TEAKEX: TESLA-Authenticated Group Key Exchange

Qinyi Li[1], Lise Millerjord[2], and Colin Boyd[2]

[1]Griffith University, Brisbane, Australia. qinyi.li@griffith.edu.au
[2]NTNU – Norwegian University of Science and Technology, Trondheim, Norway.
colin.boyd@ntnu.no, lise.millerjord@ntnu.no

Abstract

We present a highly efficient authenticated group key exchange protocol, TEAKEX, using only symmetric key primitives. Our protocol provides proven strong security, including forward secrecy, post-compromise security, and post-quantum security. For online applications we claim that TEAKEX is much simpler and more efficient than currently available alternatives. As part of our construction we also give a new abstract security definition for delayed authentication and describe its instantiation with the TESLA protocol.

1 Introduction

The purpose of a group key exchange (GKE) protocol is to securely establish a shared key for parties which are members of the group. In contrast to two-party key exchange protocols, GKE protocols often include mechanisms to securely change the group membership by adding new parties to the group or removing existing parties. These mechanisms can be much more efficient than setting up a new shared key from scratch. Thus GKE protocols with such mechanisms are very useful when group membership is dynamic.

In pioneering work in the early 2000s, Bresson, Chevassut and Pointcheval [6] (hereafter BCP) developed the first computational security analysis of GKE, including the dynamic case. They proposed a protocol based on Diffie-Hellman (DH) and proved its security in their model. However, their proposal has a number of limitations as follows.

1. Users need to store their ephemeral secrets (DH components) to allow adding or removing of parties.
2. Their security model only allows corruption of long-term secrets and not user state variables; otherwise, forward secrecy is lost.
3. Their protocol to add members is still linear in the size of the group.
4. The use of DH prevents the protocol from achieving post-quantum security.

Another significant development in the evolution of GKE was the idea of key trees. The notion was developed and analyzed independently by different authors late in the last millenium [15, 16]. The main advantage of using key trees is that group members can be added or deleted using only a logarithmic (in the number of parties) number of messages, while ensuring that added parties cannot read old group messages and removed parties cannot read new group messages. Individual parties share a long-term key with the trusted key server. These early papers [15, 16] did not include a formal security model or consider the effects of compromise of group members. The goal of this paper is to propose and prove secure a new GKE protocol based on key trees with symmetric cryptography and using the TESLA authentication protocol [14] for key authentication.

1.1 Continuous Group Key Exchange

Continuous group key exchange (CGKE) can be viewed as an extension of dynamic GKE. In many modern applications, conversations between parties are long-lived, frequently years in length. It is natural that group membership changes over time, so it is an efficient choice to set up an initial key and then evolve the key when parties join or leave. Since the session may be long-lived, it is also natural that the group key is updated periodically, even when group membership remains unchanged. CGKE sessions are therefore divided into epochs, where each epoch has a new independent group key. This leads to the following general definition of a CGKE protocol [2].

Definition 1. *A* continuous group key exchange protocol *consists of a set of potential participants* Ω *and a set of protocols* (Gen, Initialize, Add, Remove, Update)*. During the protocol execution, each user maintains an evolving state that may include short-term key materials for encryption/decryption and signing/authentication (depending on the actual protocol) as well as a value used as the current group communication key. Users may also have a long-term key which we regard as stored separately from the evolving state.*

- Gen*: The algorithm generates keys for all parties in* Ω*. The keys may be used for encryption and/or authentication (or signing) or other tasks.*
- Initialize*: initialize a communication group* $\mathcal{G} = \{U_1, U_2, \ldots, U_n\}$ *with* n *users and establish the user states for all users.*
- Add(U)*: On input an entity* $U \notin \mathcal{G}$*, the protocol creates* U*'s state and updates the states of other users in* $\mathcal{G}$*.*
- Remove(U)*: On input an entity* $U \in \mathcal{G}$*, the protocol removes* U*'s membership in* $\mathcal{G}$ *and updates the states of other users in* $\mathcal{G}$*.*
- Update(U)*: The protocol re-creates the state of user* $U \in \mathcal{G}$ *and updates the states of other users in* $\mathcal{G}$*. (The effect is the same as performing* Remove(U) *and then* Add(U)*.)*

The application target motivating the first definitions of CGKE was group messaging [2], notably in the context of Message Layer Security (MLS) standardisation [4]. Communications in such a context are naturally *asynchronous* since some parties will be offline at certain times. Our new protocol requires parties to be online; for reasons which will soon become apparent, parties need to process messages relatively quickly. This means that group messaging may not be a good application for our new protocol. However, we believe that there are many practical group communication scenarios where online working is normal, for example in working groups in corporations. Note that the BCP protocols [6] also require online parties, so in a direct comparison between BCP and our protocol we achieve both stronger security and much better efficiency.

1.2 Post-Compromise Security

When a GKE protocol session is long-lived and involves a large number of parties there is an increased likelihood that some parties will be compromised at some time. One concern is that an adversary who obtains the current group key should not be able to read group messages sent in the past, which defines forward secrecy (FS). In our protocol, as in MLS, FS is achieved because each Add, Remove or Update operation results in a new epoch with a new group key. The key material (tree root) from previous epochs has been deleted so knowledge of the current tree state is of no help to the adversary for finding previous keys.

In addition to FS, we would like to allow sessions to recover from compromise. Post-Compromise Security (PCS) [7] is the property that party U can recover private communications after the Update(U) operation. In order to achieve PCS, something must remain

hidden from the adversary. In our protocol, as in MLS, we will assume that the *long-term key* of each party remains secure even during compromise. This may be reasonable in practice if long-term keys are stored in tamper-proof devices and can only be used to execute certain operations.

In our security model, as in the MLS models [2, 11, 1], we assume that the adversary can corrupt parties and obtain any of the evolving state data. Whenever the keys in the tree are updated (using an Add, Remove or Update operation) the new values need to be authenticated so that any party can verify them. In MLS this is achieved by digital signatures, but we want to avoid public key operations. When using symmetric key cryptography, authentication is typically achieved using a message authentication code (MAC), but that is not satisfactory here since the same MAC key would then have to be shared with all group parties – compromise of that one key would then permanently break all parties who may participate. Our solution to this dilemma is to authenticate Add, Remove and Update operations using the TESLA protocol; while using only hash functions and MACs, it has the remarkable property that any party can verify authentic messages from the group manager, but the verification information can be public.

1.3 Contributions

Our *main contribution* is a new continuous group key agreement protocol, which we call TEAKEX, with features of:

- high efficiency, namely with only symmetric key primitives;
- post-quantum security, which essentially comes for free due to use of only symmetric key primitives;
- strong security, including forward secrecy and post-compromise security;
- a relatively simple security proof.

A *secondary contribution* is an abstract security definition for delayed authentication and its instantiation with TESLA.

2 Definitions and Background

This section gives some necessary definitions and then reviews related work on group key exchange, the TESLA protocol and post-quantum primitives.

2.1 Symmetric-Key Encryption

Definition 2. *Let λ be the security parameter. A symmetric-key encryption (SKE) scheme $\mathcal{E} = (\mathsf{Encrypt}, \mathsf{Decrypt})$ with plaintext space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ contains two p.p.t algorithms. $\mathsf{Encrypt}(k, m)$ encrypts the message $m \in \mathcal{M}$ using a key $k \in \{0,1\}^\lambda$ and gets the ciphertext $c \in \mathcal{C}$. The deterministic algorithm $\mathsf{Decrypt}(k, c)$ returns $m \in \mathcal{M}$, or $\perp$. We require the standard correctness for SKE, i.e., for any $k \leftarrow \{0,1\}^\lambda$, $m \in \mathcal{M}$*

$$\Pr[\mathsf{Decrypt}(k, \mathsf{Encrypt}(k, m)) = m] \geq 1 - \mathsf{negl}(\lambda)$$

where the probability is over the choices of k, and the random coins of Encrypt*. For security, we require the standard ciphertext pseudorandomness.*

Definition 3. *Let λ be the security parameter. We say that an SKE scheme $\mathcal{E} = (\mathsf{Encrypt}, \mathsf{Decrypt})$ has ciphertext pseudorandomness if for all p.p.t $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathsf{indr}}_{\mathcal{E},\mathcal{A}}(\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{indr}}_{\mathcal{E},\mathcal{A}}(\lambda) = 1] - 1/2| \leq \mathsf{negl}(\lambda),$$

where the experiment $\mathsf{Exp}^{\mathsf{indr}}_{\mathcal{E},\mathcal{A}}(\lambda)$ is defined in Fig. 1.

Experiment $\mathsf{Exp}^{\mathrm{indr}}_{\mathcal{E},\mathcal{A}}(\lambda)$:

1. $k \leftarrow \{0,1\}^{\ell}$, $b \leftarrow \{0,1\}$
2. $b' \leftarrow \mathcal{A}^{\mathcal{O}_k(\cdot,b)}(1^{\lambda})$
3. Return $(b' = b)$

Oracle $\mathcal{O}_k(m,b)$:

1. Return $c \leftarrow \mathsf{Encrypt}(k,m) \in \mathcal{C}$ if $b = 1$; Otherwise, return random $c \leftarrow \mathcal{C}$

Experiment $\mathsf{Exp}^{\mathsf{ind}}_{\mathsf{KDF},\mathcal{A}}(\lambda)$:

1. $k \leftarrow \mathcal{K}$, $b \leftarrow \{0,1\}$
2. $(x, st) \leftarrow \mathcal{A}(\cdot)(1^{\lambda})$
3. $y_0 \leftarrow \mathsf{KDF}(k,x)$, $y_1 \leftarrow \{0,1\}^{\ell}$
4. $b' \leftarrow \mathcal{A}(st, y_b)$
5. Return $(b' = b)$

Figure 1: Security Experiment for SKE and KDF

Experiment $\mathsf{Exp}^{\mathrm{euf-cma}}_{\mathsf{MAC},\mathcal{A}}(\lambda)$:

1. $\mathcal{L} \leftarrow \emptyset$,$k \leftarrow \mathsf{MacGen}(1^{\lambda}, T)$
2. $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\lambda)$
3. Return 1 if $\mathsf{MacVer}(k, m^*, \sigma^*) = 1$ and $(m^*, \sigma^*) \notin \mathcal{L}$; Else, return 0

Oracle $\mathcal{O}(m)$:

1. Return $\sigma \leftarrow \mathsf{Tag}(k,m)$
2. $\mathcal{L} \leftarrow \mathcal{L} \cup (m, \sigma)$

Figure 2: Security Experiment of CMA

2.2 Key Derivation Functions

Key derivation functions (KDFs) extract pseudorandom keys from short random sources. We simplify the definition by Krawczyk [12] by assuming a perfect λ-bit source is available where λ is the security parameter, and each sample is only used one-time. This is sufficient as every group communication key of TEAKEX is computed via KDF using a freshly generated key (seed). We emphasise that this is to simplify the security proof. We suggest instantiating KDF using HKDF [12].

Definition 4. *Let λ be the security parameter. Let $\mathcal{K}$ be a set with size $\{0,1\}^{\geq\lambda}$. We say a key derivation function $\mathsf{KDF} : \mathcal{K} \times \mathcal{X} \rightarrow \{0,1\}^{\ell}$ is secure if the advantage*

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathsf{KDF},\mathcal{A}}(\lambda) = |\mathsf{Exp}^{\mathsf{ind}}_{\mathsf{KDF},\mathcal{A}}(\lambda) = 1| \leq \mathsf{negl}(\lambda)$$

where the experiment $\mathsf{Exp}^{\mathsf{ind}}_{\mathsf{KDF},\mathcal{A}}(\lambda)$ is defined in Fig. 1

2.3 Message Authentication Code

Definition 5. *A message authentication code (MAC) scheme* MAC *has three p.p.t algorithms.* $\mathsf{MacGen}(1^{\lambda})$ *returns a secret key k.* $\mathsf{MacSign}(K,m)$ *returns a tag (a.k.a authenticator) σ on a message m.* $\mathsf{MacVer}(k,m,\sigma)$ *returns 0 or 1. The correctness of* MAC *requires that for all $k \leftarrow \mathsf{MacGen}(1^{\lambda})$:*

$$\Pr[\mathsf{MacVer}(k, m, \mathsf{MacSign}(k,m) = 1] \geq 1 - \mathsf{negl}(\lambda)$$

where $\mathsf{negl}(\lambda)$ is negligible, the probability is over the algorithms' random coins.

Definition 6. *We say* $\mathsf{MAC} = (\mathsf{MacGen}, \mathsf{MacSign}, \mathsf{MacVer})$ *has unforgeability if for all p.p.t adversary $\mathcal{A}$, the advantage*

$$\mathsf{Adv}^{\mathsf{euf-cma}}_{\mathsf{MAC},\mathcal{A}} := \Pr[\mathsf{Exp}^{\mathsf{euf-cma}}_{\mathsf{MAC},\mathcal{A}}(\lambda) = 1]$$

is negligible in λ where $\mathsf{Exp}^{\mathsf{euf-cma}}_{\mathsf{MAC},\mathcal{A}}(\lambda)$ is defined in Fig. 2. And, we say that MAC *is pseudorandom if for all p.p.t $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{pseudor}}_{\mathsf{MAC},\mathcal{A}}(\lambda)$, defined as*

$$|\Pr[\mathcal{A}^{\mathsf{MacSign}(k,\cdot)}(\lambda) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(\lambda) = 1]|$$

is negligible in λ where $k \leftarrow \mathsf{MacGen}(1^{\lambda})$ and $f \leftarrow_{\$} F$ is a random function from the message space to the tag space.

2.4 Group Key Exchange Security

Our security model for TEAKEX, detailed in Section 6, can usefully be compared with that of BCP01 [5]. The main security goal and adversary capabilities are the same so we here highlight only some key differences.

- TEAKEX, like MLS, is a CGKE protocol so that it has an Update operation in addition to the Add and Remove operations in the BCP protocols. Therefore, like MLS models [2, 11, 1], our security model also gives the adversary the ability to query Update operations.
- Corruption in the BCP model allows the adversary access to the long-term key of the corrupted party, not the ephemeral (state) data. Like MLS models [2, 11, 1], our model does the opposite: Corrupt returns only the evolving state data, not the long-term secret (the key shared with the group manager in TEAKEX, the signing key in MLS). For reasons discussed in Sec. 1.2 we believe that our choice is more natural and allows us to model PCS.

Note that despite the different terminology, the adversarial Send query in BCP models is equivalent to the Deliver/Process query in MLS models [2, 11, 1]. Both queries allow the adversary to control whether parties receive protocol messages and to observe the response, thereby modelling active attacks.

One cause of additional complexity in GKE security models is the possibility of *insider attacks* [10, 3] where valid protocol participants deviate from their defined protocol behaviour. As discussed by Alwen et al. [3], in some security models used for MLS adversarial control of the network was restricted and an idealised public key infrastructure (PKI) was assumed. We emphasise that, due to the simplicity of our protocol, such attacks are impossible. PKI is not needed during the TEAKEX protocol execution; and only the group manager can initiate any evolution of the state material for any party. Our adversary is allowed full control of network messages through its Send query.

It may be noticed that TEAKEX, like MLS, does not allow all parties to influence the group key, a property sometimes called *contributiveness*. Arguably this is not a core security property and seems impossible to achieve in any CGKE without severely impacting efficiency.

2.5 The TESLA protocol

TESLA [14, 9] is an acronym for Timed Efficient Stream Loss-tolerant Authentication. The idea is elegant and simple: the authenticator (server) generates a hash chain and (authentically) delivers the end of the hash chain to all parties who need to authenticate messages. Each message is then sent with a MAC keyed with the previous value in the hash chain. After the message and MAC are sent, the previous value in the hash chain is released so that the recipient can check it using the endpoint and then verify that MAC with the derived key.

Loosely synchronized clocks are a critical requirement to make TESLA secure. This assumption is reasonable, but only if protocol parties are online whenever key updates occur. This restricts the applicability of our TEAKEX protocol, but we believe that the increasingly reliable and online nature of real-world communications makes online working reasonable in many practical applications.

In Section 3, we give an abstract formal definition of message authentication with delayed key disclosure and show that it is satisfied by the TESLA protocol. As far as we are aware, this is the first formal analysis of TESLA. It allows us to treat server messages as formally authenticated in our protocol security proof.

2.6 Postquantum Secure Cryptography

Upgrading cryptographic primitives to avoid the impending threat of quantum computers has been one of the main research thrusts in cryptography in recent years. This effort has focussed almost exclusively on public key primitives because the previously standardised public key encryption and digital signature schemes are known to be vulnerable to quantum attacks.

In 2024, following the approval of the first standards for post-quantum secure public key systems, NIST drafted a strategy for transition to post-quantum cryptography [13]. According to this document, current NIST-approved symmetric-key primitives are believed to provide adequate security against quantum computers. For this reason, we claim that our protocol TEAKEX will also inherit post-quantum security by using currently standardised symmetric encryption, MAC and hashing algorithms. Although we have not included a formal analysis against a quantum adversary, we assert that our formal reduction to symmetric primitives shows that a successful quantum adversary against our protocol would imply a successful attack on at least one of those primitives.

3 Continuous Message Authentication

In this section, we will define the notion of continuous message authentication with delayed key disclosure (CMA-DKD), which is a crucial part of our construction. Then we will show that in practice, we can regard the TESLA protocol [14, 9] as a secure CMA-DKD.

Definition 7. *A* continuous message authentication scheme with delayed key disclosure*, CMA-DKD, consists of three p.p.t algorithms.*

- $\mathsf{KGen}(1^\lambda, T)$*: the key generation algorithm* Gen *takes as input the security parameter,* λ*, and the maximum number of time intervals* T*. It returns a set of parameters,* Param*, and a set of secret keys,* $\{AK_i\}_{i\in[T]}$*.*
- $\mathsf{Auth}(AK_i, i, m)$*: For* $i = \{T, T-1, ..., 1\}$ *the algorithm* Auth *picks the index* i *such that* AK_i *hasn't been used for* Auth*; It runs two subroutines:*
  1. $\mathsf{Tag}(AK_i, i, m)$*: Let* m *be the message to be authenticated in the* i*-th time interval, the algorithm applies the* i*-th authentication key,* AK_i*, and returns an authenticator,* σ*.*
  2. $\mathsf{Disclose}(i)$*: The secret disclosing algorithm takes as input the current time interval number* i *and returns the* i*-th authentication key,* AK_i*.*

  The values of (i, σ, m) *and then, after a suitable delay,* AK_i*, are sent to the recipient(s).*
- $\mathsf{Ver}(\mathsf{Param}, r, i, \sigma, m)$*: the verification algorithm runs three subroutines:*
  1. $\mathsf{KVer}(\mathsf{Param}, i, r)$*: The key verification algorithm takes as input* Param*, a time interval number* $i \leq T$*, and a string* r*. It returns 1 if* $r = AK_i$*, or 0, otherwise.*
  2. $\mathsf{MVer}(\mathsf{Param}, AK_i, i, \sigma_i, m)$*: The verification algorithm takes as input* Param*,* σ_i*,* m*, and a key,* AK_i*, and returns 1 or 0.*
  3. $\mathsf{Compatible}(AK_i, i, \sigma, m)$*: The compatibility checking predicate returns 1 if* AK_i *is compatible with* (i, σ, m)*, or 0, otherwise.*

  The verification algorithm returns 1 if and only if $1 = \mathsf{Compatible}(AK_i, i, \sigma, m)$*,* $1 = \mathsf{KVer}(\mathsf{Param}, i, r)$ *and* $1 = \mathsf{MVer}(\mathsf{Param}, AK_i, i, \sigma, m)$*, or 0 otherwise.*

The predicate Compatible can be different for different constructions. For the TESLA authentication protocol (see below), a mild time synchronization between the sender and receiver(s) is established. To be compatible, when an authenticator σ is received with message m during time interval i, then the key AK_i can only be released after σ and m are received.

For *correctness*, we require that for properly generated parameters, both key verification and message verification will succeed, i.e., for all $i \in [T]$, $r = AK_i$,

$$\Pr[\mathsf{KVer}(\mathsf{Param}, i, r) = 1] \geq 1 - \mathsf{negl}(\lambda)$$

and

$$\Pr[\mathsf{MVer}(\mathsf{Param}, \mathsf{Tag}(m, i, AK_i), m, r)] \geq 1 - \mathsf{negl}(\lambda)$$

where the probability is over the randomness of the algorithms. Obviously, the two conditions make $\Pr[\mathsf{Ver}(\mathsf{Param}, r, i, \sigma, m)] \geq 1 - \mathsf{negl}(\lambda)$ provided that $\mathsf{Compatible}(AK_i, i, \sigma, m) = 1$.

For *security*, we require a CMA-DKD scheme to be unforgeable. That is, no adversary can forge a valid message-authenticator pair at any time interval t without knowing the authentication keys for the later time intervals.

Definition 8. *We say a CMA-DKD scheme Σ is* unforgeable *if for all p.p.t adversary $\mathcal{A}$, the advantage*

$$\mathsf{Adv}^{\mathsf{euf}}_{\Sigma,\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Exp}^{\mathsf{euf}}_{\Sigma,\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

where the experiment $\mathsf{Exp}^{\mathsf{euf}}_{\Sigma,\mathcal{A}}(\lambda)$ is defined in Fig. 3.

Experiment $\mathsf{Exp}^{\mathrm{euf}}_{\Sigma,\mathcal{A}}(\lambda)$:

1. $\mathcal{L}_1, \mathcal{L}_2 \leftarrow \emptyset$
2. $(\mathsf{Param}, \{AK_t\}_{t\in[T]}) \leftarrow \mathsf{KGen}(1^\lambda, T)$
3. $(t^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Disclose}(\cdot),\mathsf{Tag}(\cdot,\cdot)}(\mathsf{Param}, T)$
4. If $\mathsf{MVer}(\mathsf{Param}, AK_{t^*}, t^*, \sigma^*, m^*) = 1$, $t^* > \max_{t\in\mathcal{L}_1} t$, and $((m^*, \sigma^*), t^*) \notin \mathcal{L}_2$, return 1; Else, return 0

Oracle $\mathsf{Disclose}(t)$:

1. $\mathcal{L}_1 \leftarrow \mathcal{L}_1 \cup \{t\}$
2. Return AK_t

Oracle $\mathsf{Tag}(m, t)$:

1. Return $\perp$ if $t > \max_{t'\in\mathcal{L}_1} t'$
2. $\mathcal{L}_2 \leftarrow \{(m, \sigma), t\} \cup \mathcal{L}_2$
3. Return $\sigma \leftarrow \mathsf{Tag}(AK_t, t, m)$

Figure 3: Security Experiment of CMA

TESLA – An Example of CMA-DKD. The definition of CMA-DKD can be seen as an abstract of the TESLA broadcast authentication protocol [14], a well-known broadcast authentication scheme under the delayed key disclosure paradigm. The TESLA protocol requires a cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^\ell$, modelled as a random oracle, a message authentication code $\mathsf{MAC} = (\mathsf{MacGen}, \mathsf{MacSign}, \mathsf{MacVer})$ that is unforgeable and pseudorandom (as defined in Section 2.3) and MacGen selects an element from $\{0,1\}^\ell$ uniformly at random. The protocol works as follows.

- The key generation algorithm $\mathsf{KGen}(1^\lambda, T)$ randomly selects a seed $s_0 \in \{0,1\}^\ell$. For $i \in [0, T]$, compute

$$s_{i+1} = H(s_i), \tag{1}$$

  set $s_{T+1} = h$, $AK_i = s_i$ and return $\mathsf{Param} := (H, h)$ and $\{AK_i = s_i\}_{i\in[0,T]}$. Note that Param must be communicated authentically to protocol users. In some applications, this may be conveniently achieved with one-time use of a digital signature which is elided in our efficiency analysis. In other applications, an out-of-band method can be used, for example, in a factory setup for IoT devices.

- The authentication algorithm $\mathsf{Auth}(AK_i, i, M)$ does the following:
  1. Pick the first $i \in \{T, T-1, ..., 1\}$ such that s_i hasn't been used for Auth.
  2. Run $\mathsf{Tag}(AK_i, i, m)$ as $\mathsf{MacSign}(AK_i, m||i)$.
  3. Send (i, σ, m) and then, after a suitable period, AK_i to the recipients.
- The verification algorithm $\mathsf{Ver}(\mathsf{Param}, i, \sigma, M, r)$ is defined as follows.

1. Set compatibility predicate $\mathsf{Compatible}(AK_i, i, \sigma, M) = 1$ if (i, σ, M) is received in the time interval i, and AK_i is received after a certain time delay (determined by the communication network and protocol setup). Otherwise, if the condition is unsatisfied, the predicate returns 0.
2. The key verification algorithm $\mathsf{KVer}(\mathsf{Param}, i, r)$ sets $r = r_i$, and computes $\{r_{j+1} = H(r_j)\}_{j\in[i,T]}$ and returns 1 if $h = r_{T+1}$, or 0, otherwise.
3. $\mathsf{MVer}(\mathsf{Param}, AK_i, i, \sigma, m)$ returns $\mathsf{MacVer}(AK_i, M||i, \sigma)$.

The algorithm $\mathsf{Ver}(\mathsf{Param}, i, \sigma, M, r)$ returns 1 iff: $1 = \mathsf{Compatible}(AK_i, i, \sigma, M)$, $1 \leftarrow \mathsf{KVer}(\mathsf{Param}, i, r)$ and $1 \leftarrow \mathsf{MacVer}(AK_i, M||i, \sigma)$; Else, it returns 0.

Experimental work [14] has shown that even 20 years ago, TESLA performed efficiently with delays in the worst-case measured in seconds and in the best case milliseconds, all depending on the network characteristics. Note that false assumptions on the network performance cannot lead to a recipient wrongly accepting a message – only that the recipient may reject authentic messages.

Fernández-Hernández et al. [8] have analysed the concrete security of TESLA in the context of satellite communications. Below we present a simple security analysis based on the idealisation that the hash function is a random oracle. We note that in the standard [9] and also in the analysis of Fernández-Hernández et al. [8], the hashchain used in TESLA is defined in a more complex way. Specifically, instead of the chain defined by Eq. 3, they define

$$s_{i+1} = H(s_i||i-1||\alpha), \tag{2}$$

where α is a salt value chosen randomly for each hash chain and distributed as an additional element in Param. The advantage of including α is that inverting one hashchain does not result in inverting a different hashchain. Because we model H as a random oracle this advantage is not evident in our analysis, but in practice it would be worthwhile to consider using this standardised version of TESLA.

Theorem 1. *If the message authentication code* $\mathsf{MAC} = (\mathsf{MacGen}, \mathsf{MacSign}, \mathsf{MacVer})$ *is unforgeable and the hash function H is a random oracle, the TESLA protocol described above is a CMA-DKD scheme with delayed key disclosure, and it is unforgeable. In particular,* $\mathsf{Adv}^{\mathsf{euf}}_{\mathsf{TESLA},\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$.

Proof. (Sketch) We outline the proof here. First of all, the construction demonstrates that TESLA is a CMA-DKD scheme with delayed key disclosure. For the unforgeability per Definition 8, notice that the definition essentially requires that no adversary should be able to forge an authentication tag (authenticator) on any message without knowing the key of that time interval and its later time intervals. Let i^* be the challenge time interval. Since $AK_{i+1} = H(AK_i)$ for $i \in [i^*, T]$, H is a random oracle, AK_i for $i < i^*$ are not released, the only information about AK_{i^*} are from the tags generates signed by AK_i, $i > i^*$. However, MAC is pseudorandom, meaning that those tags do not leak information about AK_{i^*} either. Hence, AK_{i^*} remains hidden from the adversary. Then, the theorem follows as the MAC is unforgeable under random and secret keys. □

4 The TEAKEX Protocol

In this section, we describe our CGKA protocol, TEAKEX. We provide the necessary notations and definitions followed by the details of our protocol.

4.1 Preliminaries and Notations

Our protocol uses full binary trees. The *depth* of a binary tree is defined as the largest number of edges from the root node to a leaf. An *internal node* of a binary tree is a

non-leaf node. The protocol involves a group of users with a group manager G denoted by $\mathcal{G} = \{G, U_1, ..., U_n\}$. It assigns each user to a leaf of the binary tree. Each user/leaf node U is associated with k_{GU}, the long-term key shared between G and U. Give a leaf U, we denote as $U \rightsquigarrow \mathsf{root}$ the path from the node U to the root root. During the protocol, internal nodes are associated with cryptographic keys. We denote the set of $\log_2(n)$ keys associated with the internal nodes on $U \rightsquigarrow \mathsf{root}$ by $\vec{k}_{U \rightsquigarrow \mathsf{root}}$. The key associated with the root node is denoted by k_{root}. (Note $\vec{k}_{U \rightsquigarrow \mathsf{root}}$ does not include the leaf key associated with U, i.e., k_{GU}.) We assume suitable time synchronization among the protocol participants is in place for the CMA-DKD scheme.

Party and Protocol States. Each party U of the protocol (excluding the group manager G) holds the user state:

$$State_U = \Big(\mathcal{G}, (\mathsf{Param}, t), \vec{k}_{U \rightsquigarrow \mathsf{root}}, (e, \mathsf{K}_e)\Big)$$

and the group manager G holds the protocol state:

$$State_{\mathcal{G}} = \Big(\mathcal{G}, (\mathsf{Param}, \{AK_i\}_{i \in [T]}, t), \{k_{GU}\}_{U \in \mathcal{G}}, \{\vec{k}_{U \rightsquigarrow \mathsf{root}}\}_{U \in \mathcal{G}}, (e, \mathsf{K}_e)\Big)$$

where 1) $\mathcal{G}$ is the current group membership including G, the group manager, 2) Param, $\{AK_i\}_i \in [T]$, t are, respectively, the public parameters, the set of secret keys, and the current time interval of the CMA-DKD scheme; 3) k_{GU} is the symmetric key U shares with G; 4) $\vec{k}_{U \rightsquigarrow \mathsf{root}}$ is the set of node keys associated to the nodes on the path $U \rightsquigarrow \mathsf{root}$; 5) the tuple (e, K_e) are the current epoch and the group communication key of the current epoch.

Cryptographic Primitives. Let λ be the security parameter. The protocol uses the following components:

- A CMA-DKD scheme (Defn. 7) $\Sigma = (\mathsf{KGen}, \mathsf{Auth}, \mathsf{Ver})$, where Auth runs Tag, $\mathsf{Disclose}$ as subroutines, and Ver runs $\mathsf{Compatible}$, KVer, MVer as subroutines.
- A symmetric key encryption (Defn. 2 in 2.1) scheme $\mathcal{E} = (\mathsf{Encrypt}, \mathsf{Decrypt})$ with key space $\{0,1\}^\lambda$.
- A key derivation function (Defn. 4 in 2.2) $\mathsf{KDF} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^{2\lambda}$.

Initializing Shared Keys. Below we specify that the group manager G will share a key, k_{GU}, with each other protocol party U. There are different ways that such keys can be initialized. For example, they can be set up using a prior TLS session using a PKI in the case of powerful computing devices. Another example is that they can be hard-wired keys to physical IoT devices set up at factory configuration. The former case allows k_{GU} to be updated after full compromise, while the latter may require the destruction of the IoT device when fully compromised. We allow applications to decide the most appropriate method.

4.2 Protocol Description

Our protocol, TEAKEX, is described as follows. We explain the five algorithms for a CGKE introduced in Defn. 1.

– $\mathsf{Gen}(1^\lambda, G)$: Given a group manager G, the algorithm outputs $(\Omega, \{k_{GU}\}_{U \in \Omega})$ where Ω is the group of users that can participate in the group communication and k_{GU} is a length-λ random key shared by G and U.

– $\mathsf{Initialize}(\mathcal{G}, n, e)$: Given a maximum group size n, an initialized group $\mathcal{G} = \{\mathrm{G}, U_1, U_2, ..., U_{n'}\}$ with $n' \leq n$, and the starting epoch $e = 0$, The protocol works as follows:

1. G sets $(\mathsf{Param}, \{AK_i\}_{i \in [0,T]}) \leftarrow \mathsf{KGen}(1^\lambda, T)$, publishes Param and T.

2. G generates a full binary tree with n leaves, assigns each user U_i with the i-th leaf, selects random, length-λ keys $\{k_i\}_{i\in[n-1]}$ for each internal node (including the root).
3. For each U_i, set $C_{0,i} \leftarrow \mathsf{Encrypt}(k_{GU_i}, \vec{k}_{U_i \rightsquigarrow \mathsf{root}})$ and $C_0 = (\{C_{0,i}\}_{i\in[n']}, \mathcal{G})$.
4. G computes $\mathsf{K}_1 = \mathsf{KDF}(k_{\mathsf{root}}, \mathcal{G}, 0)$, updates $State_{\mathcal{G}}$ with $t = 1$ and $e = 1$.
5. G sends
$$((C_0||\texttt{init}, 0, \sigma), r) \leftarrow \mathsf{Auth}(AK_0, 0, C_0)$$
to the recipient(s), where `init` specifies procedure Initialize. (Recall that $AK_0 \leftarrow \mathsf{Disclose}(0)$ and $\sigma \leftarrow \mathsf{Tag}(AK_0, 0, C_0||\texttt{init})$.)

Upon receiving the message from G, the user U_i does the following.

1. If $0 = \mathsf{Ver}(\mathsf{Param}, r, 0, \sigma_0, C_0||\texttt{init})$, U_i returns $\perp$; Otherwise, U_i parses $C_0 = (\{C_{0,i}\}_{i\in[n']}, \mathcal{G})$, returns $\top$, and does $\mathsf{Decrypt}(k_{GU_i}, C_{0,i})$ to get $\vec{k}_{U_i \rightsquigarrow \mathsf{root}}$ which includes k_{root}.
2. U_i computes the group communication key $\mathsf{K}_{e+1} = \mathsf{KDF}(k_{\mathsf{root}}, \mathcal{G}, e)$.
3. Setting $t = 1$, $e = 1$, U updates $State_U$.

As a result, the initial state $State_{\mathcal{G}}$ is established. In particular, all participants can compute the group communication key K_e for epoch $e = 1$.

– $\mathsf{Add}(U, e)$: To add a new user U to the current group $\mathcal{G}$ at the time interval $1 < t \leq T$ and epoch e, the group manager G does:

1. If there are n users in $\mathcal{G}$ or $U \in \mathcal{G}$, G keeps $State_{\mathcal{G}}$ unchanged, and exits.
2. G assigns the left-most unused leaf, e.g., the i-th leaf, to U and sets $\mathcal{G} \leftarrow \mathcal{G} \cup \{U\}$; G notifies U with $\mathcal{G}$, Param, the current time interval t and the current epoch e.
3. G identifies the current time interval t and chooses a new set of keys $\vec{k}_{U \rightsquigarrow \mathsf{root}}$ which includes the new root key. Then, G identifies all sibling nodes of the nodes on the path $U \rightsquigarrow \mathsf{root}$ and their corresponding keys.
4. Let $k_j \in \vec{k}_{U \rightsquigarrow \mathsf{root}}$ for $j \in [\log_2 n]$, and v_j be the node for k_j; G does:
   - setting $C_{t,j,0} \leftarrow \mathsf{Encrypt}(k_{j,L}, k_j)$, $C_{t,j,1} \leftarrow \mathsf{Encrypt}(k_{j,R}, k_j)$ where $k_{j,L}$ (resp. $k_{j,R}$) denotes the key associated with left child (resp. right child) of v_j, provided it exists.
   - defining C_t be the set of ciphertexts created, and sending the message $(C_t||\mathsf{add}, t, \sigma), r) \leftarrow \mathsf{Auth}(AK_t, t, C_t||\mathsf{add})$ to the group.
   - setting $t = t+1$, $e = e+1$, computing $\mathsf{K}_{e+1} \leftarrow \mathsf{KDF}(k_{\mathsf{root}}, \mathcal{G}, e)$, and updating $State_{\mathcal{G}}$.

Upon receiving the message from G, the user U_i does the following.

1. If $1 \neq \mathsf{Ver}(\mathsf{Param}, r, t, \sigma, C_t||\mathsf{add})$ then U and the users in $\mathcal{G}$ reject the ciphertexts and return $\perp$; Otherwise,
   - U_i returns $\top$ and recovers the new node keys k_j (include k_{root}) on the path $U_i \rightsquigarrow \mathsf{root}$ using the relevant ciphertext where the decryption $k_{j,L}$ or $k_{j,R}$ is known to it.(User U_i has the key associated with one of the siblings of the nodes on $U \rightsquigarrow \mathsf{root}$.)
   - U returns $\top$ and recovers $\vec{k}_{U \rightsquigarrow \mathsf{root}}$ by sequentially decrypting the ciphertexts along $U \rightsquigarrow \mathsf{root}$.
2. The users use k_{root} to compute $\mathsf{K}_{e+1} \leftarrow \mathsf{KDF}(k_{\mathsf{root}}, \mathcal{G}, e)$.
3. Setting $t = t + 1$, $e = e + 1$ user U_i updates $State_{U_i}$.

As a result, U is added to the group communication with its state $State_U$ established. The users in $\mathcal{G}$ established a fresh communication key K_e and the epoch increments to $e + 1$.

– $\mathsf{Remove}(U, e)$: To remove a user U from the current group membership $\mathcal{G}$ at epoch e, G does:

1. If $U \notin \mathcal{G}$, keep $State_{\mathcal{G}}$ unchanged, and exit.
2. G identifies the current time interval t, chooses a new set of node keys $\vec{k}_{U \rightsquigarrow \mathsf{root}}$, and identifies all sibling nodes of the internal nodes on the path $U \rightsquigarrow \mathsf{root}$ and their corresponding keys.
3. Let $k_j \in \vec{k}_{U \rightsquigarrow \mathsf{root}}$ for $j \in [\log_2 n]$, and v_j be the node for k_j. G does:
   - setting $C_{t,j,0} \leftarrow \mathsf{Encrypt}(k_{j,L}, k_j)$, $C_{t,j,1} \leftarrow \mathsf{Encrypt}(k_{j,R}, k_j)$ where $k_{j,L}$ (resp. $k_{j,R}$) denotes the key associated with the left child (resp. the right child) of v_j, provided it exists;
   - defining C_t be the set of ciphertexts and sending
   $$((C_t||\mathtt{rem}, t, \sigma), r) \leftarrow \mathsf{Auth}(AK_t, t, C_t||\mathtt{rem})$$
   to all users;
   - setting $t = 1$, $e = 1$, computing $\mathsf{K}_{e+1} \leftarrow \mathsf{KDF}(k_{\mathsf{root}}, \mathcal{G}, e)$, and updating $State_{\mathcal{G}}$.

Upon receiving the message from G, the user U_i does the following.

1. If $1 \neq \mathsf{Ver}(\mathsf{Param}, r, t, \sigma, C_t||\mathtt{rem})$, the users reject the ciphertext and returns $\perp$; Otherwise, user $U_i \neq U$ returns $\top$ and recovers k_j (including k_{root}) on the path $U_i \rightsquigarrow \mathsf{root}$ by decrypting the relevant ciphertext whose decryption key $k_{j,L}$ or $k_{j,R}$ is known.
2. The users U_i uses k_{root} to compute $\mathsf{K}_{e+1} \leftarrow \mathsf{KDF}(k_{\mathsf{root}}, \mathcal{G}, e)$.
3. Setting $t = t + 1$, $e = e + 1$, user U_i updates $State_{U_i}$.

As a result, each of the remaining users obtains the group communication key K_{e+1} and the protocol moves over to epoch $e + 1$.

– $\mathsf{Update}(U, e)$: The user U's state can be updated as follows.

1. G identifies the leaf node U.
2. G runs Step 3 and Step 4 of what the group manager (G) does in the protocol $\mathsf{Add}(U, e)$ with the message `upd` concatenated with the ciphertext set C_t instead of `add`.
3. Upon receiving the message from G, the user U does Step 1 to Step 3 of what U does in the protocol $\mathsf{Add}(U, e)$ except using the message `upd` concatenated with the ciphertext set C_t instead of `add`.

As a result, the states $State_{\mathcal{G}}$ and $State_U$ for $U \in \mathcal{G}$ are updated; the group communication key is fresh.

5 Efficiency Analysis

Let a group have n users and one group manager. We analyze the space efficiency and the computational efficiency of the protocol in terms of the variable n.

First, we consider the space overhead of users and the group manager. Recall that a user has state $State_U = (\mathcal{G}, (\mathsf{Param}, t), \vec{k}_{U \rightsquigarrow \mathsf{root}}, (e, \mathsf{K}_e))$. $State_U$ contains one long-term shared key with the group manager G, a set of node keys of the nodes from the leaf that U occupies to the root (which contains $\log_2 n$ length-λ symmetric-key keys), the current epoch, and a length-λ group communication key K_e. The total overhead of a user space is $O(\log n)$. The group manager needs to store n long-term length-λ symmetric keys plus all the node keys. This makes its space overhead $O(n)$.

Second, we analyse the computational complexity of the protocol operations Add and Remove and Update (we exclude Gen as it only needs to be done once). For Add operation on a user U, the group manager G encrypts the internal node keys along the path $U \rightsquigarrow \mathsf{root}$ to U by the sibling node keys. So, the ciphertext overhead is $2\log_2 n$. For Remove operation, there are at most $2\log_2 n$ ciphertexts which leads to an $O(\log n)$ overhead. Update operation has the same computational complexity as Add, i.e., $O(\log n)$.

Computational Overhead				Space Overhead	
Initialize	Add	Remove	Update	User	Group Manager
$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$

Table 1: Protocol Efficiency

Note that we expect that in applications the Initialize operation is called once and there will follow multiple Add, Remove and Update operations as the group evolves. The frequency of Update operations is a policy decision to be specified for the application. This is similar to MLS where it is suggested that "the right frequency is usually on the order of hours or days, not milliseconds" [4].

Our protocol has the same *asymptotic* complexity as the typical MLS-type protocols with binary tree structures [1, 2, 11]. However, since our protocol only involves symmetric-key encryption and hashing, it is in practice orders of magnitude more efficient. We stress that the favorable efficiency gain comes with the assumption that our protocol is centralized, which can make good sense in many real-world applications, as discussed in the introduction.

6 Security Model

We formally define the security model for TEAKEX using game-based security definitions. At the start of the security game, a random bit is chosen, i.e., $b \leftarrow \{0,1\}$, the challenger runs Gen to set the keys of all parties. Then $\mathcal{A}$ chooses a set of initial users, and the challenger runs Initialize to initialize the protocol participants. $\mathcal{A}$ makes queries to the oracles Add(), Send(), Remove(), Corrupt() as stated in Table 2 and receives the responses from the challenger. At some time, $\mathcal{A}$ chooses a user U^* and an epoch e^* for which the protocol state is *fresh* (see below) to make the challenge query. $\mathsf{Challenge}(U^*, e^*, b)$. If $b = 1$, $\mathcal{A}$ is given the group communication key K_{e^*} from $State_{U^*}$. If $b = 0$, $\mathcal{A}$ is given a key drawn randomly from the key space $\{0,1\}^\lambda$. Eventually, $\mathcal{A}$ outputs its guess bit b' and wins if $b' = b$. We formalize the game as $\mathsf{Exp}^{\mathsf{ind}}_{\mathsf{CGKE},\mathcal{A}}(\lambda)$, defined in Fig. 4.

$\mathsf{Send}(U, M, e)$	$\mathcal{A}$ sends a message M to the user U at epoch e receives the response.
$\mathsf{Add}(U, e)$	$\mathcal{A}$ chooses a new user U to add to the group $\mathcal{G}$ at epoch e.
$\mathsf{Remove}(U, e)$	$\mathcal{A}$ chooses an existing user U to remove from the group $\mathcal{G}$ at epoch e.
$\mathsf{Corrupt}(U, e)$	The state of $U \in \mathcal{G}$, i.e., $State_U$, at epoch e is given to $\mathcal{A}$.
$\mathsf{Update}(U, e)$	Update user U and other users' states.
$\mathsf{Challenge}\ (U, e, b)$	$\mathcal{A}$ is given either the group communication key of U in epoch e or a random key, depending on a secret bit b.

Table 2: Adversarial queries in the security game

Definition 9. *Let λ be the security parameter. We say a continuous group key exchange protocol* CGKE *is secure if for any efficient adversary $\mathcal{A}$, the advantage,*

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathsf{CGKE},\mathcal{A}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{ind}}_{\mathsf{CGKE},\mathcal{A}}(\lambda) = 1] - 1/2|.$$

is negligible in λ, if the protocol state $State_{\mathcal{G}^*}$ at epoch e^* is fresh (Definition 10). Let $\mathcal{O} := \{\mathsf{Send}(), \mathsf{Add}(), \mathsf{Remove}(), \mathsf{Update}(), \mathsf{Corrupt}()\}$. The experiment $\mathsf{Exp}^{\mathsf{ind}}_{\mathsf{CGKE},\mathcal{A}}(\lambda)$ is defined in Fig. 4.

Experiment $\mathsf{Exp}^{\mathsf{ind}}_{\mathsf{CGKE},\mathcal{A}}(\lambda)$:

1. $b \leftarrow \{0,1\}$, $\Omega \leftarrow \mathsf{Gen}(1^\lambda)$
2. $(\mathcal{G}, n, st) \leftarrow \mathcal{A}(\Omega)$
3. $State_{\mathcal{G}} \leftarrow \mathsf{Initialize}(\mathcal{G}, n, 0)$
4. $(\mathcal{G}^*, e^*, st) \leftarrow \mathcal{A}^{\mathcal{O}}(st)$
5. $\mathsf{K} \leftarrow \mathsf{Challenge}(U^*, e^*, b)$ with $U^* \in \mathcal{G}^* \setminus \{G\}$
6. $b' \leftarrow \mathcal{A}^{\mathcal{O}}(st, \mathsf{K})$
7. Return $(b' = b)$

Oracle $\mathsf{Send}(U, M, e)$:

1. U processes M at epoch e according protocol specification.

Oracle $\mathsf{Add}(U, e)$:

1. Run protocol $\mathsf{Add}(U, e)$

Oracle $\mathsf{Remove}(U, e)$:

1. Run protocol $\mathsf{Remove}(U, e)$

Oracle $\mathsf{Update}(U, e)$:

1. Run protocol $\mathsf{Update}(U, e)$

Oracle $\mathsf{Corrupt}(U, e)$:

1. Return $State_U$ at epoch e

Oracle $\mathsf{Challenge}(U^*, e^*, b)$:

1. Return a random $\mathsf{K} \leftarrow \{0,1\}$ if $State_{\mathcal{G}}$ at e^* is not fresh
2. Retrieve K_{e^*} from $State_{U^*}$
3. $\mathsf{K}_1 \leftarrow \mathsf{K}_{e^*}$, $\mathsf{K}_0 \leftarrow \{0,1\}^\lambda$
4. Return $\mathsf{K} := \mathsf{K}_b$

Figure 4: Security Experiment of Group Key Exchange

Definition 10 (Freshness)**.** *We say the state of the protocol at epoch e^*,*

$$State_{\mathcal{G}} = \Big(\mathcal{G}, (\mathsf{Param}, \{AK_i\}_{i\in[T]}, t), \{k_{GU}\}_{U\in\mathcal{G}}, \{\vec{k}_{U\rightsquigarrow\mathsf{root}}\}_{U\in\mathcal{G}}, (e^*, \mathsf{K}_{e^*})\Big)$$

is fresh *if 1) there is no $\mathsf{Corrupt}(U, e^*)$ query for $U \in \mathcal{G}$, and 2) if $\mathsf{Corrupt}(U, e')$ happened for $U \in \mathcal{G}$ and $e' < e^*$, then $\mathsf{Remove}(U, e'')$ or $\mathsf{Update}(U, e'')$ queries with $e < e'' < e^*$ happened and no $\mathsf{Corrupt}(U, e''')$ query with $e'' < e''' \leq e^*$.*

We note that our security model incorporates two important security properties:

– **forward secrecy** is provided since $\mathcal{A}$ is allowed to corrupt users in epochs *after* the Challenge epoch and the group key must still remain secure;

– **post-compromise security** is provided since the group key must be secure even after U was corrupted, as long as the state of U has been renewed before the Challenge query.

7 Security Proof

Theorem 2. *Let $\Sigma = (\mathsf{KGen}, \mathsf{Auth}, \mathsf{Ver})$ be an CMA-DKD scheme, $\mathcal{E} = (\mathsf{Encrypt}, \mathsf{Decrypt})$ with key space $\{0,1\}^\lambda$ and message space $\mathcal{K}$ be a secure symmetric-key encryption scheme, and* KDF *is secure key derivation function. Let* CGKE *be our protocol, constructed using Σ, $\mathcal{E}$, and* KDF*. For any efficient adversary $\mathcal{A}$:*

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathsf{CGKE},\mathcal{A}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{ind}}_{\mathsf{CGKE},\mathcal{A}}(\lambda) = 1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$$

where λ is the security parameter and $\mathsf{negl}(\lambda)$ is negligible.

Proof. The proof follows from the following ideas: 1) if the adversary does not alter the message flows, then we turn the adversary's advantage to break the symmetric-key encryption scheme. 2) If the adversary gains advantages from altering the message flows, we can use it to break the CMA-DKD scheme.

First, let F be the event that at a time interval, say t, of CMA-DKD that prior to the disclosure of AK_t, the adversary makes a send query $\mathsf{Send}(U, M, e)$ where $M = C_t||\mathtt{type}$ where $\mathtt{type} \in \{\mathtt{init}, \mathtt{add}, \mathtt{rem}, \mathtt{upd}\}$ such that $\mathsf{Ver}(\mathsf{Param}, AK_t, \sigma, C_t||\mathtt{type}) = 1$. That is, F represents the event that the adversary forges σ on $C_t||\mathtt{type}$ without knowing the authentication key AK_t. We have

$$\begin{aligned}\Pr[\mathsf{Exp}^{\mathsf{ind}}_{\mathsf{CGKE},\mathcal{A}}(\lambda) = 1] &= \Pr[\mathsf{Exp}^{\mathsf{ind}}_{\mathsf{CGKE},\mathcal{A}}(\lambda) = 1|F]\Pr[F] + \\ &\qquad \Pr[\mathsf{Exp}^{\mathsf{ind}}_{\mathsf{CGKE},\mathcal{A}}(\lambda) = 1|\bar{F}]\Pr[\bar{F}] \\ &\leq \Pr[F] + \Pr[\mathsf{Exp}^{\mathsf{ind}}_{\mathsf{CGKE},\mathcal{A}}(\lambda) = 1|\bar{F}]\end{aligned}$$

We first prove the following lemma which bounds $\Pr[F]$.

Lemma 1. *Let T be a (polynomial) bound on the number of time intervals for the CMA-DKD scheme. For all p.p.t adversary $\mathcal{A}_1$, we have:*

$$\Pr[F] \leq 1/T \cdot \mathsf{Adv}^{\mathsf{euf}}_{\Sigma,\mathcal{A}_1}(\lambda)$$

Proof. We construct $\mathcal{A}_1$ to interact with the protocol attacker $\mathcal{A}$. After receiving a CMA-DKD challenge (Param, T), $\mathcal{A}_1$ begins by running $\mathsf{Gen}(1^\lambda)$ to establish the shared secret keys with the users. It uses Param and T to run $\mathsf{Initialize}$ to form the group. Meanwhile, $\mathcal{A}_1$ guesses a time interval $t^* \in [T]$. Looking ahead, $\mathcal{A}_1$ hopes that at the interval t^*, the adversary forges an authenticator for the first time. $\mathcal{A}_1$ answers the queries from $\mathcal{A}$ before t^* as follows:

- $\mathsf{Send}(U, M, e)$: $\mathcal{A}_1$ follows the protocol to process M as it is in epoch e. This includes using the CMA-DKD scheme to verify the authenticity of the ciphertext sets which was included in M.
- $\mathsf{Add}(U, e)$, $\mathsf{Remove}(U, e)$ and $\mathsf{Update}(U, e)$: $\mathcal{A}_1$ acts as in the real protocols Add, Remove and Update; when there is a need to compute Auth at time interval t on a ciphertext C_t, $\mathcal{A}_1$ forwards $(C_t||\mathtt{add}, t)$ to its CMA-DKD challenger, making a query to the oracle Tag to get the authenticator σ and sending $(C_t||\mathtt{add}, t, \sigma)$ to the destinations. Then, $\mathcal{A}_1$ makes a $\mathsf{Disclose}$ query to get the authentication key AK_t and sends AK_t to the destinations within the required period.
- $\mathsf{Corrupt}(U, e)$: $\mathcal{A}_1$ follows the descriptions in Fig. 4 to get $State_U$ for $\mathcal{A}$.
- $\mathsf{Challenge}(U^*, e^*, b)$: $\mathcal{A}_1$ follows the descriptions in Fig. 4 to return the K_{e^*}.

If $\mathcal{A}_1$'s guess on t^* is incorrect, $\mathcal{A}_1$ aborts the simulation. Otherwise, at time interval t^*, a $\mathsf{Send}(U, M, e)$ query is made by $\mathcal{A}$. Instead of making a $\mathsf{Disclose}$ query to get the authentication key, she parses $M = (C_{t^*}||\mathtt{type}, t^*, \sigma)$ where $\mathtt{type} \in \{\mathtt{init}, \mathtt{add}, \mathtt{rem}, \mathtt{upd}\}$, aborts the interaction with $\mathcal{A}$, and submits $(t^*, C_{t^*}||\mathtt{type}, \sigma)$ as her output for $\mathsf{Exp}^{\mathsf{euf}}_{\Sigma,\mathcal{A}_1}(\lambda)$ and returns what the experiment returns. It is easy to see that $\mathcal{A}_1$ simulates the game properly until the time interval t^*. If $\mathcal{A}_1$ guesses t^* correctly, $\Pr[F]$ equals the probability that $\mathcal{A}_1$ breaks the CMA-DKD scheme. Hence, we derived the bound. □

Next, we bound $\Pr[\mathsf{Exp}^{\mathsf{ind}}_{\mathsf{CGKE},\mathcal{A}}(\lambda) = 1|\bar{F}]$, which is the advantage that the CGKA adversary succeeds without breaking the underlying CMA-DKD scheme. In this case, we force the adversary to break the symmetric-key encryption scheme. We prove the following lemma that formally states this.

Lemma 2. *Let λ be the security parameter, n be the number of leaves in the binary tree. For any p.p.t adversary $\mathcal{A}$, let Q, a polynomial in λ, be the number of queries that $\mathcal{A}$ made. We have* $\Pr[\mathsf{Exp}^{\mathsf{ind}}_{\mathsf{CGKE},\mathcal{A}}(\lambda) = 1|\bar{F}]$ *is bounded by*

$$\frac{1}{2} + 2n^*\dot{Q}(Q+1) \cdot \log_2(n^*) \cdot \mathsf{Adv}^{\mathsf{indr}}_{\mathcal{E},\mathcal{A}_1}(\lambda) + 2Q(Q+1) \cdot \mathsf{Adv}^{\mathsf{ind}}_{\mathsf{kdf},\mathcal{A}_2}(\lambda)$$

for all adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$.

Proof. We bound $\Pr[\mathsf{Exp}^{\mathsf{ind}}_{\mathsf{CGKE},\mathcal{A}}(\lambda) = 1|\bar{F}]$ using the security of the symmetric-key encryption scheme $\mathcal{E}$. We do not explicitly handle Send queries as condition $\bar{F}$ ensures that all Send queries only contain honest messages from the group manager G. We start with some definitions that facilitate the proof.

- The *depth* of a node of a binary tree is defined as the largest number of edges from the root node that node.
- We say an internal node is *safe* if it is not in a path $U \rightsquigarrow \mathsf{root}$ where U's state in under corrupted status. Otherwise, the node is called *dangerous*.

We proceed through a series of hybrid games G_0 to G_3, each with a well-defined binary output. We assume that Ω, the group of users that can participate in the group, has n^* users and one group manager. We also assume that $\mathcal{A}$ only asks its $\mathsf{Challenge}$ query against a fresh epoch. Otherwise, its advantage is 0. We denote by S_x the event that G_x returns 1. The games are defined as follows:

G_0**:** Is identical to $\mathsf{Exp}^{\mathsf{ind}}_{\mathsf{CGKE},\mathcal{A}}(\lambda)$ conditioned on F does not happen.

G_1**:** This game is identical to G_0 except that for a number $q \in \{0, 1, 2, ..., Q)$, the adversary's q-th query is $\mathsf{Corrupt}(U, e)$ and it is also the last $\mathsf{Corrupt}$ query *before* the $\mathsf{Challenge}$ query. The game outputs 1 if the guess is correct.

$G_{1,i}$**:** Let $G_{1,0}$ be the same as G_1. For $i \in [1, n^*]$, $G_{1,i}$ is identical to $G_{1,i-1}$ except that after the q-th query, i.e., the last $\mathsf{Corrupt}$ query, all the ciphertexts for Add, Remove, Update, queries that encrypt the node keys of *safe nodes* on path $U_i \rightsquigarrow \mathsf{root}$ are set to be random.

G_2**:** This is identical to G_{1,n^*}.

G_3**:** This game is identical to G_2, except that oracle $\mathsf{Challenge}(U, e, b)$ chooses a random $\mathsf{K}_1 \leftarrow \{0, 1\}$ instead of setting $\mathsf{K}_1 \leftarrow \mathsf{K}_{e^*}$.

Since G_0 is identical to $\mathsf{Exp}^{\mathsf{ind}}_{\mathsf{CGKE},\mathcal{A}}(\lambda)$ where F does not happen, by definition,

$$\Pr[S_0] = \Pr[\mathsf{Exp}^{\mathsf{ind}}_{\mathsf{CGKE},\mathcal{A}}(\lambda) = 1|\bar{F}] \tag{3}$$

Let E be the event that the guess was correct in G_1. Since the guess in G_1 is independent of $\mathcal{A}$'s view, we have

$$\Pr[S_1] = \Pr[S_0 \wedge E] = \Pr[S_0] \cdot \Pr[E] = \frac{1}{Q+1} \Pr[S_0] \tag{4}$$

To bound $|\Pr[S_{1,i-1}] - \Pr[S_{1,i}]|$, we further define hybrid games $G_{1,i,j}$ for the depth variable j from $\log_2(n)$ to 0:

- Game $G_{1,i,\log_2(n^*)}$ is identical to game $G_{1,i-1}$.
- Game $G_{1,i,j-1}$ is identical to $G_{1,i,j}$ except that in $G_{1,i,j-1}$ the ciphertexts for the depth $j-1$ safe node is set random, instead of being computed using $\mathsf{Encrypt}$ of the symmetric-key encryption scheme $\mathcal{E}$.
- Game $G_{1,i,0}$ is identical to $G_{1,i}$.

Recall that q-th query is the last corrupt query. All queries after that and before the challenge will not introduce any more dangerous nodes and gradually make more and more safe nodes until the challenge epoch e^* at which no nodes are on corrupted paths. That being said, given a path $U \rightsquigarrow \mathsf{root}$, once the depth-$j$ node is safe, it won't become dangerous later (U can be removed; but once it rejoins, that node remains safe). We use this idea to construct an algorithm $\mathcal{A}_1$ to bound $|\Pr[S_{1,i,j}] - \Pr[S_{1,i,j-1}]|$ using $\mathcal{A}_1$'s advantage against $\mathcal{E}$.

Assume $\mathcal{A}_1$ is running $\mathsf{Exp}^{\mathsf{indr}}_{\mathcal{E},\mathcal{A}_1}(\lambda)$ and aims to produce guess b' of a bit value b. $\mathcal{A}_1$ creates a simulation by running $\mathsf{Initialize}$ to set up the protocol and follow $G_{1,i,j}$ to interact with the adversary $\mathcal{A}$. Recall that to user U_i, only $\mathsf{Add}(U_i, e)$, $\mathsf{Remove}(U_i, e)$, $\mathsf{Update}(U_i, e)$ queries can happen after $\mathcal{A}$'s q-th query. Given one such query during this period, there is a need to produce a ciphertext of the depth-j node key on $U_i \rightsquigarrow \mathsf{root}$ encrypted using the key of the depth-j node. At that point, $\mathcal{A}_1$ receives a ciphertext c from the oracle $\mathcal{O}$ in the experiment $\mathsf{Exp}^{\mathsf{indr}}_{\mathcal{E},\mathcal{A}_1}(\lambda)$ (see Definition 3) and uses c to be the encryption of the node key. $\mathcal{A}_1$ follows $G_{1,i,j}$ for the rest of the simulation. Finally, $\mathcal{A}_1$ returns what $\mathcal{A}$ returns.

Now we analyse the simulation. It is easy to see that if $\mathcal{O}$ is parametrised by $b = 1$, c is a proper encryption of k. Here, k is implicitly treated as the key of the depth-j node on $U_i \rightsquigarrow \mathsf{root}$, which makes the case $G_{1,i,j}$. On the other hand, c is random, which corresponds to the case in $G_{1,i,j}$. Meanwhile, the simulation can answer the queries perfectly without the knowledge of key k – there are cases.

- Before $\mathsf{Challenge}(U, e^*)$: $\mathcal{A}$ cannot see k as no $\mathsf{Corrupt}$ query happens.
- After $\mathsf{Challenge}(U, e^*)$: k is replaced by a key chosen by $\mathcal{A}_1$ as in the real protocol (G_1). So, $\mathsf{Corrupt}$ queries can be answered.

As a result, we have

$$\begin{aligned}
|\Pr[S_{1,i,j}] - \Pr[S_{1,i,j-1}]| &= 2 \cdot |\frac{1}{2}(\Pr[S_{1,i,j}] + 1 - \Pr[S_{1,i,j-1}]) - \frac{1}{2}| \\
&= 2 \cdot |\frac{1}{2}(\Pr[\mathcal{A} \Rightarrow 1 | b = 1] + 1 - \Pr[\mathcal{A} \Rightarrow 1 | b = 0]) - \frac{1}{2}| \\
&= 2 \cdot |\frac{1}{2}(\Pr[\mathcal{A}_1 \Rightarrow 1 | b = 1] + \Pr[\mathcal{A}_1 \Rightarrow 0 | b = 0]) - \frac{1}{2}| \\
&= 2 \cdot \mathsf{Adv}^{\mathsf{indr}}_{\mathcal{E},\mathcal{A}}(\lambda)
\end{aligned}$$

This gives

$$\begin{aligned}
|\Pr[S_{1,i-1}] - \Pr[S_{1,i}]| &\leq |\Pr[S_{1,i,\log_2(n^*)}] - \Pr[S_{1,i,0}]| \\
&\leq \sum_{j=1}^{\log_2(n)} |\Pr[S_{1,i,j}] - \Pr[S_{1,i,j-1}]| \\
&\leq \sum_{j=1}^{\log_2(n^*)} 2 \cdot \mathsf{Exp}^{\mathsf{indr}}_{\mathcal{E},\mathcal{A}_1}(\lambda) \\
&= 2 \cdot \log_2(n^*) \cdot \mathsf{Adv}^{\mathsf{indr}}_{\mathcal{E},\mathcal{A}}(\lambda)
\end{aligned} \tag{5}$$

For the relation between G_1 and G_2, applying the inequality (5), we obtained

$$\begin{aligned}
|\Pr[S_1] - \Pr[S_2]| &= |\Pr[S_{1,0}] - \Pr[S_{1,n^*}]| \\
&\leq \sum_{i=1}^{n^*} |\Pr[S_{1,i-1}] - \Pr[S_{1,i}]| \\
&\leq 2n^* \cdot \log_2(n^*) \cdot \mathsf{Adv}^{\mathsf{indr}}_{\mathcal{E},\mathcal{A}}(\lambda)
\end{aligned} \tag{6}$$

The difference between G_2 and G_3 can be easily bounded by the security of KDF. Notice that in G_2, the key associated with the root for the challenge query $\mathsf{Challenge}(U, e^*)$

is randomly chosen and independent of the adversary $\mathcal{A}$'s view. A routine reduction shows

$$|\Pr[S_2] - \Pr[S_3]| \leq 2 \cdot \mathsf{Adv}^{\mathsf{ind}}_{\mathsf{KDF},\mathcal{A}}(\lambda) \tag{7}$$

for some adversary $\mathcal{A}_2$ against KDF. Finally, we can see that the group communication key $\mathsf{K}_{\mathsf{e}^*}$ is random. So, the adversary $\mathcal{A}$ has no advantage, i.e.,

$$\Pr[S_3] = 1/2 \tag{8}$$

Combining (3), (4), (6), (7) and (8), we derived the bound in Lemma 2. □

Under our assumptions that the symmetric-key encryption scheme $\mathcal{E}$ and the key derivation function KDF are secure, the quantities $\mathsf{Adv}^{\mathsf{indr}}_{\mathcal{E},\mathcal{A}_1}(\lambda)$ and $\mathsf{Adv}^{\mathsf{ind}}_{\mathsf{kdf},\mathcal{A}_2}(\lambda)$ are negligible in λ. Meanwhile, Q and T are polynomials of λ. Therefore, putting the bounds from Lemma 1 and Lemma 2, Theorem 2 is proved. □

8 Conclusion

We have designed a continuous group key exchange (CGKE) protocol based on symmetric-key primitives. While achieving all desirable security properties for CGKE, compared to MLS and related protocols using public-key primitives, the protocol enjoys high efficiency, simplicity, and post-quantum security. The core of the protocol is a novel use of the famous TESLA authentication protocol.

Acknowledgements.

Boyd and Millerjord were supported by the Research Council of Norway under Project No. 288545.

References

[1] Alwen, J., Auerbach, B., Noval, M.C., Klein, K., Pascual-Perez, G., Pietrzak, K., Walter, M.: CoCoA: Concurrent continuous group key agreement. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022. LNCS, vol. 13276, pp. 815–844. Springer (2022), https://doi.org/10.1007/978-3-031-07085-3_28

[2] Alwen, J., Coretti, S., Dodis, Y., Tselekounis, Y.: Security analysis and improvements for the IETF MLS standard for group messaging. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020 - Part I. LNCS, vol. 12170, pp. 248–277. Springer (2020), https://doi.org/10.1007/978-3-030-56784-2_9

[3] Alwen, J., Jost, D., Mularczyk, M.: On the insider security of MLS. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022 - Part II. LNCS, vol. 13508, pp. 34–68. Springer (2022). https://doi.org/10.1007/978-3-031-15979-4_2, https://doi.org/10.1007/978-3-031-15979-4_2

[4] Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., Cohn-Gordon, K.: The messaging layer security (MLS) protocol. RFC 9420 (July 2023). https://doi.org/10.17487/RFC9420, https://www.rfc-editor.org/info/rfc9420

[5] Bresson, E., Chevassut, O., Pointcheval, D.: Provably authenticated group Diffie-Hellman key exchange - the dynamic case. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 290–309. Springer (2001), https://doi.org/10.1007/3-540-45682-1_18

[6] Bresson, E., Chevassut, O., Pointcheval, D.: Provably-secure authenticated group Diffie-Hellman key exchange. ACM Trans. Inf. Sys. Sec. **10**(3) (2007), http://www.di.ens.fr/%7epointche/Documents/Papers/2007_tissec.pdf

[7] Cohn-Gordon, K., Cremers, C., Garratt, L.: On post-compromise security. In: CSF 2016. pp. 164–178. IEEE (2016), https://doi.org/10.1109/CSF.2016.19

[8] Fernández-Hernández, I., Ashur, T., Rijmen, V.: Analysis and recommendations for MAC and key lengths in delayed disclosure GNSS authentication protocols. IEEE Trans. Aerosp. Electron. Syst. **57**(3), 1827–1839 (2021), https://doi.org/10.1109/TAES.2021.3053129

[9] ISO: Information Security – Lightweight Cryptography – Part 7: Broadcast Authentication Protocols, ISO/IEC 29192-7 (2019), international Standard

[10] Katz, J., Shin, J.S.: Modeling insider attacks on group key-exchange protocols. In: Atluri, V., et al. (eds.) Proceedings of CCS 2005. pp. 180–189. ACM (2005), https://doi.org/10.1145/1102120.1102146

[11] Klein, K., et al.: Keep the dirt: Tainted TreeKEM, adaptively and actively secure continuous group key agreement. In: IEEE Security and Privacy. pp. 268–284. IEEE (2021), https://doi.org/10.1109/SP40001.2021.00035

[12] Krawczyk, H.: Cryptographic extraction and key derivation: The HKDF scheme. In: Annual Cryptology Conference. pp. 631–648. Springer (2010)

[13] Moody, D., Perlner, R., Regenscheid, A., Robinson, A., Cooper, D.: Transition to post-quantum cryptography standards. NIST IR 8547 (Nov 2024). https://doi.org/10.6028/NIST.IR.8547.ipd, https://csrc.nist.gov/pubs/ir/8547/ipd

[14] Perrig, A., Canetti, R., Tygar, J.D., Song, D.X.: Efficient authentication and signing of multicast streams over lossy channels. In: Security and Privacy. pp. 56–73. IEEE Computer Society (2000). https://doi.org/10.1109/SECPRI.2000.848446, https://doi.org/10.1109/SECPRI.2000.848446

[15] Wallner, D., Harder, E., Agee, R.: Key Management for Multicast: Issues and Architectures. RFC 2627 (Jun 1999). https://doi.org/10.17487/RFC2627, https://www.rfc-editor.org/info/rfc2627

[16] Wong, C.K., Gouda, M.G., Lam, S.S.: Secure group communications using key graphs. IEEE/ACM Trans. Netw. **8**(1), 16–30 (2000). https://doi.org/10.1109/90.836475, https://doi.org/10.1109/90.836475