QUIC-MLS: Making a Space Security Draft Standard Resilient for Disconnected Environments

Benjamin Dowling[1][0000−0003−3234−6527], Britta Hale[2][0000−0003−1131−2109], Xisen Tian[2][0000−0001−6171−2309], and Bhagya Wimalasiri[1][0000−0002−7688−504X]

[1] King's College London, UK {benjamin.dowling, bhagya.wimalasiri}@kcl.ac.uk
[2] Naval Postgraduate School, USA {britta.hale, xisen.tian1}@nps.edu

Abstract. Among standardization efforts for space and interplanetary network security, the Internet Engineering Task Force (IETF) is driving work on space network security, accounting for the unique properties of space environments that make space communication challenging. This includes long, variable-length delays, packet loss, and intermittent end-to-end connectivity. Within these efforts, there is a focus on using IP-based protocols for security, and in particular the use of the QUIC protocol. This is unsurprising given QUIC's growing popularity and offer of optimization intended for reducing latency. However, QUIC uses the Transport Layer Security (TLS) key exchange handshake protocol, which was originally designed for 'connect and forget' style Internet connections at scale. It is also session-based, where protocol participants require reestablishment of the session for each reconnection – a costly maneuver in the space setting. Furthermore, TLS by default does not achieve strong post-compromise security properties within sessions, exhibiting a risk under long-lived connections, and need for synchronous handshakes to counteract this are in functional contrast to the space environment, which has intermittent end-to-end connectivity.
We address both drawbacks of QUIC by introducing QUIC-MLS: a variant of QUIC which replaces the session-based, synchronous TLS handshake with the standardized continuous key agreement protocol, Messaging Layer Security (MLS), which achieves asynchronous forward secrecy and post-compromise security. In addition to the design itself, we implement our design and provide benchmarks, and analyze our new construction in a formal cryptographic model.

1 Introduction

Space may be a vacuum, but humans are helping fill it up. According to the United Nations Office for Outer Space Affairs [1], humans have sent more than 2,500 objects into space in 2024, a trend that continues to increase. Since many of these objects are unmanned, they require communication protocols to enable control systems on Earth to make mission-critical adjustments.

However, secure communication protocols used in the terrestrial setting are not generally appropriate as-is for use in space. One can see this in the Internet

Engineering Task Force (IETF) where the Taking IP to Other Planets (TIPTOP) working group has been outlining characteristics and use case requirements for space, that deployed protocols must address, including challenges such as “variable delays”, “frequent and long interruptions of communications" and “one-way / unidirectional links”. Within the same working group, the first and primary focus for deployable secure IP-based protocols for space networking is the QUIC protocol [2,3].

The internet-wide adoption of the QUIC protocol is contributing to its use IP-based space communications [4,5,6], delay tolerant networks [7], and vehicular adhoc mobile networks [8]. In internet usage, generic QUIC traffic held the #3 spot for total volume and accounted for 6-8% of global internet traffic in 2023 [9]. An arguable driver in the use of QUIC over more traditional protocols such as TLS 1.3 is the efficiency gains and latency reductions from its integrated approach across network layers. While Transport Layer Security (TLS) version 1.3 must be combined with some transport protocol, and therefore is not optimized for combination with any particular one, QUIC has the benefit of optimized integration of transport and security protocol aspects. Multiplexing capabilities are another benefit of the QUIC protocol. However, QUIC has some characteristics that make it difficult to support a deepspace use-case.

QUIC inherited the legacy key establishment characteristics of its predecessor, TLS 1.3, specifically for key exchange (KEX). QUIC uses the TLS 1.3 KEX, which incurs at least one full round trip (RTT) even with the QUIC optimizations [3]. The 0-RTT TLS mode could intuitively be considered as a solution to this inefficiency, but has known weaknesses [10,11]. This leads to a tension between security and the QUIC goals of reducing latency. Within terrestrial uses these issues are not prominent, but within space settings, where transmission delays can take minutes or even hours, the both latency and security risks become more prominent. We highlight that the core functionality of the TLS 1.3 KEX is negatively impacted by the challenges of the deepspace setting. A 1 round trip time (RTT) synchronous handshake requires bidirectional, synchronous communication in order to achieve forward secrecy.

However, a fundamental review of the core challenges reveals a path forward. The TLS 1.3 KEX is a session-based key agreement mechanism, meaning that parties are not assumed to maintain state between sessions. This is an artifact of TLS’s original design for use on the Internet, where tens of thousands of new and changing connections make it undesirable to maintain state between instances of client connections. In contrast, in uses such as space settings, where the set of communication parties is usually small and can even be static, state can be maintained between communications. This obviates the need for the original session-based design and opens the aperture for lower latency together with improved security through use of continuous session protocols. Moreover, given that the use of TLS KEX within QUIC is itself a selected handshake option, a such more appropriate handshake alternative could likewise replace it.

Continuous session protocols, a.k.a. continuous key agreement or ratcheted protocols, are stateful protocols that enable key rotation on demand, perhaps

according to a schedule, which removes the need for session handshakes at the start of every connection and therefore reduces RTT. This does not necessarily come at the cost of security; in fact, continuous session protocols have been shown to achieve properties that were unattainable under traditional TLS [12]. Normal TLS sessions provide forward secrecy (FS), where *past* communications remain secure if the current session is compromised. In contrast, a continuous session protocol can achieve both FS and post-compromise security (PCS), thus "self-healing" and locking out adversarial access *after* a key compromise [13].

TLS and QUIC have slowly been moving towards a continuous setting, namely with additions for session resumption. Yet these session-based protocols have never been analyzed as continuous session protocols nor fundamentally designed for those goals, and even recent work on session resumption is still synchronous in nature, contrary to the functional needs of space communications. Thus, in this work we show how to leverage another standardized, yet continuous key agreement protocol, Messaging Layer Security (MLS) [14], as an alternative KEX within QUIC in order to create a continuous session protocol QUIC variant. This reduces the latency and security issues of QUIC within space that were previously noted [10,11], supports PCS, and still maintains the networking functionality optimizations of QUIC. Furthermore, it opens up potential for moving uses of QUIC beyond the client-server setting [4,8]. As an added benefit, since MLS was designed for scalable asynchronous group key agreement [14], our solution introduces options beyond pairwise QUIC, namely reducing the overhead of key updates in QUIC for larger multi-device configurations, bringing it down from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \cdot log(n))$ for groups of size n [15]. This benefit is optional, as even 1-to-1 (group size of 2) settings obtain the other benefits of the construction.

In this work we provide a cryptographic design for using the MLS KEX with QUIC, a security model and computational analysis of QUIC-MLS, and benchmark testing of the solution. For generality we model and analyze for the more complex multi-party case, where group key agreement resulting in a shared secure QUIC data channel. This is fully extensible to the 1-to-1 communication scenario where QUIC-MLS is used in a typical pair-wise case, e.g., client-to-server or client-to-client. Our QUIC-MLS variant turns QUIC from a stateless session based protocol to a stateful continuous protocol opening doors for wider adoptions beyond the traditional web applications, and is suitable for the challenges highlighted by the IETF.

2 Preliminaries

Here we describe the abstract models used to capture key establishment methods such as TLS, QUIC, and MLS.

2.1 Secure Channel Establishment

Session-based key establishment and continuous key agreement (CKA) represent two distinct paradigms for establishing a secure channel. Session-based key

exchange protocols like TLS or Internet Key Exchange (IKE) execute so-called handshakes to negotiate cryptographic parameters and perform cryptographic operations to derive an ephemeral session key used between two parties that wish to securely communicate. In particular, handshakes are executed independently, with certificates transmitted at each establishment to authenticate the new connection. This approach is used for one-to-one sessions that are short-lived and benefit from statelessness between sessions.

In contrast, CKA protocols, such as the Signal protocol, have often been used as the cryptographic core of secure messaging: they establish a shared cryptographic state between communicating parties and allow for new keys to be generated and updated incrementally and asynchronously (i.e. *without* additional handshakes upon re-connection). This allows for asynchronous key updates, granular control over key lifetime between refresh and update, and recovery from adversarial state compromise. For systems that can afford to manage and store cryptographic states, CKA protocols offer additional security guarantees—such as post-compromise security—compared to their session-based counterparts. CKAs are extended to the multi-user setting as Continuous Group Key Agreement (CGKA), enabling groups of users to establish a series of shared keys consistently. The TreeKEM sub-protocol of the MLS protocol [14] is a prime example of a deployed CGKA. Here we provide the formal definition of a CGKA.

Definition 1 (CGKA from [16]). *A continuous group key-agreement (CGKA) scheme* $\mathsf{CGKA} = (\mathsf{KGen}, \mathsf{Create}, \mathsf{Add}, \mathsf{Join}, \mathsf{Upd}, \mathsf{Rem}, \mathsf{Commit}, \mathsf{Process}, \mathsf{Key})$ *consists of the following algorithms:*

- ***Key Generation****:* $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathsf{KGen}()$ *samples a fresh public/secret key pair.*
- ***Group creation****:* $\gamma \leftarrow \mathsf{Create}()$ *takes no input and returns a fresh group state* γ *containing only the party running the algorithm. This represents the first* epoch *of a new session.*
- ***Add****:* $(\gamma', \mathrm{p}) \leftarrow \mathsf{Add}(\gamma, \mathsf{id}_t, \mathrm{pk}_t)$ *proposes adding a new member to the group. On input a protocol state* γ*, identity of the new member* id_t *and their public key (generated by* KGen*), it outputs an updated state* γ' *and an* add proposal message p.
- ***Remove****:* $(\gamma', \mathrm{p}) \leftarrow \mathsf{Rem}(\gamma, \mathsf{id}_t)$ *proposes removing a member from the group. On input a protocol state* γ *and identity* id_t*, it outputs an updated state* γ' *and* remove proposal message p.
- ***Update****:* $(\gamma, \mathrm{p}) \leftarrow \mathsf{Upd}(\gamma)$ *proposes updating the member's key material. It outputs an updated state* γ' *and an* update proposal message p.
- ***Commit****:* $(\gamma', \mathrm{commit}, \mathrm{welc}) \leftarrow \mathsf{Commit}(\gamma, \overrightarrow{\mathrm{p}})$ *applies (aka* commits*) a vector of proposals to a group. The output consists of an updated protocol state* γ'*,* commit message *and a (potentially empty)* welcome message *(depending on if any add proposal messages were included in* $\overrightarrow{\mathrm{p}}$*).*
- ***Join****:* $(\gamma', \overrightarrow{\mathsf{G}}, \mathsf{id}_i) \leftarrow \mathsf{Join}(\mathrm{sk}, \mathrm{welc})$ *allows a party with secret key* sk *(generated by* KGen*) to join a group with a welcome message* welc*. The outputs are: an updated protocol state* γ'*, a group* roster $\overrightarrow{\mathsf{G}}$ *(i.e. a set of IDs listing the group members), an epoch ID, and the ID of the inviter (i.e. the party that created the welcome message).*

– ***Process:*** $(\gamma', \text{info}) \leftarrow \mathsf{Process}(\gamma, \text{commit}, \overrightarrow{\text{p}})$ *processes an incoming commit message and the corresponding proposals to output a* commit info message info *and an updated group state* γ' *which represents a new epoch in the ongoing CGKA session. The commit info message captures the semantics of the processed commit and has the form:* $\text{info} = (\mathsf{id}, (\text{propSem}_1, \ldots, \text{propSem}_z))$ *where* id *is the ID of the sender of the commit message the* propSem*vector conveys the semantics of the committed add and remove via triples of the form* $\text{propSem} = (\mathsf{id}_s, \mathsf{op}, \mathsf{id}_t)$. *Here,* id_s *denotes the identity of the proposal's sender* $\mathsf{op} \in \{\text{“addP”}, \text{“remP”}\}$ *is the proposal's type and* id_t *is the identity of the proposal's target (i.e. the party being added or removed).*
– ***Get Group Key:*** $(\gamma', K) \leftarrow \mathsf{Key}(\gamma)$ *outputs the current group key for use by a higher-level application and deletes it from the state.*

We next cover the TLS, QUIC, and MLS standards of the of the IETF, and how their components fit together.

2.2 Transport Layer Security

As a session based protocol, main goal of TLS [17] has always been to establish a secure communications channel across an untrusted network (the Internet). In what began with Secure Socket Layer (SSL) protocol created by Netscape, the TLS protocol has been upgraded multiple times by the IETF, resulting in the current iteration of TLS 1.3 (hereto simply referred as TLS) [17]. TLS itself is comprised of sub-protocols: the handshake protocol which establishes keys (TLS-HS), the record layer protocol (TLS-RL) which uses those keys to provide a secure channel for data transmission using authenticated encryption with associated data (AEAD) and an alert protocol. TLS supports session resumption using an established secret for faster establishment of future sessions with the same client (e.g. 0RTT).

2.3 QUIC

QUIC was first developed by Google in 2012 and later adopted and refined as an internet standard by the IETF [3]. Because QUIC uses TLS-HS for security key establishment [18], its evolution has been tied to TLS's evolution. For analysis purposes, we will also use a similar break down in reference to QUIC, namely as having a handshake component (QUIC-HS $\approx$ TLS-HS) for key agreement and a record layer (QUIC-RL) component. Unlike TLS, once a session key is established in QUIC (i.e., after QUIC-HS), it can be updated using a key-phase bit which is flipped between parties to indicate the use of a new key from a key derivation function (KDF) applied to the old key. This is a symmetric key ratchet, and should not be interpreted as introducing new randomness or achieving PCS. Furthermore, QUIC offers a 0-RTT connection mode for previously connected clients through relying on a server-signed configuration file stored and presented by the client; this is based on the TLS 0-RTT mode.

2.4 Messaging Layer Security (MLS) Protocol

The MLS (RFC 9420) [14] standard was designed for end-to-end encryption, providing users with secure and scalable communications. Similarly to QUIC, MLS can be broken down into key exchange and security channel/*record layer* phases; however, as it was originally intended for application messages, the *record layer* phase is at the application layer. Notably, as with TLS and QUIC, the networking layer use of the overall protocol is determined by the *record layer* protocol definition—the key exchange portion of MLS is not tied to a particular networking layer use. This allows us to modularly change the QUIC-HS from TLS-HS to MLS, as shown in Figure 1.

Instead of session-based handshake negotiation for key exchange, MLS and similar CKAs establish the cryptographic session state once and then update keys as needed or as scheduled (potentially asynchronously) without additional round-trip negotiations. Such key updates are also what enables PCS. MLS allows for a notable differentiation from prior pair-wise CKA protocols in that it supports *groups*, meaning that not only is it a CKA, but a *continuous group key agreement* (CGKA). Group members can efficiently perform the CGKA with entity authentication and key indistinguishability security guarantees [19]. Group members/clients can also be added or removed, enabling adaption.

3 Related Works

MLS has undergone extensive computational analysis as a CGKA [16,20] resulting in major changes to the protocol by the IETF. Similarly, symbolic analysis (in the Dolev-Yao model) has been used to analyze MLS [21]; such symbolic analysis abstracts cryptographic primitives as "black boxes". In this work, we use the computational analysis approach to analyze and prove security QUIC-MLS, leveraging existing models from work on CGKAs.

Work on CKAs include analyses on the Signal protocol. While there is extensive work in this domain [22,23,24,25,26], we opt to use MLS as it is already standardized. This fits with the overall theme of shifting out one standardized key establishment (TLS-HS) with another (MLS) within the QUIC protocol.

Other related works include analyses of QUIC, especially the work done on record layer analysis [27,28] and TLS vs. QUIC [29]. As noted before, prior work has raised concerns on use of the TLS 0-RTT option [10,30].

4 Protocol Design and Mechanics

In this section we provide a high level overview of how MLS can be incorporated into QUIC in terms of requirements analysis and protocol construction. QUIC combines key agreement and secure channel management into one protocol: for analysis purposes we break these out as Handshake (QUIC-HS) and Record Layer (QUIC-RL) responsibilities, respectively, to reflect the TLS 1.3 components that they are derived from (i.e. TLS Handshake and Record Layers). Low level details such as wire formatting are out of scope for this work.

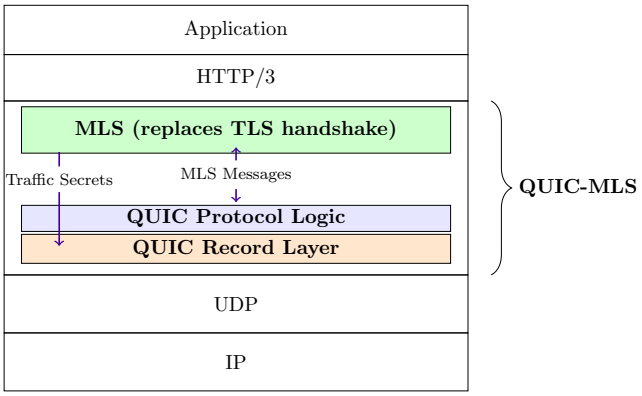

Fig. 1: High level overview of QUIC-MLS, showing slot-in replacement of TLS-HS with MLS.

4.1 General Requirements

QUIC has three general requirements of its handshake component: 1) authenticated key exchange, 2) transport parameter authentication, and 3) authenticated application protocol negotiation [3]. MLS can be used as a drop-in replacement for TLS from the perspective of QUIC-RL, as MLS meets the minimal requirements to be an authenticated key agreement protocol. With minimal additions to the MLS `GroupInfo` to specify transport parameters and application protocols the other requirements are met with MLS as well (see Section 4.4).

In satisfying the QUIC requirements, MLS has remaining requirements of its own to function properly. Specifically, to facilitate client communication, MLS relies on two abstract services: an Authentication Service (AS) and a Delivery Service (DS). The AS, as previously mentioned, attests to the bindings of identifier to public key material which can be adapted from existing PKI network services. For our analysis purposes, we assume that the AS functions as it currently does through existing PKI as used by TLS. The DS receives and distributes messages between group members and ensures the ordered delivery of MLS handshake messages [31]. A *strongly consistent* DS ensures messages arrive in order and that all members agree on that order. In web applications, for example, a strongly consistent DS can be enforced through reliable and order enforcing transport protocols or via a centralized server. An *eventually consistent* DS only guarantees that messages arrive (eventually) to all members but makes no guarantees on their ordering, leaving clients responsible for reconciliation. An eventually consistent DS is endemic to decentralized, peer-to-peer, or hop-by-hop networks.

While a protocol designed for an eventually consistent DS can be streamlined for an strongly consistent DS, the reverse is not true. Thus, we design QUIC-MLS for an eventually consistent DS with tunable parameters to fit the reality of the underlying network up to a strongly consistent DS. To provide a clear mechanism to ensure deterministic handling of group state updates and avoid race conditions arising from concurrent update proposals, we propose introducing

a designated group administrator responsible for all membership changes which may change in a fixed order (e.g. based on the sequence in which members joined the group) from epoch to epoch. This is consistent with the notion of epoch leaders in previous work [**?**] except that only the administrator may initiate add or remove proposals; any such proposals from other group members will be ignored. Additionally, to ensure all members can synchronize group state, a "commit transcript" of a tunable length is appended to all messages sent. In the worst case of an eventually consistent DS, this may contain all previous commits since the start of the session. The length can be parameterised to the reliability of the network; with a strongly consistent DS, the length is zero.

4.2 Construction Overview

Our QUIC-MLS construction, detailed in Figure 2, is composed of KGen, Init, Enc, and Dec. KGen is used to generate public keys, simply by calling $\mathsf{CGKA.KGen}$. In Init, a group admin instantiates the group session (using the intended partner's public keys) using the CGKA algorithms Create, Add and Commit. All welcome messages generated by Init are sent to recipients via QUIC-RL who call Init as a participant ($\rho = \mathsf{p}$) upon receipt to join the group (via $\mathsf{CGKA.Join}$).

Enc enables payload encryption via QUIC 0RTT, and generation of group control proposals (e.g. $\mathsf{Add}, \mathsf{Rem}, \mathsf{Upd}$). Similarly, Dec decrypts payload messages and processes proposals and commits. To ensure reliability, ciphertexts contain "a commit transcript" ast with a tunable length tl to reconcile group state synchronization. In an asynchronous scenario with a two-party group and strongly consistent DS the tl can be set to 1 when assuming parties are never more than 1 epoch apart. If Dec updates the session's group state, they derive a new key (via $\mathsf{CGKA.Key}$) for payload encryption via QUIC 0RTT.

4.3 Limitations

While our approach is intended as a clean substitution of TLS in QUIC, it introduces a limitation on non-repudiation due to the nature of group communications versus two-party communications. QUIC does not sign messages because non-repudiation is implicit in the channel among two parties who have completed authenticated key exchange. In a group channel, however, insiders can impersonate one-another if messages are not signed with the sender's private key. To avoid this, group channel messages for QUIC-MLS can also be signed using public and private keys of clients used for MLS. However, this composition requires further analysis as we must then also consider allowing adversary access to a signing and verification oracle within the context of the CGKA security model. We leave this for future work and assign handling non-repudiation of insiders to the application layer.

4.4 Changes to Respective Protocol Standards

We make no changes to the underlying cryptographic components of MLS or QUIC-RL as standardized. However, we change the semantics of some QUIC

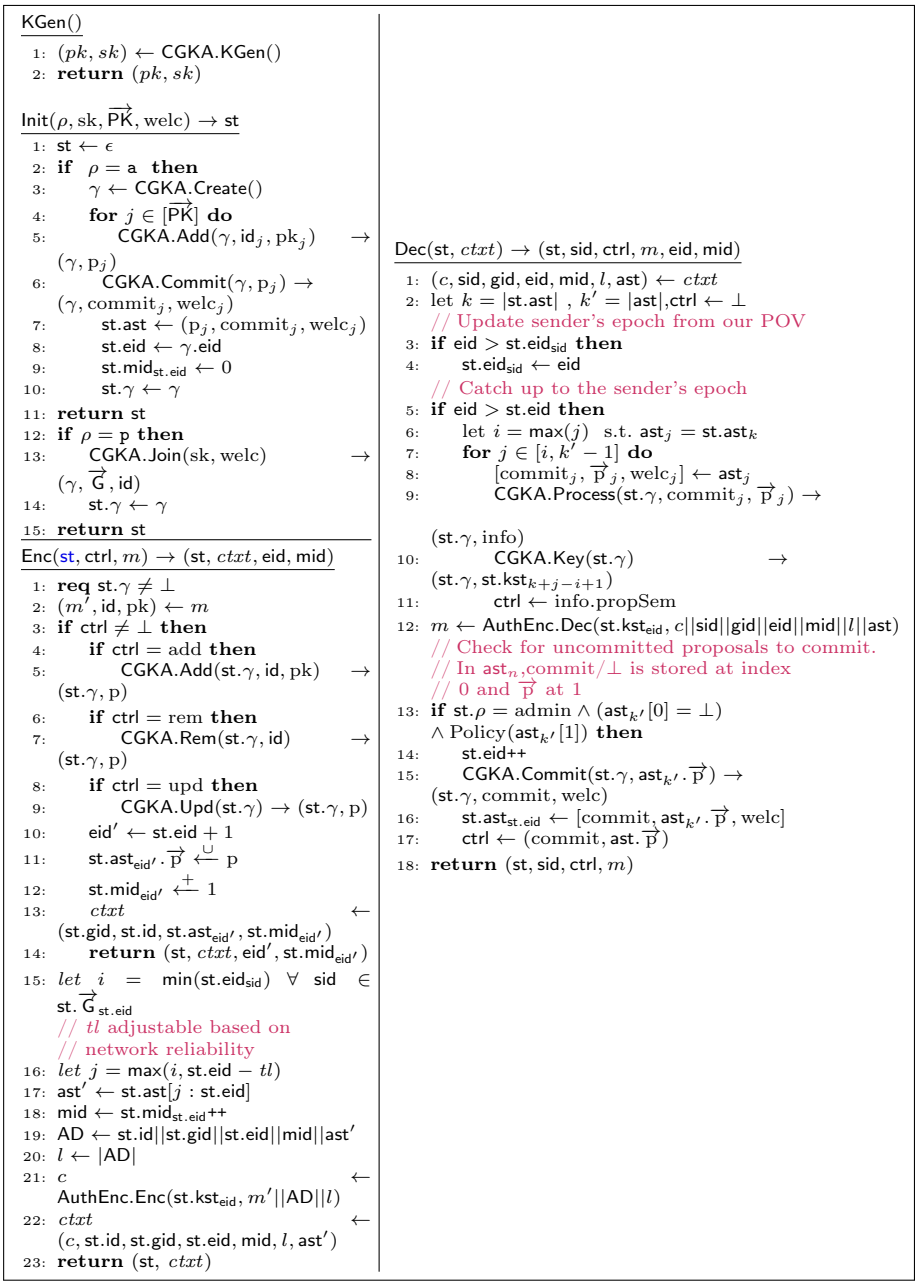

Fig. 2: QUIC-MLS algorithms. All algorithms abort if any CGKA sub-algorithms return $\perp$.

terminology in order to adapt MLS. The association of QUIC security levels to certain message types throughout the QUIC handshake is obviated by the single

stage MLS key agreement mechanism. As observed in Figure 3, clients continue to use 0-RTT messages for sending data after the initial volley of messages. This choice conveys the efficiency of QUIC-MLS and that MLS security is obtained from the session start.

To address the requirements for QUIC-HS to offer authenticated transport parameter and application protocol negotiation, we utilize the MLS Safe Extension [32] to add further context to the group. Specifically, for application-layer protocol agreement, a list of protocols would be included in the `data` vector of the `ComponentData` struct within the `app_data_dictionary` in the `GroupContext`. For transport layer parameters advertisement, this can simply be a list added to the `aad_item_data` with an appropriate `component_id` in the `SafeAAD` struct to be more transparent. These changes have been proposed to the MLS working group (see pull requests for [32]).

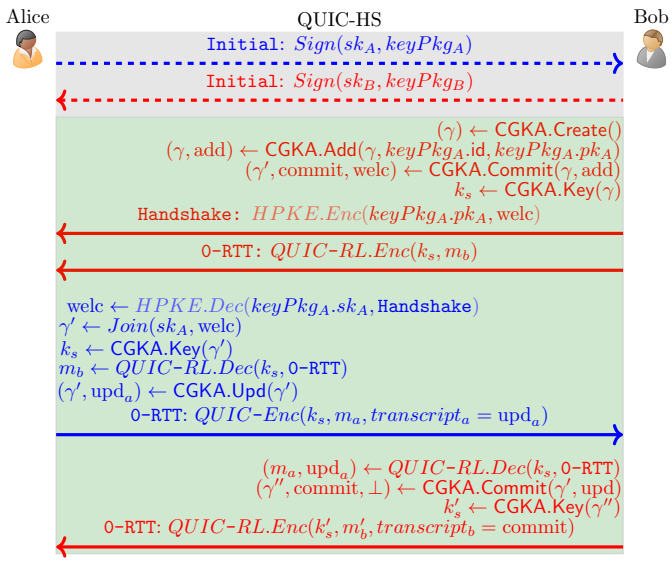

Fig. 3: QUIC-MLS execution with two parties. The use of Initial QUIC packets to send MLS Key Packages is optional if key packages are installed out-of-band/prior.

5 Benchmarking QUIC-MLS

In this section, we present benchmarking results for our proposed QUIC-MLS construction. As part of our benchmarking efforts, we developed a reference implementation [3] prototyping QUIC-MLS functionality in Go, leveraging the existing libraries `(quic-go)`, `(go-mls)`, and `(crypto/aes)` `(crypto/tls)`, [33][34],

[3] `https://osf.io/qun62/?view_only=e814346aa4334fe1be43a81a89f01c8d`

[35]. We note, at the time of writing, our reference implementation is not a complete integration of MLS into QUIC. Instead, it serves as a prototype for assessing the computational and bandwidth costs associated with our proposed QUIC-MLS construction. This is due in part to the tight coupling of the `(quic-go)` [33] library with Go's native TLS stack, making it non-trivial to fully replace TLS with MLS without extensive refactoring. However, we simulate the core functionality of the QUIC record layer by encrypting application data using AEAD schemes with session secrets derived from MLS, enabling a functionally comparable evaluation of performance and overhead.

We instantiated QUIC-MLS with the cipher suite `P256_AES128GCM_SHA256_P256` and benchmarked its average performance across group sizes of 2, 5, 10, 20, 50, and 100 members, using 100 iterations per group size. For each epoch, the derived `ExporterSecret` was utilized for `0-RTT` encryption of application-layer data.

In Table 1, we compare the average performance of the cryptographic procedures involved in the key package and group creation stages of QUIC-MLS. During key package creation, QUIC-MLS issues each member a set of credentials that can be leveraged in subsequent group operations. We observe that average creation times increase steadily with group size, scaling gradually as more members are added. Within the group creation stage, QUIC-MLS facilitates the formation and management of dynamic groups, where a designated group *admin* handles Add and Remove proposals by processing Join requests and issuing corresponding Welcome messages. Reflecting the increased overhead of managing group state and the complexity of underlying cryptographic operations, average group creation time exhibits a more dramatic rise, particularly as the group becomes large.

Group Size	KeyPackage Creation	Group Creation
2	$\approx 0.235 \pm 0.373$	$\approx 0.723 \pm 0.946$
5	$\approx 1.42 \pm 0.564$	$\approx 6.72 \pm 1.781$
10	$\approx 2.12 \pm 0.508$	$\approx 15.79 \pm 2.261$
20	$\approx 3.11 \pm 0.9$	$\approx 45.97 \pm 7.129$
50	$\approx 10.09 \pm 2.613$	$\approx 191.49 \pm 27.395$
100	$\approx 17.566 \pm 6.253$	$\approx 712.57 \pm 267.381$

Table 1: Average runtime (in milliseconds over 100 iterations) of cryptographic operations during key package and group creation in QUIC-MLS, evaluated on an **Intel(R) Core(TM) i7-11370H CPU @ 3.30GHz**.

In Table 2, we compare the cost of QUIC-MLS Update operations in terms of both performance and bandwidth overhead. In particular, we model a use case involving a two-member group, where one member processes committed Update messages from the other after 5, 10, 20, 50, and 100 Commit operations. This setup is intended to simulate a scenario in which a QUIC-MLS group member remains part of the group but is inactive for an extended period. When the inactive member resumes communication, they must update their local group state to the latest epoch before re-engaging with the group. Figure 4 illustrates

the expected format of a QUIC-MLS `Update` packet within our construction. Crucially, to resume sending encrypted messages over the QUIC record layer, the inactive member must first derive the most recent `ExporterSecret` from the updated group state. In a catch-up scenario, the inactive member processes previously committed `Commit` messages to update its local state. The `path` field provides the encrypted key material necessary to advance to the latest epoch.

Our evaluation shows that epoch synchronization time grows predictably with the number of missed updates, reflecting the accumulation of cryptographic state. Sync message size follows a similar trend, increasing with the volume of group information that must be communicated.

```
QUICMLSUpdate group_id, epoch, sender, authenticated_data, content: Commit proposals
: [empty], path: DirectPath nodes: [...] , signature, confirmation_tag
```

Fig. 4: QUIC-MLS packet structure for maintaining admin state (See Fig. 2 ast)

Missed Commit Count	State Sync Time	Message Size
5	$\approx 5.41 \pm 0.501$	$\approx 6,119$
10	$\approx 10.02 \pm 2.873$	$\approx 11,204$
20	$\approx 20.01 \pm 3.325$	$\approx 21,369$
50	$\approx 51.55 \pm 8.062$	$\approx 51,903$
100	$\approx 101.43 \pm 22.756$	$\approx 102,740$

Table 2: Average runtime and bandwidth evaluation (in milliseconds for sync time and bytes for message size over 100 iterations) during epoch Update for inactive member in QUIC-MLS evaluated on **Intel(R) Core(TM) i7-11370H CPU @ 3.30GHz**.

6 Security Analyis of QUIC-MLS

Both TLS and QUIC have been extensively analyzed under various provable security approaches that either separate the key exchange and secure channel phases or consider them together as an intertwined protocol. Security goals commonly analyzed across the various approaches include key indistinguishability, authenticated and confidential channel establishment, protection against replay attacks, defense against state reveals, and forward secrecy. Jager et al. [36] formalized the monolithic approach as *authenticated and confidential channel establishment* (ACCE) protocols, which are separated into pre-accept and post-accept phases (similar to the TLS Handshake and Record Layer phases [37]). Follow-on work from Lychev et al.in [38] take a more holistic approach in analyzing QUIC and define the Q-ACCE model based on how QUIC uses TLS with multiple keys for key agreement and data messages in a two stage approach. Fischlin and Günther in [39] analyze QUIC in their multi-stage key exchange (MSKE) model in the Bellare and Rogaway style of [40], highlighting their model's composability with any symmetric-key application protocol. Tangential to this composability emphasized approach, Dowling et. al. in [41] propose the Flexible ACCE (fACCE) model which eliminates the boundary between the handshake and channel yet is able to measure fine-grained security guarantees at each phase.

6.1 gACCE for QUIC-MLS

We use a modified fACCE model we call gACCE in our analysis of QUIC. To this end, we make some changes to the fACCE primitive to accommodate group channel security in what we dub gACCE (Definition 2).

Definition 2 (Flexible ACCE for Groups ([41]). *A flexible group ACCE protocol* gACCE *is a tuple of algorithms* $\mathsf{gACCE} = (\mathsf{KGen}, \mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ *defined over a secret key space* $\mathcal{SK}$*, a public key space* $\mathcal{PK}$ *and a state space* $\mathcal{ST}$*. The syntax of a* gACCE *protocol is as follows:*

- $\mathsf{KGen} \rightarrow_{\$} (\mathrm{sk}, \mathrm{pk})$ *generates a long-term key pair where* $\mathrm{sk} \in \mathcal{SK}, \mathrm{pk} \in \mathcal{PK}$*.*
- $\mathsf{Init}(\rho, sk_i, \overrightarrow{\mathsf{PK}}_{\overrightarrow{\mathsf{G}}}, \mathrm{welc}, ad) \rightarrow_{\$} st$ *initializes a session to begin communication, where* sk_i *(optionally) is the caller* i*'s long-term secret key,* $\overrightarrow{\mathsf{PK}}_{\overrightarrow{\mathsf{G}}}$ *(optionally) are the long-term public keys of the intended group session partners* $\overrightarrow{\mathsf{G}}$ *such that* $i \notin \overrightarrow{\mathsf{G}}$*,* $\rho \in \{\mathsf{a}, \mathsf{p}\}$ *the session's role (i.e. admin or partner), ad is the data associated with this session, and* $sk \in \mathcal{SK} \cup \{\perp\}, \mathrm{pk} \in \mathcal{PK} \; \forall \; \mathrm{pk} \in \overrightarrow{\mathsf{PK}}_{\overrightarrow{\mathsf{G}}}, ad \in \{0,1\}^*, st \in \mathcal{ST}$*.*
- $\mathsf{Enc}(\mathsf{st}, \mathsf{ctrl}, m) \rightarrow_{\$} (\mathsf{st}', ctxt, \mathsf{eid}, \mathsf{mid})$ *enables a group member to encrypt a message, or to propose to* evolve *the group by adding, removing or updating members of the group. It takes as input a state* st*, an optional control message* ctrl *and a message* m*, and outputs new state* st'*, and a ciphertext ctxt (where* $\mathsf{st}, \mathsf{st}' \in \mathcal{ST}, \mathsf{ctrl} \in \{\perp, \mathsf{Add}, \mathsf{Upd}, \mathsf{Rem}\}, m \in \{0,1\}^*$ *when* $\mathsf{ctrl} = \perp$ *otherwise* $m \leftarrow (\mathsf{st.id}, \mathrm{pk}_{\mathsf{st.id}})$*,* $ctxt \in \{0,1\}^* \cup \{\mathrm{propSem}\}, \mathsf{eid} \in \mathbb{N}$ and $\mathsf{mid} \in \mathbb{N}$*).*
- $\mathsf{Dec}(\mathsf{st}, ctxt) \rightarrow_{\$} (\mathsf{st}', \mathsf{id}, \mathsf{ctrl}, m, \mathsf{eid}, \mathsf{mid})$ *processes any updates to the group or decrypts messages by other group members. It takes as input a state* st *and ctxt and outputs new state* st'*, sending partner's identifier* id*, message* m*, control message* $\mathsf{ctrl} \in \{\perp, \mathsf{Upd}, \mathsf{Add}, \mathsf{Rem}, \mathsf{Commit}\}$*, and epoch and message counters* eid *and* mid*, where* $\mathsf{st} \in \mathcal{ST}, st' \in \mathcal{ST} \cup \{\perp\}, m \in \{\{0,1\}^* \cup \perp\}$*,* $ctxt \in \{0,1\}^*, \mathsf{mid} \in \mathbb{N}$ and $\mathsf{eid} \in \mathbb{N}$*. If* $m = \perp$*, then this denotes a failure to process the message.*

To capture security in a group setting, we need to modify the fACCE's matching conversations and correctness definitions. Two group members share a *matching conversation* when they agree on the *control messages* used to evolve the group, and that any received payload messages are the same as the original sent message. The control messages are held in CT and the payload application messages are held in AT. Due to the propose-and-commit nature of MLS, we introduce the concepts of a *confirmed transcript* and *opportunistic transcript* for CT as an expansion of the message-stage-ciphertext transcripts in [41]: group members can propose changes that are (validly) discarded by the admin.

The confirmed transcript maintained by a group member party is a pair of structures ordered by the currently member $\mathsf{id} \in \overrightarrow{\mathsf{G}}$ and indexed by epoch eid. $(\mathrm{CT}^{\mathrm{c,e}}_{\mathsf{id},\mathsf{eid}}, \mathrm{CT}^{\mathrm{c,d}}_{\mathsf{id},\mathsf{eid}})$: where $\mathrm{CT}^{\mathrm{c,e}}_{\mathsf{id},\mathsf{eid}}$ records all messages that have been encrypted (sent) and $\mathrm{CT}^{\mathrm{c,d}}_{\mathsf{id},\mathsf{eid}}$ that holds decrypted messages (received) from a party $n \in \overrightarrow{\mathsf{G}}$

in epoch eid. The opportunistic transcript is similarly composed of a $\mathrm{CT}^{\mathrm{o,e}}_{n.\mathsf{eid}}$ and $\mathrm{CT}^{\mathrm{o,d}}_{n.\mathsf{eid}}$ except these only hold *proposed* group *control messages*, respectively. The opportunistic transcript control messages have not been committed: when they do, they are moved to the *confirmed* transcript. Together, these allow gACCE to both match application messages and match epoch evolutions through staging pending proposals until a successful decryption of a commit message triggers a new epoch. We now define correct execution of an gACCE protocol. Finally, the application transcript $\mathrm{AT}^{x}_{\mathsf{id},\mathsf{eid}}$ (where $x \in \{e, d\}$) contain all application messages (i.e. payload) encrypted and decrypted by id in epoch eid. Note that the decrypted $\mathrm{CT}^{\mathrm{d}}_{\mathsf{id},\mathsf{eid}}[\mathsf{id}']$ is indexed by the claimed sender partner identifier id'.

Definition 3 (Correctness of gACCE (adopted from [41]). *A* gACCE *protocol is* correct *if, for any pair of keys* $(\mathrm{sk}_i, \mathrm{pk}_i)$ *and* $(\mathrm{sk}_j, \mathrm{pk}_j)$ *output by* KGen *such that* $(i, j) \in \overrightarrow{\mathsf{G}}$*, with* $i \neq j$*, the following holds. Let* $ad \in \{0,1\}^*$*. The session states are initialized as* $\mathsf{st}^{\mathsf{a}}_i \xleftarrow{\$} \mathsf{Init}(\mathsf{a}, \mathrm{sk}_i, \{\mathrm{pk}_{\overrightarrow{\mathsf{G}}}\}, ad)$ *and* $\mathsf{st}^{\mathsf{p}}_j \xleftarrow{\$} \mathsf{Init}(\mathsf{p}, \mathrm{sk}_j, \{\mathrm{pk}_{\overrightarrow{\mathsf{G}}}\}, ad)$*. Application transcripts are initialized as* $\mathrm{AT}^{*}_{*} \leftarrow \epsilon$*, and control transcripts as* $(\mathrm{CT}^{*,*}_{i,*}, \mathrm{CT}^{*,*}_{j,*}) \leftarrow \epsilon$ *(indexed by* $\mathsf{id} \in \overrightarrow{\mathsf{G}}$*,* eid*). Then, for all sequences of operations* $\big((op^0, \mathsf{id}^0, \mathsf{ctrl}^0, \rho^0, m^0), \ldots, (op^n, \mathsf{id}^n, \mathsf{ctrl}^n, \rho^n, m^n)\big)$*, with* $0 \leq l \leq n$*, the following holds:* $op^l \in \{e, d\}$*,* $\mathsf{id}^l \in \{i, j\}$*, control actions* $\mathsf{ctrl}^l \in \{\bot, \mathsf{Add}, \mathsf{Rem}, \mathsf{Upd}, \mathsf{Commit}\}$*, role* $\rho^l \in \{\mathsf{a}, \mathsf{p}\}$*, and message* $m^l \in \{0,1\}^*$*. This* gACCE *correctness definition holds provided the protocol is executed precisely as follows.*

- *if* $op^l = e$*, invoke:* $\mathsf{Enc}(\mathsf{st}^{\rho^l}_{\mathsf{id}^l}, \mathsf{ctrl}^l, m^l) \xrightarrow{\$} (\mathsf{st}^{\rho^l}_{\mathsf{id}^l}, ctxt^l, \mathsf{eid}, \mathsf{mid})$ *and,*
  - *if* $\mathsf{ctrl}^l = \bot$*, update:* $\mathrm{AT}^{\mathrm{e}}_{\mathsf{id}^l,\mathsf{eid}} \leftarrow \mathrm{AT}^{\mathrm{e}}_{\mathsf{id}^l,\mathsf{eid}^l} \,\|\, (\mathsf{id}^l, \mathsf{mid}, m^l, ctxt^l)$
  - *else if* $\mathsf{ctrl}^l \in \{\mathsf{Add}, \mathsf{Rem}, \mathsf{Upd}\}$*:*
    - *if* $\rho_{\mathsf{id}^l} = \mathsf{a}$*, update:* $\mathrm{CT}^{\mathrm{c,e}}_{\mathsf{id}^l,\mathsf{eid}^l} \leftarrow \mathrm{CT}^{\mathrm{c,e}}_{\mathsf{id}^l,\mathsf{eid}^l} \,\|\, (\mathsf{eid}^l, \mathsf{ctrl}^l, m^l, ctxt^l)$
    - *if* $\rho_{\mathsf{id}^l} = \mathsf{p}$*, update:* $\mathrm{CT}^{\mathrm{o,e}}_{\mathsf{id}^l,\mathsf{eid}^l} \leftarrow \mathrm{CT}^{\mathrm{o,e}}_{\mathsf{id}^l,\mathsf{eid}^l} \,\|\, (\mathsf{eid}^l, \mathsf{ctrl}^l, m^l, ctxt^l)$
- ***OR****, if* $op^l = d$*, invoke:* $\mathsf{Dec}(\mathsf{st}^{\rho^l}_{\mathsf{id}^l}, ctxt^l) \xrightarrow{\$} (\mathsf{st}^{\rho^l}_{\mathsf{id}^l}, \mathsf{id}_*, \mathsf{ctrl}^l_*, m^l_*, \mathsf{eid}^l_*, \mathsf{mid}^l_*)$
  - *if* $\mathsf{ctrl}^l_* = \bot$*, update:*
$$\mathrm{AT}^{\mathrm{d}}_{\mathsf{id}_*,\mathsf{eid}^l_*}[\mathsf{id}_*] \leftarrow \mathrm{AT}^{\mathrm{d}}_{\mathsf{id}_*,\mathsf{eid}^l_*}[\mathsf{id}_*] \,\|\, (\mathsf{id}_*, \mathsf{mid}^l_*, m^l_*, ctxt^l)$$
  - *else if* $\mathsf{ctrl}^l_* \neq \bot$*, update:*
$$\mathrm{CT}^{\mathrm{c,d}}_{\mathsf{id}^l,\mathsf{eid}^l_*}[\mathsf{id}_*] \leftarrow \mathrm{CT}^{\mathrm{c,d}}_{\mathsf{id}^l,\mathsf{eid}^l_*}[\mathsf{id}_*] \,\|\, (\mathsf{eid}^l_*, \mathsf{ctrl}^l_*, m^l_*, ctxt^l)$$

If $m^l_* \neq \bot$*, then encrypted and decrypted messages, control messages, and epoch and message counters outputs equal* $m^l_* = m^l_o, \mathsf{ctrl}^l_* = \mathsf{ctrl}^l_o, \mathsf{eid}^l_* = \mathsf{eid}^l_o, \mathsf{mid}^l_* = \mathsf{mid}^l_o$*, that epoch counter outputs increase monotonically* $(\forall l^* < l$ *with* $op^l = op^{l^*} = e$ *and* $\rho^{l^*} = \rho^l$ *it holds that* $\mathsf{eid}^{l^*} \leq \mathsf{eid}^l_o)$*, and the following transcript invariants hold:*

1. $\mathrm{CT}^{\mathrm{c,d}}_{\mathsf{id},\mathsf{eid}}[\mathsf{id}'][1:] \subseteq \{\mathrm{CT}^{\mathrm{c,e}}_{\mathsf{id}',\mathsf{eid},\mathsf{mid}} \cup \mathrm{CT}^{\mathrm{o,e}}_{\mathsf{id}',\mathsf{eid}}\} \wedge$
2. $\mathrm{AT}^{\mathrm{d}}_{\mathsf{id},\mathsf{eid}}[\mathsf{id}'] \subseteq \mathrm{AT}^{\mathrm{e}}_{\mathsf{id}',\mathsf{eid}}$

The gACCE model analyzes QUIC's security properties in various epochs. fACCE captures certain security properties which are given values corresponding to a stage of the protocol that they are guaranteed thereafter: Authentication and integrity, key compromise impersonation (KCI) resistance, forward secrecy, resistance against replay attacks, and reveals against executions' random coins. MLS already provides guarantees for FS and replay attacks which we do not model for brevity. For gACCE, we add PCS as an additional security property denoted by the counter `pcs` to the ten presented in [41]. Similar to the `fs` counter, `pcs` defines the epoch from which post-compromise security (with respect to the group session) is reached. It should be hard, for an epoch $\mathsf{eid} \geq \mathtt{pcs}$, to break the confidentiality of ciphertexts, even if all group members were corrupted previously (*unless* one of the members' random coins were revealed to the adversary). We modify the freshness notion to incorporate PCS security in Figure 5.

$\underline{\text{Fresh}_{\mathtt{fs-pcs}}}$

1: **for** $(i,j) \in [n_p]$ **do**
2: **for** $(s,t) \in [n_s]$: **do**
3: $\mathtt{ctr} \leftarrow \min(\mathsf{eid}^* : \pi_i^s.fr_{\mathsf{eid}^*} = 1)$
4: **if** $corr_i = 1$ **then**
5: $\mathtt{ctr} \leftarrow \max(\mathtt{ctr}, \mathtt{fs})$
6: $\tau \leftarrow \pi_i^s.\mathsf{eid}$
7: $\pi_i^s.fr_{\mathsf{eid}^*} \leftarrow 0$ for all $\mathsf{eid}^* < \mathtt{ctr}$
8: **if** $(\mathsf{Upd} \in \{\pi_i^s.\mathrm{CT}^{\mathrm{c,e}}_{i,\tau'} \cup \pi_i^s.\mathrm{CT}^{\mathrm{o,e}}_{i,\tau'}\}) \wedge ((\pi_j^t.\mathrm{CT}^{\mathrm{c,d}}_{j,\tau'}[i] = \pi_i^s.\mathrm{CT}^{\mathrm{c,e}}_{i,\tau'})$
$\vee\ (\pi_j^t.\mathrm{CT}^{\mathrm{c,d}}_{j,\tau'}[i] = \pi_i^s.\mathrm{CT}^{\mathrm{o,e}}_{i,\tau'}))\ \forall\ \tau' > \tau$ **then**
9: $\pi_i^s.fr_{\mathsf{eid}^*}, \pi_j^t.fr_{\mathsf{eid}^*} \leftarrow 1$ for all $\mathsf{eid}^* \geq \tau'$
10: $\mathtt{pcs} \leftarrow \max(\mathtt{pcs}, \tau')$

Fig. 5: Freshness notion for fACCE with PCS modifications in blue. Freshness (fr) is regained after corruption through all parties reaching a new shared group state resulting from processing an update. We abuse notation on line 8 in the second clause to convey that the *commit* to π_i^s's update must appear in π_j^t's transcript (i.e. $(\mathsf{Commit}, \mathsf{Upd}) \in \mathrm{CT}^{\mathrm{c,d}}_{j,\tau+1}[i]$).

The execution environment of fACCE is also suitable to analyze QUIC. Particularly, it's worth noting that the Honest Partner definition can extend to the group channel formed by QUIC-MLS.

Definition 4 (Honest Group Partner (adapted from [41])). *π_j^t is an honest group partner of π_i^s if $(i,j) \in \overrightarrow{\mathsf{G}}$ where $j \neq i$ and $\overrightarrow{\mathsf{G}} = \{i, \ldots, n\}$ are group partners and all initial variables match (i.e. $\pi_i^s.\overrightarrow{\mathsf{G}}_{\mathsf{eid}} = \pi_j^t.\overrightarrow{\mathsf{G}}_{\mathsf{eid}}$, $\pi_i^s.\mathsf{st}.\gamma = \pi_j^t.\mathsf{st}.\gamma \neq \bot$, $\pi_i^s.\mathsf{eid} = \pi_j^t.\mathsf{eid}$ where $\rho_\iota = \mathsf{a}$ for some $\iota \in \overrightarrow{\mathsf{G}}$), and the received control transcript is a prefix of the partner's sent control transcript for their common epochs, respectively, where at least one of them is not empty (i.e. , for any epoch n s.t. $0 \leq n \leq m$ where m is the $\min(\pi_i^s.\mathsf{eid}, \pi_j^t.\mathsf{eid})$, where $a = \left|\pi_j^t.\mathrm{CT}^{\mathrm{c,d}}_{i,n}[j]\right|, b = \left|\pi_i^s.\mathrm{CT}^{\mathrm{c,d}}_{j,n}[i]\right|$ such that if $a > 0$ and $b > 0$ then $\forall\ 0 \leq \alpha < a$: $(\pi_i^s.\mathrm{CT}^{\mathrm{c,e}}_{i,n}[\alpha] = \pi_j^t.\mathrm{CT}^{\mathrm{c,d}}_{j,n}[i][\alpha])$ and $\forall 0 \leq \beta < b : (\pi_i^s.\mathrm{CT}^{\mathrm{c,d}}_{i,\mathsf{eid}}[j][\beta] = \pi_j^t.\mathrm{CT}^{\mathrm{o,e}}_{j,n}[\beta])$). If π_i^s already received ciphertexts from any π_j^t, then π_j^t is an honest group partner of π_i^s only if no other honest partner π^* exists outside the set $\pi^i_{s.G}$.*

Our honest group partner notion changes the honest partnering from fACCE in two major ways. First, gACCE allows for partnering to be a 1-to-many relationship such that there can be up to $n-1$ honest partners for a particular initiator after decrypting a ciphertext. This reflects the fact that any member of an MLS group can encrypt and decrypt application messages and group control messages. Second, as opposed to matching conversations across all message types, we only require matching control message sequences within CT between honest group partners to enforce group state agreement.

6.2 Adversarial Model

We adopt the adversarial model of fACCE in [41] for our security experiment with modifications for the group setting. Like fACCE, an adversary $\mathcal{A}$ plays a security game with a challenger in which they attempt break the confidentiality (i.e. via guessing a sampled bit b) or integrity (i.e. via forgery) of messages in the gACCE protocol. At the beginning of the experiment, the challenger runs KGen to generate keys for all participants of the protocol and gives $\mathcal{A}$ the list of their public keys. $\mathcal{A}$ is given oracle access to OInit, OEnc, ODec, OReveal, and OCorrupt and can control communications between participants. One way the $\mathcal{A}$ can win the security game by is by triggering a $\texttt{win} \leftarrow 1$ (indicating breaking the authenticity or integrity of ciphertexts). Otherwise, at the end of the experiment $\mathcal{A}$ terminates and outputs a bit b' against the challenge bit $\pi_i^s.b_{\mathsf{eid}}$. The challenger checks the freshness conditions and if $\mathcal{A}$ has not violated any of the freshness conditions outputs $(b' = \pi_i^s.b_{\mathsf{eid}}) \vee \texttt{win}$. We detail the modifications to the relevant oracles below:

- $\mathsf{OInit}(i, s, \overrightarrow{\mathsf{G}}, \rho)$ initializes a session π_i^s (if not yet initialized) of party i to be in a group with parties in the list $\overrightarrow{\mathsf{G}} = (i, \ldots, n)$, invoking $\mathsf{gACCE.KGen}() \xrightarrow{\$} (\mathrm{sk}_n, \mathrm{pk}_n)$ for each party in $\overrightarrow{\mathsf{G}}$ (if not yet generated) and then $\mathsf{gACCE.Init}(sk_i, \overrightarrow{\mathsf{PK}}_{\overrightarrow{\mathsf{G}}}, \rho, ad) \xrightarrow{\$} \pi_i^s.st$ under $\$ \leftarrow \pi_i^s.rand$. It also sets $\pi_i^s.\rho \leftarrow \rho$ and $\pi_i^s.\overrightarrow{\mathsf{G}} \leftarrow \overrightarrow{\mathsf{G}}$. This oracle provides no return value. All subsequent invocations of Enc, Dec of this session participant use $\pi_i^s.rand$ for obtaining randomness. Finally freshness flags are updated by invoking $\mathrm{Fresh}_{\texttt{fs-pcs}}$ (see Figure 5).
- $\mathsf{OEnc}(i, s, \mathsf{ctrl}, m_0, m_1; \pi_i^s.rand)$ Upon satisfaction of freshness conditions, this triggers either the encryption of a message or a group action proposal according to the given ctrl instruction for the encryption of message m_b for $b = \pi_i^s.b_{\mathsf{eid}}$ by invoking $\mathsf{gACCE.Enc}(\pi_i^s.st, \mathsf{ctrl}, m_b) \rightarrow (st', c, \mathsf{eid}, \mathsf{mid})$ for an initialized π_i^s. It returns $(c, \mathsf{eid}, \mathsf{mid})$ to the adversary. If $m_0 \neq m_1$, the challenger checks the freshness conditions (given by Figure 5) and outputs $\perp$ if the $\mathcal{A}$ issued this query and $\pi_i^s.fr_{\mathsf{eid}} = 0$. Otherwise, if encryption succeeds (i.e. $c \neq \perp$), this oracle updates the session specific variables $\pi_i^s.st \leftarrow st'$, and appends the private results to the challenger held encryption CT and AT in π_i^s according to Definition 3.

– $\mathsf{ODec}(i, s, c; \pi_i^s.rand)$ triggers invocation of $\mathsf{gACCE.Dec}(\pi_i^s.st, c) \rightarrow (st', m, \mathsf{ctrl}, \mathsf{id}, \mathsf{eid}, \mathsf{mid})$ for initialized π_i^s and returns $(m, \mathsf{eid}, \mathsf{mid})$ to the adversary. If c was not generated by an honest partner and the epoch was fresh according to Definition 4 and Figure 5, respectively, then $\mathtt{win} = 1$ and the challenger ends the game. Otherwise, if decryption was successful (i.e. $m \neq \perp$), then it appends the results to the decryption CT and AT in π_i^s according to Definition 3.

We define the the (insecurity) of gACCE below:

Definition 5 (Breaking gACCE Security). *Let Π be a flexible group ACCE protocol $\Pi = \{\mathsf{KGen}, \mathsf{Init}, \mathsf{Enc}, \mathsf{Dec}\}$. Let $\mathsf{Exp}^{\mathsf{gACCE}}_{\Pi,\mathcal{A}}(n)$ be the gACCE security experiment described in Section 6.2 in which an adversary $\mathcal{A}$ breaks the security of gACCE security when $\mathcal{A}$ terminates and outputs (i, s, eid, b'), if there either exists a session π_i^s such that $\pi_i^s.b_{\mathsf{eid}} = b'$ and $\pi_i^s.fr_{\mathsf{eid}} = 1$, or $\mathtt{win} = 1$. We define the advantage of n adversary $\mathcal{A}$that breaks a group flexible ACCE protocol gACCE as: $\mathsf{Adv}^{\mathsf{gACCE}}_{\Pi,\mathcal{A}}(n) = |2 * \mathsf{Pr}(\mathsf{Exp}^{\mathsf{gACCE}}_{\Pi,\mathcal{A}}(n) = 1) - 1| + \mathsf{Pr}[\mathtt{win} = 1]$.*

6.3 Security Proof Outline

Broadly speaking, our construction is simply a composition of a CGKA protocol, Authenticated Encryption (AuthEnc) protocol, and some book-keeping functionalities. There is no change to either protocols nor their primitives. Thus, the game-hopping based proof is split into two cases which are structurally similar:

Case 1 The adversary $\mathcal{A}$ attempts to forge a message. That is, $\mathcal{A}$ succeeds in getting a party to decrypt a ciphertext c for which no honest partner produced.

Case 2 The adversary $\mathcal{A}$ guesses the correct bit b with non-negligible probability.

a) Case 1: We proceed in a series of game-hops as follows:

Game 0 (G0) This is the gACCE experiment as described in section 6.2 with no modifications.

Game 1 (G1) This version is the same as G0 except that we replace all of the keys used for encryption or decryption during an epoch marked fresh by in the gACCE experiment (i.e. $\mathcal{A}$has not triggered a trivial attack) with uniformly random values from a CGKA challenger $\mathcal{C}_{\mathsf{CGKA}}$. This is sound under the standard hybrid argument since CGKA provides uniformly random keys for fresh epochs. Different CGKA challengers are extended, without loss of generality (WLOG), to any of group sessions allowed under the gACCE experiment. For each group session, a CGKA reduction is made as follows. Freshness in the gACCE game then corresponds to freshness in the CGKA game. When $\mathcal{A}$ issues a OReveal query, the challenger first checks the freshness flag of the epoch and if $fr = 1$, it returns a uniformly random value from $\mathcal{C}_{\mathsf{CGKA}}$, otherwise it returns the real key. When $\mathcal{A}$ issues a OCorrupt query against a party for a given epoch, if the epoch in question is not fresh then the real long-term keys of party are exposed, otherwise the query is not allowed. Future corruptions in new epochs against that party are allowed until pcs flips fr back to 1.

Game 2 (G2) This game is the same as G1 except that we replace all instances of encryption and decryption in OEnc and ODec with ones provided by an $\mathsf{AuthEnc}$ challenger $\mathcal{B}_{\mathsf{AuthEnc}}$. WLOG, this requires one fresh $\mathsf{AuthEnc}$ reduction per epoch per group session (each group has distinct epochs and each epoch has new keys). In each instance, the adversary's advantage in forgery reduces to the $\mathsf{AuthEnc.auth}$ security of the $\mathsf{AuthEnc}$ scheme.

b) Case 2: This proof follows the series of game hops exactly as in Case 1 except that in G2, the adversary's advantage in guessing bit b reduces the advantage of an adversary guess bit b in the the $\mathsf{AuthEnc.conf}$ game.

Theorem 1 (gACCE Security for QUIC-MLS). *Let n_s, n_G, and n_e be the number of sessions, number of groups, and number of epochs given by some security parameter n used in the experiment. The QUIC-MLS protocol presented in Figure 2 is gACCE-secure. That is, for any PPT algorithm $\mathcal{A}$ against the gACCE security experiment (described in Section 6.2), the $\mathsf{Adv}^{\mathsf{gACCE}}_{\Pi,\mathcal{A}}(n)$ is negligible under the cgka security of CGKA and the auth and conf security of the $\mathsf{AuthEnc}$ primitives.*

7 Conclusion

This work introduces the first formalization and analysis of QUIC-MLS, an idea initially proposed for space networking security. We demonstrate feasibility of the concept and provide a cryptographic security proof. With growing needs for low-latency and efficiency, as well as security, this points a way forward for using novel key exchange as a replacement for legacy handshake approaches within networking protocols.

Acknowledgements. The research of B. Dowling was supported by EPSRC grant EP/X016226/2. The research of B. Dowling and B. Wimalasiri was supported by UKRI grant UKRI156.

References

1. U. N. O. for Outer Space Affairs (UNOOSA), "Online Index of Objects Launched into Outer Space," 2025. Retrieved from `https://www.unoosa.org/oosa/osoindex/search-ng.jspx`.
2. E. Vyncke, "Taking IP To Other Planets Charter," 2025. Retrieved from `https://datatracker.ietf.org/doc/charter-ietf-tiptop/`.
3. J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport." RFC 9000, May 2021.
4. M. Blanchet, W. Eddy, and T. Li, "An Architecture for IP in Deep Space," Internet-Draft draft-many-deepspace-ip-architecture-01, Internet Engineering Task Force, Nov. 2024. Work in Progress.
5. J. Smailes, R. David, S. Kohler, S. Birnbach, and I. Martinovic, "Poster: spacequic: Securing communication in computationally constrained spacecraft," 2023.

6. B. Dowling, B. Hale, X. Tian, and B. Wimalasiri, "Key establishment in the space environment," 2025.
7. M. Sommer, A. Sterz, M. Vogelbacher, H. Bellafkir, and B. Freisleben, "QUICL: A QUIC Convergence Layer for Disruption-tolerant Networks," in *Proceedings of the Int'l ACM Conference on Modeling Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '23, p. 37–46, Association for Computing Machinery, 2023.
8. G. Amponis, T. Lagkas, V. Argyriou, P. Radoglou-Grammatikis, K. Kyranou, I. Makris, and P. Sarigiannidis, "Channel-Aware QUIC Control for Enhanced CAM Communications in C-V2X Deployments Over Aerial Base Stations," *IEEE Transactions on Vehicular Technology*, vol. 73, no. 7, pp. 9320–9333, 2024.
9. Sandvine, "The Global Internet Phenomena Report," Jan. 2023.
10. N. Aviram, K. Gellert, and T. Jager, "Session Resumption Protocols and Efficient Forward Security for TLS 1.3 0-RTT," *J. Cryptol.*, vol. 34, July 2021.
11. F. Günther, B. Hale, T. Jager, and S. Lauer, "0-rtt key exchange with full forward secrecy," in *Advances in Cryptology – EUROCRYPT 2017*, pp. 519–548, Springer International Publishing, 2017.
12. K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, "A Formal Security Analysis of the Signal Messaging Protocol," *Journal of Cryptology*, vol. 33, pp. 1914–1983, 2020.
13. K. Cohn-Gordon, C. Cremers, and L. Garratt, "On Post-Compromise Security," in *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pp. 164–178, IEEE, 2016.
14. R. Barnes, B. Beurdouche, R. Robert, J. Millican, E. Omara, and K. Cohn-Gordon, "The Messaging Layer Security (MLS) Protocol." RFC 9420, July 2023.
15. M. Anastos, B. Auerbach, M. A. Baig, M. C. Noval, M. Kwan, G. Pascual-Perez, and K. Pietrzak, "The Cost of Maintaining Keys in Dynamic Groups with Applications to Multicast Encryption and Group Messaging," in *Theory of Cryptography: 22nd International Conference, TCC 2024*, p. 413–443, Springer-Verlag, 2024.
16. J. Alwen, S. Coretti, D. Jost, and M. Mularczyk, "Continuous Group Key Agreement with Active Security," in *Theory of Cryptography: 18th International Conference, TCC 2020*, (Berlin, Heidelberg), p. 261–290, Springer-Verlag, 2020.
17. E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3." RFC 8446, Aug. 2018.
18. M. Thomson and S. Turner, "Using TLS to Secure QUIC." RFC 9001, May 2021.
19. C. Brzuska, E. Cornelissen, and K. Kohbrok, "Security analysis of the MLS key derivation," in *2022 IEEE Symposium on Security and Privacy*, pp. 2535–2553, IEEE Computer Society Press, May 2022.
20. C. Cremers, B. Hale, and K. Kohbrok, "The Complexities of Healing in Secure Group Messaging: Why Cross-Group Effects Matter," pp. 1847–1864, 2021.
21. T. Wallez, J. Protzenko, and K. Bhargavan, "TreeKEM: A Modular Machine-Checked Symbolic Security Analysis of Group Key Agreement in Messaging Layer Security," 2025. Publication info: Preprint.
22. K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, "A Formal Security Analysis of the Signal Messaging Protocol," in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 451–466, 2017.
23. B. Dowling and B. Hale, "Secure Messaging Authentication against Active Man-in-the-Middle Attacks," in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 54–70, 2021.
24. R. Fiedler and F. Günther, "Security Analysis of Signal's PQXDH Handshake," in *Public-Key Cryptography – PKC 2025*, p. 137–169, Springer-Verlag, 2025.

25. A. Bienstock, J. Fairoze, S. Garg, P. Mukherjee, and S. Raghuraman, "A More Complete Analysis of the Signal Double Ratchet Algorithm," in *Advances in Cryptology – CRYPTO 2022*, p. 784–813, Springer-Verlag, 2022.
26. B. Dowling and B. Hale, "Authenticated Continuous Key Agreement: Active MitM Detection and Prevention." Cryptology ePrint Archive, Paper 2023/228, 2023.
27. M. Fischlin, F. Günther, and C. Janson, "Robust Channels: Handling Unreliable Networks in the Record Layers of QUIC and DTLS 1.3," *J. Cryptol.*, vol. 37, Jan. 2024.
28. A. Delignat-Lavaud, C. Fournet, B. Parno, J. Protzenko, T. Ramananandro, J. Bosamiya, J. Lallemand, I. Rakotonirina, and Y. Zhou, "A Security Model and Fully Verified Implementation for the IETF QUIC Record Layer," in *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 1162–1178, 2021.
29. S. Chen, S. Jero, M. Jagielski, A. Boldyreva, and C. Nita-Rotaru, "Secure Communication Channel Establishment: TLS 1.3 (over TCP Fast Open) versus QUIC," *J. Cryptol.*, vol. 34, July 2021.
30. R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru, "How Secure and Quick is QUIC? Provable Security and Performance Analyses," in *2015 IEEE Symposium on Security and Privacy*, pp. 214–231, 2015.
31. B. Beurdouche, E. Rescorla, E. Omara, S. Inguva, and A. Duric, "The Messaging Layer Security (MLS) Architecture." RFC 9750, Apr. 2025.
32. R. Robert, "The Messaging Layer Security (MLS) Extensions," Internet-Draft draft-ietf-mls-extensions-07, Internet Engineering Task Force, July 2025. Work in Progress.
33. Marten Seemann, "quic-go," 2025. `https://pkg.go.dev/github.com/quic-go/quic-go`.
34. Cisco Systems, "go-mls," 2021. `https://pkg.go.dev/github.com/cisco/go-mls`.
35. Google, "crypto/tls," 2025. `https://pkg.go.dev/crypto/tls`.
36. T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, "On the Security of TLS-DHE in the Standard Model," in *Annual Cryptology Conference*, pp. 273–293, Springer, 2012.
37. T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, "On the security of TLS-DHE in the standard model," in *CRYPTO 2012* (R. Safavi-Naini and R. Canetti, eds.), vol. 7417 of *LNCS*, pp. 273–293, Springer, Berlin, Heidelberg, Aug. 2012.
38. R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru, "How secure and quick is quic? provable security and performance analyses," in *Proceedings of the 16th Annual Information Security Symposium*, CERIAS '15, (West Lafayette, IN), CERIAS - Purdue University, 2015.
39. M. Fischlin and F. Günther, "Multi-stage key exchange and the case of Google's QUIC protocol," in *ACM CCS 2014* (G.-J. Ahn, M. Yung, and N. Li, eds.), pp. 1193–1204, ACM Press, Nov. 2014.
40. M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *CRYPTO'93* (D. R. Stinson, ed.), vol. 773 of *LNCS*, pp. 232–249, Springer, Berlin, Heidelberg, Aug. 1994.
41. B. Dowling, P. Rösler, and J. Schwenk, "Flexible Authenticated and Confidential Channel Establishment (fACCE): Analyzing the Noise Protocol Framework," in *Public-Key Cryptography – PKC 2020*, p. 341–373, Springer-Verlag, 2020.

A QUIC-MLS Protocol Construction

From a high level overview, QUIC-MLS establishes a secure group channel among two or more parties which uses the output of the MLS continuous key agreement component as the input to the QUIC record layer used to send encrypted application layer messages. We detail our assumptions and intuition on different components of the protocol below:

- *Operational Settings* We assume that there is no centralized strongly consistent DS (e.g. server style DS). Rather, we rely on QUIC transport mechanisms to provide reliable in-order delivery of all messages. Furthermore, we assume that QUIC-MLS participants already have the certificate information (i.e. public keys and associated identities) of all other participants as provided by some AS implementation (e.g. PKI from CAs, out-of-band, preinstalled, etc.).
- *Initialization* The initiator of a session may choose one or more participants to form a group. The initiator uses the public keys of the group member(s) to start an MLS session by unilaterally committing their own add proposal and sends individualized Welcome messages out (via their public keys).
- *Group Control* Because we assume there is no DS to arbitrate valid proposals simultaneously sent by multiple group members, we assign a leader to each epoch that is allowed to commit to group control proposals to ensure no conflicts can arise. Leader assignment can be time based, schedule based, or arbitrarily selected to fit a use-case. For example, epoch leaders can be chosen based on a pre-assigned schedule upon group initialization which is exported (i.e. in the `Group Context`) and updated with any group control action. Or, perhaps more elegantly, epoch leaders can be assigned based on their position in the ratchet tree (leftmost to right).
- *Encryption and Decryption* We make no changes to the QUIC AEAD Encryption or Decryption as previously discussed in section 4.3. However, if non-repudiation between group members is desired for application messages, then the use of a sign-then-encrypt procedure can be adapted to messages input to QUIC's AEAD.

Term	Description
$\mathrm{AT}^{\mathrm{x}}_{\mathsf{id},\mathsf{eid}}[\mathsf{id}']$	The application message transcript used in the gACCE security model where $x \in (e, d)$. Note, $\mathrm{AT}^{\mathrm{e}}_{\mathsf{id},\mathsf{eid}}$ does not maintain separate vectors for partners (i.e. $\mathrm{AT}^{\mathrm{e}}_{\mathsf{id},\mathsf{eid}} = \mathrm{AT}^{\mathrm{e}}_{\mathsf{id},\mathsf{eid}}[\mathsf{id}]$).
$\mathrm{CT}^{\mathrm{a,b}}_{\mathsf{id},\mathsf{eid}}[\mathsf{id}']$	The group control message transcript used in the gACCE security model where $a \in \{c, o\}$ signifies confirmed or opportunistic, $b \in \{e, d\}$ refers to encrypted or decrypted, id refers to the transcript owner and id′ refers to a partner id, and eid refers to the epoch number. Similarly, id does not maintain encryption transcripts of partners (i.e. $\mathrm{CT}^{\cdot,\mathrm{e}}_{\mathsf{id},\mathsf{eid}} = \mathrm{CT}^{\cdot,\mathrm{e}}_{\mathsf{id},\mathsf{eid}}[\mathsf{id}]$).
st	The gACCE state stored in memory of each group member which includes the following vectors: γ - The CGKA group state as defined in 1 ρ - The role of the st owner which can take the values of $\{\mathrm{a}, \mathrm{p}\}$ for admin or participant. Only admin can commit to proposals sent by other group members. ast - The administrative state which holds a transcript of commits and proposals, and (optional) welcome messages leading to the current epoch formatted as: $[\mathrm{commit}, \overrightarrow{\mathrm{p}}, \mathrm{welc}]$ or a proposed epoch (where $\mathrm{commit} = \perp$). eid - The epoch number associated with a particular id: $\mathsf{st.eid}$ is shorthand for $\mathsf{st.eid}_{\mathsf{st.id}}$. gid - The group identifier of the current session id - The ID of the st owner kst - The set of keys stored by the st owner indexed by eid. $\mathsf{mid}_{\mathsf{eid}}$ - The message transcript indexed by eid $\overrightarrow{\mathsf{G}}_{\mathsf{eid}}$ - The IDs of group members in a particular epoch

Table 3: Summary of variable terminology