USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

Exploring How to Authenticate Application Messages in MLS: More Efficient, Post-Quantum, and Anonymous Blocklistable

Keitaro Hashimoto, *National Institute of Advanced Industrial Science and Technology* (*AIST*); Shuichi Katsumata, *National Institute of Advanced Industrial Science and Technology* (*AIST*) *and PQShield;* Guillermo Pascual-Perez, *Institute of Science and Technology Austria* (*ISTA*)

https://www.usenix.org/conference/usenixsecurity25/presentation/hashimoto-blocklistable

This paper is included in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

Exploring How to Authenticate Application Messages in MLS: More Efficient, Post-Quantum, and Anonymous Blocklistable

Keitaro Hashimoto
National Institute of Advanced Industrial Science and Technology (AIST)

Shuichi Katsumata
AIST & PQShield

Guillermo Pascual-Perez
Institute of Science and Technology Austria (ISTA)

Abstract

The Message Layer Security (MLS) protocol has recently been standardized by the IETF. MLS is a scalable secure group messaging protocol expected to run more efficiently compared to the Signal protocol at scale, while offering a similar level of strong security. Even though MLS has undergone extensive examination by researchers, the majority of the works have focused on *confidentiality*.

In this work, we focus on the *authenticity* of the *application messages* exchanged in MLS. Currently, MLS authenticates every application message with an EdDSA signature and while manageable, the overhead is greatly amplified in the post-quantum setting as the NIST-recommended Dilithium signature results in a 40x increase in size. We view this as an invitation to explore new authentication modes that can be used instead. We start by taking a systematic view on how application messages are authenticated in MLS and categorize authenticity into four different security notions. We then propose several authentication modes, offering a range of different efficiency and security profiles. For instance, in one of our modes, COSMOS^{++}, we replace signatures with one-time tokens and a MAC tag, offering roughly a 75x savings in the post-quantum communication overhead. While this comes at the cost of weakening security compared to the authentication mode used by MLS, the lower communication overhead seems to make it a worthwhile trade-off with security.

1 Introduction

1.1 Background

A *secure group messaging* (SGM) protocol allows a group of users to asynchronously communicate in an end-to-end encrypted fashion. The *Messaging Layer Security* (MLS) protocol [7, 13], a recently standardized SGM protocol by the IETF, is a proposal developed in a joint effort by academics and industry for a *scalable* SGM protocol supporting groups with tens of thousands of users. Similarly to the Signal protocol [22, 23, 31], considered the gold standard for two-user SGMs, it offers a strong level of forward secrecy and post-compromise security, limiting the scope of device compromise. The draft versions of MLS are already running in production in Cisco's Webex [46] and RingCentral [54], and other companies, including AWS, Cloudflare, and Google, are planing deployment.[1] Furthermore, with the recent adoption of the Digital Markets Act by the European Union, a standard like MLS is hoped to be a potential solution for the interoperability problem in secure messaging [42].

The security of MLS (and its variants) has undergone extensive examination by researchers during the standardization process, e.g., [2–6, 14, 17, 35, 36, 40, 57], and the protocol has been continuously updated leading up to 20 drafts in total[2] until the issuance of the RFC. The majority of works on MLS have focused on the *confidentiality* of the exchanged messages (or the shared group secret key). In contrast, relatively less attention has been directed towards the *authenticity* of messages, which is often viewed as a means to establish confidentiality.

In MLS, there are two types of messages being authenticated [7, Sec. 2]: *application* and *handshake* messages. While the former carry the actual payloads such as chat texts, the latter carry group operations affecting the group state (e.g., authenticating that user u added a new user v to the group). In this work, we revisit how MLS authenticates application messages motivated by the following two issues.

Issue 1: Heavy Reliance on Signatures. In MLS, every user u has a signature key pair $(\mathsf{vk}_u, \mathsf{sk}_u)$ and signs the application message am for authentication. It further independently encrypts the message am and signature sig_u using a symmetric key encryption scheme whose key is derived from the group secret key to conceal the application message and its identity from the delivery server. The resulting (tuple of) ciphertext ct_u is then sent to the group. We call this mode of authentication *Enc-Sign mode*.[3] The recent work by Hashimoto et

[1] https://www.ietf.org/blog/mls-protocol-published/

[2] https://datatracker.ietf.org/doc/rfc9420/

[3] In contrast, handshake messages can be sent in *Sign mode*, where the user simply sends the pair $(\mathsf{m}, \mathsf{sig}_u)$.

al. [36] proposed adding an additional signature $\overline{\mathsf{sig}}_G$ on top of ct_u, using a signing key derived from the group secret key. This mode, called the *Sign-Enc-Sign mode*, is a simple but powerful enhancement of the Enc-Sign mode, allowing to anonymously block outsiders from injecting malicious messages to the group, similarly to Signal's two-user Sealed Senders [43].

While adding signatures provides stronger authenticity guarantees, it comes with an increase in the communication and computational costs. This is currently manageable as MLS uses an EdDSA signature with an overhead of 64 B. However, this overhead is greatly amplified in the post-quantum setting. For instance, the NIST-recommended Dilithium signature is 2.4 KB, a 40x increase to EdDSA signatures. Given that a typical application message contains less than 100 B [32], the overhead has a noticeable effect. We thus view this as an invitation to explore alternatives designs. We note that while handshake messages incur the same overhead when turning to post-quantum security, the effect is marginal as the size of the handshake message is larger, and the rate at which group operations are performed is less frequent compared to sending application messages.

Issue 2: Lack of Formal Model for Authentication. Compared to the comprehensive study of the confidentiality guarantees of MLS, authentication has drawn less attention. This lack of focus on authenticity may lead to unforeseen attacks on MLS that do not contradict confidentiality but still harm the protocol. As an illustrative example, the MLS is prone to abuse from malicious insiders (e.g., [6]). Notice that both Enc-Sign and Sign-Enc-Sign modes conceal the sender from the server. This allows a malicious insider to craft a malformed message and send it to the group. If the signature sig_u included in the ciphertext is malformed, even the group users cannot trace back the sender, meaning that a malicious sender can stealthily repeat the attack. While the users can reject these malformed messages, this can only happen *after* downloading them from the server and processing them. This opens the door for a malicious insider to mount a DoS attack on the group. A similar issue was pointed out by Tyagi et al. [55] for Signal's two-user Sealed Senders [43], who experimentally verified that such an attack can easily drain a recipient's battery in a short period of time.

A formal security model that comprehensively captures these properties allows us to better understand the strengths and limitations of a given authentication mode.

1.2 Our Contributions

In this work, we explore new approaches to authenticate application messages in MLS. Our contribution is explained below in more detail and an overview is provided in Tab. 1.

Formal Model for Authentication. In Sec. 2, we study how application messages are authenticated in MLS and systematically analyze the types of adversaries and threat models

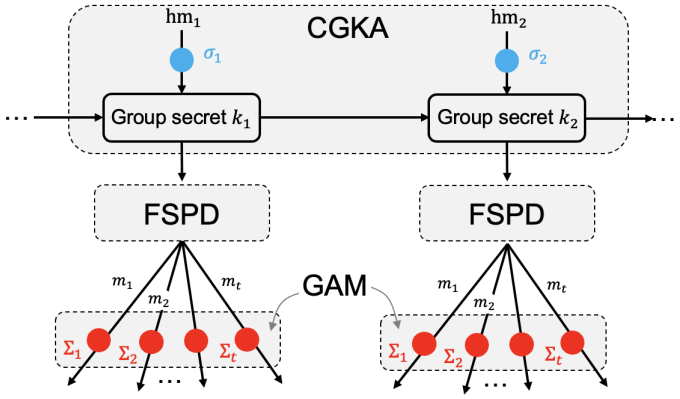

Figure 1: Relation between a CGKA, FSPD, and GAM protocols. hm denotes the handshake message used by the CGKA protocol. m_i denotes the output of the FSPD protocol; in MLS this is an *encryption* of the application message am. The blue and red circles indicate that the handshake and application messages are authenticated.

needed to be considered. More technically, the core of MLS can be regarded as a combination of two protocols: a *continuous group key agreement* (CGKA) and a *forward-secure payload delivery* (FSPD) protocol [3].[4] The former (resp. latter) handles handshake (resp. application) messages. In this paper, we formalize the authentication guarantees of the application messages handled by the FSPD protocol and introduce four different security notions: unforgeability, anonymity, anonymous blocklisting, and tracing soundness. To the best of our knowledge, this is the first work to put a focus on the authenticity of the application message; previous works on MLS studied the different types of CGKA protocol and focused on the confidentiality of the application message [2–6, 14, 17, 35, 36, 40, 57].

Group Authenticated Messaging Protocol. In Sec. 3, we propose the new notion of *group authenticated messaging* (GAM) protocol, allowing us to focus solely on the authenticity of the application messages while abstracting the confidentiality guarantees. More specifically, the FSPD protocol already entails the confidentiality of application messages and our GAM protocol can be viewed as adding authenticity guarantees to them. Fig. 1 gives an illustration on how a GAM protocol interacts with the CGKA and FSPD protocols. For instance, in MLS, m and Σ_1 are the encryptions of the application message am and signature sig_u on am, respectively (i.e., Enc-Sign mode).

New Authentication Modes. In Secs. 4 and 5, we introduce five new GAM protocols: COSMOS, COSMAC, QUASAR, STARS, and GEMSTARS. All are based on generic building blocks such as one-way functions (OWFs), message authentication codes (MACs), and key encapsulation mechanisms (KEMs) that are instantiable from both classical and post-quantum assump-

[4] Alwen et al. [3] uses the term forward-secure group AEAD instead of FSPD.

Table 1: Comparison between different authentication modes for secure messaging protocols. N and T denote the size of the group and the number of messages each user sends. "Communication Cost Overhead per Msg per User" is defined as the sum of 1 (offline/online) upload cost and $(N-1)$ (offline/online) download cost for each user normalized by NT. For readability, we use the simplification $(N+1)/N \approx 1$. sig, osig, and gsig denote a standard signature, a one-time signature, and a group signature, respectively. ovk denotes the verification key of a one-time signature. ct denotes a KEM ciphertext. κ denotes the security parameter, set to 128 bits. ✔$^{(*)}$ denotes that it satisfies a weaker notion of unforgeability compared to ✔ (see Sec. 2.3). "State Updates" comes with "-", "local", and "global", where "-" means no state update is necessary (see Remark 3.2). COSMOS and COSMAC come with an optimized variants indicated by $(^{+})$ and $(^{++})$, whose respective total communication cost overheads and state updates are provided in parentheses.

Authentication Modes	Anon.	Unf.	Anonymous Blocklistable	Tracing Soundness	Comm. Cost Overhead per Msg per User	State Updates
Enc-Sign [7]	✔	✔	✘	✘	$\lvert\mathsf{sig}\rvert$	-
Sign-Enc-Sign [36]	✔	✔	✔	✘	$2\cdot\lvert\mathsf{sig}\rvert$	-
COSMOS$(^{+},^{++})$ (Secs. 4)	✘	✔$^{(*)}$	✔	✔	$3\cdot\kappa$ $(3\cdot\kappa\,,\,(2+\frac{3}{T})\cdot\kappa)$	local (- , local)
COSMAC$(^{+},^{++})$ (Secs. 4)	✔	✔$^{(*)}$	✔	✘	$4\cdot\kappa$ $(4\cdot\kappa\,,\,(3+\frac{4}{T})\cdot\kappa)$	local (-, local)
QUASAR (Sec. 5.1)	✔	✔$^{(*)}$	✔	✔	$6\cdot\kappa+\frac{2\cdot(\kappa+\lvert\mathsf{ct}\rvert)}{T}$	global
STARS (Sec. 5.2)	✔	✔	✔	✔	$\lvert\mathsf{ovk}\rvert+2\cdot\lvert\mathsf{osig}\rvert+\frac{\kappa+2\cdot\lvert\mathsf{ct}\rvert}{T}$	global
GEMSTARS (Sec. 5.2)	✔	✔	✔	✔	$\lvert\mathsf{sig}\rvert+\lvert\mathsf{gsig}\rvert$	-

tions. Each mode fills a specific part of the design space with strengths and weaknesses, summarized in Tab. 1. In particular, COSMOS and COSMAC do not rely on signatures and the overhead (for their optimized variants) is merely 32 B and 48 B, respectively. This offers a roughly 75x savings in the post-quantum communication overhead compared to MLS, though at the cost of slightly weakening the unforgeability guarantee; we assume the malicious server does not collude with the malicious insider. See Sec. 2.3 for more detail. We believe this significantly lower communication overhead makes it a worthwhile trade-off with security.

Efficiency Analysis. In Sec. 7, we instantiate our proposed GAM protocols from both classical and post-quantum assumptions and compare their efficiency. For completeness, we also detail in Sec. 6 how to use each of our proposed GAM protocols inside MLS. While it is mostly a simple drop in, there are minor issues that require some explanation, since the syntax of GAM protocols intentionally leaves out some functionality provided by MLS, such as what is typically captured by the CGKA protocol (e.g., welcoming group members).

Lastly, we leave it as an important future work to analyze MLS in its entirety when using our notion of GAM protocol as a building block. While Alwen et al. [3] analyze MLS by composing the CGKA and FSPD protocols with a PRF-PRNG, the output of the FSPD protocol is explicitly signed (and not encrypted); that is, they assume the vanilla (unencrypted) GAM protocol used by MLS. Replacing this with a general GAM protocol and analyzing MLS is an interesting future work. We discuss further open problems in Sec. 8.

Other related work and preliminaries are deferred to the full version of this paper.

2 Setting: Authentication in SGM

This work focuses on secure group messaging (SGM) protocols, where group users share a unique common group secret key. In this section we use MLS as our primary example, but all of our constructions apply equally to most MLS variants (e.g., [1, 2, 4, 5, 35, 36, 40]) that rely on a group secret key to exchange messages.

Below, we give a brief background on how MLS authenticates application messages. We then take a close look at different security notions under the umbrella of authenticity and formally categorize them. Building on the systematization provided in this section, we introduce the concept of a *group authenticated messaging* (GAM) protocol in Sec. 3 and formally define the relevant security notions.

2.1 Secure Group Messaging and Our Goal

Following Alwen et al. [3], we view MLS as a combination of the CGKA and FSPD protocols (see Fig. 1 for illustration). At a high level, one can draw a parallel to hybrid encryption, where the heavy public key operations are handled by the CGKA protocol and the exchange of application messages is handled by the lightweight FSPD protocol.

In more detail, the CGKA protocol allows a group of users to agree on a continuous sequence of shared (symmetric) group secret keys. By regularly updating the group secret key (and user specific keys), strong notions of *forward secrecy* and *post-compromise security* [2–4, 6, 24] are guaranteed. The protocol is also responsible for handling group operations such as adding and removing users. *Handshake messages* is an umbrella term for the exchanged messages by the CGKA protocol, used to achieve the above objectives. A handshake

message, or an encryption of it, is signed using the user's signing key to authenticate the sender. This plays an important role in guaranteeing the consistency and integrity of the group state. As can also be seen from Fig. 1, the authenticity of handshake messages is analyzed implicitly as a means to show confidentiality of the group secret keys; this is similar to standard (two-user) authenticated key exchange protocols where authenticity guarantees are implicit [10, 19].

The FSPD protocol then uses the established group secret key by the CGKA protocol to securely exchange *application messages*, containing various types of payload such as chat texts, images, and stamps. Compared to the CGKA protocol, the FSPD protocol is much simpler since there are no group operations (i.e., static groups) and the objective is only confidentiality with forward secrecy; authenticity is not a security requirement. In MLS, the output of the FSPD protocol — an encryption of the application message — is then signed using a signature scheme and encrypted (i.e., Enc-Sign mode), adding the necessary authenticity guarantee. This rather ad hoc way of adding authenticity seems to be justified by the simple nature of the FSPD protocol, and indeed most works on MLS mainly focus on the security of the CGKA protocol [2,4–6,14,17,35,36,40,57]. To the best of our knowledge, Alwen et al. [3] is the only prior work to analyze MLS in its entirety. They do so by modularly combining the CGKA and FSPD protocols with a PRF-PRNG, assuming the output of the FSPD protocol is authenticated by a digital signature.

The goal of our paper is thus to put a spotlight onto the authentication of application messages or, to be more precise, the output of the FSPD protocol (see Fig. 1). We introduce a new primitive called group authenticated messaging (GAM) protocol and aim to more clearly and systematically explore alternative choices to the currently (implicitly) used GAM protocol by MLS, which is the Enc-Sign mode.

2.2 Environment

We first explain the environment in which MLS operates in. This involves introducing the relevant entities and outlining the network model under consideration.

2.2.1 Entities

Group Users: The set of users in a group. Depending on the considered security notion, the users are modeled to either be all honest or some malicious. For instance, we consider the latter case when modeling a security notion where a malicious insider (e.g., [6]) tries to impersonate an honest user.

Server: Any *asynchronous* messaging protocol requires a server to curate the messages between the group users. We consider two types of servers: honest and malicious. While servers are typically considered to be malicious by default in prior work, this is because the focus is mainly on the confidentiality of the CGKA protocol. For authentication, it makes sense to consider honest servers as well. For example, the recent work by Hashimoto et al. [36] considers an honest server to anonymously block outsiders from injecting malicious messages to the group.

Outsiders: Any adversary that is not a group user or the server. For instance, a user of the secure messaging application not in the group.

2.2.2 Network Model

Due to asynchronicity, when group users exchange messages, they must upload and download these to and from the server. Depending on the anonymity guarantee we aim to achieve, there are two types of communication channels that can be used between the group users and the server.

Non-Anonymous: If the server is allowed to know the users in the group, then we assume a user-server authenticated channel is used. For instance, TLS or Noise [49] with user-side password-based authentication can be used.

Anonymous: If the group users are required to remain anonymous to the server, then we assume an authenticated anonymous channel such as TOR [27, 51] or a VPN is used.

We note dealing with authentication in the non-anonymous setting is trivial since the server can simply maintain the group list and explicitly authenticate the group users. In contrast, in the anonymous setting, such trivial solutions no longer exist and the issue of authentication becomes non-trivial. Indeed, prior works on *anonymous* secure messaging, e.g., [20, 36, 43, 55], overcome this by relying on some type of anonymous group authentication protocol.

2.3 Threat Model for Authentication

We now categorize authenticity into four different security notions: unforgeability, anonymity, anonymous blocklisting, and tracing soundness. This categorization of the application message is motivated by the security definitions used in well-studied anonymous authentication schemes, such as group signatures [8, 16, 21] and accountable ring signatures [12, 58].

Below, for each security notion, we explain *who* the adversary is, *what* the goal of the security notion is, and *why* we consider it. For simplicity, we leave outsider adversaries out of most security notions as they are strictly weaker than malicious servers and group users. The following security notions will be formalized in Sec. 3.2.

Goal 1: Unforgeability.
Adversary: Malicious group users and/or a malicious server.
Goal: No adversary can forge a signature[5] of an honest user.

[5]Throughout this section, we use the term "signature" loosely and note that signatures are not the only way to authenticate. Using the terminology of our GAM protocol, this is more formally an "authentication token".

This is the default notion that any secure messaging protocol must ensure. We can consider two levels of unforgeability: we call it unforgeable if the set of malicious group users and the malicious server can collude, and *non-colluding* unforgeable otherwise. The former guarantees that even a colluding malicious insider and server cannot forge a signature of an honest group user. In contrast, the latter restricts the adversary to be either the set of malicious group users or the malicious server; that is, unforgeability holds only if there is no collusion. While (standard) unforgeability is the more secure notion, sacrificing security against collusion of a malicious insider and server could be a reasonable compromise for better efficiency.

Goal 2: Anonymity.
Adversary: A malicious server.
Goal: The server cannot deanonymize and link the activity of the group users. E.g., the server cannot distinguish whether two uploaded messages came from the same user or from two different users.

For this security notion we must rely on an anonymous network model as, otherwise, communication will be linkable at the network level. We further assume all the group users to be honest, since a malicious user can always inform the server of who is in the group or who authored a message.

Goal 3: Anonymous Blocklisting.
Adversary: An outsider.
Goal: An honest server can block any outsider trying to upload messages on behalf of the group.

Observe that non-anonymous blocklisting is trivial to satisfy, since the server can perform access control by explicitly authenticating the group users. We therefore use the term "anonymous" to emphasize that the motivation of the server is to blocklist non-group users while preserving the anonymity of the users. The purpose of anonymous blocklisting is for the server to be able to prevent outsiders from launching a DoS attack on the group. Importantly, although group users can verify the authenticity of the messages by downloading them from the server, we require the server to directly reject invalid messages on behalf of the group. This is satisfied for example by Sealed Sender [43] used in the Signal protocol and the metadata-hiding MLS protocol by Hashimoto et al. [36].

Goal 4: Tracing Soundness.
Adversary: Malicious group users.
Goal: The set of honest group users can trace any (possibly maliciously crafted) signature back to a *unique* group user; if an honest user traces a signature back to a user u in the group, then all other honest users trace it back to the same user u.

Tracing soundness allows to keep the view of the honest users consistent. For instance, consider a malicious insider mounting a DoS attack against the group by spamming garbage application messages. With tracing soundness, the honest users can unanimously agree on who the malicious insider was and remove him from the group. One can draw a parallel to anonymous blocklisting, that prevents such attacks from outsiders. Moreover, while similar, it is worth noting that tracing soundness is an orthogonal notion to unforgeability. Consider a malicious insider u that modifies the signature of an honest user v in such a way that for half of the group members it traces back to u, but for the other half traces back to v. While this does not contradict unforgeability, as the malicious user is effectively just "repurposing" somebody's message, it clearly breaks the consistency of the group's view.

2.4 Modeling Choices and Simplifications

Before introducing our GAM protocol in the next section, we clarify the modeling choices and simplifications we make.

Trusted Setup. The GAM protocol assumes the states of both the group users and the server are generated honestly by an initialization phase. This simplification is justified for protocols like MLS, since users are assumed to start the GAM protocol with the group secret key, derived from the CGKA protocol, already in their states.

Static Groups. The GAM protocol assumes static groups, following the way in which MLS' FSPD protocol operates. Recall that in MLS a new FSPD protocol for a static group is initialized every time group membership changes, as this will trigger a new CGKA protocol epoch (see Fig. 1). More generally, though, we could consider a *continuous* GAM protocol where we do not need to reinitialize the protocol with every group change, similarly to a CGKA protocol. However, such a definition must be intertwined with that of the CGKA protocol responsible for group state updates, rendering the definition to be as complex as modeling MLS in its entirety. As a study investigating new security goals of authentication, we opt for making the security notions tractable and to improve the overall readability. Nonetheless, we explain in Sec. 6.2 with concrete examples on how each of our proposed GAM protocols can handle dynamic operations.

Out-of-Order Messages. In our work, we do not model authentication when messages arrive out-of-order. While this is arguably important for a comprehensive model, we highlight that, unlike confidentiality, lack of authentication does not harm the usability of the FSPD protocol. In the context of the MLS protocol, *immediate decryption* of the messages will still be maintained. The only difference between MLS is that we may lose *immediate authentication* when messages arrive out-of-order. Importantly, though security is lost while some messages are missing, assuming that every message eventually arrives, then out-of-order messages do not affect security. Instead, if some messages are permanently dropped, we can allow the recipients to fetch this missing authentication information, which they can do assuming the proper indexing of the messages required by out-of-order decryption. We note that in MLS [13, Section 5.2], whether messages

eventually arrive or not is controlled by the application that sets the policy.

3 Group Authenticated Messaging Protocol

We introduce *group authenticated messaging* (GAM) protocols and the associated security requirements.

3.1 Definition

A GAM protocol is defined between a server and a group G of users. As explained above, there exists an initialization algorithm Init that prepares the initial state for the group users, possibly further preparing a secret key for the server. To send a message m (e.g., the output of a FSPD protocol), a user $u \in \mathsf{G}$ runs the Send algorithm, outputting a *group* authentication token Σ_{G}. A server verifies $(\mathsf{m}, \Sigma_{\mathsf{G}})$ using the Verify algorithm and prepares *user* authentication tokens $(\sigma_i)_{i\in[N]}$, where $N = |\mathsf{G}|$. For example, in the context of the Sign mode in MLS (see Footnote 3), Σ_{G} is simply u's signature and $\sigma_i := \Sigma_{\mathsf{G}}$. To capture anonymity, we assume the server only knows the size of the group G[6] and assume a bijective map $\mathsf{idx} : \mathsf{G} \to [N]$ is secretly known by the group users. Namely, a user u such that $i = \mathsf{idx}(u)$ fetches σ_i from the server. It then runs the $\mathsf{Receive}$ algorithm to verify (m, σ_i) and traces the purported user $v \in \mathsf{G}$ that generated σ_i. It is worth highlighting that we make a distinction between a group authentication token Σ_{G} and a user authentication token σ_i to capture an optimization technique called *selective downloading* [5, 35]. This technique allows the server to sanitize the group authentication token Σ_{G} in a straightforward manner by delivering to each group user just the strictly necessary amount of data σ_i, while maintaining the same level of (dis)trust.

Finally, we endow a GAM protocol with an *offline-online* feature. In the offline phase, when the message is still unknown, a user can perform a possibly heavy state update, and share the update with the server and the group via the $\mathsf{UpdSend}$ algorithm. This algorithm is accompanied by algorithms $\mathsf{UpdVerify}$ and $\mathsf{UpdReceive}$ similarly to above. Once the message is known in the online phase, the user can send it using its updated state.[7] Formally, we have the following.

Definition 3.1. *A* GAM *protocol for message space $\mathcal{M}$ between a server* Sv *and a set of users in a group* G *consists of the following algorithms, where* $\mathsf{idx} : \mathsf{G} \to [N]$ *is a bijective function with* $N := |\mathsf{G}|$*. Below, if an algorithm outputs $\perp$, we assume it reverts to the state before running the algorithm.*

$\mathsf{Init}(1^\kappa, \mathsf{G}) \to \left(\mathsf{pp}, \mathsf{sk}_{\mathsf{Sv}}, (\mathsf{st}_u)_{u\in\mathsf{G}}\right)$: *On input the security parameter 1^κ and group information $\mathsf{G} \subset \{0,1\}^*$, it outputs public parameters* pp*, a secret key* $\mathsf{sk}_{\mathsf{Sv}}$ *for the server* Sv*, and an initial state* st_u *for all users $u \in \mathsf{G}$. We assume $\mathsf{G} \in \mathsf{st}_u$.*

$\mathsf{Send}(\mathsf{st}_u, \mathsf{m}) \to (\mathsf{st}'_u, \Sigma_{\mathsf{G}})$ *or* $\perp$: *On input a state* st_u *for user $u \in \mathsf{G}$ and a message $\mathsf{m} \in \mathcal{M}$, user u outputs an updated state* st'_u *and a* group authentication token Σ_{G}*, or* $\perp$*.*

$\mathsf{Verify}(\mathsf{pp}, \mathsf{sk}_{\mathsf{Sv}}, \Sigma_{\mathsf{G}}, \mathsf{m}) \to (\mathsf{pp}', (\sigma_i)_{i\in[N]})$ *or* $\perp$: *On input public parameters* pp*, a server secret key* $\mathsf{sk}_{\mathsf{Sv}}$*, a group authentication token Σ_{G}, and a message $\mathsf{m} \in \mathcal{M}$, the server* Sv *outputs updated public parameters* pp' *and N* user authentication tokens $(\sigma_i)_{i\in[N]}$*, or* $\perp$*.*

$\mathsf{Receive}(\mathsf{st}_u, \sigma, \mathsf{m}) \to (\mathsf{st}'_u, b \in \{\top, \perp\}, v \in \mathsf{G} \cup \{\perp\})$: *On input a state* st_u *for user $u \in \mathsf{G}$, a user authentication token σ, and a message $\mathsf{m} \in \mathcal{M}$, user u outputs an updated state* st'_u*, a bit b indicating whether the token was valid ($b = \top$) or invalid ($b = \perp$), and a purported user $v \in \mathsf{G} \cup \{\perp\}$, where $v = \perp$ if tracing fails.*

$\mathsf{UpdSend}(\mathsf{st}_u) \to (\mathsf{st}'_u, \widehat{\Sigma}_{\mathsf{G}}, \widehat{\mathsf{ct}}_{\mathsf{G}})$: *On input a state* st_u *for user $u \in \mathsf{G}$, user u outputs an updated state* st'_u*, a* group update authentication token $\widehat{\Sigma}_{\mathsf{G}}$*, and* group update information $\widehat{\mathsf{ct}}_{\mathsf{G}}$*.*

$\mathsf{UpdVerify}(\mathsf{pp}, \mathsf{sk}_{\mathsf{Sv}}, \widehat{\Sigma}_{\mathsf{G}}, \widehat{\mathsf{ct}}_{\mathsf{G}}) \to \left(\mathsf{pp}', \left(\widehat{\sigma}_i, \widehat{\mathsf{ct}}_i\right)_{i\in[N]}\right)$ *or* $\perp$: *On input public parameters* pp*, a server secret key* $\mathsf{sk}_{\mathsf{Sv}}$*, a group update authentication token $\widehat{\Sigma}_{\mathsf{G}}$, and group update information $\widehat{\mathsf{ct}}_{\mathsf{G}}$, the server* Sv *outputs updated public parameters* pp' *and a list of* user update authentication tokens *and* user update information $\left(\widehat{\sigma}_i, \widehat{\mathsf{ct}}_i\right)_{i\in[N]}$*, or* $\perp$*.*

$\mathsf{UpdReceive}(\mathsf{st}_u, \widehat{\sigma}, \widehat{\mathsf{ct}}) \to (\mathsf{st}'_u, b \in \{\top, \perp\}, v \in \mathsf{G} \cup \{\perp\})$: *On input a state* st_u *for user $u \in \mathsf{G}$, a user update authentication token $\widehat{\sigma}$, and user update information $\widehat{\mathsf{ct}}$, user u outputs an updated state* state'_u*, a bit b indicating whether the token was valid ($b = \top$) or invalid ($b = \perp$), and a purported user $v \in \mathsf{G} \cup \{\perp\}$, where $v = \perp$ if tracing fails.*

Remark 3.2 (Local and Global State Updates). For some protocols the user state may only allow signing up to T messages, and it may need to be updated before the user can sign again. There are two ways to perform state updates: *locally* and *globally*. In the former, a user regains the ability to send messages once it has updated its own state. In the latter, a user regains the ability to send messages only after every user in the group updates their states. Since global state updates are much more costly than local state updates, they are only useful if one state update allows to send a large number of messages T. Further, global updates can only guarantee security if users are online, a clear disadvantage over local updates. For the schemes presented in this paper there are no risks of a deadlock — i.e., a situation where a global state update cannot be completed and users are prevented to keep sending messages — as long as the users perform updates once coming online. However,

[6] While we could consider further hiding the size of the group to the server, we choose not to since it would resort in an inefficient padding strategy. This is the same level of anonymity satisfied by previous anonymous SGM protocols e.g., [20, 36].

[7] Naturally, protocols need not have such a differentiation and can simply only perform online state updates. This optimization allows us to improve the real-world usability of those protocols that do, as they can more evenly distribute their computation and communication over time.

the general definition of global updates does not guarantee that such a deadlock does not occur.

Correctness of GAM protocols is defined in the full version of this paper.

3.2 Security

We formalize the threat models explained in Sec. 2.3: unforgeability, anonymity, anonymous blocklisting, and tracing soundness, via a security game defined in Fig. 2. The probability of the game outputting 1 against an efficient adversary must be negligible for every game except for anonymity. For anonymity, as it is a distinguishing game, the game must output 1 with probability negligibly close to $\frac{1}{2}$.

For every game, the adversary is given access to oracles $\{ O_{\mathsf{Send}}, O_{\mathsf{UpdSend}} \}$, allowing it to invoke honest users to create group (update) authentication tokens. The adversary is further given access to either $\{ O_{\mathsf{Receive}}, O_{\mathsf{UpdReceive}} \}$ or $\{ O_{\mathsf{GroupReceive}}, O_{\mathsf{GroupUpdReceive}} \}$. The former allows the adversary to directly invoke honest users to process (update) authentication tokens. This capture malicious server capabilities and is used by the unforgeability and anonymity games. In contrast, the latter only allows the adversary to query for *group* (update) authentication tokens. The oracle then individually invokes each honest users on the correctly processed (update) authentication tokens. Namely, this captures honest server behavior and models the fact that malicious users cannot directly send messages to group users. It is worth noting that, in this case, the authentication tokens created in $\{ O_{\mathsf{Send}}, O_{\mathsf{UpdSend}} \}$ are directly processed by $\{ O_{\mathsf{GroupReceive}}, O_{\mathsf{GroupUpdReceive}} \}$, modeling the fact that the communication channel between an honest user and server is secure.

To aid readability, we highlight some features of the security game. We model two types of unforgeability by $\mathsf{Game}^{\mathsf{X}}_{\mathcal{A}}$ with $\mathsf{X} \in \{ \mathsf{ncUnf}, \mathsf{Unf} \}$. In standard unforgeability, as the adversary models both a malicious user and server, it has unrestricted access to all oracles. In contrast, for non-colluding unforgeability, we have two case distinctions depending on whether the set of corrupted users $\mathcal{C} = \emptyset$ or not. In the former case the adversary is a malicious server, so the adversary is given the server secret key $\mathsf{sk}_{\mathsf{Sv}}$ and has access to $\{ O_{\mathsf{Receive}}, O_{\mathsf{UpdReceive}} \}$. In the latter case the adversary is a set of malicious users, so the adversary is instead given the corrupted users' states and only has access to $\{ O_{\mathsf{GroupReceive}}, O_{\mathsf{GroupUpdReceive}} \}$. For both types of unforgeability, an adversary wins if it can output a valid user (update) authentication token for an honest user that it has not seen before. For anonymity, we model a malicious server by giving the adversary the server secret key $\mathsf{sk}_{\mathsf{Sv}}$. The adversary outputs two users and messages and the game creates the group authentication tokens for both users. To non-trivialize the game, we restrict the (group) authentication tokens to be valid. To perform this check, the adversary needs to further output a (possibly malformed) public parameter $\overline{\mathsf{pp}}$ so the game can run algorithm Verify. The adversary can further perform oracle queries under the restriction that it does not query the receive oracles on the challenge authentication tokens.

More discussion can be found in the full version of this paper.

4 COSMOS: Authentication with One-Time Tokens

In this section we propose a GAM protocol named COSMOS (Compact authenticated Secure Messaging with randomized One-time tokenS). When anonymity is not necessary, COSMOS is the most efficient and simplest protocol among all our proposed protocols. The additional total communication overhead is only 3κ compared to a protocol where messages are sent without any authentication, where κ is the security parameter. Additionally, we show a simple method to bootstrap COSMOS to satisfy anonymity and anonymous blocklisting, which we name COSMAC. The added overhead to COSMOS is a single MAC tag. Lastly, we show how to optimize both protocols by batching sends and updates together.

4.1 Construction of COSMOS

The high level idea is as follows: each group user mints tokens $(x_i, y_i) \in \{0,1\}^\kappa \times \{0,1\}^\kappa$ for $i \in [T]$ such that $y_i = \mathsf{OWF}(x_i)$; stores the private tokens $(x_i)_{i\in[T]}$ in its state; uploads the public tokens $(y_i)_{i\in[T]}$ to the server in an *offline* phase; and ideally sends (x_i, m_i) to the server once the message m_i is defined in an *online* phase, where x_i acts as the authentication token, and delete x_i from its state. However, since x_i is not cryptographically tied to m_i, this is insecure. Thus, the user additionally MACs (x_i, m_i) using a MAC key only known among the group users. We highlight that such a MAC key can be generated from the *common* group secret key gsk maintained by the CGKA protocol.

More formally, after the initialization phase, each user $u \in \mathsf{G}$ and server maintain a list of public tokens $\mathsf{PubTOKEN} \in (\{0,1\}^\kappa)^{NT}$, where $N = |\mathsf{G}|$ and T is the number of messages a user can send before needing to update its state. $\mathsf{PubTOKEN}$ is a list such that, for each user $u \in \mathsf{G}$, $\mathsf{PubTOKEN}[u] \in (\{0,1\}^\kappa)^T$ stores the T public tokens $(y_i)_{i\in[T]}$ used by user u. User u also maintains a list of private tokens $\mathsf{PrivTOKEN}_u \in (\{0,1\}^\kappa)^T$ storing the T private tokens $(x_i)_{i\in[T]}$.

To send a message m, user u retrieves an unused private token x from $\mathsf{PrivTOKEN}_u$, along with the counter $\mathsf{ctr} \in [T]$ such that $\mathsf{PubTOKEN}[u][\mathsf{ctr}] = \mathsf{OWF}(x)$, and sends $(x, \mathsf{ctr}, \Sigma_{\mathsf{MAC}})$ as the group authentication token Σ_{G} to the server, where Σ_{MAC} is a MAC tag using $\mathsf{k}_{\mathsf{MAC}}$. The server then checks if the token x is valid (i.e., $y_{\mathsf{ctr}} = \mathsf{OWF}(x)$) and relays $(x, \mathsf{ctr}, \Sigma_{\mathsf{MAC}})$ as the user authentication token σ_i to all the users. Here, Σ_{MAC} does not need to (nor can it) be verified

$\mathsf{Game}^{\mathsf{X}}_{\mathcal{A}}(1^\kappa)$: $\mathsf{X} \in \{\,\mathsf{ncUnf}, \mathsf{Unf}\,\}$

$\mathcal{C} \leftarrow\$\, \mathcal{A}(1^\kappa)$
$\mathcal{H} := \mathsf{G} \backslash \mathcal{C}$
$L_{\mathsf{msg}}, L_{\mathsf{upd}} := \emptyset$ / Book keeping
$(\mathsf{pp}, \mathsf{sk}_{\mathsf{Sv}}, (\mathsf{st}_u)_{u \in \mathsf{G}}) \leftarrow\$\, \mathsf{Init}(1^\kappa, \mathsf{G})$
if $\mathsf{X} = \mathsf{Unf}$ **then**
  $(\mathtt{label}, \mathtt{obj}) \leftarrow\$\, \mathcal{A}^{O}(\mathsf{pp}, \mathsf{sk}_{\mathsf{Sv}}, (\mathsf{st}_u)_{u \in \mathcal{C}})$
else / No collusion between malicious user and server
  if $\mathcal{C} = \emptyset$ / Honest users
    $(\mathtt{label}, \mathtt{obj}) \leftarrow\$\, \mathcal{A}^{O}(\mathsf{pp}, \mathsf{sk}_{\mathsf{Sv}})$
  else / Honest server
    $(\mathtt{label}, \mathtt{obj}) \leftarrow\$\, \mathcal{A}^{O^\star}(\mathsf{pp}, (\mathsf{st}_u)_{u \in \mathcal{C}})$
if $\mathtt{label} = \mathtt{msg}$ **then**
  parse $(u, \sigma, \mathsf{m}) \leftarrow \mathtt{obj}$
  req $u \in \mathcal{H}$
  $(\mathsf{st}'_u, b_v, v_u) \leftarrow \mathsf{Receive}(\mathsf{st}_u, \sigma, \mathsf{m})$
  req $b_v = \top$ / Valid authentication token
  $b \leftarrow [\![v_u \in \mathcal{H} \wedge (v_u, *, \mathsf{m}) \notin L_{\mathsf{msg}}]\!]$
elseif $\mathtt{label} = \mathtt{upd}$ **then**
  parse $(u, \widehat{\sigma}, \widehat{\mathsf{ct}}) \leftarrow \mathtt{obj}$
  req $u \in \mathcal{H}$
  $(\mathsf{st}'_u, b_v, v_u) \leftarrow \mathsf{UpdReceive}(\mathsf{st}_u, \widehat{\sigma}, \widehat{\mathsf{ct}})$
  req $b_v = \top$ / Valid authentication token
  $b \leftarrow [\![v_u \in \mathcal{H} \wedge (v_u, *, \widehat{\mathsf{ct}}) \notin L_{\mathsf{upd}}]\!]$
return b

$\mathsf{Game}^{\mathsf{Anon}}_{\mathcal{A}}(1^\kappa)$

$\mathcal{H} := \mathsf{G}$ / No corrupt users
$\mathsf{Chall}_{\mathsf{msg}} := \emptyset$
$\mathsf{coin} \leftarrow\$\, \{0,1\}$
$(\mathsf{pp}, \mathsf{sk}_{\mathsf{Sv}}, (\mathsf{st}_u)_{u \in \mathsf{G}}) \leftarrow\$\, \mathsf{Init}(1^\kappa, \mathsf{G})$
$(\overline{\mathsf{pp}}, u_0, u_1, \mathsf{m}_0, \mathsf{m}_1) \leftarrow\$\, \mathcal{A}^{O}(\mathsf{pp}, \mathsf{sk}_{\mathsf{Sv}})$
foreach $b \in \{0,1\}$ **do**
  $(\mathsf{st}'_{u_b}, \Sigma^b_{\mathsf{G}}) \leftarrow\$\, \mathsf{Send}(\mathsf{st}_{u_b}, \mathsf{m}_{b \oplus \mathsf{coin}})$
  $(\overline{\mathsf{pp}}', (\sigma^b_i)_{i \in [N]})$
    $\leftarrow \mathsf{Verify}(\overline{\mathsf{pp}}, \mathsf{sk}_{\mathsf{Sv}}, \Sigma^b_{\mathsf{G}}, \mathsf{m}_{b \oplus \mathsf{coin}})$
  / Require the authentication token to be valid
  req $\overline{\mathsf{pp}}' \neq \bot$
  foreach $u \in \mathcal{H}$ **do**
    $(\mathsf{st}'_u, b_u, v_u) \leftarrow \mathsf{Receive}(\mathsf{st}_u, \sigma^b_{\mathsf{idx}(u)}, \mathsf{m}_{b \oplus \mathsf{coin}})$
    req $b_u \neq \bot$
  $\mathsf{Chall}_{\mathsf{msg}} \leftarrow \mathsf{Chall}_{\mathsf{msg}} \cup \{\, \sigma^b_i \,\}_{i \in [N]}$
  $\overline{\mathsf{pp}} \leftarrow \overline{\mathsf{pp}}'$
$\widehat{\mathsf{coin}} \leftarrow\$\, \mathcal{A}^{O}(\mathsf{Chall}_{\mathsf{msg}})$
return $[\![\mathsf{coin} = \widehat{\mathsf{coin}}]\!]$

$\mathsf{Game}^{\mathsf{AnonBlock}}_{\mathcal{A}}(1^\kappa)$

$\mathcal{H} := \mathsf{G}$ / No corrupt users
$L_{\mathsf{msg}}, L_{\mathsf{upd}} := \emptyset$ / Book keeping
$(\mathsf{pp}, \mathsf{sk}_{\mathsf{Sv}}, (\mathsf{st}_u)_{u \in \mathsf{G}}) \leftarrow\$\, \mathsf{Init}(1^\kappa, \mathsf{G})$
$(\mathtt{label}, \mathtt{obj}) \leftarrow\$\, \mathcal{A}^{O^\star}(\mathsf{pp})$ / Malicious outsides
if $\mathtt{label} = \mathtt{msg}$ **then**
  parse $(\Sigma_{\mathsf{G}}, \mathsf{m}) \leftarrow \mathtt{obj}$
  $(\mathsf{pp}', (\sigma_i)_{i \in [N]}) \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathsf{sk}_{\mathsf{Sv}}, \Sigma_{\mathsf{G}}, \mathsf{m})$
  req $\mathsf{pp}' \neq \bot$ / Require Verify to succeed
  / Sv accepts new non-member token
  $b \leftarrow [\![(*, \Sigma_{\mathsf{G}}, *) \notin L_{\mathsf{msg}}]\!]$
elseif $\mathtt{label} = \mathtt{upd}$ **then**
  parse $(\widehat{\Sigma}_{\mathsf{G}}, \widehat{\mathsf{ct}}_{\mathsf{G}}) \leftarrow \mathtt{obj}$
  $\left(\mathsf{pp}', (\widehat{\sigma}_i, \widehat{\mathsf{ct}}_i)_{i \in [N]}\right)$
    $\leftarrow \mathsf{UpdVerify}(\mathsf{pp}, \mathsf{sk}_{\mathsf{Sv}}, \widehat{\Sigma}_{\mathsf{G}}, \widehat{\mathsf{ct}}_{\mathsf{G}})$
  req $\mathsf{pp}' \neq \bot$ / Require UpdVerify to succeed
  / Sv accepts new non-member token
  $b \leftarrow [\![(*, \widehat{\Sigma}_{\mathsf{G}}, *) \notin L_{\mathsf{upd}}]\!]$
return b

$\mathsf{Game}^{\mathsf{TraceSound}}_{\mathcal{A}}(1^\kappa)$

$\mathcal{C} \leftarrow\$\, \mathcal{A}(1^\kappa)$
$\mathcal{H} := \mathsf{G} \backslash \mathcal{C}$
$L_{\mathsf{tr}} := \emptyset$ / Book keeping
$(\mathsf{pp}, \mathsf{sk}_{\mathsf{Sv}}, (\mathsf{st}_u)_{u \in \mathsf{G}}) \leftarrow\$\, \mathsf{Init}(1^\kappa, \mathsf{G})$
$(\mathtt{label}, \mathtt{obj}) \leftarrow\$\, \mathcal{A}^{O^\star}(\mathsf{pp}, (\mathsf{st}_u)_{u \in \mathcal{C}})$
if $\mathtt{label} = \mathtt{msg}$ **then**
  parse $(\Sigma_{\mathsf{G}}, \mathsf{m}) \leftarrow \mathtt{obj}$
  $(\mathsf{pp}', (\sigma_i)_{i \in [N]}) \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathsf{sk}_{\mathsf{Sv}}, \Sigma_{\mathsf{G}}, \mathsf{m})$
  req $\mathsf{pp}' \neq \bot$ / Require UpdVerify to succeed
  foreach $u \in \mathcal{H}$ **do**
    $(\mathsf{st}'_u, b_v, v_u) \leftarrow \mathsf{Receive}(\mathsf{st}_u, \sigma_{\mathsf{idx}(u)}, \mathsf{m})$
    if $b_v = \top$ **then**
      $L_{\mathsf{tr}} \leftarrow L_{\mathsf{tr}} \cup \{\, v_u \,\}$ / If tracing fails, $v_u = \bot$
elseif $\mathtt{label} = \mathtt{upd}$ **then**
  parse $(\widehat{\Sigma}_{\mathsf{G}}, \widehat{\mathsf{ct}}_{\mathsf{G}}) \leftarrow \mathtt{obj}$
  $\left(\mathsf{pp}', (\widehat{\sigma}_i, \widehat{\mathsf{ct}}_i)_{i \in [N]}\right)$
    $\leftarrow \mathsf{UpdVerify}(\mathsf{pp}, \mathsf{sk}_{\mathsf{Sv}}, \widehat{\Sigma}_{\mathsf{G}}, \widehat{\mathsf{ct}}_{\mathsf{G}})$
  req $\mathsf{pp}' \neq \bot$ / Require UpdVerify to succeed
  foreach $u \in \mathcal{H}$ **do**
    $(\mathsf{st}'_u, b_v, v_u)$
      $\leftarrow \mathsf{UpdReceive}(\mathsf{st}_u, \widehat{\sigma}_{\mathsf{idx}(u)}, \widehat{\mathsf{ct}}_{\mathsf{idx}(u)})$
    if $b_v = \top$ **then**
      $L_{\mathsf{tr}} \leftarrow L_{\mathsf{tr}} \cup \{\, v_u \,\}$ / If tracing fails, $v_u = \bot$
/ Does not uniquely trace user
$b \leftarrow [\![\nexists v \in \mathsf{G} : L_{\mathsf{tr}} = \{\, v \,\}]\!]$
return b

Oracle $O_{\mathsf{Send}}(u \in \mathcal{H}, \mathsf{m})$

$s_{\mathsf{Rec}} := \emptyset$
$(\mathsf{st}'_u, \Sigma_{\mathsf{G}}) \leftarrow\$\, \mathsf{Send}(\mathsf{st}_u, \mathsf{m})$
$L_{\mathsf{msg}} \leftarrow L_{\mathsf{msg}} \cup \{\, (u, \Sigma_{\mathsf{G}}, \mathsf{m}) \,\}$
/ Server honestly processes group authentication
if $\mathcal{A}$ has access to $O^\star$ **then**
  $s_{\mathsf{Rec}} \leftarrow O_{\mathsf{GroupReceive}}(\Sigma_{\mathsf{G}}, \mathsf{m})$
return $(\Sigma_{\mathsf{G}}, s_{\mathsf{Rec}})$

Oracle $O_{\mathsf{Receive}}(u \in \mathcal{H}, \sigma, \mathsf{m})$

req $\sigma \notin \mathsf{Chall}_{\mathsf{msg}}$ / Only used by anonymity
$(\mathsf{st}'_u, b_u, v_u) \leftarrow \mathsf{Receive}(\mathsf{st}_u, \sigma_{\mathsf{idx}(u)}, \mathsf{m})$
return (b_u, v_u)

Oracle $O_{\mathsf{GroupReceive}}(\Sigma_{\mathsf{G}}, \mathsf{m})$

$(\mathsf{pp}', (\sigma_i)_{i \in [N]}) \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathsf{sk}_{\mathsf{Sv}}, \Sigma_{\mathsf{G}}, \mathsf{m})$
if $\mathsf{pp}' = \bot$ **then return** $\bot$
foreach $u \in \mathcal{H}$ **do**
  $(\mathsf{st}'_u, b_u, v_u) \leftarrow \mathsf{Receive}(\mathsf{st}_u, \sigma_{\mathsf{idx}(u)}, \mathsf{m})$
return $(b_u, v_u)_{u \in \mathcal{H}}$

Oracle $O_{\mathsf{UpdSend}}(u \in \mathcal{H})$

$s_{\mathsf{UpdRec}} := \emptyset$
$(\mathsf{st}'_u, \widehat{\Sigma}_{\mathsf{G}}, \widehat{\mathsf{ct}}_{\mathsf{G}}) \leftarrow\$\, \mathsf{UpdSend}(\mathsf{st}_u)$
$L_{\mathsf{upd}} \leftarrow L_{\mathsf{upd}} \cup \{\, (u, \widehat{\Sigma}_{\mathsf{G}}, \widehat{\mathsf{ct}}_{\mathsf{G}}) \,\}$
/ Server honestly processes group authentication
if $\mathcal{A}$ has access to $O^\star$ **then**
  $s_{\mathsf{UpdRec}} \leftarrow O_{\mathsf{GroupUpdReceive}}(\widehat{\Sigma}_{\mathsf{G}}, \widehat{\mathsf{ct}}_{\mathsf{G}})$
return $((\widehat{\Sigma}_{\mathsf{G}}, \widehat{\mathsf{ct}}_{\mathsf{G}}), s_{\mathsf{UpdRec}})$

Oracle $O_{\mathsf{UpdReceive}}(u \in \mathcal{H}, \widehat{\sigma}, \widehat{\mathsf{ct}})$

$(\mathsf{st}'_u, b_u, v_u)$
  $\leftarrow \mathsf{UpdReceive}(\mathsf{st}_u, \widehat{\sigma}_{\mathsf{idx}(u)}, \widehat{\mathsf{ct}}_{\mathsf{idx}(u)})$
return (b_u, v_u)

Oracle $O_{\mathsf{GroupUpdReceive}}(\widehat{\Sigma}_{\mathsf{G}}, \widehat{\mathsf{ct}}_{\mathsf{G}})$

$\left(\mathsf{pp}', (\widehat{\sigma}_i, \widehat{\mathsf{ct}}_i)_{i \in [N]}\right)$
  $\leftarrow \mathsf{UpdVerify}(\mathsf{pp}, \mathsf{sk}_{\mathsf{Sv}}, \widehat{\Sigma}_{\mathsf{G}}, \widehat{\mathsf{ct}}_{\mathsf{G}})$
if $\mathsf{pp}' = \bot$ **then return** $\bot$
foreach $u \in \mathcal{H}$ **do**
  $(\mathsf{st}'_u, b_u, v_u)$
    $\leftarrow \mathsf{UpdReceive}(\mathsf{st}_u, \widehat{\sigma}_{\mathsf{idx}(u)}, \widehat{\mathsf{ct}}_{\mathsf{idx}(u)})$
return $(b_u, v_u)_{u \in \mathcal{H}}$

Figure 2: Security games for (non-colluding) unforgeability, anonymity, anonymous blocklisting, and tracing soundness. We define a set of oracles $O := \{\, O_{\mathsf{Send}}, O_{\mathsf{Receive}}, O_{\mathsf{UpdSend}}, O_{\mathsf{UpdReceive}} \,\}$ and $O^\star := \{\, O_{\mathsf{Send}}, O_{\mathsf{GroupReceive}}, O_{\mathsf{UpdSend}}, O_{\mathsf{GroupUpdReceive}} \,\}$. We assume the game maintains the public parameter pp and (secret) user states st_u. Moreover, we assume the updated state st'_u is implicitly set as st_u and omit the substitution $\mathsf{st}'_u \leftarrow \mathsf{st}_u$ for readability. When the condition in **req** does not hold, we assume the game outputs a random bit in the anonymity game and 0 in all other games. Lastly, for readability, we sometimes ignore creating the lists $L_{\mathsf{tr}}, L_{\mathsf{msg}}, L_{\mathsf{upd}}$ when they are not required by the game.

by the server. Now, since PubTOKEN and $\mathsf{k}_{\mathsf{MAC}}$ is shared among the group, the users can verify the MAC tag and trace the user u that sent σ_i.

When only one private token x is left, user u performs a state update and mints new tokens. It generates a new batch of T tokens $(x_i, y_i)_{i\in[T]}$ and uploads $(y_i)_{i\in[T]}$ using the final token x along with a MAC tag. The server and users check that the newly minted tokens are from user u by validating x and update $\mathsf{PubTOKEN}[u] \leftarrow (y_i)_{i\in[T]}$. Once user u's state is updated, u can send T messages again. Importantly, COSMOS is *locally* state-updatable since a user can start sending messages once they update their state. Due to page limitations, the protocol is formally given in the full version of this paper.

Lastly, COSMOS satisfies all the security notions except for anonymity: non-colluding unforgeability, (anonymous) blocklisting, and tracing soundness. At a high level, we argue non-colluding unforgeability by considering two cases: against a malicious server the authentication token is unforgeable as $\mathsf{k}_{\mathsf{MAC}}$ is unknown. Importantly, the same authentication token $(x, \mathsf{ctr}, \Sigma_{\mathsf{MAC}})$ cannot be reused by the malicious server since the users have already deleted the associated public token y when it receives x the second time. Otherwise, against a malicious user, it is unforgeable as the private token x is unknown. In the latter, we use the fact that an honest server correctly processes the private token sent from an honest user (i.e., delete it from the server), preventing a malicious user from replaying it. One can check that it is not *standard* unforgeable since if a malicious server and insider collude, both $\mathsf{k}_{\mathsf{MAC}}$ and private tokens x will be known to the adversary, allowing for a trivial forgery. Moreover, we note that even though the MAC tag attached to the group authentication token cannot be verified by the server, and hence can be stealthily modified to a garbage MAC tag, this will not harm tracing soundness as we only use the private tokens for tracing. The formal security proof is deferred to the full version of this paper.

4.2 COSMAC: An Anonymous COSMOS with Anonymous Blocklisting

While COSMOS is efficient, it lacks anonymity. A server can link two tokens by looking at their corresponding locations in PubTOKEN. We present a simple method to transform COSMOS to have anonymity and anonymous blocklisting at an overhead of only one MAC tag. We name this GAM protocol COSMAC (COSMOS with MAC). Note that in exchange for anonymity, COSMAC loses tracing soundness.

The high level idea is for each user in the group to additionally derive a unique MAC key $\overline{\mathsf{k}_{\mathsf{MAC}}}$ and a symmetric-key encryption (SKE) key $\mathsf{k}_{\mathsf{SKE}}$ from gsk where, unlike in COSMOS, $\overline{\mathsf{k}_{\mathsf{MAC}}}$ is uploaded to the server. When a group user uploads some content to the server, it runs the Send (resp. UpdSend) algorithm of COSMOS, encrypts the group (resp. update) authentication token using $\mathsf{k}_{\mathsf{SKE}}$, and MACs the ciphertext with $\overline{\mathsf{k}_{\mathsf{MAC}}}$. The server only accepts contents that have a valid tag under $\overline{\mathsf{k}_{\mathsf{MAC}}}$. A group user can verify the user authentication token by first decrypting the ciphertext using $\mathsf{k}_{\mathsf{SKE}}$, followed by the same check as COSMOS. The protocol is formally given in the full version of this paper.

Observe that the authentication tokens are now encrypted and the server no longer learns the identity of the user. This is how anonymity is achieved. Non-colluding unforgeability almost immediately follows from the non-colluding unforgeability of COSMOS. This is because from the users point of view, COSMAC and COSMOS are almost identical. The only difference is that COSMAC requires to first perform a decryption using $\mathsf{k}_{\mathsf{SKE}}$; this does not make forging anymore easier for the adversary. Indeed, prove non-colluding unforgeability of COSMAC assuming the non-colluding unforgeability of COSMOS. Moreover, COSMAC satisfies anonymous blocklisting since an outsider without knowledge of $\overline{\mathsf{k}_{\mathsf{MAC}}}$ cannot upload contents which the server will accept. Lastly, on the other hand, unlike COSMOS, a malicious user can now stealthily perform a DoS attack on the group since the server can only check the validity of the MAC tag and not the content. In particular, COSMAC loses tracing soundness, as the content, which can now be a malformed ciphertext, may not include the sender's identity. We show in the full version of this paper that COSMAC is non-colluding unforgeable, anonymous, and anonymous blocklistable.

4.3 Optimizations of COSMOS and COSMAC

We take advantage of the fact that COSMOS (and COSMAC) have an efficient *local* state update and apply two optimizations leading to $\mathtt{COSMOS}^{+}$ and $\mathtt{COSMOS}^{++}$ (and $\mathtt{COSMAC}^{+}$ and $\mathtt{COSMAC}^{++}$, respectively). We focus on COSMOS as the case for COSMAC is almost identical.

Removing Local Updates: $\mathtt{COSMOS}^{+}$. Notice that local state-updates allow a user to execute the UpdSend algorithm as part of the Send algorithm. That is, users can send a message and perform an update *at the same time*. Concretely, this requires to maintain just one public token y per user $u \in \mathsf{G}$. To send a message m, u first mints a new token (x', y') and uploads both the message and public token (m, y') using the private token x, along with a MAC tag binding (m, x, y') together. The server and other group members replace y with y'. User u can repeat the process using the new private token x'. Effectively, the protocol now consists of only running an online phase, since the update is implicitly performed during a send. Compared to COSMOS, $\mathtt{COSMOS}^{+}$ balances the throughput of the user without harming the total communication cost $3 \cdot \kappa$, while also reducing the storage cost of public tokens.

Minimizing Communication Cost: $\mathtt{COSMOS}^{++}$. This optimization reduces the communication cost of $\mathtt{COSMOS}^{+}$ by $1/3$ while keeping the local update of COSMOS.[8] The main idea

[8] This optimization was suggested to us by an anonymous reviewer; afterwards, another reviewer informed us that a similar idea is used in the S/KEY

is to make the private authentication token become the public token for the next message. As in COSMOS$^+$, the server maintains a single public token $y_{i,c}$ per user, where $c \in \mathbb{N}$ will be the number of times user u ran UpdSend. As a result of running UpdSend the user will upload a public token $y_{0,c} = \mathsf{OWF}^T(x_{T,c})$, i.e., T^{th} invocation of OWF. This updated public token can be used to send T messages. To authenticate the i-th message m_i, user u simply sends token $x_{i,c} = \mathsf{OWF}^{T-i}(x_{T,c})$ along with a MAC tag on $(m_i, x_{i,c})$. The server and other group members update the public token to $y_{i,c} := x_{i,c}$. Since the public token generated in the offline phase is useful for sending T messages, the amortized cost of sending one message is $(2+\frac{1}{T})\cdot\kappa$. For a sufficiently large T, this reduces the communication cost of COSMOS$^+$ by $1/3$.

5 Anonymous and Tracing Sound GAMs

In this section we introduce three GAM protocols that simultaneously achieve anonymity and tracing soundness. These are the first authentication modes in the literature to do so. The first two protocols: QUASAR (Quick Authenticated Secure *Anonymous* messaging with Randomized one-time tokens) and STARS (Strongly-Authenticated anonymous messaging with Randomized one-time Signatures) satisfy these stronger authenticity guarantees at the cost of being only *global* as opposed to *local* state-updatable like COSMOS and COSMAC. Once a user $u \in \mathsf{G}$ exhausts its private tokens, it must wait till all other users perform an update before being able to send a message again. We discuss some ideas to mitigate the shortcoming of global state updates in the full version of this paper. Our third protocol GEMSTARS (Group Signature Modified STARS), eliminates updates altogether by relying on group signatures. Below we give intuitive overviews of the protocols, deferring the formal descriptions and security proofs to the full version of this paper.

5.1 QUASAR: Anonymous Authentication with Tokens

We first consider a non-anonymous variant of QUASAR and add anonymity later. Its core idea is to perform a relatively expensive *offline* phase (i.e., UpdSend) to make the *online* phase (i.e., Send) very cheap.

Basic Idea. Assume a group $\mathsf{G} = (u_i)_{i\in[N]}$. Each user u_i mints tokens $(x^{(t)}_{j\to i}, y^{(t)}_{j\to i}) \in \{0,1\}^\kappa \times \{0,1\}^\kappa$ for $(j,t) \in [N]\times[T]$ such that $y^{(t)}_{j\to i} = \mathsf{OWF}(x^{(t)}_{j\to i})$. Here, $j \to i$ indicates that user u_i approves u_j to send a message to him. User u_i sends the private tokens $(x^{(t)}_{j\to i})_{t\in[T]}$ to u_j by encrypting it with a public-key encryption (PKE) scheme using u_j's public key. Moreover, u_i uploads the public tokens $(y^{(t)}_{j\to i})_{(j,t)\in[N]\times[T]}$ to the server.

one-time password authentication protocol [33, 34].

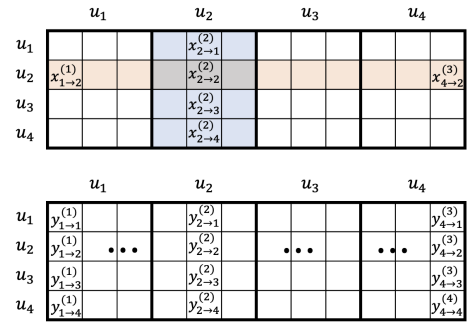

Figure 3: A toy example of a non-anonymous variant of QUASAR. $\mathsf{G} = (u_i)_{i\in[4]}$ and $T = 3$. The upper box stores the private tokens $(x^{(t)}_{i\to j})_{t\in[3]}$ for $i, j \in [4]$. The blue columns are tokens that user u_2 uses to send messages and the red rows are tokens that user u_2 minted. The bottom box stores the public tokens $(y^{(t)}_{i\to j})_{t\in[3]}$ for $i, j \in [4]$ held by the server.

A toy example is provided in Fig. 3. Throughout the protocol, each user u_i maintains two types of tokens: *sender* tokens $(x^{(t)}_{i\to j})_{(j,t)\in[N]\times[T]}$ (blue box in Fig. 3) and *receiver* tokens $(x^{(t)}_{j\to i})_{(j,t)\in[N]\times[T]}$ (red box in Fig. 3). To send the t^*-th $(t^* < T)$ message m, u_i retrieves the sender tokens $(x^{(t^*)}_{i\to j})_{j\in[N]}$ and uploads this as the group authentication token Σ_G along with m. Similarly to COSMOS and COSMAC, Σ_G will include N MAC tags for each message tuple $(x^{(t^*)}_{i\to j}, \mathsf{m})$, binding the tokens to m. The server checks that Σ_G maps to a specific column of public tokens in its database. If so, it parses Σ_G, sets the user authentication token σ_j for user u_j as $x^{(t^*)}_{i\to j}$ and the j-th MAC tag. User u_j can verify and trace σ_j back to u_i by searching through its receiver tokens. Once the users exhaust their tokens, they perform an update by minting new one-time tokens and distributing them to the group as done above. Note that this is where we require global state updates: once the boxes in Fig. 3 become empty, every user has to update in order to fill them back again.

Informally, user traceability holds since the group authentication token Σ_G corresponds to a unique column in the database held by the server. All honest users can use this unique column index to trace the same sender.

Amortized Efficiency. To reduce the communication cost of the offline phase and make the dependence on the ciphertext size minimal (particularly important in the post-quantum regime), we replace the private tokens $(x^{(t)}_{j\to i})_{t\in[T]}$ by one PRF seed $\mathsf{seed}_{j\to i}$, allowing each user to locally derive the corresponding tokens. This reduces the number of ciphertexts by a factor of T, making the overhead in the offline communication cost to be $2\cdot\left(\frac{|\mathsf{ct}|}{T}+\kappa\right)$ per message, where ct denotes a PKE ciphertext and κ is the bit-length of the public tokens.

Adding Anonymity. Fig. 3 illustrates how if, e.g., u_2 sends two messages using private tokens from the blue box, the

server will be able to link these two messages together. Our final description of QUASAR fixes this by permuting the column indices using a permutation key derived from the group secret key. The server stills checks the group authentication tokens Σ_G with respect to a single column, and the users can map this randomly permuted column index to a unique sender.

Similarly to all the GAM protocols so far, QUASAR is non-colluding unforgeable since each users maintain a database of the public token and user pair. The added complexity only comes from adding tracing soundness while maintaining anonymity. Moreover, QUASAR is not standard unforgeable as a malicious server and malicious insider can collude to impersonate any honest user.

5.2 STARS and GEMSTARS

STARS: This is almost equivalent to QUASAR, except that it additionally achieves *standard* unforgeability by replacing the usage of one-time tokens (i.e., private and public tokens (x, y) such that $\mathsf{OWF}(x) = y$) with *one-time signatures* (OTS). Importantly, the usage must remain *one-time* as otherwise two messages sent with the same signing key becomes linkable.
GEMSTARS: This is essentially STARS without state updates. In order to remove updates without weakening security, we use a *group signature* (GS) [8, 16, 21], enabling a user to anonymously sign, while allowing a special entity called the group tracer to trace the signature back to the user. In our context, we will view the set of group users as one instantiation of the group tracer. Unfortunately, due to a mismatch in the scope of "groups" considered by the GAM protocol and GS, a naive solution does not work. We will resolve this issue by proving group membership of the GAM protocol using a standard signature, similar to the idea in the metadata-hiding protocol of [36]. Although GEMSTARS removes state updates while satisfying all the desired security, it is quite inefficient in the post-quantum setting due to the lack of efficient GS.

6 Running GAM Protocols on MLS

In this section, we explain how to integrate a GAM protocol into MLS. While this is fairly straightforward, some discussion is required since our GAM protocol assumes a trusted initialization algorithm Init that prepares the group user states. For the particular GAM protocol (i.e., Enc-Sign mode) currently used by MLS this is not an issue, since the initial user state (i.e., the group secret key and verification keys of the group users) is implicitly provided by the CGKA protocol. However, in general, this may not be the case.

Below, we explain this integration in two steps. We first consider the base case where a GAM protocol operates on an already established set of user states. We then consider how our *specific* GAM protocols can handle the Init algorithm without a trusted setup. Recall in MLS, a single user u initializes a group $\mathsf{G} = \{u\}$ and then dynamically adds users

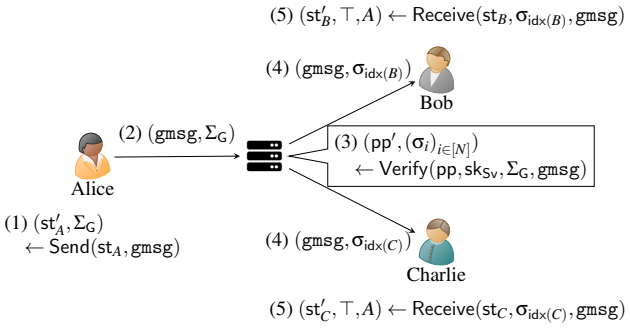

Figure 4: Using a GAM protocol on top of MLS' FSPD protocol. When retrieving a message, a user u specifies its index $\mathsf{idx}(u)$ to the server; the server then returns the user authentication token $\sigma_{\mathsf{idx}(u)}$ along with gmsg.

by sending welcome messages. We follow this approach and explain how the Init algorithm of a GAM protocol can be implemented through these dynamic group operations. As explained in Sec. 2.4, a formal model for allowing dynamic groups in GAM protocols is left as an important future work.

6.1 Authentication in a Static Group

Assume the base case where the user states of the GAM protocol are already set up. Recall an application message of MLS' FSPD protocol have the following format [7, Sec. 6]:

$$(\mathsf{gid}, \mathsf{epoch}_{\mathsf{CGKA}}, \mathsf{ct}_{\mathsf{senderID}}, \mathsf{ct}_{\mathsf{Contents}}, \mathsf{AuthData}) \quad (1)$$

gid is the group identity; $\mathsf{epoch}_{\mathsf{CGKA}}$ is a counter[9]; senderID is the sender's identity; Contents stores the payload; ct_X is an SKE encryption of X under an SKE key derived from the group secret key; and AuthData is an authentication data field, including an encryption of the signature (i.e., Enc-Sign mode). Below, we denote gmsg as Eq. (1), excluding AuthData.

With this structure in mind, using a GAM protocol on top of MLS' FSPD is straightforward. This is depicted in Fig. 4 (see also Fig. 1). To send a message gmsg, user $u \in \mathsf{G}$ runs the Send algorithm to create a group authentication token Σ_G and uploads $(\mathtt{gmsg}, \mathsf{AuthData} := \Sigma_\mathsf{G})$ to the server. The server verifies Σ_G and prepares *user* authentication tokens $(\sigma_i)_{i\in[N]}$, where $N = |\mathsf{G}|$. When a user $v \in \mathsf{G}$ with index $i = \mathsf{idx}(v)$ contacts the server, the server returns $(\mathtt{gmsg}, \mathsf{AuthData}_i := \sigma_i)$. User v can then verify and trace user u by running the Receive algorithm of the GAM protocol on $(\mathtt{gmsg}, \sigma_i)$.

Lastly, in case the GAM protocol requires state updates (e.g., QUASAR and STARS), this can be simply run on top of FSPD by directly embedding the group update information $\widehat{\Sigma}_\mathsf{G}$ into Contents or $\mathsf{ct}_{\mathsf{Contents}}$ depending on the anonymity guarantee we require.

[9] Note that this $\mathsf{epoch}_{\mathsf{CGKA}}$ is maintained by the CGKA protocol and differs from those used by QUASAR and STARS.

6.2 Authentication in a Dynamic Group

We now discuss how to add users to a preexisting GAM protocol using the welcome message functionality provided by MLS (or the CGKA protocol to be precise). Similarly to how the CGKA protocol is implemented, this procedure can be used to execute the Init algorithm. Since the process is inherently protocol specific, we explain them individually below.

COSMOS. This is simple since each user state is independent of the others. When an outsider joins the group via a welcome message, it mints a new token (x_0, y_0) and uploads the public token y_0 to the server and group. The outsider can additionally authenticate y_0 by including it in its key package maintained by the Delivery Service [13, Sec. 5] or by directly signing it to the group with its long-term key maintained by the Authentication Service [13, Sec. 4]. Once y_0 is shared among the group, it can use x_0 to start sending messages. Moreover, the outsider can fetch the current public tokens of the other users from the server. In case they want to be sure that these public tokens were indeed minted by the respective users, they can obtain the tokens directly from the senders. Note that the process is very similar to the Enc-Sign mode currently used by MLS' FSPD protocol. The case for the optimized COSMOS is similar.

COSMAC. This is almost identical to COSMOS. The only difference is that the outsider, after processing the welcome message, recovers the current group secret key gsk. It then uses the gsk to derive the MAC key $\mathsf{k}_{\mathsf{MAC}}$ and an SKE key $\mathsf{k}_{\mathsf{SKE}}$. Note that we can update $\mathsf{k}_{\mathsf{MAC}}$ and $\mathsf{k}_{\mathsf{SKE}}$ anytime the CGKA protocol performs a commit, which would allow some form of post-compromise security of the anonymity and anonymous blocklisting properties (see Sec.8). Moreover, if we need sender anonymity of the welcome message, we can rely on existing anonymous *two-party* messaging protocols, such as Sealed Sender [43] or Orca [55], where the latter protocol also provides user traceability.

QUASAR *and* STARS. These are the most involved due to global state updates. Since the protocol flows of QUASAR and STARS are identical, we only focus on QUASAR. Looking at the toy example from Fig. 3, we cannot simply append columns corresponding to the new users since the server can always link public tokens associated to these appended columns. Thus, to ensure anonymity, the group users must all update their state and refresh the public tokens held by the server. In more detail and similarly to COSMOS, the outsider o, after having processed the welcome message, mints their one-time tokens, uploads the public tokens $\left(y^{(t)}_{j\to o}\right)_{(j,t)\in[N]\times[T]}$ to the server, and sends the PRF seeds $\mathsf{seed}_{j\to o}$ to each group user so that they can recover the corresponding private tokens. After every other user updates their states by further minting T extra public tokens for the new user o, the user can start sending messages.

An optimization of the above approach will have each group member j send a new PRF seed $\mathsf{seed}_{o\to j}$ to o, but, for each other group member i, derive the corresponding seed for the new epoch from the seed $\mathsf{seed}_{j\to i}$ (respectively $\mathsf{seed}_{i\to j}$) for the previous epoch, and upload the corresponding public tokens $y^{(t)}_{*\to j}$ to the server. This could be done by setting the new seed to be $\mathsf{OWF}(\mathsf{seed}_{j\to i}\|\mathsf{epoch}\|o)$ (respectively $\mathsf{OWF}(\mathsf{seed}_{i\to j}\|\mathsf{epoch}\|o)$). The advantage of such an approach is that each group member is only required to send a single ciphertext, and download nothing, as opposed to uploading and downloading N ciphertexts.

GEMSTARS. This is the only protocol that requires the MLS server to additionally run a group signature scheme. When a user joins the secure group messaging application, the server provides the user with a certificate (see the full version of this paper for details). Assuming this step has been finished, then joining a group is straightforward. This is because the only thing the outsider requires to run GEMSTARS is the groups tracer key and signature key, which are both derived from the group secret key gsk. Hence, following the same discussion in COSMAC, we can easily add new users to GEMSTARS.

Lastly, we note that removing users is straightforward for all of our GAM protocols. Since COSMOS is not anonymous, the server can simply maintain a list of group members. COSMAC and GEMSTARS natively supports removal at the CGKA layer since, once a commit occurs, the MAC key $\mathsf{k}_{\mathsf{MAC}}$ is updated along with the group secret key gsk. Finally, for QUASAR, the users can simply remove the unused public tokens corresponding to the removed users from the server.

7 Bandwidth Efficiency Analysis

In this section we analyze the efficiency of our proposed GAM protocols and compare them with existing authentication modes. We are specifically interested in the bandwidth overhead incurred by each authentication mode compared to a messaging protocol where the application message (e.g., chat texts) is sent in the clear, i.e., without authentication.

7.1 Instantiation

We target the NIST Level I security[10] stating that breaking the protocol is no easier than key-recovery on a block cipher with a 128-bit key (e.g., AES-128). This provides a meaningful baseline to discuss post-quantum security and ignoring quantum attacks corresponds to a classical security level of 128 bits. The main cryptographic primitives used in our GAM protocols is summarized below. The sizes of the cryptographic artifacts used in our instantiations are shown in Tab. 2.

OWF: We use SHA-256 and truncate its output to 16 B.

[10] https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/evaluation-criteria/security-(evaluation-criteria)

Table 2: Instantiation of the building blocks for our GAM protocols. SIG and GS stand for signature schemes and group signatures. sig, osig, and gsig denote the signature a standard signature, an OTS, and a group signature, respectively. ovk denotes the verification key of an OTS. ek and ct denote a KEM encapsulation key and ciphertext, respectively. All sizes are given in bytes. † indicates the output of SHA-256 is truncated to 16 B. We use κ to denote the security parameter, set to 128 bits.

Primitives	Classic		Post-quantum		Related Auth. Modes
OWF	SHA-256:	$\kappa = 16^{\dagger}$	SHA-256:	$\kappa = 16^{\dagger}$	COSMOS, COSMAC, QUASAR
MAC	HMAC-SHA-256:	$\kappa = 16^{\dagger}$	HMAC-SHA-256:	$\kappa = 16^{\dagger}$	COSMAC
OTS	EdDSA:	$\lvert ovk\rvert = 32$, $\lvert osig\rvert = 64$	WOTS^{+}:	$\lvert ovk\rvert = 1320$, $\lvert osig\rvert = 1072$	STARS
SIG	EdDSA:	$\lvert sig\rvert = 64$	Dilithium:	$\lvert sig\rvert = 2420$	Sign, Enc-Sign, Sign-Enc-Sign, GEMSTARS
GS	BBS:	$\lvert gsig\rvert = 336$	LNP:	$\lvert gsig\rvert = 9.2 \times 10^4$	GEMSTARS
KEM	Hashed ElGamal:	$\lvert ek\rvert = \lvert ct\rvert = 32$	Kyber:	$\lvert ek\rvert = 800$, $\lvert ct\rvert = 768$	QUASAR, STARS

MAC and PRF: We use HMAC [41] with SHA-256 and truncate its output to 16 B. Note that a deterministic MAC can be viewed as a PRF.

Pseudo-random permutation: We require a PRP to permute the set of NT tokens. We have the option of using either FastPRP [53] or the Thorp shuffle [47].

One-time signature schemes: For classical security, we use EdDSA [11]. For post-quantum security, we use WOTS^{+} [37, 38] used as a building block of the NIST PQC standard signature SPHINCS+ [39]. We set the Winternitz parameter $w = 16$, and use SHA-256 as the underlying hash function.

Signature schemes: For classical security, we use EdDSA [11]. For post-quantum security, we use the NIST PQC standard Dilithium [44]. We did not consider Falcon [50], another PQC standard, since Dilithium is selected as the primary algorithm and NIST recommends it for most use cases.

Group signatures: For classical security, we use the pairing-based BBS scheme [15] with the BLS12-381 pairing-friendly curve. For post-quantum security, we use the lattice-based scheme proposed by Lyubashevsky, Nguyen, and Plançon (LNP) [45].

KEM schemes: For classical and post-quantum security, we use the Hashed ElGamal KEM [25] and the NIST PQC standard Kyber [52], respectively.

7.2 Efficiency

Cost Metric. Following [5, 36], analyzing the bandwidth efficiency of MLS (and its variants), we analyze our GAM protocol through three metrics: the upload and download cost, and the total cost. Each of these costs are further broken down into offline and online costs; online (resp. offline) cost is associated to the cost of uploading and downloading contents generated by Send and Verify (resp. UpdSend and UpdVerify). We assume a user can send at most T messages once their states are updated. For protocols that require no updates, we can simply set $T = 1$ as there are no offline costs. In more detail, we have the following, where the costs are defined per user in a group of size N.

Total upload cost: The cost of uploading T outputs of Send (online) and one output of UpdSend (offline).

Total download cost: The cost of downloading NT outputs of Send (online) and N outputs of UpdSend (offline).

Total cost: The sum of the upload and download costs.

The download cost is a factor N times larger than the upload cost since each user has $N = |\mathsf{G}|$ users to download from. Here, for COSMOS^{++}[11], COSMAC^{++}, and GEMSTARS, we can slightly optimize the download cost by allowing the users to not download the messages they upload. In contrast, for QUASAR and STARS, the users *must* download what they uploaded to preserve anonymity. At a high level, looking at Fig. 3, if a user does not download what it uploaded, then the server can link the (permuted) columns together. Lastly, observe that even if the offline cost is larger compared to the online cost, it gets amortized by T: as T grows larger, the total online cost starts to dominate the total offline cost.

Comparison of Communication Costs. We analyze the communication cost of our proposed GAM protocols and compare them with existing authentication modes. The number of exchanged cryptographic elements in the GAM protocols is summarized in Tab. 3. We classify the GAM protocols into the following three categories and compare them.

(1) Non-anonymous protocols: Sign mode and COSMOS^{++}.

(2) Anonymous protocols without tracing soundness: Enc-Sign mode, Sign-Enc-Sign mode, and COSMAC^{++}.

(3) Anonymous protocols with tracing soundness: QUASAR, STARS, and GEMSTARS.

Table 3: The number of total cryptographic elements exchanged in GAM protocols. N is the group size and T is the number of online messages per one offline update. κ denotes the security parameter. sig, osig, and gsig denote a standard signature, a one-time signature, and a group signature, respectively. ovk denotes the verification key of a one-time signature. ct denotes a KEM ciphertext.

	Offline						Online							
	Upload			Download			Upload				Download			
Auth. Mode	κ	|ovk|	|ct|	κ	|ovk|	|ct|	κ	|osig|	|sig|	|gsig|	κ	|osig|	|sig|	|gsig|
Enc-Sign (MLS)									T					$(N-1)T$	
Sign-Enc-Sign [36]									T					$(N-1)T$	
COSMOS^{++}	3			$3(N-1)$			$2T$				$2(N-1)T$				
COSMAC^{++}	4			$4(N-1)$			$3T$				$3(N-1)T$				
QUASAR	$2NT$		N	$2N$		N	$2NT$				$2NT$				
STARS		NT	N	N		N		NT				NT			
GEMSTARS									T	T			$(N-1)T$	$(N-1)T$	

Table 4: Communication cost of each GAM protocols. The sizes are in bytes. N is the group size and T is the number of online messages per one offline update. In Tabs. 4a and 4b, we ignore the $O(N)$ cost of the offline phase for readability. The column "Total" is normalized by NT, denoting the total overhead cost per message. The column "PQ?" is ✗ (resp. ✓) for classical (resp. post-quantum) security. The mode "Sign" in Tab. 4a corresponds to the naïve approach of simply signing messages.

a: Non-anonymous GAM protocols

Auth. Mode	Online		Total	PQ?
	Upload	Download		
Sign	$64 \cdot T$	$64 \cdot (N-1)T$	64	✗
	$2420 \cdot T$	$2420 \cdot (N-1)T$	2420	✓
COSMOS^{++}	$32 \cdot T$	$32 \cdot (N-1)T$	$16 \cdot (2+\frac{3}{T})$	✓

b: Anonymous GAM protocols without tracing soundness

Auth. Mode	Online		Total	PQ?
	Upload	Download		
Enc-Sign (MLS)	$64 \cdot T$	$64 \cdot (N-1)T$	64	✗
	$2420 \cdot T$	$2420 \cdot (N-1)T$	2420	✓
Sign-Enc-Sign [36]	$128 \cdot T$	$128 \cdot (N-1)T$	128	✗
	$4840 \cdot T$	$4840 \cdot (N-1)T$	4840	✓
COSMAC^{++}	$48 \cdot T$	$48 \cdot (N-1)T$	$16 \cdot (3+\frac{4}{T})$	✓

c: Anonymous GAM protocols with tracing soundness

Auth. Mode	Offline		Online		Total	PQ?
	Upload	Download	Upload	Download		
QUASAR	$32 \cdot NT + 32 \cdot N$	$64 \cdot N$	$64 \cdot NT$	$32 \cdot NT$	$(96+\frac{96}{T}) \cdot \frac{N+1}{N}$	✗
	$32 \cdot NT + 768 \cdot N$	$800 \cdot N$	$32 \cdot NT$	$32 \cdot NT$	$(96+\frac{1568}{T}) \cdot \frac{N+1}{N}$	✓
STARS	$32 \cdot NT + 32 \cdot N$	$48 \cdot N$	$64 \cdot NT$	$64 \cdot NT$	$(160+\frac{80}{T}) \cdot \frac{N+1}{N}$	✗
	$1320 \cdot NT + 768 \cdot N$	$784 \cdot N$	$1072 \cdot NT$	$1072 \cdot NT$	$(3464+\frac{1552}{T}) \cdot \frac{N+1}{N}$	✓
GEMSTARS			$400 \cdot T$	$400 \cdot (N-1)T$	400	✗
			$9.4 \times 10^4 \cdot T$	$9.4 \times 10^4 \cdot (N-1)T$	9.4×10^4	✓

Category (1). Tab. 4a compares the Sign mode (cf. Footnote 3) used in MLS and COSMOS^{++}. Technically, the Sign mode is used exclusively by the CGKA protocol in MLS, and not used to authenticate the output of the FSPD protocol. Nonetheless, this mode was used to analyze MLS by Alwen et al. [3] and we view it as the vanilla non-anonymous GAM protocol. Considering that any cryptographic element added to satisfy authentication should be at least 128-bits, COSMOS^{++} is near optimal. Compared to the Sign mode, the total post-quantum communication cost is a factor 75x smaller. On the other hand, Sign mode achieves *standard* unforgeability while COSMOS^{++} does not. We believe COSMOS^{++} offers a worthwhile tradeoff between efficiency and security.

[11] We only consider the most efficient version of COSMOS and COSMAC here.

Category (2). Tab. 4b compares the Enc-Sign mode used in MLS, Sign-Enc-Sign mode [36], and COSMAC^{++}. Recall Enc-Sign mode does not achieve anonymous blocklisting while Sign-Enc-Sign and COSMAC^{++} do (see Tab. 1). Out of the three protocols, COSMAC^{++} has the lowest communication cost. Compared to the Enc-Sign mode used in MLS, the total post-quantum communication cost is a factor 50x smaller. As with COSMOS^{++}, COSMAC^{++} only achieves non-colluding unforgeability while Enc-Sign and Sign-Enc-Sign modes do.

Category (3). Tab. 4c compares QUASAR, STARS, and GEMSTARS. These are the only anonymous GAM protocols with tracing soundness. QUASAR and STARS have a variable total communication cost that becomes smaller as T (and N) increases. This is because the KEM ciphertext encrypting the

PRF seed, exchanged during the offline phase, can be used to mint T tokens. Specifically, the cost of sending a large ciphertext is amortized by the number of messages T sent in the online phase. By setting $T = 1000$, the cost of sending a KEM ciphertext relative to the total cost is only 2 B or less per message, even in the post-quantum setting.

Out of the three protocols, QUASAR provides the most total-cost-efficient protocol. In fact, QUASAR is even comparable to COSMAC^{++} that has no tracing soundness, e.g., it is 106 B when $(N,T) = (10,1000)$. While larger than QUASAR, STARS also offers a relatively small total overhead, albeit more computationally expensive due to running an OTS. The benefit of using STARS over QUASAR is that it achieves *standard* unforgeability. Lastly, while both QUASAR and STARS have an $O(N)$ online upload cost (i.e., maximum bandwidth consumption) per message, the concrete cost is only 16 KB for QUASAR even for a relatively large group of $N = 1024$. STARS uploads 64 KB and 1 MB of data in the classical and post-quantum settings, respectively. Lastly, recall one of the weakness of QUASAR and STARS are that they are only *globally* state-updatable. GEMSTARS removes updates altogether, with the cost of a larger total communication overhead; in the post-quantum setting, it is 94 KB.

8 Open Problems and Future Work

Other than those discussed in Sec. 2.4, we consider the following as interesting future work.

FS and PCS. Both forward secrecy (FS) and post-compromise security (PCS) are standard security notions in secure messaging. A natural question is then, *given the compromise of (one or more) users states to the adversary, what is the effect of this on the unforgeability, anonymity, anonymous blocklisting, and user traceability of past and future messages?* This opens interesting directions, both towards formalizing these notions and towards constructing authentication modes satisfying them.

Regarding FS, we first note that unforgeability, anonymous blocklisting, and user traceability are not relevant, as in our setting these notions are only concerned with the moment messages are processed by users.[12] Anonymity, on the other hand, is more interesting: can a state compromise allow an adversary to de-anonymize past messages from a user? The answer for both MLS and our proposals is "Yes", at least in some cases. Indeed, messages in each MLS' FSPD instantiation share an epoch and are thus all signed with the same signing key, and their sender identity and signature are encrypted with the shared secret in the epoch. In the latter, either key material is static (like in GEMSTARS), or anonymity relies on the key material that only gets rotated between CGKA epochs, like the MAC key used in COSMAC, or the permutation key used in QUASAR or STARS. Thus, natural questions are: *how do we formalize "forward anonymity"?* and *can we design authentication modes that satisfy it?*

The matter regarding PCS for authentication is more involved, since all the security notions make sense in this setting, as indicated by the original work on PCS by Cohn-Gordon, Cremers, and Garratt [24]. For unforgeability, the work of Cremers, Hale, and Kohbrok [26] introduces the notion of PCS signatures, where key-pairs can be evolved to "heal" from a compromise. For anonymity, ideas from unlinkable sanitizable signatures [18, 30] could be useful. We leave concrete construction of a PCS GAM protocol as an interesting problem.

Optimal Security with PQ Efficiency. We provide several GAM protocols with different efficiency and security profiles, some of which offering much better post-quantum efficiency compared to the GAM protocol used in MLS, albeit weakening unforgeability. So far, the only GAM protocol satisfying optimal security (i.e., standard unforgeability, anonymity, anonymous blocklisting, tracing soundness) with no global state updates is GEMSTARS. However, this comes at a great cost as post-quantum group signatures are much more costly than signatures. We view it as an interesting open problem to find a GAM protocol achieving all the desireable properties while retaining efficiency.

[12] The concept of *forward-secret signatures* [9], motivated by the will that the compromise of the current secret key does not enable an adversary to forge signatures pertaining to the past, is thus not relevant.

Acknowledgments

This research was partially supported by JST CREST JPMJCR22M1, Japan and funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No.665385.

Ethical Considerations

Our work follows the ethical guidelines of the conference. It proposes improvements for a popular and widely studied IETF standard: MLS. Most of the improvements target efficiency, and the formalization of authenticity notions, while new, does not allow for the creation of non-previously-known primitives that could be seen as bringing controversial or negative consequences, i.e., anonymous (the only notion that could be seen as potentially having a negative outcome) messaging systems already exist and, in particular, anonymity on the FSPD layer is already an aim of MLS. All the other authenticity notions enabled by this work are hard to imagine to have a negative outcome if implemented in the real world. Finally, our work did not include any experiments with live systems and does not lead to negative results for already used or implemented systems.

Open Science Policy

We do not have any artifacts (e.g., datasets, scripts, binaries) related to this paper.

References

[1] Joël Alwen, Benedikt Auerbach, Miguel Cueto Noval, Karen Klein, Guillermo Pascual-Perez, Krzysztof Pietrzak, and Michael Walter. CoCoA: Concurrent continuous group key agreement. In Dunkelman and Dziembowski [29], pages 815–844.

[2] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 248–277. Springer, Heidelberg, August 2020.

[3] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Modular design of secure group messaging protocols and the security of MLS. In Vigna and Shi [56], pages 1463–1483.

[4] Joël Alwen, Sandro Coretti, Daniel Jost, and Marta Mularczyk. Continuous group key agreement with active security. In Pass and Pietrzak [48], pages 261–290.

[5] Joël Alwen, Dominik Hartmann, Eike Kiltz, and Marta Mularczyk. Server-aided continuous group key agreement. In Yin et al. [59], pages 69–82.

[6] Joël Alwen, Daniel Jost, and Marta Mularczyk. On the insider security of MLS. In Dodis and Shrimpton [28], pages 34–68.

[7] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. RFC 9420, July 2023.

[8] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629. Springer, Heidelberg, May 2003.

[9] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 431–448. Springer, Heidelberg, August 1999.

[10] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Heidelberg, August 1994.

[11] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 124–142. Springer, Heidelberg, September / October 2011.

[12] Ward Beullens, Samuel Dobson, Shuichi Katsumata, Yi-Fu Lai, and Federico Pintore. Group signatures and more from isogenies and lattices: Generic, simple, and efficient. In Dunkelman and Dziembowski [29], pages 95–126.

[13] Benjamin Beurdouche, Eric Rescorla, Emad Omara, Srinivas Inguva, and Alan Duric. The Messaging Layer Security (MLS) Architecture. Internet-Draft draft-ietf-mls-architecture-15, Internet Engineering Task Force, August 2024. Work in Progress.

[14] Karthikeyan Bhargavan, Benjamin Beurdouche, and Prasad Naldurg. Formal Models and Verified Protocols for Group Messaging: Attacks and Proofs for IETF MLS. Research report, Inria Paris, December 2019.

[15] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, Heidelberg, August 2004.

[16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, and Jens Groth. Foundations of fully dynamic group signatures. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16*, volume 9696 of *LNCS*, pages 117–136. Springer, Heidelberg, June 2016.

[17] Chris Brzuska, Eric Cornelissen, and Konrad Kohbrok. Security analysis of the MLS key derivation. In *2022 IEEE Symposium on Security and Privacy*, pages 2535–2553. IEEE Computer Society Press, May 2022.

[18] Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Unlinkability of sanitizable signatures. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 444–461. Springer, Heidelberg, May 2010.

[19] Ran Canetti and Hugo Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 143–161. Springer, Heidelberg, August 2002. https://eprint.iacr.org/2002/120/.

[20] Melissa Chase, Trevor Perrin, and Greg Zaverucha. The signal private group system and anonymous credentials supporting efficient verifiable encryption. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1445–1459. ACM Press, November 2020.

[21] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 257–265. Springer, Heidelberg, April 1991.

[22] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 451–466, 2017.

[23] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. *Journal of Cryptology*, 33(4):1914–1983, October 2020.

[24] Katriel Cohn-Gordon, Cas J. F. Cremers, and Luke Garratt. On post-compromise security. In Michael Hicks and Boris Köpf, editors, *CSF 2016 Computer Security Foundations Symposium*, pages 164–178. IEEE Computer Society Press, 2016.

[25] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[26] Cas Cremers, Britta Hale, and Konrad Kohbrok. The complexities of healing in secure group messaging: Why cross-group effects matter. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 1847–1864. USENIX Association, August 2021.

[27] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In Matt Blaze, editor, *USENIX Security 2004*, pages 303–320. USENIX Association, August 2004.

[28] Yevgeniy Dodis and Thomas Shrimpton, editors. *CRYPTO 2022, Part II*, volume 13508 of *LNCS*. Springer, Heidelberg, August 2022.

[29] Orr Dunkelman and Stefan Dziembowski, editors. *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*. Springer, Heidelberg, May / June 2022.

[30] Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part I*, volume 9614 of *LNCS*, pages 301–330. Springer, Heidelberg, March 2016.

[31] Signal Foundation. Signal protocol: Technical documentation. (Accessed on 04/25/2023).

[32] GreenNet. Understanding file sizes. https://www.greennet.org.uk/support/understanding-file-sizes. (Accessed on 04/25/2023).

[33] Neil M. Haller. The S/KEY one-time password system. In *Proceedings of the ISOC Symposium on Network and Distributed System Security*, February 1994.

[34] Neil M. Haller. The S/KEY One-Time Password System. RFC 1760, February 1995.

[35] Keitaro Hashimoto, Shuichi Katsumata, Eamonn Postlethwaite, Thomas Prest, and Bas Westerbaan. A concrete treatment of efficient continuous group key agreement via multi-recipient PKEs. In Vigna and Shi [56], pages 1441–1462.

[36] Keitaro Hashimoto, Shuichi Katsumata, and Thomas Prest. How to hide MetaData in MLS-like secure group messaging: Simple, modular, and post-quantum. In Yin et al. [59], pages 1399–1412.

[37] Andreas Hülsing. W-OTS+ - shorter signatures for hash-based signature schemes. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *AFRICACRYPT 13*, volume 7918 of *LNCS*, pages 173–188. Springer, Heidelberg, June 2013.

[38] Andreas Hülsing. WOTS+ – shorter signatures for hash-based signature schemes. Cryptology ePrint Archive, Report 2017/965, 2017. https://eprint.iacr.org/2017/965.

[39] Andreas Hulsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kolbl, Tanja Lange, Martin M Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. SPHINCS+. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[40] Karen Klein, Guillermo Pascual-Perez, Michael Walter, Chethan Kamath, Margarita Capretto, Miguel Cueto, Ilia Markov, Michelle Yeo, Joël Alwen, and Krzysztof Pietrzak. Keep the dirt: Tainted TreeKEM, adaptively and actively secure continuous group key agreement. In *2021 IEEE Symposium on Security and Privacy*, pages 268–284. IEEE Computer Society Press, May 2021.

[41] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-hashing for message authentication. IETF Internet Request for Comments 2104, February 1997.

[42] Julia Len, Esha Ghosh, Paul Grubbs, and Paul Rösler. Interoperability in end-to-end encrypted messaging. Cryptology ePrint Archive, Report 2023/386, 2023. https://eprint.iacr.org/2023/386.

[43] Joshua Lund. Technology preview: Sealed sender for signal. https://signal.org/blog/sealed-sender/, October 2018. (Accessed on 04/19/2023).

[44] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[45] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Maxime Plançon. Lattice-based zero-knowledge proofs and applications: Shorter, simpler, and more general. In Dodis and Shrimpton [28], pages 71–101.

[46] Jaroslav Martan. Webex meetings security. Technical report, September 2021. https://www.cisco.com/c/dam/m/cs_cz/training-events/webinars/tech-club-webinars/webex-meetings-security.pdf.

[47] Ben Morris, Phillip Rogaway, and Till Stegers. Deterministic encryption with the Thorp shuffle. *Journal of Cryptology*, 31(2):521–536, April 2018.

[48] Rafael Pass and Krzysztof Pietrzak, editors. *TCC 2020, Part II*, volume 12551 of *LNCS*. Springer, Heidelberg, November 2020.

[49] Trevor Perrin. The noise protocol framework, 2018. (Accessed on 04/23/2023).

[50] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[51] Guardian Project. Orbot: Proxy with tor. (Accessed on 04/19/2023).

[52] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[53] Emil Stefanov and Elaine Shi. FastPRP: Fast pseudorandom permutations for small domains. Cryptology ePrint Archive, Report 2012/254, 2012. https://eprint.iacr.org/2012/254.

[54] Roman Tobe. Ringcentral expands end-to-end encryption to phone and messaging. https://www.ringcentral.com/us/en/blog/ringcentral-update-e2ee-2022/, 2022. (Accessed on 04/25/2023).

[55] Nirvan Tyagi, Julia Len, Ian Miers, and Thomas Ristenpart. Orca: Blocklisting in sender-anonymous messaging. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 2299–2316. USENIX Association, August 2022.

[56] Giovanni Vigna and Elaine Shi, editors. *ACM CCS 2021*. ACM Press, November 2021.

[57] Matthew Weidner. Group messaging for secure asynchronous collaboration. Mphil dissertation, University of Cambridge, Cambridge, UK, 2019.

[58] Shouhuai Xu and Moti Yung. Accountable ring signatures: A smart card approach. In *Smart Card Research and Advanced Applications VI*, pages 271–286. Springer, 2004.

[59] Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors. *ACM CCS 2022*. ACM Press, November 2022.