scheme automatically sets the SFrame KID and CTR values based on the sender's MLS state, and automatically derives per-sender keys and nonces.

The metadata input to SFrame computations comprises the unique MOQT identifier for the object, namely the track namespace, track name, group ID, and object ID.

TODO: Define a serialization for the metadata.

5.1. Key Roll-Over

TODO: In practice, it has been useful to have coordinated key roll-over, where clients report that they have received a new key and wait for a quorum before starting to encrypt with it.

6. Security Considerations

TODO: Document security properties, in particular the significance of forward secrecy and post-compromise security in this setting. Note that these properties do not take effect until the media keys have rolled over.

TODO: Define some sort of identity story. How might clients authenticate one another? Does this interact with the MOQ authorization story?

7. References

7.1. Normative References

[MLS] 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>.

[MOQT] Nandakumar, S., Vasiliev, V., Swett, I., and A. Frindell, "Media over QUIC Transport", Work in Progress, Internet-Draft, draft-ietf-moq-transport-12, 23 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-moq-transport-12>.

[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>.

[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/rfc/rfc8446>.

[RFC9605] Omara, E., Uberti, J., Murillo, S. G., Barnes, R., Ed., and Y. Fablet, "Secure Frame (SFrame): Lightweight Authenticated Encryption for Real-Time Media", RFC 9605, DOI 10.17487/RFC9605, August 2024, <https://www.rfc-editor.org/rfc/rfc9605>.

[SFrame] Omara, E., Uberti, J., Murillo, S. G., Barnes, R., Ed., and Y. Fablet, "Secure Frame (SFrame): Lightweight Authenticated Encryption for Real-Time Media", RFC 9605, DOI 10.17487/RFC9605, August 2024, <https://www.rfc-editor.org/rfc/rfc9605>.

7.2. Informative References

[I-D.ietf-mls-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>.

[I-D.kiefer-mls-partial]
Kiefer, F., Bhargavan, K., Barnes, R., Jo谷l, and M. Mularczyk, "Partial MLS", Work in Progress, Internet-Draft, draft-kiefer-mls-partial-00, 28 June 2025, <https://datatracker.ietf.org/doc/html/draft-kiefer-mls-partial-00>.

[I-D.mularczyk-mls-splitcommit]
Jo谷l and M. Mularczyk, "MLS Split Commits", Work in Progress, Internet-Draft, draft-mularczyk-mls-splitcommit-00, 21 October 2024, <https://datatracker.ietf.org/doc/html/draft-mularczyk-mls-splitcommit-00>.

Authors' Addresses

Cullen Jennings
Cisco
Email: fluffy@cisco.com

Suhas Nandakumar
Cisco
Email: snandaku@cisco.com

Richard L. Barnes
Cisco
Email: rlb@ipv.sx