On the Cost of Post-Compromise Security in Concurrent Continuous Group-Key Agreement

Benedikt Auerbach, Miguel Cueto Noval, Guillermo Pascual-Perez, and Krzysztof Pietrzak

ISTA, Austria
{bauerbac, mcuetono, gpascual, pietrzak}@ista.ac.at

Abstract

Continuous Group-Key Agreement (CGKA) allows a group of users to maintain a shared key. It is the fundamental cryptographic primitive underlying group messaging schemes and related protocols, most notably TreeKEM, the underlying key agreement protocol of the Messaging Layer Security (MLS) protocol, a standard for group messaging by the IETF. CKGA works in an asynchronous setting where parties only occasionally must come online, and their messages are relayed by an untrusted server. The most expensive operation provided by CKGA is that which allows for a user to refresh their key material in order to achieve forward secrecy (old messages are secure when a user is compromised) and post-compromise security (users can heal from compromise). One caveat of early CGKA protocols is that these update operations had to be performed sequentially, with any user wanting to update their key material having had to receive and process all previous updates. Late versions of TreeKEM do allow for concurrent updates at the cost of a communication overhead per update message that is linear in the number of updating parties. This was shown to be indeed necessary when achieving PCS in just two rounds of communication by [Bienstock *et al.* TCC'20].

The recently proposed protocol CoCoA [Alwen *et al.* Eurocrypt'22], however, shows that this overhead can be reduced if PCS requirements are relaxed, and only a logarithmic number of rounds is required. The natural question, thus, is whether CoCoA is optimal in this setting.

In this work we answer this question, providing a lower bound on the cost (concretely, the amount of data to be uploaded to the server) for CGKA protocols that heal in an arbitrary k number of rounds, that shows that CoCoA is very close to optimal. Additionally, we extend CoCoA to heal in an arbitrary number of rounds, and propose a modification of it, with a reduced communication cost for certain k.

We prove our bound in a combinatorial setting where the state of the protocol progresses in rounds, and the state of the protocol in each round is captured by a set system, each set specifying a set of users who share a secret key. We show this combinatorial model is equivalent to a symbolic model capturing building blocks including PRFs and public-key encryption, related to the one used by Bienstock *et al.*.

Our lower bound is of order $k \cdot n^{1+1/(k-1)}/\log(k)$, where $2 \leq k \leq \log(n)$ is the number of updates per user the protocol requires to heal. This generalizes the n^2 bound for $k = 2$ from Bienstock *et al.*. This bound almost matches the $k \cdot n^{1+2/(k-1)}$ or $k^2 \cdot n^{1+1/(k-1)}$ efficiency we get for the variants of the CoCoA protocol also introduced in this paper.

Contents

1 Introduction

A fundamental task underlying various cryptographic protocols is to agree upon, and maintain, a secret key amongst a group of users. A prominent example is *continuous group-key agreement* (CGKA) [ACDT20], which underlies group messaging applications. Here, a group of users wants to maintain a shared secret key, that then can be used for private communication amongst the group members.

CGKA is defined in an asynchronous setting, where parties are online only occasionally, and the exchanged messages are relayed through an untrusted server (only trusted to provide liveness and thus correctness). CGKA allows for users to be added or removed from the group. Moreover users can update their keys, which allows the group to achieve forward secrecy (FS) and post-compromise security (PCS). FS guarantees that, should a user's secrets be compromised, messages sent in the past remain secure. PCS, in turn, allows the group to "heal", i.e. to recover privacy after a compromise occurs.

The most efficient existing protocols for CGKA are TreeKEM [BBR18] and variants thereof [Mat19, KPPW+21, ACDT20, AAN+22a, AAN+22b], which are inspired by logical key hierarchies (LKH) [WHA99], a popular protocol for multicast encryption (ME) [CGI+99]. The study of these protocols has received a great deal of attention recently, motivated by the IETF working group on *Message Layer Security* (MLS) [BBR+23], which aims to output standard for instant group messaging. Said standard employs TreeKEM as the underlying CGKA. These schemes all arrange keys from a public-key encryption scheme in trees, known as *ratchet trees*, where each node is associated with a key, each user is associated with a leaf, and users should know exactly the (secret) keys on the path from their leaf to the root (also known as the *tree invariant*).

A simple ratchet tree with four users is illustrated in Figure 1. The advantage of using such a hierarchical tree structure is that replacing a user's keys in a group of size n just requires the creation of $\lceil \log(n) \rceil$ ciphertexts, while e.g. maintaining pairwise keys between the users would require $n - 1$.

Concurrent Updates. Updating keys in a ratchet tree as illustrated in Figure 1 only works if updates are sequential. That is, if two users want to update, then they need to do it in order, with the second processing the first user's update before creating their own. TreeKEM supports concurrent updates through the "propose and commit" (P&C) paradigm, but handling concurrency in this way degrades the nice tree structure and thus efficiency of the protocol. Indeed, after several users update concurrently, all of their paths to the root but one will lose their keys, a.k.a. become *blank*, increasing the in-degrees of nodes in the tree and thus the cost of that and subsequent operations. This incurs an overhead that is linear in the number of updating parties, something which was shown to be optimal by Bienstock, Dodis and Rösler [BDR20], whenever PCS is to be achieved as soon as all corrupted users update once.

CoCoA [AAN+22a] takes a different approach, and simply choses a "winner" whenever there is a conflict, i.e., when two users want to concurrently replace the same key, as illustrated in Figure 2. As opposed to the previous scenario, this does not immediately "heal" the state of the concurrently updating parties (in the Figure, key $K_{\bar{7}}$ is not secure if Dave's key K_6 was compromised). However, in [AAN+22a] it is shown that the group heals (i.e., achieves PCS) after all corrupted users participate in $\log(n)$ (possibly concurrent) update rounds. This is a middle ground between the immediate concurrent healing of P&C TreeKEM, and the n sequential rounds needed for non-concurrent versions of TreeKEM.

In this work we prove a lower bound on the communication cost of CGKA protocols that heal in any number of (up to logarithmic in the group size) rounds.

A Combinatorial Model. Conceptually, our lower bound proof proceed in two steps. We first derive the lower bounds in a clean and simple combinatorial model which proceeds in rounds. The state of the protocol for n users in round t is captured by a set system $\mathcal{S}_t \subseteq 2^{[n]}$, where $S \in \mathcal{S}_t$ means that after round t there is a shared secret amongst the users S *not* known to the adversary. In particular, $[n] \in \mathcal{S}_t$ means the group $[n] = \{1, \ldots, n\}$ shares a secret, which has to be satisfied in all rounds with a secure group key.

For example, in the ratchet tree example from Fig. 1 with (where users are denoted $\{A, B, C, D\}$ not $\{1, 2, 3, 4\}$), the sets corresponding to the keys $K_1, .., K_7$ are

$$\mathcal{S}_t = \{\{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{3, 4\}, \{1, 2, 3, 4\}\} \ .$$

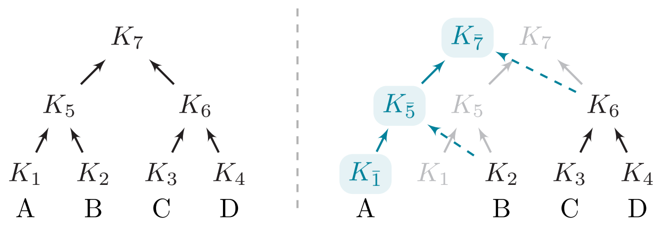

Figure 1: Left: Illustration of a ratchet tree with $n = 4$ users $\{A, B, C, D\}$ where each key $K_i = (pk_i, sk_i)$ is a public/secret key tuple. Right: To update and achieve PCS Alice rotates keys $\{K_1, K_5, K_7\}$ by sampling new keys $\{K_{\bar{1}}, K_{\bar{5}}, K_{\bar{7}}\}$ (blue, shaded background) and encrypting each secret key under the public key of their parent (blue, dashed arrows), e.g. $K_2 \rightarrow K_{\bar{5}}$ corresponds to a ciphertext $\mathsf{Enc}_{pk_2}(sk_{\bar{5}})$. Given those ciphertexts, all users can learn the new keys on their path to the root. For example Bob must decrypt the ciphertexts $\mathsf{Enc}_{pk_2}(sk_{\bar{5}})$ and $\mathsf{Enc}_{pk_{\bar{5}}}(sk_{\bar{7}})$. This requires $2\lceil\log(n)\rceil = 4$ ciphertexts. However, by deriving the keys $\{K_{\bar{1}}, K_{\bar{5}}, K_{\bar{7}}\}$ deterministically from a single seed using a PRG as suggested in [CGI+99], we can save the ciphertexts for the solid blue arrows and only need $\lceil\log(n)\rceil = 2$ ciphertexts.

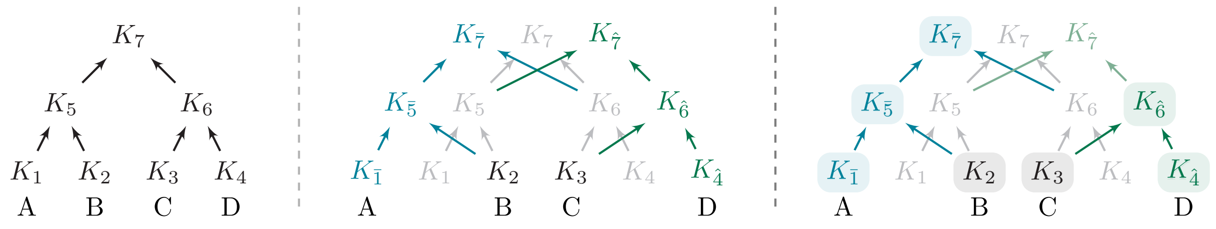

Figure 2: Illustration of CoCoA where Alice and Dave concurrently rotate their keys. Left: state of the ratchet tree before the updates. Middle: Keys $\{K_{\bar{1}}, K_{\bar{5}}, K_{\bar{7}}\}$ and $\{K_{\hat{4}}, K_{\hat{6}}, K_{\hat{7}}\}$ generated by Alice and Dave's updates respectively. There's a collision at the (root) key K_7, and the server chooses a "winner" (any rule for choosing winners will do), in this case Alice. Right: New state of the ratchet tree (Keys in the tree depicted with shaded background). The new root key is $K_{\bar{7}}$ while Dave's $K_{\hat{7}}$ is ignored. As $K_{\bar{7}}$ was encrypted to K_6 we do not achieve PCS if Dave's state $\{K_4, K_6, K_7\}$ prior to the update was compromised. But latest once all corrupted parties updated $\log(n)$ times PCS will be achieved (in particular, if Dave updates once more PCS is achieved).

If a user u gets compromised, all secrets corresponding to sets containing u become known to the adversary and thus the sets must be removed, e.g., if we compromise user 1, the set system becomes

$$\mathcal{S}_{t+1} = \{\{2\}, \{3\}, \{4\}, \{3, 4\}\} \ .$$

A user u can update and create new sets (keys) as follows. They can always locally sample a key, creating the singleton $\{u\}$. For two sets $S, S' \in \mathcal{S}$, where $u \in S$ (or $u \in S'$), they can create a new set $S \cup S'$, by deterministically deriving a secret from that of S using a PRF, and encrypting it under the public key of S'. This would get added to $\mathcal{S}$ in the next round. Note that indeed all users in $S \cup S'$ are able to derive the secret either deterministically from the secret associated to S or by decrypting the ciphertext. In the simplest version of TreeKEM, user 1 performs an update by creating $\{1\}$, then $\{1, 2\} = \{1\} \cup \{2\}$, then $\{1, 2, 3, 4\} = \{1, 2\} \cup \{3, 4\}$ (i.e., the keys $K_{1'}, K_{5'}, K_{7'}$ in Fig. 1). Of course, u is not restricted to create new sets as the union of only two sets, but could also encrypt the secret using the keys of sets $S_1, \ldots, S_k$ to form the set $S \cup \bigcup_{i=1}^{k} S_i$. The communication cost of this operation, i.e., the number of ciphertexts that have to be uploaded to the server to communicate the new secret to all members of the corresponding set, would in this example be k. In Section 3.1 we extend this idea into a self-contained combinatorial model consisting of set system $\mathcal{S}_t$ and an accompanying cost function Cost required to satisfy properties matching the intuition given above. In Section 3.2 the model is used to prove our lower bounds.

The Symbolic Model. While the combinatorial model offers a clean model for proving lower bounds, it is not obvious how it captures real-world protocols. We show that any lower bound in the combinatorial model implies a lower bound in a symbolic model capturing pseudorandom functions and public-key encryption. Most existing CGKA protocols can be captured in this symbolic model and the fact that lower bounds in the combinatorial model carry over to the symbolic model justifies the interest of the combinatorial model we propose. Symbolic models were introduced by Dolev and Yao [DY83] in public key encryption, used in multicast encryption by Micciancio and Panjwani [MP04] and in CGKA by Bienstock, Dodis, and Rösler [BDR20] and Alwen *et al.* [AAB+21]. In the symbolic model pseudorandom functions and public-key encryption are treated in an idealized way by seeing their inputs and outputs as variables with a data type, which, in turn, follow some grammar rules, and ignoring other considerations that an actual construction may have. The functionality and security of these primitives are captured by the grammar rules and entailment relations described in Section 4.1.

1.1 Our Bounds.

In this work we prove lower bounds on the communication cost of CGKA protocols achieving PCS. Moreover, in Section 5, we introduce a new protocol, a modification and generalization of CoCoA. It introduces the necessary number of rounds to heal as a parameter and, in some cases, improves over the natural generalization of CoCoA in this setting.

We measure the cost of a protocol in terms of the number of ciphertexts that users in a group must create (and upload to a server for the other users to download) to achieve post-compromise security.[1] Sometimes, we additionally put a bound on the number of rounds required for parties to heal. We do not require forward-secrecy and will also consider groups of a fixed size, i.e., without removals or additions of users, just updates. Note that both of these make the lower bounds stronger, as an adversary could always choose to not use add/removes. Additionally, FS is relatively well understood, and can be achieved without any asymptotic overhead [ACDT20].

We consider the setting where the users do not know who is compromised or who else will update in any given communication round, and the adversary schedules who does updates in each round. This is similar to that of [BDR20].

When a user is corrupted, we assume its entire secret state is leaked to the adversary, who can also observe all computations – in particular all local randomness – of the user during the corruption. We call this the "randomness corruption" model (RC for short), but we will also consider a weaker "no-randomness corruption" ($\neg$RC for short) model, where only the secret state is leaked. In this model, a corrupted user can still create encrypted secrets for other users. Most protocols are proven secure in the stronger RC model, whereas lower bounds are naturally stronger in the $\neg$RC model. We point out that our lower bound require some additional restrictions on the CGKAs. We expand on this at the end of the introduction.

Lower bound. The number k of updates a user is required to make before their state is guaranteed to heal plays a crucial role. We consider a security game parameterized by the number of users n and k. The adversary schedules who updates in each round, and we require that at any point the group key is secure assuming that every party that was corrupted in the past was asked to update at least k times (since their last corruption). Table 1 states our lower bound and upper bound, as well as existing ones. Our lower bound is roughly $n^{1+1/k} \cdot k/\log(k)$.

The main message here is that we need to allow for logarithmically many rounds for healing (as in CoCoA) if we want a small logarithmic sender communication cost per user. In particular, if we insist on a constant number of rounds, the average cost per user will be of order $n^{1/k}$.

[1]It is possible, as in [AAN+22a, AHKM22], to reduce recipient communication by introducing additional reliance on the server. We focus on sender communication.

Upper bounds

Scheme	Communication	Rounds	Rand. corr.	See
TreeKEM and related	n^2	2	RC	[BBR18]
Bienstock, Dodis, Rösler	n^2	2	$\neg$RC	[BDR20]
CoCoA on $\sqrt[k-1]{n}$-ary trees	$n\,k^2\,\sqrt[k-1]{n}$	k	RC	Sec. 5.2
CoCoA on 2-ary trees	$n\log(n)^2$	$\log(n)$	RC	[AAN+22a]
CoCoALight on $\sqrt[(k-1)/2]{n}$-ary trees	$n\,k\,\sqrt[(k-1)/2]{n}$	k	RC	Sec. 5.3

Lower bounds

Restrictions	Communication	Rounds	Rand. corr.	See
None	n^2	2	$\neg$RC	[BDR20]
NDW, NNE, PCU*	$n\log(n)/\log(\log(n))$	$\log(n)$	$\neg$RC	Cor. 6
NDW, NNE, PCU*	$\varepsilon\cdot n\cdot\sqrt[(1+\varepsilon)k-1]{\alpha_\varepsilon n}\cdot k/\log(k)$	k	$\neg$RC	Cor. 6

Table 1: Upper-bounds (top) and lower-bounds (bottom) in the no-information setting for $\Omega(n)$ corrupted users. Communication is measured as total number of ciphertexts sent to recover from corruption, column "Rounds" indicates the number of update rounds after which schemes are required to recover from corruption, column "Rand. corr.", whether the security model allows the adversary to learn internal randomness of algorithms. The protocol [BDR20] improves over TreeKEM in that concurrent operations do not degrade future performance, which is not captured in the table. Our lower-bounds require CGKA to not allow distributed work (NDW) and not use nested encryption (NNE). Our bound holds without the extra assumption requiring the protocols to have publicly-computable update cost (PCU). However, additional properties of it hold when this assumption is present. We refer the reader to the discussion in Section 1.3 below for more details. Here, $\alpha_\varepsilon \approx \varepsilon$ is some constant depending on ε.

Upper bound. We introduce in Section 5 the protocol CoCoALight, a modification of CoCoA that achieves PCS in $k \in [4, 2\lceil\log(n)\rceil + 1]$ rounds. This protocol has a cost $k \cdot n^{1+2/(k-1)}$, which matches the lower bound up to a factor $\log(k)/n^{1/(k-1)}$. In particular, our protocol is only a factor of $\log(\log(n))$ from optimal for k in the order of $\log(n)$. In turn, CoCoA (or rather, a straightforward generalization of it we propose for $k \in [2, \lceil\log(n)\rceil + 1]$, as opposed to $k = \lceil\log(n)\rceil + 1$ in the original protocol) has better efficiency for low values of k. The key insight in our protocol is that users do not need to update all the keys in their path to heal. In fact, it suffices for them to update keys one by one, as long as every key in the path is updated twice. We formalize and discuss this further in Section 5.

1.2 Our Proofs

Proof of the Lower Bound. To prove the lower bound we first show that, if the protocol can heal from c corruptions in k rounds, then there is some user whose cost is $c^{1/k}$. In particular, if $c = \Theta(n)$, we get a cost of $\Theta(n^{1/k})$. The intuition for this is quite simple, let us give it for $c = n$. Initially everyone is corrupted, so our set system is simply $\{1\}, \ldots, \{n\}$, after the kth round $[n] = \{1, \ldots, n\}$ is in our set system. If we denote with s_i the size of the largest set in round i, we have $s_1 = 1, s_k = n$, which means there must be a round i where $s_{i+1}/s_i \geq n^{1/(k-1)}$. Therefore, in this round, the user creating the new set of size s_i has cost $\geq n^{1/(k-1)}$. A slightly more careful argument shows that the maximum cost of a user in each round adds up to $k \cdot c^{1/(k-1)}$.

To prove our bound we will show that for $c = \Theta(n)$ corruptions, we can adversarially schedule the updates so that a $1/\log(k)$ fraction of users (and not just a single one) can be forced to pay close to the maximum cost in each round, which then adds up to $n^{1+1/(k-1)} \cdot k/\log(k)$.

This adversarial scheduling goes as follows: before each round, the adversary investigates each user's cost, should they be asked to update in the next round. Then, it simply picks a $1/\log(k)$ fraction of users, all having either very small cost or, if such a set does not exist, a set of users with roughly the same cost (we show that such a set of users always exists).

Proof of the Upper Bound. We prove our protocol secure by following the framework set by [KPPW$^+$21], which reduces the adaptive security of a CGKA protocol to that of a game played on graphs. One first defines a so-called *safe predicate*, which captures the settings in which security should be guaranteed (i.e. every corrupted user performed k updates since their last corruption, in our case). This is implicit in our security game. Then, in order to apply previous results, one needs to essentially show that key satisfying the safe predicate trivially leaked as a result of a user corruption during the execution. We do this in Theorem 8, where we associate to each group key in the execution a *recovery graph*, made up of those keys that trivially allow recovery of the group key. Then, through a combinatorial argument, we show that if the safe predicate holds all keys ever leaked through a corruption cannot belong to the recovery graph of the challenge key. Security of the protocol thus follows using the aforementioned framework, in a fashion similar to that of previous works, such as [AAN$^+$22a].

1.3 Overcoming Lower Bounds.

Proving lower bounds for important protocols serves several purposes. On the one hand, it can tell us when constructions *falling into the model of the lower-bound* cannot be further improved. As we identify a protocol that almost match our lower bound, this question is basically answered.

However, lower bound proofs can also hint as to where one should look for constructions overcoming them. One such possibility is to consider building blocks not captured by the bounds, or seemingly technical assumptions, which seem crucial for the lower-bound proofs to go through.

More Powerful Building Blocks. The symbolic model we consider (and which is captured by our combinatorial model) allows the basic primitives of PRFs or public-key encryption, and thus does not rule out protocols overcoming our lower bounds if they use more sophisticated tools.

The “big hammer” in this context is multiparty non-interactive key-agreement (mNIKE). With this primitive, each user could simply create a single message to be broadcast, after which any subset of users can locally compute a shared secret. While this overcomes our lower-bounds, it is just of theoretical interest, as currently no practical instantiations of mNIKE exist.

There already do exist CGKA protocols using primitives not captured by our model, in particular rTreeKEM [ACDT20] and DeCAF [AAN$^+$22b]. The variant rTreeKEM uses secretly-updatable public-key encryption [JMM19] (skUPKE), but this primitive is used to improve the forward secrecy of the protocol, with no difference to the (asymptotic) communication cost of the protocol. The CGKA DeCAF also uses skUPKE, but in order to improve the round complexity for healing: instead of $\log(n)$ rounds as in CoCoA, DeCAF only needs $\log(c)$ rounds, with c being the number of users corrupted.

Note that our lower bound is independent of the number of corrupted users, but the proof argues based on an adversary which corrupts $c = \Theta(n)$ parties. In this setting DeCAF’s cost matches the one of CoCoA and thus adheres to our lower bound However, under the promise that very few, say constant, users are actually corrupted, DeCAF heals in a constant number of rounds with cost $\mathcal{O}(n \log(n))$.

Finally, two recent works [HKP$^+$21, AHKM22], explore the use of multi-recipient multi-message PKE (mmPKE), which allows for much more efficient updates. However, the improvements save a constant factor in the ciphertext size, and do not have an influence in the asymptotic cost of the protocols.

Distributing work. Our bound is restricted to schemes that do not “distribute the workload of communicating a secret on several users”, in the following sense. We require that, if in any round a user gets access to a secret they did not previously possess, then they must have recovered it from a single update message, or sampled it by themselves. In particular, for such schemes, it holds that whenever a user encrypts a secret, they know which other users will have access to it at the end of the round. We point out that all CGKA protocols we are aware of satisfy this requirement.

Nested encryption. Finally, we require that users do not create layered ciphertexts, i.e., those of the form $\mathsf{Enc}(pk_1, \mathsf{Enc}(pk_2, m))$. Again, this is a property that is satisfied by all CGKA protocols we are aware of. This condition has a similar flavor as the one of distributing work, with the difference being that, instead of splitting the communication cost between several users, using this kind of layered encryption enables a user to spread out communication cost over several rounds.

Publicly-computable update cost. We show that there exists a sequence of updates such that the lower bound holds. However, this does not mean that the sequence can be found using only public information. We introduce this assumption to guarantee that the adversary can tell what cost a user will incur if asked to update in round t using only public information available at the end of round $t-1$ and this suffices to find the update sequence used in the proof of the lower bound. We also introduce a stronger version of this property, which we call offline publicly-computable update cost, that makes it possible to use public information available at the end of the initialization phase and the sequence of users who have performed updates in the previous rounds. While the strong property is satisfied by all protocols we are aware of, it is conceivable that protocols exist that overcome our bound, by having a user toss a coin when asked to update, with the outcome determining whether a "cheap" or "expensive" update is made.

1.4 Related work

Protocols. The primitive of CGKA was introduced by Alwen *et al.* [ACDT20], but constructions existed earlier, notably ART [CCG+18] and TreeKEM [BBR18]. These two were the starting point for the Message Layer Security (MLS) working group by the IETF. A variety of protocols have since been published, aiming to improve TreeKEM across different axes.

First, in the non-concurrent setting, [ACDT20] propose the use of UPKE in order to improve on forward secrecy; Klein *et al.* [KPPW+21] propose an alternative way to handle dynamic operations with a lower communication cost in certain scenarios; Devigne, Duguey and Fouque [DDF21] propose to use zero-knowledge proof to enhance the protocol robustness; Alwen *et al.* [AAB+21] initiates the study of efficiency of CGKAs in the multi-group setting; Hashimoto, Katsumata and Prest [HKP22] provide a wrapper upgrading non-metadata-hiding CGKAs into metadata-hiding ones.

Concurrency was already mentioned in the initial TreeKEM versions, and indeed, as mentioned, its new versions allow for a certain degree of it. The first protocol to explore the idea was Weidner's Causal TreeKEM [Mat19] with the idea of updates by re-randomizing (and combining) key material, instead of overwriting it. The work of Weidner *et al.* [WKHB21] puts forth the notion of decentralized CGKA. Alwen *et al.*'s CoCoA [AAN+22a] analyzes a variant allowing for concurrent healing in $\log(n)$ rounds. A follow-up of this work by Alwen *et al.* [AAN+22b] picked up the idea of [Mat19] and extended it and formally analyzed it, showing that it allows for PCS in a logarithmic number of rounds in the number of corrupted parties.

Lowerbounds. The main approach is to make use of the symbolic security model, first introduced by Dolev and Yao [DY83] and later used by Micciancio and Panjwani [MP04] to prove worst case bounds on the update cost of multicast encryption schemes for a single group.

Regarding CGKAs, in the non-concurrent setting, Alwen *et al.* [AAB+21] provide lower bounds for the average update cost of an update in any CGKA protocol in the symbolic model, following and generalizing the approach of [MP04]. This shows TreeKEM or other related protocols are indeed optimal in this setting. In the concurrent setting, i.e. that where we consider the case of healing c corruptions in less than c, the study of lower bounds was initiated by Bienstock, Dodis and Rösler [BDR20], who establish lower bounds for protocols achieving PCS concurrently in precisely 2 rounds.[2] Last, Bienstock *et al.* [BDG+22] establish a lower bound on the cost arising from certain sequences of adds and removes. In particular, they show that any CGKA has a worst-case communication cost linear in the number of users.

[2]Here, we have a tradeoff between the time needed to achieve PCS, and the communication needed to do so. The picture is slightly more complicated, as in protocols like TreeKEM, or the protocol proposed in [BDR20], the bigger tradeoff is in the increased cost of subsequent updates.

Security. Finally, security of CGKAs has been studied by multiple papers. Security against adaptive adversaries with a sub-exponential loss was first proved by Klein *et al.* [KPPW$^+$21]. Active security has been studied by Alwen *et al.* [ACJM20, AJM22] in the UC model. PCS in the multi-group setting has been studied by Cremers, Hale and Kohbrok [CHK21], who show shortcomings of a certain version of MLS compared to the (inefficient) pairwise-channels construction. Brzuska, Cornelissen and Kohbrok [BCK21] apply the State Separating Proofs methodology to analyze the security of a certain version MLS.

2 Preliminaries

2.1 Definitions and Results from Combinatorics

We define the minimal set cover of a set with respect to a set system and recall the well-known inequality of arithmetic and geometric means.

Definition 1 (Minimal set cover). *Let $n \in \mathbb{N}$ and $\mathcal{S} \subseteq 2^{[n]}$. Then for $X \subseteq [n]$ we define the min cover of X with respect to $\mathcal{S}$. A* minimal set cover *(min cover)* $\text{minCover}_{\supseteq}(X, \mathcal{S})$ *of X with respect to $\mathcal{S}$ is a set $\mathcal{T} \subseteq \mathcal{S}$ of minimal cardinality such that $X \subseteq \bigcup_{T \in \mathcal{T}} T$, i.e., a minimal subset of $\mathcal{S}$ that covers X. Note that we only require S be contained in the union but no equality.*

Proposition 1 (Inequality of arithmetic and geometric means). *For $k \in \mathbb{N}$ let $x_1, \ldots, x_k \in \mathbb{R}$ be non-negative such that $\sum_{i=1}^{k} x_i = x$. Then*

$$\prod_{i=1}^{k} x_i \leq \left(\frac{x}{k}\right)^k \quad .$$

2.2 Continuous Group-Key Agreement

We now establish syntax for *continuous group-key agreement* (CGKA) schemes. A CGKA scheme allows a group G of users to agree on a group key that is to be used to secure communication within the group. In order to be able to recover from corruption users can also, possibly concurrently, send update messages, which rotate their key material. On top of this, CGKA schemes normally allow for group membership to evolve throughout the execution, by adding or removing users. However, while schemes allowing for theses additional operations are desirable in practice, the main goal of this work is to establish lower bounds on the communication complexity of recovering from corruption by concurrent updates. Thus, we restrict our view to static groups, i.e., we do not require the functionality of adding users to or removing users from the group. Not considering adds and removes allows for less technical notation, and we point out that lower bounds only profit from this restriction, as they hold even for schemes restricted to static groups. In doing so, our syntax essentially follows that of [BDR20], with a couple of small differences mentioned below.

A continuous group-key agreement scheme CGKA specifies algorithms Setup, Init, Update, $\mathsf{Process}$, and GetKey. Algorithms Setup and Init can be used to initialize the a group G, that since we restrict our view to static groups, throughout this work we simply identify with $G = [n]$ for some $n \in \mathbb{N}$. Here, Setup is used to generate every user's initial internal state and can be thought of as the users generating a key pair and registering it with a PKI. Init, on the other hand, is called by one of the users to initialize the group. It generates a control message that, when processed by the other users, establishes the initial group-key K_G^0. Afterwards, the scheme proceeds in rounds t, in each of which a subset of users in G concurrently generate update messages using algorithm Update. The update messages are in turn processed by the group members resulting in a new group key K_G^t that can be recovered from a user's internal state using algorithm GetKey.

More formally,

- $\mathsf{Setup}(n; r)$ on input the group size n and random coins r belonging to randomness space Rnd outputs public information pub, as well as an initial state st_u for every user $u \in G = [n]$.
- $\mathsf{Init}(st_u, pub; r)$ receives as input a user's (initial) state, the public information pub, and random coins r. Its output (st'_u, MI_u) consists of the initializing user's updated state and a control message MI_u.

```
Game $\mathrm{CORRECT}^{\mathsf{CGKA}}(n, u_0, (U_t)_{t=1}^{t_{\max}})$
$t \leftarrow 0$
$\mathsf{INIT}(n, u_0)$
while $t \leq t_{\max}$:
    $t \leftarrow t + 1$
    $\mathsf{ROUND}(U_t)$
return 0

Oracle $\mathsf{INIT}(n, u_0)$
$(pub, (st_u)_{u \in [n]}) \leftarrow \mathsf{Setup}(n)$
$(st_{u_0}, MI^0_{u_0}) \leftarrow \mathsf{Init}(st_{u_0}, pub, n; r)$
for $u \in [n]$:
    $st_u \leftarrow \mathsf{Process}(st_u, pub, MI^0_{u_0})$
    $K^u_G \leftarrow \mathsf{GetKey}(st_u)$
if $\exists u, v \in [n] : K^u_G \neq K^v_G$:
    return 1

Oracle $\mathsf{ROUND}(U_t)$
$MU \leftarrow \emptyset$
for $u \in U_t$:
    $(st_u, MU^t_u) \leftarrow \mathsf{Update}(st_u, pub)$
    $MU \leftarrow_{\cup} MU^t_u$
for $u \in [n]$:
    $st_u \leftarrow \mathsf{Process}(st_u, pub, MU)$
    $K^u_G \leftarrow \mathsf{GetKey}(st_u)$
if $\exists u, v \in [n] : K^u_G \neq K^v_G$:        \\ disagreement on group key
    return 1
```

Figure 3: Correctness game for continuous group-key agreement scheme CGKA.

- $\mathsf{Update}(st_u, pub; r)$ in round t takes as input a user's current state, the public information pub, and random coins r. It returns updated state st'_u and a update message MU^t_u.
- Deterministic algorithm $\mathsf{Process}(st_u, pub, M)$ gets as input a user u's state, the public information pub, and a set M of control messages that either consists of a single group initialization message MI_v, or a family of update messages $(MU_v)_v$. Its output is the processing user's updated state st'_u.
- Deterministic algorithm $\mathsf{GetKey}(st_u)$ on input a user's state returns u's view of current group key K_G belonging to key space $\mathsf{CGKA.KS}$.

When comparing to the syntax of [BDR20], one can find two differences. On the one hand, we chose not to merge algorithms Setup and Init, as in [BDR20], although this would be possible since we consider the simple setting where a single static group is created. On the other hand, we include the algorithm GetKey, present in the original CGKA definition from [ACDT20]. This makes it easier to argue the connection between the combinatorial and symbolic models. The two main properties that we require from a CGKA scheme are correctness and security.

Correctness. For correctness we essentially require that, for every valid sequence of operations, in every round t, all users agree on the current group key K^t_G where $G = [n]$ is a static group of cardinality $n \in \mathbb{N}$. We capture the notion of correctness formally in the game of Figure 3. The game gets as input n, the user $u_0 \in G$ initializing the group, and a sequence $(U_t)_t$ of updates to be applied in every round where $U_t \subseteq G$. The game returns the value 0 if the execution was correct and 1 otherwise. Accordingly, we say that a scheme CGKA is *perfectly correct* if, for every input $(n, u_0, (U_t)_{t=1}^{t_{\max}})$ and all choices of random coins, we have that $0 = \mathrm{CORRECT}^{\mathsf{CGKA}}(n, u_0, (U_t)_{t=1}^{t_{\max}})$. As, at every point in time, for a perfectly correct scheme, all users agree on the current group key, we will denote it simply by K_G, instead of K^u_G.

Security. For security, we require that the group key of a CGKA scheme recovers from corruption assuming that every party did at least k updates since their last corruption. More formally, we consider the security notions of indistinguishability of the group key from random (IND-k-PCS$_{\mathrm{mode}}$), and one-wayness (OW-k-PCS$_{\mathrm{mode}}$). Here, mode $\in \{\neg\mathrm{RC}, \mathrm{RC}\}$ indicates whether the corruption of a user reveals only their private state in the current round, or also additionally the random coins they sampled in the round. Thus, we end up with 4 different security notions. The weakest, OW-k-PCS$_{\neg\mathrm{RC}}$, is used for our lower bounds in Section 3.2 and the strongest, IND-k-PCS$_{\mathrm{RC}}$, for our new upper bound of Section 5.

The security games are formally defined in Figure 4. They provide the adversary A with an initialization oracle INIT that allows for a single query, that has to be made before using any of the other oracles. It enables A to set up a universe of users and initialize a group. Using oracle ROUND, the adversary can specify sets of users to concurrently perform updates. All operations are then processed by the members of the group, and the round counter t is increased. Further, A can, at any point in time, use the corruption oracle $\mathsf{CORR}(u)$ to reveal user u's current internal state st_u, and, in the case that mode = RC, additionally the random coins u sampled in the current round while updating. Finally, A, at an arbitrary point in time t^*, can make a single query to the challenge oracle CHALL, which in Game $\text{IND-}k\text{-PCS}^{\mathsf{CGKA}}_{\text{mode}}(\mathsf{A})$, depending on challenge bit b^*, returns either the current group key or a uniformly random key. The adversary wins if it is able to correctly guess b^* and safety predicate safe-k-PCS holds. In Game $\text{OW-}k\text{-PCS}^{\mathsf{CGKA}}_{\text{mode}}(\mathsf{A})$, the oracle instead stores the current group key as challenge key K^*. This has to be computed by A in order to win, again with the restriction that safe-k-PCS holds. The predicate safe-k-PCS verifies that, for every user that at time t^* is a member of the group, (a) they were never corrupted after t^* and (b) since their last corruption before t^* they performed at least k updates.

Definition 2 (k-PCS security). *Let* CGKA *be a continuous group-key agreement scheme,* $k \in \mathbb{N}$*, and* mode $\in \{\text{RC}, \neg\text{RC}\}$*. Then* CGKA *is* $\text{IND-}k\text{-PCS}_{\text{mode}}$ secure*, if for every PPT adversary the advantage function*

$$\left|\Pr[\text{IND-}k\text{-PCS}^{\mathsf{CGKA}}_{\text{mode}}(\mathsf{A}) \Rightarrow 1 \mid b^* = 1] - \Pr[\text{IND-}k\text{-PCS}^{\mathsf{CGKA}}_{\text{mode}}(\mathsf{A}) \Rightarrow 1 \mid b^* = 0]\right| \leq \text{negl}$$

is negligible.

Further, CGKA *is* $\text{OW-}k\text{-PCS}_{\text{mode}}$ secure*, if for every PPT adversary the advantage function*

$$\Pr[\text{OW-}k\text{-PCS}^{\mathsf{CGKA}}_{\text{mode}}(\mathsf{A}) \Rightarrow 1] \leq \text{negl}$$

is negligible.

Remark 1. *We make the following observation about the security model.*

(i) In this work we are interested in the communication cost of achieving post-compromise security, and thus ignore attacks breaching forward secrecy, i.e., learning group keys from previous rounds by corrupting users. This is encoded in lines 38 and 39 of the safe predicate, which disallow corrupting users after the challenged round t^*.

(ii) Our security model is quite weak. In particular, all initialization and update operations are honestly generated and immediately processed by all users in synchronous rounds. We point out that this only strengthens our lower bounds, as they hold even for a security notion far weaker than what one would aim for in practice. While this leaves open the possibility of improving on our bounds by switching to a stronger security notion, we point out that they are closely matched by the upper bound of Section 5, which we expect to be easily made secure in asynchronous settings with a semi-honest server using standard techniques to ensure consistency (e.g. signatures, a key schedule, transcript and parent hashes, etc.[AAN+22a, BBR18, AJM22]).

Restrictions. Our lower bounds apply to CGKA schemes CGKA satisfying the following two restrictions.

- CGKA does not use nested encryption (NNE). This means that users do not create layered ciphertexts of the form $\mathsf{Enc}(pk_1, \mathsf{Enc}(pk_2, m))$.
- CGKA does not distribute work (NDW). This means that, if in any round a user get access to a secret they did not previously possess, then they must have either sampled it by themselves or recovered it from the update message of a single user.

The properties are not directly exploited in our proofs in the combinatorial model. Instead, we use them to show that bounds in the combinatorial model also hold in the symbolic model. We defer the restrictions' formal definitions to Section 4 (Def. 5 and Def. 7), where we will also formally justify their impact on the combinatorial model. We point out that all CGKA schemes that we are aware of satisfy both properties.

We also consider an additional property. We say that CGKA has publicly-computable update cost (PCU) if it is always possible to determine the size $|MU_u|$ of an update that a user u would produce if asked to

```
Game IND-$k$-PCS$^{\mathsf{CGKA}}_{\mathrm{mode}}$(A)
$b^* \leftarrow_\$ \{0,1\}$
$b \leftarrow \mathsf{A}^{\mathsf{INIT,ROUND,CORR,CHALL}}$
return $[b = b^*] \wedge \mathrm{safe}_{k\text{-PCS}}$

Game OW-$k$-PCS$^{\mathsf{CGKA}}_{\mathrm{mode}}$(A)
$K \leftarrow \mathsf{A}^{\mathsf{INIT,ROUND,CORR,CHALL}}$
return $[K^* = K] \wedge \mathrm{safe}_{k\text{-PCS}}$

Oracle INIT$(n, u_0)$                    \\ one call; called first
$t \leftarrow 0$
$\mathrm{Cor}[u] \leftarrow \emptyset$, $\mathrm{Upd}[u] \leftarrow \emptyset$ for all $u \in [n]$
$\mathrm{Coins}[u,t] \leftarrow \emptyset$ for all $u \in [n]$, $\forall t$
$r_{\mathbf{setup}} \leftarrow_\$ \mathit{Rnd}$
$(\mathit{pub}, (st_u)_{u\in[n]}) \leftarrow \mathsf{Setup}(n; r_{\mathbf{setup}})$
$r \leftarrow_\$ \mathit{Rnd}$; $\mathrm{Coins}[u_0,t] \leftarrow_\cup \{r\}$
$(st_{u_0}, \mathit{MI}^0_{u_0}) \leftarrow \mathsf{Init}(st_{u_0}, \mathit{pub}, n; r)$
for $u \in [n]$:
    $st_u \leftarrow \mathsf{Process}(st_u, \mathit{pub}, \mathit{MI}^0_{u_0})$
    $K_G \leftarrow \mathsf{GetKey}(st_u)$
return $\mathit{pub}, \mathit{MI}^0_{u_0}$

Oracle CHALL    \\ one call, Game IND-$k$-PCS$^{\mathsf{CGKA}}_{\mathrm{mode}}$(A)
$t^* \leftarrow t$
$K_0 \leftarrow_\$ \mathsf{CGKA.KS}$; $K_1 \leftarrow K_G$
return $K_{b^*}$

Oracle CHALL    \\ one call, Game OW-$k$-PCS$^{\mathsf{CGKA}}_{\mathrm{mode}}$(A)
$t^* \leftarrow t$
$K^* \leftarrow K_G$

Oracle CORR$(u)$
$\mathrm{Cor}[u] \leftarrow_\cup \{t\}$
return $(st_u, \mathrm{Coins}[u,t])$          \\ if mode = RC
return $st_u$                                 \\ if mode = ¬RC

Oracle ROUND$(U_t)$
$t \leftarrow t+1$
$\mathit{MU} \leftarrow \emptyset$
for $u \in U_t$:
    require $u \in [n]$
    $\mathrm{Upd}[u] \leftarrow_\cup \{t\}$
    $r \leftarrow_\$ \mathit{Rnd}$; $\mathrm{Coins}[u,t] \leftarrow_\cup \{r\}$
    $(st_u, \mathit{MU}^t_u) \leftarrow \mathsf{Update}(st_u, \mathit{pub}; r)$
    $\mathit{MU} \leftarrow_\cup \mathit{MU}^t_u$
for $u \in [n]$:
    $st_u \leftarrow \mathsf{Process}(st_u, \mathit{pub}, \mathit{MU})$
    $K_G \leftarrow \mathsf{GetKey}(st_u)$
return $\mathit{MU}$

Predicate $\mathrm{safe}_{k\text{-PCS}}$
for $u \in [n]$:
    if $\mathrm{Cor}[u] \neq \emptyset$
        if $\exists t \in \mathrm{Cor}[u] : t \geq t^*$:          \\ corruption after challenge
            return 0
        $t^c_u \leftarrow \max\{t \in \mathrm{Cor}[u]\}$
        if $|\{t \in \mathrm{Upd}[u] : t^c_u < t \leq t^*\}| < k$:   \\ too few updates
            return 0
return 1
```

Figure 4: Security games IND-k-PCS$_{\mathrm{mode}}$ for indistinguishability and OW-k-PCS$_{\mathrm{mode}}$ for one-wayness of group keys with respect to mode of randomness-corruption mode $\in \{\mathrm{RC}, \neg\mathrm{RC}\}$. The game is defined with respect to continuous group-key agreement scheme CGKA and adversary A. We require that the adversary's first call is to oracle INIT, which can only be queried once.

update given access only to public information, i.e., *pub*, as well as the sets of update messages sent so far. With this additional property we can show that not only there exists a sequence of updates for which the total communication cost is at least roughly $n^{1+1/k} \cdot k/\log(k)$, but it is also possible to find the sequence using only public information. Formally, CGKA schemes with publicly-computable update cost are defined as follows.

Definition 3 (Publicly-computable update cost)**.** *Let* CGKA *be a CGKA scheme. Consider an execution of the security game* IND-k-PCS$_{\mathrm{mode}}$ *(or* OW-k-PCS$_{\mathrm{mode}}$*). We say that* CGKA *has* publicly-computable update cost *if, for every round t and for every user $u \in G_t$ with internal state st^t_u and public information pub, it is possible to efficiently compute $|\mathit{MU}|$, where $(st', \mathit{MU}) \leftarrow \mathsf{Update}(st^t_u, \mathit{pub}; r)$ would be the output of calling the update procedure, from public information at the end of round $t-1$ (i.e., all messages $\mathit{MI}^0_{u_0}$ and $\mathit{MU}^{t'}_u$ sent in any round $t' \leq t-1$, the sequence $(U_{t'})_{t'\leq t-1}$, n, k, pub and u_0). Note that, in particular, the size of MU must be independent of the random coins r used to generate the update message. We say that that* CGKA *has* offline publicly-computable update cost *if the same property holds using only the initialization messages $\mathit{MI}^0_{u_0}$, the sequence $(U_{t'})_{t'\leq t-1}$, n, k, pub and u_0.*

We point out that all CGKA schemes that we are aware of have offline publicly-computable update cost. For example, for schemes based on ratchet trees, as for example TreeKEM or CoCoA, the size of every user's next update is fully determined by the position of blank and non-blank nodes in the ratchet tree, which can be determined given just the sequence of update / propose-commit operations.

3 Lower Bounds in the Combinatorial Model

In this section we define a self-contained combinatorial model capturing CGKA schemes recovering from corruption in k rounds of updates and then prove a lower bound on the communication complexity of such schemes. The model is given in Section 3.1, the bound in Section 3.2.

3.1 The Combinatorial Model

We now present a purely combinatorial model capturing an adversary interacting with a correct and secure CGKA scheme built from public-key encryption and pseudorandom functions. The interaction proceeds in rounds, in each of which the users schedule update, add, and remove operations and at the end of which a set of users is corrupted.

High-level structure. An instance of the combinatorial model is characterized by a tuple $(n, k, t_{\max}, C_0)$ and a sequence $(U_t, C_t)_{t=1}^{t_{\max}}$, where $n, k, t_{\max} \in \mathbb{N}$, $C_0 \subseteq [n]$, and $U_t, C_t \subseteq [n]$ for all t. Intuitively, this corresponds to setting up the group $G = [n]$ and in round 0 corrupting the set of users C_0. The sequence $(U_t, C_t)_{t=1}^{t_{\max}}$ determines the operations performed in the following $t_{\max}$ rounds, where

- U_t is the set of users performing an update in round t, and
- C_t is the set of users corrupted at the end of round t.

Integer k determines the safety requirement imposed on the CGKA scheme. More precisely, we aim to capture CGKA schemes that recover from corruption after k updates, meaning that if every user did at least k updates since the last round in which they were corrupted, then the group must agree on a secure key. Formally, consider an instantiation of the combinatorial model with respect to $(n, k, t_{\max}, C_0)$ and $(U_t, C_t)_t$ as described above. We say that a round $t \in \{0, \ldots, t_{\max}\}$ is *safe*, if for every user $u \in G$ such that $u \in C_{t'}$ for some t' there exist rounds $t_1, \ldots, t_k$ such that

$$u \in U_{t_i} \text{ for all } i \in \{1, \ldots, k\} \quad \text{and} \quad \max\{t_c \in \{0, \ldots, t_{\max}\} : u \in C_{t_c}\} < t_1 < \cdots < t_k \leq t \ . \tag{1}$$

Recall that since we want to only argue about post-compromise security but not forward-secrecy the condition also excludes the corruption of users *after* round t.

Set system and cost function. The main intuition behind the combinatorial model is to associate the secure PKE and PRF keys present in the CGKA scheme in round t to the set of users in G that have access to them at this point in time, i.e, can recover them from their current internal state. Here 'secure key' refers to keys that were established by update operations and cannot be trivially recovered from the adversary. The adversary is able to get access to keys directly by corrupting users' states, or by recovering them from protocol messages. The latter is possible if the message contains an encryption of the key under a key the adversary has access to, or if it contains the key in plain. In every round the sets $S(sk)$ of users having access to secure keys sk form a subset of 2^G. Intuitively, security and correctness of a CGKA scheme imply that the system of associated sets should satisfy certain properties, and that adding sets to it by scheduling updates comes at the cost of sending ciphertexts. These properties are stated below and, looking ahead, will serve as the main tools to derive our lower bound. For an illustration of the set system corresponding to a ratchet tree as described in the introduction see Figure 5 (Top).

Formally, consider an instantiation of the combinatorial model with respect to $(n, k, t_{\max}, C_0)$ and $(U_t, C_t)_t$. We require the existence of a cost function Cost and a sequence $(\mathcal{S}_t)_{0 \leq t \leq t_{\max}}$ of set systems $\mathcal{S}_t \subseteq 2^G$. The cost function and sequences are required to satisfy three properties to be given further below. The cost function takes as input

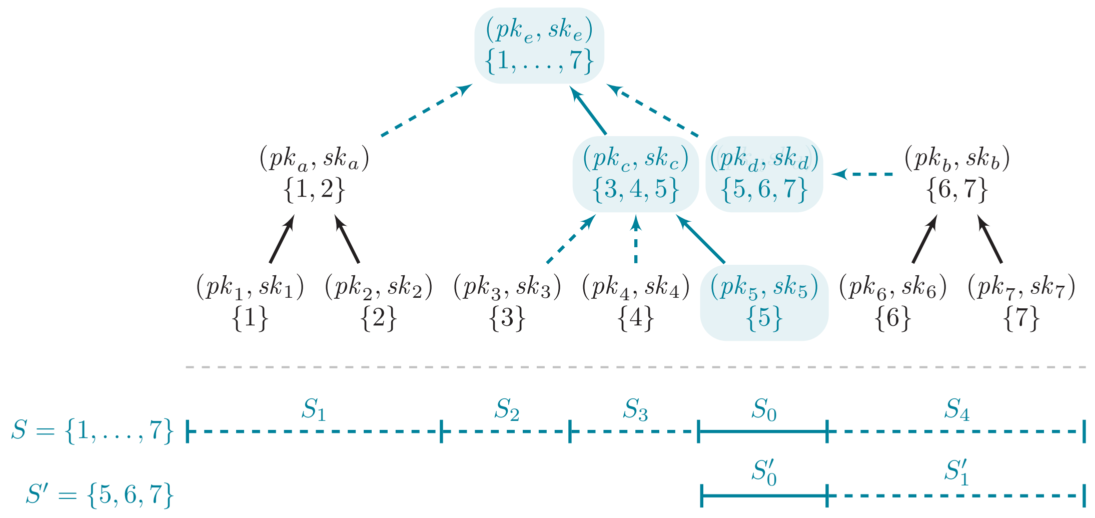

Figure 5: Top: Illustration of a ratchet tree and its associated set system $\mathcal{S}_t$. Vertices contain key-pairs (above) and the associated set (below). Keys already present in the system at time $t-1$ are depicted in black and keys added by user 5 in round t in blue with shaded background. Edges indicate that knowledge secret key of the source implies knowledge of the one of the sink. Dashed, blue edges correspond to ciphertexts $\mathsf{Enc}_{pk_{\text{source}}}(sk_{\text{sink}})$ sent by user 5 in round t, solid edges either to keys derived using a PRF in round t (depicted in blue) or to keys communicated in a previous round (depicted in black). Accordingly, user 5 generated key-pairs (pk_5, sk_5) and (pk_d, sk_d) using fresh randomness, and (pk_c, sk_c) and (pk_e, sk_e) using a PRF. Bottom: Depiction of the sets required to exist by property (iii) using the examples $S = \{1,\dots,7\} = \bigcup_{i=0}^{k} S_i$ and $S' = \{5,6,7\} = \bigcup_{i=0}^{k'} S'_i$ with $k = 4$ and $k' = 1$ corresponding to the secret keys sk_e and sk_d respectively. We have $S_0 = \{5\} = S'_0$, $S_1 = \{1,2\}$, $S_2 = \{3\}$, $S_3 = \{4\}$, and $S_4 = \{6,7\} = S'_1$. Note that the number of ciphertexts sent to communicate the secret keys corresponding to S and S' to their members are $5 > k$ and $1 = k'$ respectively, thus satisfying the inequality on the user's cost function required by property (iii).

- the user $u \in G$ performing the update operation,
- the round t with $1 \le t \le t_{\max}$, and
- the history $M_t = (n, k, (U_{t'})_{1 \le t' < t})$ of sets of users performing updates in the previous rounds.

Its output is an integer $\text{Cost}(u, t, M_t)$. For better legibility, we will simply write $\text{Cost}(u, t)$ whenever the third input is clear from context.

Note that while the cost of a user's update in a given round depends on the operations performed in previous rounds, it does not depend on the sets $C_{t'}$ of users corrupted in previous rounds. The latter is justified, since, looking ahead, in the security game in the symbolic model that we aim to capture users are not aware whether they are corrupted or not. However, if they are asked to update by the adversary, they may decide to create particular ciphertexts depending on the the history of operations performed so far as these may have had an impact on their internal state.

Requirements on set system and cost function. We now give three properties to be satisfied by the cost function and the set system.

(i) Correctness of the CGKA scheme implies that the members of the group share a common key. Further, by security, whenever a round is safe, the key they shared among all users of the group must not be known to the adversary at this point in time.

Formally, if round t is safe we require that $G = [n] \in \mathcal{S}_t$.

(ii) If a user is corrupted in some round, all keys they currently have access to can also be recovered by the adversary and therefore should be considered insecure. This is represented by $\mathcal{S}_t$ not containing any sets that include a party corrupted in round t.

Formally, for all $t \in \{0, \ldots, t_{\max}\}$ and all $u \in C_t$ we have that $S \in \mathcal{S}_t$ implies that $u \notin S$.

(iii) The third property captures how users agree on new keys when using basic cryptographic primitives (PRFs and PKE) and which cost in terms of ciphertexts sent is incurred by communicating these keys to other users. A user $u \in G$ can always sample a new key locally. Further, from such a key or one already present in the system they can derive a chain of new keys using PRF evaluations. To communicate the key sk to other users they can encrypt it it under a public key pk' that either must have been present in the system at the end of round $t-1$ or must previously have been generated by the u in round t. From the resulting ciphertext every user with access to the corresponding sk' is able to derive sk as well as all keys derived from sk using PRF evaluations. Note that if sk' is insecure, then the adversary can recover sk.

In terms of sets this essentially means that the set $S \in \mathcal{S}_t$ of users able to recover sk can be covered by a union of sets in $\mathcal{S}_{t-1}$ (and potentially a singleton $\{u\}$ in case user u generated the starting point of the PRF evaluation chain from fresh randomness) and that the cost of the user communicating sk to the other members of S should be at least the number of sets forming the union (where sometimes one ciphertext can be saved, as the key serving as a starting point of a chain of PRF evaluations does not have to be communicated).

Formally, for every $t \geq 1$ and every $S \in \mathcal{S}_t$ we require that there exist $h \in \mathbb{N}_{\geq 0}$ and $S_0, \ldots, S_h$, such that either

$$S_0 = \{u\} \text{ for some } u \in U_t \quad \text{or} \quad S_0 \in \mathcal{S}_{t-1} \ , \tag{2}$$

and if $h \geq 1$ then $S_i \in \mathcal{S}_{t-1}$ for all $i \in \{1, \ldots, h\}$. Further, we require that

$$S \subseteq \bigcup_{i=0}^{h} S_i \tag{3}$$

and, regarding the cost function, that

$$\exists u \in S_0 \cap U_t \text{ such that } \mathrm{Cost}(u,t) \geq h \ . \tag{4}$$

Note that if in Equation 2 we have $S_0 = \{u\}$ then the user in Equation 4 must be u. Finally, we can connect the cost of adding a set to the set system $\mathcal{S}_t$ to its MinCover with respect to $\mathcal{S}_{t-1}$. Indeed, for $S \in \mathcal{S}_t$, if u is the user required to exist by Equation 4, then by Equations 2, 3, and 4 it holds that

$$\mathrm{Cost}(u,t) \geq |\mathrm{minCover}_{\supseteq}(S, \mathcal{S}_{t-1} \cup \{u\})| - 1 \ . \tag{5}$$

The precise connection between the combinatorial model and the symbolic model is established later in Section 4. There, we essentially show that an adversary playing the one-wayness security game in the symbolic model with respect to a correct and secure CGKA scheme that satisfies the restrictions described in Section 2.2 implies the existence of a set system $\mathcal{S}_t$ satisfying Properties (i)-(iii) if one uses the number of ciphertexts sent by a user u in round t as cost function $\mathrm{Cost}(u,t)$.

3.2 Lower Bound in the Combinatorial Model

We now prove a lower bound on the communication cost required to recover from compromise within k rounds in the combinatorial model. Conceptually, our proof proceeds in two steps. First, we lower bound the sum of the maximal per-user update cost over all rounds. This bound is a best-case bound, i.e., it holds with respect to every sequence $(U_t)_t$ of updating users. In a second step we then prove our main result, a bound on the total cost required to recover from corruption. This bound is worst case, i.e., it holds with respect to an adversarially chosen sequence of updating users. Concretely, we will exploit that the cost of a user $u \in U_t$ updating in round t does not depend on the cost of other members of U_t updating concurrently.

This enables us to find a sequence $(U_t)_t$ for which all members of U_t have roughly the same update cost, which yields the desired bound as the bound on the maximal per-user update cost implies that the cost of the users in U_t in sufficiently many rounds t must be quite large.

Lower bound on the maximal per-user update cost. We first consider the scenario that after an arbitrary setup phase of t_c rounds a set of c users in $G = [n]$ is corrupted and that after m subsequent rounds of updates we have $G \in \mathcal{S}_t$ (intuitively corresponding to the existence of a secure group key). Below, we bound the sum of the maximal per-user update cost over the m rounds. Note that this bound holds irrespective of how the sets U_t of updating users are chosen.

Proposition 2. *Let $n, k, t_c, m \in \mathbb{N}$, $C \subsetneq [n]$, and $c = |C|$ such that $\ln(c) \geq m - 1$. Let $t_{\max} = t_c + m$ and consider an instantiation of the combinatorial model with respect to $(n, k, t_{\max}, C_0)$, $(U_t, C_t)_t$, where $C_{t_c} = C$, $C_t = \emptyset$ for $t \neq t_c$, and $(U_t)_t$ is an arbitrary sequence.*

If $G = [n]$ is contained in the set-system $\mathcal{S}_{t_{\max}}$ at the end of round $t_{\max} = t_c + m$, then we have

$$\sum_{t=1}^{m} \max_{u \in U_{t_c+t}} \mathrm{Cost}(u, t_c + t) \geq (m-1)\left(\sqrt[m-1]{c} - 1\right) .$$

Proof. For $t \in \{1, \ldots, m\}$ let $s_t = \max\{|S \cap C| : S \in \mathcal{S}_{t_c+t}\}$ denote the largest number of (previously) corrupted users that are contained in an element of $\mathcal{S}_{t_c+t}$.

Let $t_{\min} \in \{1, \ldots, m\}$ be minimal such that $s_{t_{\min}} > 0$. We now argue that $s_{t_{\min}} = 1$. Note that by Property (ii), in round t_c no $S \in \mathcal{S}_{t_c}$ contains any of the users in C. Now, for contradiction assume $s_{t_{\min}} > 1$, implying the existence of a set $S \in \mathcal{S}_{t_c+t_{\min}}$ containing more than one formerly corrupted user. Let $S_0, \ldots, S_h$ be the sets guaranteed to exists by Property (iii). Then S_0 must contain at most one user of C as $C \cap S = \emptyset$ for all $S \in \mathcal{S}_{t_c+t_{\min}-1}$. Further, if $h \geq 1$ then all S_i with $i \geq 1$ are elements of $\mathcal{S}_{t_c+t_{\min}-1}$ and thus cannot contain any users in C. This is in contradiction to $s_{t_{\min}} > 1$. Thus, we have $s_{t_{\min}} = 1$ as well as $s_m = c$, which directly follows from $[n] \in \mathcal{S}_{t_c+m}$. This implies

$$\prod_{t=t_{\min}}^{m} s_t / s_{t-1} = s_m / s_{t_{\min}} = c \ . \tag{6}$$

Further, we have $\max_{u \in U_{t_c+t}} \mathrm{Cost}(u, t_c + t) \geq s_t/s_{t-1} - 1$ for every $t_{\min} + 1 \leq t \leq m$. To see this consider $S \in \mathcal{S}_{t_c+t}$ with $|S \cap C| = s_t$. By Property (iii) we have that $S \subseteq S_0 \cup \bigcup_{i=1}^{h} S_i$ and $\mathrm{Cost}(u, t_c + t) \geq h$ for some user $u \in U_t$, $h \in \mathbb{N}_{\geq 0}$, $S_0 \in \mathcal{S}_{t_c+t-1} \cup \{u\}$, and $S_i \in \mathcal{S}_{t_c+t-1}$. Since every set in $\mathcal{S}_{t_c+t-1}$ and in turn also S_0 contains at most s_{t-1} users in C we have that the cover of S must consist of at least s_t/s_{t-1} sets, i.e., we have $\max_{u \in U_{t_c+t}} \mathrm{Cost}(u, t_c + t) + 1 \geq h + 1 \geq s_t/s_{t-1}$ as desired.

To conclude, we will make use of the inequality of arithmetic and geometric means (Proposition 1). By setting $m' = m - t_{\min}$ and $x_t = s_{t_{\min}+t}/s_{t_{\min}+t-1}$ for $t = 1, \ldots, m'$ and defining $x = \sum_{t=1}^{m'} x_t$ we obtain from Equation 6 that $c \leq \prod_{t=1}^{m'} x_t \leq (x/m')^{m'}$. Solving for x gives

$$\sum_{t=1}^{m} \max_{u \in U_{t_c+t}} \mathrm{Cost}(u) \geq \sum_{t=t_{\min}+1}^{m} (s_t/s_{t-1} - 1) = x - m' \geq m'\left(\sqrt[m']{c} - 1\right) \geq (m-1)\left(\sqrt[m-1]{c} - 1\right) ,$$

the statement of the proposition. Here, in the last step we used the fact that the function $m' \mapsto m' \sqrt[m']{c}$ is monotonously decreasing on the interval $[1, \ln(c)]$ and that by assumption $m - 1 \leq \ln(c)$. $\square$

From maximal per-user cost to total-communication cost. We now show that for an adversarially chosen sequence $(U_t)_t$ of sets of updating users actually almost all users have to adhere to the bound derived in the previous paragraph. Intuitively, after an arbitrary warm up phase of t_c rounds and corrupting a linear fraction of users in round t_c, we construct $(U_t)_t$ such that either all updating users have roughly the same update cost, or all users have a very small update cost. This procedure will then be repeated for sufficiently many rounds to force that a linear fraction of all users in the group has updated at least k times. In this case the final round $t_{\max}$ must be secure enforcing that $G \in \mathcal{S}_{t_{\max}}$. This allows us to use the bound derived in the previous paragraph to show that the communication cost of rounds corresponding to the former case must be substantial. We obtain the following.

Theorem 3. *Let $k, n, t_c \in \mathbb{N}$ and $0 < \varepsilon < 2/5$ be a constant such that $(1+\varepsilon)k \in \mathbb{N}$. Set $\alpha_\varepsilon = \frac{\varepsilon - 5/2\varepsilon^2 + \varepsilon^3}{8(1+\varepsilon)} > 0$ and $t_{\max} = t_c + (1+\varepsilon)k$. If $3 \le k \le \ln(\alpha_\varepsilon n)$, then for every sequence $(U_t)_{t=1}^{t_c}$ there exists a set $C \subseteq [n]$ of size $\lceil \alpha_\varepsilon n \rceil$ and a sequence $(U_t)_{t=t_c+1}^{t_{\max}}$ such that the instantiation of the combinatorial model with respect to $(n, k, t_{\max}, \emptyset)$ and $(U_t, C_t)_{t=1}^{t_{\max}}$, where $C_{t_c} = C$ and $C_t = \emptyset$ if $t \neq t_c$, satisfies*

$$\sum_{t=1}^{(1+\varepsilon)k} \mathrm{Cost}(U_{t_c+t}) \ge \frac{(k-1)}{4} \cdot \left\lfloor \frac{2\varepsilon n}{5(1+\varepsilon)\lceil \log(k) \rceil} \right\rfloor \left((\alpha_\varepsilon n)^{\frac{1}{(1+\varepsilon)k-1}} - 1 \right) .$$

Proof. We postpone the definition of C and first describe how the sets of updating users are chosen. In round $t_c + t$ for every user u consider the communication cost $\mathrm{Cost}(u, t_c + t)$ incurred if the user would update in this round. Let $\delta = ((k-1)/(2(1+\varepsilon)k)) \cdot (\sqrt[(1+\varepsilon)k-1]{\alpha_\varepsilon n} - 1)$. Given $(U'_t)_{t' < t_c+t}$ the set U_{t_c+t} of users to update is chosen according to the following case distinction.

(a) Let $n' = n(1 - \varepsilon/(4\log(k)))$. If at least $n - n'$ users have an update cost $\mathrm{Cost}(u, t_c + t) > k \cdot \delta$, then U_{t_c+t} is chosen to be a set of $\lceil n - n' \rceil$ of those users.

(b) Else, if at least $n'(1 - \varepsilon/2)$ users have an update cost $\mathrm{Cost}(u, t_c + t) < \delta$ then U_{t_c+t} is chosen to be a set of $\lceil n'(1 - \varepsilon/2) \rceil$ of those users.

(c) Else, as we will show below, we can find a set $U_{t_c+t} \subseteq [n]$ of $\lceil n'(1-\varepsilon/2) \rceil$ users u with update costs for round $t_c + t$ satisfying

$$\sum_{u \in U_{t_c+t}} \mathrm{Cost}(u, t_c + t) \ge \max_{u \in U_{t_c+t}} \mathrm{Cost}(u, t_c + t) \cdot \frac{1}{2} \left\lfloor \frac{\varepsilon n'}{2 \lceil \log(k) \rceil} \right\rfloor . \tag{7}$$

Note, that if in any of the rounds case (a) occurs, then the total upload update cost is at least $\varepsilon n/(4\log(k)) \cdot k\delta = \varepsilon n(k-1)/(8(1+\varepsilon)\log(k))(\sqrt[(1+\varepsilon)k-1]{\alpha_\varepsilon n} - 1)$ and in particular exceeds the desired bound.

This allows us to restrict to the case, in which (a) never occurs, in the remainder of the proof. Note that in this case in every round $\lceil n'(1-\varepsilon/2) \rceil$ users update. We will show below, that this implies that after $(1+\varepsilon)k$ rounds of updates the set $U_{k\text{-}\mathsf{Update}}$ of users that updated in at least k of the rounds after t_c must have size at least $\lceil n\alpha_\varepsilon \rceil$. This, however, means that if we set $C = U_{k\text{-}\mathsf{Update}}$, then the round $t_{\max} = t_c + (1+\varepsilon)k$ must be safe as defined in Equation 1 which by Property (i) of the combinatorial model implies that $G = [n] \in \mathcal{S}_{t_{\max}}$. Thus we are in the setting of Proposition 2 for $m = (1+\varepsilon)k$ and $c = \lceil n\alpha_\varepsilon \rceil$ and obtain

$$\sum_{t=1}^{(1+\varepsilon)k} \max_{u \in U_{t_c+t}} \mathrm{Cost}(u, t_c + t) \ge ((1+\varepsilon)k - 1)(\sqrt[(1+\varepsilon)k-1]{\alpha_\varepsilon n} - 1) \ge (k-1)(\sqrt[(1+\varepsilon)k-1]{\alpha_\varepsilon n} - 1) . \tag{8}$$

In the following we denote the update cost of user u in round t_c+t by $c_u^t := \mathrm{Cost}(u, t_c+t)$. Note that in every round t_c+t, in which case (b) occurs, we have that the maximal update cost must be at most $\max_{u \in U_{t_c+t}} c_u^t < \delta = (k-1)/(2(1+\varepsilon)k) \cdot (\sqrt[(1+\varepsilon)k-1]{\alpha_\varepsilon n} - 1)$. Thus, if we denote the set of such rounds in $\{t_c+1, \dots, t_{\max}\}$ by $T_{(b)}$ and the set of rounds in which case (c) occurs by $T_{(c)}$, then we obtain $\sum_{t \in T_{(b)}} \max_{u \in U_{t_c+t}} c_u^t \le (k-1)/2 \cdot (\sqrt[(1+\varepsilon)k-1]{\alpha_\varepsilon n} - 1)$. Plugging this in Equation 8 yields $\sum_{t \in T_{(c)}} \max_{u \in U_{t_c+t}} c_u^t \ge (k-1)/2 \cdot (\sqrt[(1+\varepsilon)k-1]{\alpha_\varepsilon n} - 1)$. This by Equation 7 implies the theorem's statement, as

$$\begin{aligned} \sum_{t=1}^{(1+\varepsilon)k} \sum_{u \in U_{t_c+t}} c_u^t &\ge \sum_{t \in T_{(c)}} \sum_{u \in U_{t_c+t}} c_u^t \ge \frac{1}{2} \left\lfloor \frac{\varepsilon n'}{2\lceil \log(k) \rceil} \right\rfloor \sum_{t \in T_{(c)}} \max_{u \in U_t} c_u^t \\ &\ge \frac{(k-1)}{4} \cdot \left\lfloor \frac{2\varepsilon n}{5(1+\varepsilon)\lceil \log(k) \rceil} \right\rfloor (\sqrt[(1+\varepsilon)k-1]{\alpha_\varepsilon n} - 1) , \end{aligned}$$

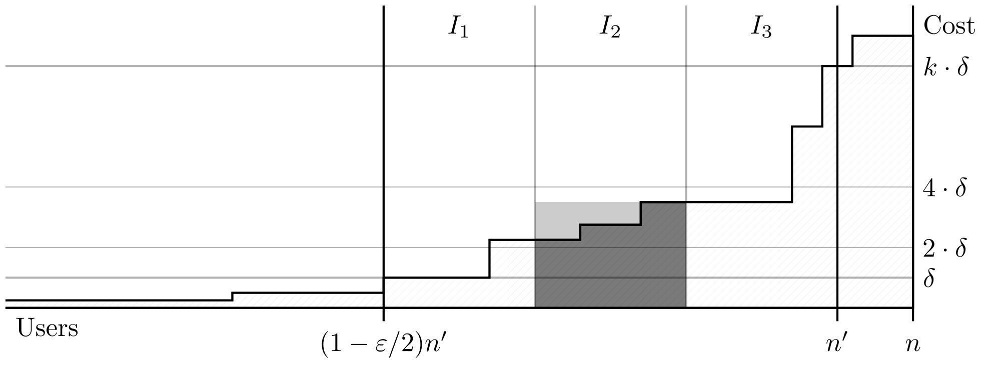

Figure 6: Existence of U_t satisfying the property of case (c) for $k = 8$. The interval $[(1 - \varepsilon/2)n', n']$, of users with update cost between δ and $k \cdot \delta$ is split into $\log(k)$ subintervals I_1, I_2, I_3 of length $n'\varepsilon/(2\log(k))$. While within the left and right subintervals the minimal and maximal cost of users differs by more than a factor of 2, it does not for the middle subinterval. Such a subinterval must always exist as else the cost of user n' would exceed $k \cdot \delta$. As a consequence the sum of the costs of the users in the middle interval (shaded in dark gray) covers at least half of $\max_{u\in I_2} \mathrm{Cost}(u, t_c + t) \cdot |I_2|$, the maximal cost of a user in the interval times the interval length (area shaded in (dark and light) gray).

where in the last step we used that $n' = (1 - \varepsilon/(4\log(k)))n \geq 4n/5$ for $k \geq 3$, and that $1 + \varepsilon > 1$.

Thus, it only remains to show that (A) in case (c) a set satisfying Equation (7) exists, and (B) that $|U_{k\text{-}\mathsf{Update}}| \geq \lceil \alpha_\varepsilon n \rceil$.

Regarding (A), note that in case (c) at least $\lceil n' \rceil$ users have an update cost of at most δk and at least $\lceil \varepsilon n'/2 \rceil$ of these users have an update cost larger or equal to δ. Assume w.l.o.g. that the users are ordered by update cost, i.e., $c^t_1 \leq c^t_2 \leq \cdots \leq c^t_n$. The intuition behind the statement we aim to prove is that $\delta \leq \mathrm{Cost}(u, t_c + t) \leq \delta k$ for the users in the interval $[\lfloor (1 - \varepsilon/2)n' \rfloor, \lceil n' \rceil]$. Thus, if we divide the interval into $\log(k)$ subintervals of length approximately $n'\varepsilon/(2\log(k))$ then for at least one of those intervals the cost of the start u_{j-1} and end point u_j must differ by at most a factor of 2. Thus if one lets the $\lceil (1 - \varepsilon/2)n' \rceil$ users preceding u_j update, already the sum of the update cost of the users contained in the subinterval satisfies Inequality 7. For an illustration of this see Figure 6.

Formally, for $i = 0, \ldots, \lceil \log(k) \rceil$ consider the update cost of user $u_i := \lfloor (1 - \varepsilon/2)n' \rfloor + i \cdot \lfloor \lceil n'\varepsilon/2 \rceil / \lceil \log(k) \rceil \rfloor$. We then have $c^t_{u_o} \geq \delta$ and $c^t_{u_{\lceil \log(k) \rceil}} \leq k\delta$ (since $\lfloor (1 - \varepsilon/2)n' \rfloor + \lceil \log(k) \rceil \cdot \lfloor \lceil n'\varepsilon/2 \rceil / \lceil \log(k) \rceil \rfloor \leq \lceil n' \rceil$) implying that there exists $j \geq 2$ such that $c^t_{u_j}/c^t_{u_{j-1}} \leq 2$. Indeed, otherwise we would have $c^t_{u_{\log(k)}}/c^t_{u_1} = \prod_{i=1}^{\lceil \log(k) \rceil} c^t_{u_i}/c^t_{u_{i-1}} > 2^{\lceil \log(k) \rceil} \geq k$. Now by choosing $U_{t_c+t} = \{u_j - \lceil (1 - \varepsilon/2)n' \rceil + 1, \ldots, u_j\}$ and using that $(1 - \varepsilon/2)n' \geq \varepsilon n'/(2\log(k))$ we obtain

$$\sum_{u \in U_{t_c+t}} c^t_u = \sum_{u=u_j - \lceil (1-\varepsilon/2)n' \rceil + 1}^{u_j} c^t_u \geq \sum_{u=u_{j-1}}^{u_j} c^t_u \geq \left\lfloor \left\lceil \frac{n'\varepsilon}{2} \right\rceil \cdot \frac{1}{\lceil \log(k) \rceil} \right\rfloor \cdot \frac{c^t_{u_j}}{2} \geq \max_{u \in U}(c^t_u) \cdot \frac{1}{2} \left\lfloor \frac{\varepsilon n'}{2 \lceil \log(k) \rceil} \right\rfloor \ .$$

We now show (B). Note that if case (a) never occurs, then in every round a set of $\lceil (1 - \varepsilon/2)n' \rceil$ users updates. This implies that after $(1 + \varepsilon)k$ rounds $(1 + \varepsilon) \lceil (1 - \varepsilon/2)n' \rceil k$ updates were issued. Let $0 \leq \alpha \leq 1$ denote the fraction of users in $[n]$ that updated in at least k rounds. These users must have made at most $(1+\varepsilon)k \cdot \alpha n$ updates, while the sum of updates of the remaining users must have been at most $(k-1) \cdot (1-\alpha)n$. As the sum of these two terms must exceed the overall number of updates made, we obtain

$$(1 + \varepsilon)k\alpha n + (k - 1) \cdot (1 - \alpha)n \geq (1 + \varepsilon) \lceil (1 - \varepsilon/2)n' \rceil k \geq (1 + \varepsilon)(1 - \varepsilon/2)(1 - \varepsilon/(4\log(k)))nk \ ,$$

which by solving for α yields

$$\alpha \geq \frac{(1+\varepsilon)(1-\varepsilon/2)(1-\varepsilon/(4\log(k)))k/(k-1)-1}{(1+\varepsilon)k/(k-1)-1}$$
$$\geq \frac{(1+\varepsilon)(1-\varepsilon/2)(1-\varepsilon/4)-1}{2+2\varepsilon} \geq \frac{\varepsilon-5/2\varepsilon^2+\varepsilon^3}{8(1+\varepsilon)} = \alpha_\varepsilon \ .$$

finally, since the number αn of users updating in at least k rounds must be an integer, we even obtain that $\alpha n \geq \lceil \alpha_\varepsilon n \rceil$. □

While we phrased Theorem 3 as a single-stage experiment, i.e., only consider the communication required to recover from corruptions made in a single round, it easily carries over to a repeated experiment consisting of repeatedly corrupting a linear fraction of the users from which the group has to recover within $(1+\varepsilon)k$ rounds of updates. Note, that the setting of Theorem 3 allows for an arbitrary setup phase $(U_t)_{t\leq t_c}$ of t_c rounds. Thus, by simply applying the arguments in the proof iteratively to each recovery phase, we obtain that the derived bound holds even in an amortized sense, i.e., even in this setting the recovery from each corruption requires communication of order $nk\sqrt[(1+\varepsilon)k]{n}/\log(k)$.

Corollary 4. *Let k, n, t_c, ε, and α_ε be as in Theorem 3. Let $z_{\max} \in \mathbb{N}$ and for $0 \leq z < z_{\max}$ set $t_{c,z} = t_c + z \cdot (t_c + (1+\varepsilon)k)$ and $t_{\max} = z_{\max} \cdot (t_c + (1+\varepsilon)k)$. For all collections of sequences $(U_t)_{t=t_{c,z}-t_c+1}^{t_{c,z}}$ with $0 \leq z < z_{\max}$, there exist sets $C_z \subseteq [n]$ each of size $\lceil \alpha_\varepsilon n \rceil$ and collections of updates $(U_t)_{t=t_{c,z}+1}^{t_{c,z}+(1+\varepsilon)k}$ such that for every instantiation of the combinatorial model with respect to $(n, k, t_{\max}, \emptyset)$ and $(U_t, C_t)_{t=1}^{t_{\max}}$, where $C_{t_{c,z}} = C_z$ and $C_t = \emptyset$ if $t \notin \{t_{c,z} \mid 0 \leq z < z_{\max}\}$, we have*

$$\sum_{t=t_{c,z}+1}^{t_{c,z}+(1+\varepsilon)k} \mathrm{Cost}(U_{t_{c,z}+t}) \geq \frac{(k-1)}{4} \cdot \left\lfloor \frac{2\varepsilon n}{5(1+\varepsilon)\lceil \log(k) \rceil} \right\rfloor \left((\alpha_\varepsilon n)^{\frac{1}{(1+\varepsilon)k-1}} - 1 \right)$$

for very $0 \leq z < z_{\max}$.

4 Lower Bounds in the Symbolic Model

In this section define CGKA in a symbolic model, an approach introduced for public key encryption by Dolev and Yao [DY83], following the work on multicast encryption by Micciancio and Panjwani [MP04], and generalized to CGKA by Bienstock, Dodis, and Rösler [BDR20], who considered concurrent updates for schemes recovering in two rounds. A similar model was also used to lower bound the communication incurred by users in CGKA schemes in order to achieve PCS, in a setting of multiple groups [AAB+21]. We show how the questions we are interested in can be translated from the symbolic to the combinatorial model of Section 3, which allows us to conclude that the bounds derived in the combinatorial model also hold with respect to the symbolic model.

4.1 The Symbolic Model

We consider schemes constructed from pseudorandom functions and public-key encryption, both modeled as idealized primitives that take as input symbolic variables, and output symbolic variables. To more easily distinguish these from non-symbolic variables we use typewriter font. We use the following syntax.

(i) *Pseudorandom function:* Algorithm PRF takes as input a key `K` and a message `m` and returns a key `K'` = PRF(`K`, `m`).

(ii) *Public-key Encryption:* A PKE scheme consists of algorithms (PKE.Gen, PKE.Enc, PKE.Dec), where PKE.Gen on input of secret key `sk` returns the corresponding public key `pk`. PKE.Enc takes as input a public key `pk` and a message `m`, and outputs a ciphertext `c` ← PKE.Enc(`pk`, `m`) with message data type.

$\mathsf{PKE.Dec}$ takes as input a secret key $\mathtt{sk}$ and a ciphertext $\mathtt{c}$, and outputs a message $\mathtt{m} = \mathsf{PKE.Dec}(\mathtt{sk}, \mathtt{c})$. We assume perfect correctness: $\mathsf{PKE.Dec}(\mathtt{sk}, \mathsf{PKE.Enc}(\mathtt{pk}, \mathtt{m})) = \mathtt{m}$ for all $\mathtt{sk}$, $\mathtt{pk} = \mathsf{PKE.Gen}(\mathtt{sk})$, and messages $\mathtt{m}$.

As data types, we consider messages, public keys, secret keys, symmetric keys, and random coins, the latter being a terminal type. Which variables can be recovered from a set of messages $\mathtt{M}$, is captured by the *entailment relation* $\vdash$.

Data type		Grammar rules
Message $\mathtt{m}$	$\leftarrow$	$\mathtt{sk}, \mathtt{pk}, \mathsf{PKE.Enc}(\mathtt{pk}, \mathtt{m})$
Public key $\mathtt{pk}$	$\leftarrow$	$\mathsf{PKE.Gen}(\mathtt{sk})$
Secret key $\mathtt{sk}$	$\leftarrow$	$\mathtt{K}$
Key $\mathtt{K}$	$\leftarrow$	$\mathtt{r}, \mathsf{PRF}(\mathtt{K}, \mathtt{m})$
Random coin $\mathtt{r}$		terminal type
Entailment relation		
$\mathtt{m} \in \mathtt{M}$	$\Rightarrow$	$\mathtt{M} \vdash \mathtt{m}$
$\mathtt{M} \vdash \mathtt{m}, \mathtt{pk}$	$\Rightarrow$	$\mathtt{M} \vdash \mathsf{PKE.Enc}(\mathtt{pk}, \mathtt{m})$
$\mathtt{M} \vdash \mathtt{K}$	$\Rightarrow$	$\mathtt{M} \vdash \mathsf{PRF}(\mathtt{K}, \mathtt{m})$ for all $\mathtt{m}$
$\mathtt{M} \vdash \mathsf{PKE.Enc}(\mathtt{pk}, \mathtt{m}), \mathtt{sk} : \mathtt{pk} = \mathsf{PKE.Gen}(\mathtt{sk})$	$\Rightarrow$	$\mathtt{M} \vdash \mathtt{m}$

Note that the entailment relation captures (ideal) correctness and (ideal) security of PRF and PKE, as recovering a PRF output or an encrypted message from a ciphertext requires knowledge of the secret key. Security is effectively captured by the of a sequence of entailment relations that recover the appropriate message. Examples and further comments (in the setting of multicast encryption) can be found in [MP04, Section 3.2]. The set of messages which can be recovered from $\mathtt{M}$ using relation $\vdash$ is denoted by $\mathsf{Der}(\mathtt{M}) := \{\mathtt{m} : \mathtt{M} \vdash \mathtt{m}\}$.

We point out that the model of [BDR20] covers more primitives, concretely, dual PRFs, updatable PKE, and broadcast encryption. It is an interesting open question to consider whether a translation to our combinatorial model is also possible if one takes these additional primitives into account. For a brief discussion on challenges to overcome if one would allow dual PRFs see Remark 3 after the proof of this section's main result.

Continuous group-key agreement in the symbolic model. A CGKA scheme CGKA in the symbolic model follows the syntax of Section 2.2. Additionally, we require some of the inputs to CGKA's algorithms to be symbolic variables. Concretely, we require that the group keys $\mathtt{K}$, public and internal states $\mathtt{pub}$ and $\mathtt{st}$, random coins $\mathtt{r}$ as well as the control messages $\mathtt{MI}$ and $\mathtt{MU}$ are symbolic. They can also have a non-symbolic counterpart which we omit as the properties we study and the security game we consider in the symbolic model do not depend on the non-symbolic variables. However, we often distinguish between symbolic random coins $\mathtt{r}$ and non-symbolic randomness r as this is used in some of the proofs. Intuitively, symbolic randomness represents the new secrets being sampled, while non-symbolic randomness allows to capture the fact that the algorithms may flip a coin in order to determine their actions (e.g., the update algorithm might flip random coins to decide whether to generate certain ciphertexts or not). Further, we assume that the context symbolic variables, e.g., which key corresponds to a certain ciphertext, or which keys correspond to a particular set of users, are implicitly known to the algorithms.

We use the game of Figure 3 to define correctness of CGKA, where we additionally require that, for every algorithm, each of its symbolic outputs can be derived from its symbolic inputs using the entailment relation $\vdash$. E.g., if user u computes $(\mathtt{st}'_u, \mathtt{MU}_u) \leftarrow \mathsf{Update}(\mathtt{st}_u, \mathtt{pub}; \mathtt{r}, r)$, then we require that $\mathtt{st}'_u, \mathtt{MU}_u \in \mathsf{Der}(\{\mathtt{st}_u, \mathtt{pub}, \mathtt{r}\})$, and similarly if $\mathtt{st}'_u \leftarrow \mathsf{Process}(\mathtt{st}_u, \mathtt{pub}, \mathtt{M})$ then it must hold that $\mathtt{st}'_u \in \mathsf{Der}(\{\mathtt{st}_u, \mathtt{pub}, \mathtt{M}\})$.

Regarding security, we target the notion of OW-k-PCS$_{\neg\text{RC}}$ of Definition 2. Note that, since our goal is to prove lower bounds, using one-wayness as the targeted security notion only makes our results stronger compared to using indistinguishability.

We structure the game in rounds, that correspond to the oracle calls that occur between two subsequent calls to oracle ROUND. We say a query to some oracle was made in round 0 if it was made before the first query to ROUND, and in round t for $t \in \{1, \dots, t_{\max}\}$, if it was either the tth query to ROUND, or, for calls to CHALL or CORR, if it was made after the tth and before the $(t+1)$st query to ROUND. This allows us to fully characterize adversaries A by the sequence of inputs to the oracles made in each round. For round 0, these are the input (n, G_0, u_0) to INIT and the set C_0 of corrupted users; for round t, the set U_t of updating users queried to ROUND, as well as the set C_t of users corrupted during the round; and finally, t^* indicating in which round the single call to CHALL is made. An explicit description of the OW-k-PCS$_{\neg\mathrm{RC}}$ security game in the symbolic model can be found in Figure 8.

Definition 4 (Symbolic k-PCS security)**.** *Let* CGKA *be a continuous group-key agreement scheme,* $k \in \mathbb{N}$*. Then* CGKA *is* OW-k-PCS$_{\neg\mathrm{RC}}$ secure*, if for all* $(n, u_0, C_0, (U_t, C_t)_{t=1}^{t_{\max}}, t^*)$ *it holds that*

$$\Pr[\text{OW-}k\text{-PCS}_{\neg\mathrm{RC}}^{\mathsf{CGKA}}(n, u_0, C_0, (U_t, C_t)_{t=1}^{t_{\max}}, t^*) \Rightarrow 1] = 0$$

where the probability is taken over the non-symbolic randomness.

This notion of security in which the game is lost for any sequence $(n, u_0, C_0, (U_t, C_t)_{t=1}^{t_{\max}}, t^*)$ is standard in the literature of symbolic security and used, for instance, in [MP04] and [BDR20]. The requirement that the probability be zero, implies that the game is not won for every possible choice of non-symbolic randomness. The reason for this choice rather than requiring that it be a negligible function in $\log|R|$, where R denotes the set of non-symbolic randomness, is that it may very well be the case that $|R|$ is small since this is not the randomness used to sample new keys (when it would be reasonable to work with $\log|R|$ as a security parameter). For instance, one could just flip a coin (i.e., $R = \{0, 1\}$).

In the game we require that all symbolic random coins used by users are generated disjointly. More precisely, if $\mathtt{r}_u^t$ denotes the set of random coins used by user u in round t in the init/update procedures, then we require that $\mathtt{r} \in \mathtt{r}_u^t$ implies $\mathtt{r} \notin \mathtt{r}_{u'}^{t'}$ for all $(u, t) \neq (u', t')$.

We now define a property of CGKA schemes that we will require for our bounds. It essentially forbids schemes to generate layered ciphertexts of the form $\mathsf{PKE.Enc}(\mathtt{pk}_2, \mathsf{PKE.Enc}(\mathtt{pk}_2, \mathtt{m}))$. For some intuition on how it factors into our translation to the combinatorial model see Remark 3 after the proof of this section's main result.

Definition 5 (No nested encryption)**.** *We say a scheme* CGKA *does not use* nested encryption *if, for all ciphertexts* $\mathtt{c} \leftarrow \mathsf{PKE.Enc}(\mathtt{pk}, \mathtt{m})$*, the encrypted message is either a secret key or a random coin, i.e., of type* $\mathtt{sk}, \mathtt{K}$*, or* $\mathtt{r}$*.*

Our goal for the remainder of this section is to show that the task of deriving lower bounds on the communication complexity of correct and secure CGKA schemes translates to the analogue in the combinatorial model. To this end, we define *useful secrets*, i.e., secret symbolic variables that the adversary is not able to derive, and associate them to the set of users with knowledge of them. We prove that these sets satisfy the properties of the combinatorial model described in Section 3.1 for a cost function that counts the number of messages sent in a given round by a user.

Useful secrets and associated sets. First, we establish some notation. Consider an adversary playing OW-k-PCS$_{\neg\mathrm{RC}}$. We denote the set of public messages sent up to and including round t by $\mathtt{M}_t$, i.e., for $t = 0$ we set $\mathtt{M}_0 = \{\mathtt{pub}, \mathtt{MI}_{u_0}^0\}$ to be the output of oracle Init; and for every round $t \geq 1$ we extend the set by the output of oracle ROUND: $\mathtt{M}_t \leftarrow \mathtt{M}_{t-1} \cup \mathtt{MU}$, where $\mathtt{MU} = \mathsf{ROUND}(U_t)$. Further, for $t \geq 0$ we track all variables the adversary learned up to and including round t, via the corruption oracle, in a set $\mathtt{COR}_t$. I.e., at the beginning of round t, the set $\mathtt{COR}_t$ is initialized to $\mathtt{COR}_{t-1}$ and, if user u is corrupted in round t, then their current state $\mathtt{st}_u$ (meaning the one after all oracle calls of the round) is added to the set. Note that $\mathtt{COR}_t$ matches the set $\mathtt{COR}_t$, defined in game OW-k-PCS$_{\neg\mathrm{RC}}$, that tracks the values known to the adversary via corruption. This allows us to define the notion of useful secrets $\mathtt{s}$, i.e., variables of type $\mathtt{r}$, $\mathtt{K}$, and $\mathtt{sk}$ that cannot be derived by the adversary, and associate to them the set of users that in round t have access to $\mathtt{s}$.

Definition 6 (Useful secrets and associated sets)**.** *Consider adversary* A *playing game* OW-k-PCS$_{\neg\text{RC}}$ *in the symbolic model and let* $t \in \mathbb{N}$*, and* $\mathtt{s}$ *be a variable of type* $\mathtt{r}$*,* $\mathtt{K}$*, or* $\mathtt{sk}$ *generated during the game, before or in round* t*. We say that* $\mathtt{s}$ *is* useful *in round* t *if* $\mathtt{s} \notin \mathsf{Der}(\{\mathtt{M}_t, \mathtt{COR}_{t-1}\})$*. Let* $\mathtt{s}$ *be useful in round* t*. We define the associated set of* $\mathtt{s}$ *in round* t *as*

$$S(\mathtt{s}, t) := \{u \in [n] \mid \mathtt{s} \in \mathsf{Der}(\mathtt{st}_u^t, \mathtt{M}_t)\} \subseteq [n]$$

and the set system *in round* t *as*

$$\mathcal{S}_t := \{S(\mathtt{s}, t) \mid \mathtt{s} \textit{ is useful in round } t\} \subseteq 2^{[n]} \ .$$

Further, we define the set of sets remaining useful after the next round of corruptions as $\mathrm{Sec}(\mathcal{S}_t) := \{S(\mathtt{s}, t) \in \mathcal{S}_t \mid \mathtt{s} \notin \mathsf{Der}(\{\mathtt{M}_t, \mathtt{COR}_t\})\}$*.*

We now define what it means for a scheme to not allow users to distribute work. Intuitively, this requirement says that whenever a secret (be it already existing or newly generated) is communicated to a set of users, who did not yet have access to it, then this communication must have been done by a *single* user. For example, this notion excludes schemes in which two users u_1, u_2, already sharing a common key, communicate this key to users u_3 and u_4 by having u_1 encrypt it to u_3, and u_2 encrypt it to u_4. See Figure 7 for an illustration of a scheme that *does* make use of distributed work at hand of a ratchet tree.

Definition 7 (No distributed work)**.** *Consider a scheme* CGKA *and an execution of game* OW-k-PCS$_{\neg\text{RC}}$ *with respect to* CGKA *in the symbolic model. For user* u *and round* t *let* $\mathtt{st}_u^t$ *denote the user's state in round* t *and* $\mathtt{r}_u^t$ *the random coins generated in round* t*. We say that* CGKA *does* not allow users to distribute work *if, for all* t *and every secret symbolic variable* $\mathtt{s}$*, we have that there exists a user* u' *such that for every* $u \in S(\mathtt{s}, t) \setminus (S(\mathtt{s}, t-1) \cup \{u'\})$ *it holds that* $\mathtt{s} \in \mathsf{Der}(\mathtt{st}_u^{t-1}, \mathtt{M}_{t-1}, \mathtt{MU}_{u'}^t)$ *and* $\mathtt{s} \in \mathsf{Der}(\mathtt{st}_{u'}^{t-1}, \mathtt{r}_{u'}^t, \mathtt{M}_{t-1})$*.*

Connection to combinatorial model. In the following we show that the three properties required in the combinatorial model are satisfied by the symbolic model's associated set system. The first two are quite natural observations, the last essentially corresponds to a generalization of a statement that is shown in the proof of [BDR20, Thm. 2] and can be seen as quantifying the cost of adding new sets to the set system $\mathcal{S}_t$ by updating. We measure the cost in terms of the number of symbolic variables sent by a user u in round t and denote this quantity $|\mathtt{MU}_u^t|$. When interested in the cost of a round t, we take the sum over all users $u \in U_t$.

Intuitively, Property (i) is enforced by correctness and security, as on one hand every member of G_t must be able to derive the current group key from their state, and the safety predicate being satisfied implies that the group key at time t^* must be useful, i.e., $G_{t^*} \in \mathcal{S}_{t^*}$. Property (ii) corresponds to the simple fact that no secret derivable from $\mathtt{st}_u$ can be useful in a round in which the user gets corrupted, as in this case it can be derived by the adversary as well. Equivalently, a set $S \in \mathrm{Sec}(\mathcal{S}_t)$ cannot contain any users in C_t. Finally, Property (iii) corresponds to the intuition, that the secret $\mathtt{s}$ belonging to a new set $S = S(\mathtt{s}, t)$ needs to be communicated to (at least) every member u of S. If $\mathtt{s}$ cannot be derived using PRF evaluations from a secret already known to u, then either it, or a secret which can be derived from using PRF, must be communicated to u by encrypting it to a useful key that was known to the party in the previous round, i.e., in round $t-1$. In other words, this determines a covering of the set S with sets in $\mathrm{Sec}(\mathcal{S}_{t-1})$ and possibly a singleton $\{u\}$ for some updating user $u \in U_t$ with the property that the number of symbolic variables contained in the messages exchanged in round t is at least the number of sets in the said cover minus one. When we consider schemes in which users do not distribute work, we obtain a simpler statement for property (iii) and it matches Equation 4 from the combinatorial model.

Lemma 5. *Let* CGKA *be a perfectly correct continuous group-key agreement scheme that is* OW-k-PCS$_{\neg\text{RC}}$*-secure and does not use nested encryption. Consider an adversary playing game* OW-k-PCS$_{\neg\text{RC}}$ *of Figure 8 in the symbolic model.*

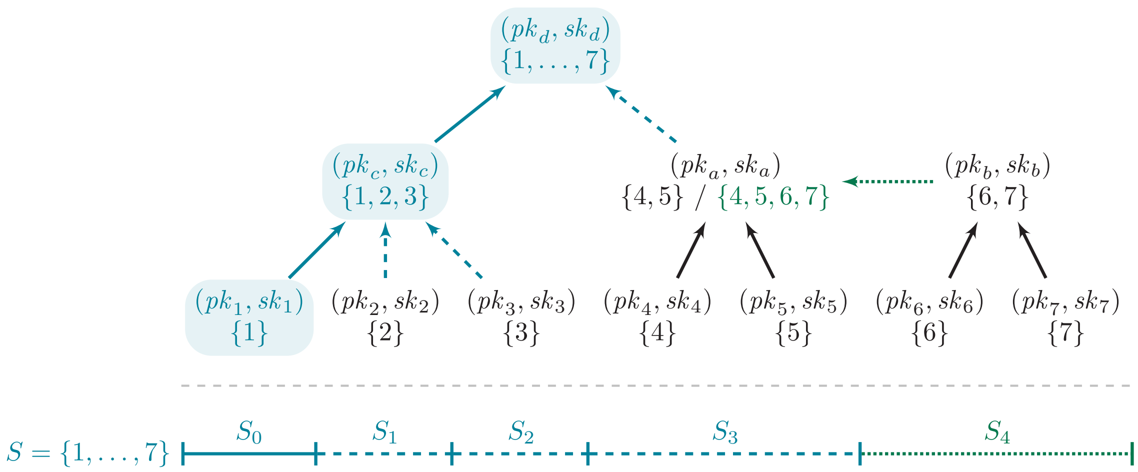

Figure 7: Top: Example of a ratchet tree and its associated set system $\mathcal{S}_t$ making use of distributed work. Vertices contain key-pairs (above) and the associated set (below). Edges indicate that knowledge of the secret key of the source implies knowledge of the one of the sink. Dashed edges correspond to ciphertexts $\mathsf{Enc}_{pk_{\text{source}}}(sk_{\text{sink}})$ sent by user 1 in round t, solid edges either to keys derived using a PRF in round t or to keys communicated in a previous round. Keys already present in the system at time $t-1$ are depicted in black and keys added by user 1 in round t in blue with shaded background. The dotted edge corresponds to a ciphertext sent by user 5 in round t. Note that the associated set of (pk_a, sk_a) changes in round t as an effect of this ciphertext, and that users 6 and 7 need to decrypt ciphertexts sent by two different users, namely users 1 and 5, in order to recover sk_d, implying that the scheme does indeed use distributed work. Bottom: Depiction of the sets proven to exist in Lemma 5 using the example $S = \{1,\ldots,7\} = \bigcup_{i=0}^{k} S_i$ in the set system depicted above. Note that $k = 4$ matches the number of ciphertexts sent in round t to establish S, which, however, stem from more than a single user (compare Lemma 5; (iii)).

(i) If $\mathtt{K}_{G_t}$ is the group key in round t, then $S(\mathtt{K}_{G_t}, t) = G_t$. In particular, if oracle CHALL *is queried in round t^* and the safety predicate is satisfied then we have $G_{t^*} \in \mathcal{S}_{t^*}$.*

(ii) If user u was corrupted by the adversary in round t, then for every $S \in \mathrm{Sec}(\mathcal{S}_t)$ it holds that $u \notin S$.

(iii) Let $t \geq 1$. Recall, that $U_t \subseteq [n]$ indicate the users that updated and for user u the sets $\mathtt{MU}_u^t$ correspond to the control messages generated by performing the corresponding update operation. Then for every set $S \in \mathcal{S}_t$ there exist $k \in \mathbb{N}_{\geq 0}$ and sets $\{S_i\}_{i=0}^k$ such that either

$$(a)\ S_0 = \{u\} \text{ for some } u \in U_t, \quad \text{or } (b)\ S_0 \in \mathrm{Sec}(\mathcal{S}_{t-1})$$

and, if $k \geq 1$, $S_i \in \mathrm{Sec}(\mathcal{S}_{t-1})$ for every $i = 1, \ldots, k$. Furthermore, it holds that

$$S \subseteq \bigcup_{i=0}^{k} S_i \quad \text{and} \quad \sum_{u \in U_t} |\mathtt{MU}_u^t| \geq k \ .$$

If CGKA *does not allow users to distribute work, then the last statement can be replaced by the following stronger expression:*

$$\exists u \in S_0 \cap U_t \text{ such that } |\mathtt{MU}_u^t| \geq k \ .$$

Before turning to the Lemma's proof we observe the following.

Remark 2. *Looking ahead, we observe the following. Consider an execution of game* OW-k-PCS$_{\neg\mathrm{RC}}$ *in the symbolic model, where* CGKA *is a perfectly correct,* OW-k-PCS$_{\neg\mathrm{RC}}$*-secure CGKA scheme which does not use*

```
Game $\text{OW-}k\text{-PCS}^{\mathsf{CGKA}}_{\neg\text{RC}}(n, u_0, C_0, (U_t, C_t)_{t=1}^{t_{\max}}, t^*)$
00 $\mathsf{INIT}(n, u_0)$
01 for $u \in C_0$: call $\mathsf{CORR}(u)$
02 for $t = 1, \ldots, t_{\max}$:
03     $\mathsf{ROUND}(U_t)$
04     $\mathtt{COR}_t \leftarrow \mathtt{COR}_{t-1}$
05     for $u \in C_t$: call $\mathsf{CORR}(u)$
06     if $t = t^*$: call $\mathsf{CHALL}$
07 return $[\mathtt{K}^* \in \mathsf{Der}(\mathtt{COR}_{t_{\max}}, \mathtt{M}) \wedge \text{safe}_{k\text{-PCS}}]$

Oracle $\mathsf{INIT}(n, u_0)$                         \\ one call; called first
08 $t \leftarrow 0$
09 $\mathtt{M} \leftarrow \emptyset$, $\mathtt{COR}_0 \leftarrow \emptyset$
10 $\text{Cor}[u] \leftarrow \emptyset$, $\text{Upd}[u] \leftarrow \emptyset$ for all $u \in [n]$
11 $\text{Coins}[u, t] \leftarrow \emptyset$ for all $u \in [n]$, $\forall t$
12 $\mathtt{r_{setup}} \leftarrow_{\$} \mathit{Rnd}$
13 $(\mathtt{pub}, (\mathtt{st}_u)_{u \in [n]}) \leftarrow \mathsf{Setup}(n; \mathtt{r_{setup}})$
14 $\mathtt{r} \leftarrow_{\$} \mathit{Rnd}$; $\text{Coins}[u_0, t] \leftarrow_{\cup} \{\mathtt{r}\}$
15 $(\mathtt{st}_{u_0}, \mathtt{MI}^0_{u_0}) \leftarrow \mathsf{Init}(\mathtt{st}_{u_0}, \mathtt{pub}, n; \mathtt{r})$
16 for $u \in [n]$:
17     $\mathtt{st}_u \leftarrow \mathsf{Process}(\mathtt{st}_u, \mathtt{pub}, \mathtt{MI}^0_{u_0})$
18     $\mathtt{K}_G \leftarrow \mathsf{GetKey}(\mathtt{st}_u)$
19 $\mathtt{M} \leftarrow_{\cup} \{\mathtt{pub}, \mathtt{MI}^0_{u_0}\}$

Oracle $\mathsf{CHALL}$                                \\ one call
20 $\mathtt{K}^* \leftarrow \mathtt{K}_G$

Oracle $\mathsf{CORR}(u)$
21 $\text{Cor}[u] \leftarrow_{\cup} \{t\}$
22 $\mathtt{COR}_t \leftarrow_{\cup} \{\mathtt{st}_u\}$

Oracle $\mathsf{ROUND}(U_t)$
23 $\mathtt{MU} \leftarrow \emptyset$
24 for $u \in U_t$:
25     $\text{Upd}[u] \leftarrow_{\cup} \{t\}$
26     $\mathtt{r} \leftarrow_{\$} \mathit{Rnd}$; $\text{Coins}[u, t] \leftarrow_{\cup} \{\mathtt{r}\}$
27     $(\mathtt{st}_u, \mathtt{MU}^t_u) \leftarrow \mathsf{Update}(\mathtt{st}_u, \mathtt{pub}; \mathtt{r})$
28     $\mathtt{MU} \leftarrow_{\cup} \{\mathtt{MU}^t_u\}$
29 for $u \in [n]$:
30     $\mathtt{st}_u \leftarrow \mathsf{Process}(\mathtt{st}_u, \mathtt{pub}, \mathtt{MU})$
31     $\mathtt{K}_G \leftarrow \mathsf{GetKey}(\mathtt{st}_u)$
32 $\mathtt{M} \leftarrow_{\cup} \mathtt{MU}$

Predicate $\text{safe}_{k\text{-PCS}}$
33 for $u \in [n]$:
34     if $\text{Cor}[u] \neq \emptyset$
35         if $\exists t \in \text{Cor}[u] : t \geq t^*$:        \\ corruption after challenge
36             return 0
37         $t^c_u \leftarrow \max\{t \in \text{Cor}[u]\}$
38         if $|\{t \in \text{Upd}[u] : t^c_u < t \leq t^*\}| < k$:    \\ too few updates
39             return 0
40 return 1
```

Figure 8: Security game OW-k-PCS$_{\neg\text{RC}}$ in the symbolic model with respect to adversary A that is completely characterized by its sequence $(n, u_0, C_0, (U_t, C_t)_{t=1}^{t_{\max}}, t^*)$ of inputs to the oracles.

nested encryption and does not allow users to distribute work, and set $\text{Cost}(u, t) = |\mathtt{MU}^t_u|$ *to be the number of symbolic variables sent by u in round t. Then, by Lemma 5 the associated set system* $\text{Sec}(\mathcal{S}_t)$ *(Definition 6) and* Cost *satisfy all properties of the combinatorial model described in Section 3.1. As a consequence, to prove lower bounds on the communication cost of* CGKA*, i.e., the number of ciphertexts sent during the execution of the game, it is sufficient to lower bound the cost function for a scheme satisfying the combinatorial model.*

Proof of Lemma 5. In Figure 8 we recall the security game for OW-k-PCS$_{\neg\text{RC}}$, where $G = [n]$, with respect to the symbolic model and such adversaries.

(i) By correctness (line 19 of the correctness game), every member u of G must be able to derive the current group key, $\mathtt{K}_G$, from their state, $\mathtt{st}^t_u$. By definition, this implies that $G \subseteq S(\mathtt{K}_G, t)$ and it always holds that $S(\mathtt{K}_G, t) \subseteq G$. The safety predicate being satisfied implies that the group key at time t^* must be useful, i.e., $G_{t^*} \in \mathcal{S}_{t^*}$.

(ii) If $u \in S(\mathtt{s}, t)$ and u was corrupted at time t, then $\mathtt{s} \in \mathsf{Der}(\mathtt{st}^t_u, \mathtt{M}_t) \subseteq \mathsf{Der}(\{\mathtt{M}_t, \mathtt{COR}_t\})$. By definition, $S(\mathtt{s}, t) \notin \text{Sec}(\mathcal{S}_t)$.

(iii) Let $G_t = (V_t, E_t)$ be the graph with nodes $V_t = \{\mathtt{s} \mid \mathtt{s} \text{ is useful at time } t\}$ and edges $E_t = E^{\mathsf{PRF}}_t \cup E^{\mathsf{st}}_t \cup E^{\mathsf{Enc}}_t$ where

$$E^{\mathsf{PRF}}_t = \{(\mathtt{s}_1, \mathtt{s}_2) \in V_t^2 \mid \exists \mathtt{ad}\colon s_2 = \mathsf{PRF}(\mathtt{s}_1, \mathtt{ad})\}$$
$$E^{\mathsf{st}}_t = \{(\mathtt{s}_1, \mathtt{s}_2) \in V_t^2 \mid \exists u \in [n] \exists c_1, \ldots, c_l \in \mathtt{st}^{t-1}_u \colon c_i = \mathsf{Enc}(\mathtt{pk}_i, \mathtt{s}_{i+1}), \mathtt{pk}_i = \mathsf{Gen}(\mathtt{s}_i), \mathtt{sk}_1 = \mathtt{s}_1 \text{ and } \mathtt{sk}_{l+1} = \mathtt{s}_2\}$$
$$E^{\mathsf{Enc}}_t = \{(\mathtt{s}_1, \mathtt{s}_2) \in V_t^2 \mid \exists c \in \mathtt{M}_t \colon c = \mathsf{Enc}(\mathtt{pk}_1, \mathtt{s}_2) \text{ and } \mathtt{pk}_1 = \mathsf{Gen}(\mathtt{s}_1)\} \ .$$

The assumption that the scheme does not use nested encryption guarantees that all ciphertexts are of the form of those used in the definition of E_t^{st} and E_t^{Enc}.

Let $\mathtt{s}$ be a useful secret at time t and $u \in S(\mathtt{s}, t)$ and consider the sets

$$\begin{aligned} V_{t,\mathtt{s},u} =& \{\mathtt{s}' \in V_t \mid \mathtt{s}' \in \mathsf{Der}(\mathtt{st}_u^{t-1}, \mathtt{r}_u^t, \mathtt{M}_t)\} \\ E_{t,\mathtt{s},u} =& \{(\mathtt{s}_1, \mathtt{s}_2) \in E_t \mid \mathtt{s}_1, \mathtt{s}_2 \in V_{t,\mathtt{s},u}\} \ . \end{aligned}$$

Intuitively these sets represent the secrets u knows at the beginning of the round and how it derives the secrets it knows at the end of the round. Let $P_{t,\mathtt{s},u} = (\mathtt{s}^1_{t,\mathtt{s},u}, \ldots, \mathtt{s}^{\ell_u}_{t,\mathtt{s},u})$ be a path with nodes in $V_{t,\mathtt{s},u}$ and edges $(s^i_{t,\mathtt{s},u}, s^{i+1}_{t,\mathtt{s},u}) \in E_{t,\mathtt{s},u}$ for every $i \in [\ell_u - 1]$, $\mathtt{s}^{\ell_u}_{t,\mathtt{s},u} = \mathtt{s}$ and either $\mathtt{s}^1_{t,\mathtt{s},u} = \mathtt{r}_u^t$ or $u \in S(\mathtt{s}^1_{t,\mathtt{s},u}, t-1)$.

We observe that there always exists such a path $P_{t,\mathtt{s},u}$ since $\mathtt{s} \in \mathsf{Der}(\mathtt{st}_u^{t-1}, \mathtt{r}_u^t, \mathtt{M}_t)$. This is actually the case because $u \in S(\mathtt{s}, t)$, i.e., $\mathtt{s} \in \mathsf{Der}(\mathtt{st}_u^t, \mathtt{M}_t)$, and $\mathsf{Der}(\mathtt{st}_u^t, \mathtt{M}_t) \subseteq \mathsf{Der}(\mathtt{st}_u^{t-1}, \mathtt{r}_u^t, \mathtt{M}_t)$ as we require that symbolic outputs of algorithms can always be derived symbolically from their symbolic inputs. And it is a path since PRFs and PKE take only one secret input.

We now consider the graph $G_{t,\mathtt{s}} = (V_{t,\mathtt{s}}, E_{t,\mathtt{s}})$ where

$$\begin{aligned} V_{t,\mathtt{s}} &= \bigcup_{u \in S(\mathtt{s},t)} \mathsf{nodes}(P_{t,\mathtt{s},u}) = \bigcup_{u \in S(\mathtt{s},t)} \{\mathtt{s}^1_{t,\mathtt{s},u}, \ldots, \mathtt{s}^{\ell_u}_{t,\mathtt{s},u}\} \ , \\ E_{t,\mathtt{s}} &= \bigcup_{u \in S(\mathtt{s},t)} \mathsf{edges}(P_{t,\mathtt{s},u}) = \bigcup_{u \in S(\mathtt{s},t)} \{(\mathtt{s}^i_{t,\mathtt{s},u}, \mathtt{s}^{i+1}_{t,\mathtt{s},u}) \colon i \in [\ell_u - 1]\} \ . \end{aligned}$$

We associate a set $S_{(\mathtt{s}_1,\mathtt{s}_2)}$ to each edge $(\mathtt{s}_1, \mathtt{s}_2)$ in $(E_t^{\mathsf{Enc}} \setminus E_{t-1}^{\mathsf{Enc}}) \cap E_{t,\mathtt{s}}$:

$$S_{(\mathtt{s}_1,\mathtt{s}_2)} = \begin{cases} \{u\}, & \text{if } \mathtt{s}_1 \in \mathsf{Der}(\mathtt{r}_u^t) \\ S(\mathtt{s}_1, t-1), & \text{otherwise.} \end{cases}$$

This set is well-defined, i.e., for an edge $(\mathtt{s}_1, \mathtt{s}_2)$ there is at most one user u such that $\mathtt{s}_1 \in \mathtt{r}_u^t$, because randomness is generated independently. Since $\mathtt{s}_1$ is useful at time t, $\{u\} \in \mathcal{S}_t$ if $\mathtt{s}_1 \in \mathsf{Der}(\mathtt{r}_u^t)$, or $S(\mathtt{s}_1, t-1) \in \mathrm{Sec}(\mathcal{S}_{t-1})$ otherwise.

We define the set S_0 as follows:

$$S_0 = \begin{cases} \{u\}, & \text{if } \mathtt{s} \in \mathsf{Der}(\mathtt{r}_u^t) \\ S(\mathtt{s}, t-1), & \text{otherwise.} \end{cases}$$

which is well-defined, i.e., there is at most one user u such that $\mathtt{r}_u^t \in \mathtt{s}_1$, because randomness is generated independently. Since $\mathtt{s}$ is useful at time t, $\{u\} \in \mathcal{S}_t$ if $\mathtt{s} \in \mathsf{Der}(\mathtt{r}_u^t)$, or $S(\mathtt{s}, t-1) \in \mathrm{Sec}(\mathcal{S}_{t-1})$ otherwise.

By construction we obtain the following lower bound:

$$|(E_t^{\mathsf{Enc}} \setminus E_{t-1}^{\mathsf{Enc}}) \cap E_{t,\mathtt{s}}| \geq |\mathcal{F}| - 1 \ .$$

where $\mathcal{F} = \{S_0\} \cup \{S_{(\mathtt{s}_1,\mathtt{s}_2)} \mid (\mathtt{s}_1, \mathtt{s}_2) \in (E_t^{\mathsf{Enc}} \setminus E_{t-1}^{\mathsf{Enc}}) \cap E_{t,\mathtt{s}}\}$.

Moreover, for every $u \in S(\mathtt{s}, t)$ the construction of $P_{t,\mathtt{s},u}$ guarantees that either $\mathtt{s}^1_{t,\mathtt{s},u} = \mathtt{r}_u^t$ or $u \in S(\mathtt{s}^1_{t,\mathtt{s},u}, t-1)$. This implies that there exists a set $S \in \mathcal{F}$ such that $u \in S$, namely, the set associated to the first edge in $P_{t,\mathtt{s},u}$ that belongs to $(E_t^{\mathsf{Enc}} \setminus E_{t-1}^{\mathsf{Enc}}) \cap E_{t,\mathtt{s}}$, or S_0 if $P_{t,\mathtt{s},u}$ contains no edges that belong to $(E_t^{\mathsf{Enc}} \setminus E_{t-1}^{\mathsf{Enc}}) \cap E_{t,\mathtt{s}}$.

This shows that $S(\mathtt{s}, t)$ can be covered with sets in $\mathcal{F}$. It remains to prove that we can find a cover $\mathcal{T} \subseteq \mathcal{F}$ such that for every $S \in \mathcal{T}$ of the form $S = S_{(\mathtt{s}_1,\mathtt{s}_2)}$, it holds that $S = S(\mathtt{s}_1, t-1)$.

Let $u \in S(\mathtt{s}, t)$. We consider three cases:

– If $P_{t,\mathtt{s},u}$ contains no edges that belong to $(E_t^{\mathsf{Enc}} \setminus E_{t-1}^{\mathsf{Enc}}) \cap E_{t,\mathtt{s}}$, then $u \in S_0$.

– If there exist $u' \neq u$ and $i \in [\ell_u - 1]$ such that $(\mathtt{s}^i_{t,\mathtt{s},u}, \mathtt{s}^{i+1}_{t,\mathtt{s},u}) \in (E^{\mathsf{Enc}}_t \setminus E^{\mathsf{Enc}}_{t-1}) \cap E_{t,\mathtt{s}}$ and $\mathtt{s}^i_{t,\mathtt{s},u} \in \mathsf{Der}(\mathtt{r}^t_{u'})$, then there exists $j < i$ such that $(\mathtt{s}^j_{t,\mathtt{s},u}, \mathtt{s}^{j+1}_{t,\mathtt{s},u}) \in (E^{\mathsf{Enc}}_t \setminus E^{\mathsf{Enc}}_{t-1}) \cap E_{t,\mathtt{s}}$ and $u \in S(\mathtt{s}^j_{t,\mathtt{s},u}, t-1) = S_{(\mathtt{s}^j_{t,\mathtt{s},u}, \mathtt{s}^{j+1}_{t,\mathtt{s},u})}$.

– Else assume that for every edge $(\mathtt{s}_1, \mathtt{s}_2)$ in $P_{t,\mathtt{s},u}$ that belongs to $(E^{\mathsf{Enc}}_t \setminus E^{\mathsf{Enc}}_{t-1}) \cap E_{t,\mathtt{s}}$, $\mathtt{s}_1 \in \mathsf{Der}(\mathtt{r}^t_u)$. Let $i \in [\ell_u - 1]$ such that $(\mathtt{s}^i_{t,\mathtt{s},u}, \mathtt{s}^{i+1}_{t,\mathtt{s},u}) \in (E^{\mathsf{Enc}}_t \setminus E^{\mathsf{Enc}}_{t-1}) \cap E_{t,\mathtt{s}}$ and for every $j > i$ $(\mathtt{s}^i_{t,\mathtt{s},u}, \mathtt{s}^{i+1}_{t,\mathtt{s},u}) \notin (E^{\mathsf{Enc}}_t \setminus E^{\mathsf{Enc}}_{t-1}) \cap E_{t,\mathtt{s}}$. By assumption, $\mathtt{s}^i_{t,\mathtt{s},u} \in \mathsf{Der}(\mathtt{r}^t_u)$. Therefore the public key that corresponds to $\mathtt{s}^i_{t,\mathtt{s},u}$ is only available after round t, the only user that can send the ciphertext that corresponds to the edge $(\mathtt{s}^i_{t,\mathtt{s},u}, \mathtt{s}^{i+1}_{t,\mathtt{s},u})$ is u. This implies that u could also derive the secret being encrypted, namely, $\mathtt{s}^{i+1}_{t,\mathtt{s},u}$. Therefore we have two cases depending on whether $\mathtt{s}^{i+1}_{t,\mathtt{s},u} \in \mathsf{Der}(\mathtt{r}^t_u)$. If $\mathtt{s}^{i+1}_{t,\mathtt{s},u} \in \mathsf{Der}(\mathtt{r}^t_u)$, then the edges in $\{(\mathtt{s}^j_{t,\mathtt{s},u}, \mathtt{s}^{j+1}_{t,\mathtt{s},u})\}_{j \geq i+1}$ belong to E^{PRF}_t as by assumption they are not in E^{Enc}_t and they cannot be in $E^{\mathtt{st}}_t$ or else there would be a ciphertext in $\mathtt{st}^{t-1}_u$ under a public key generated in round t, which yields a contradiction. Thus if $\mathtt{s}^{i+1}_{t,\mathtt{s},u} \in \mathsf{Der}(\mathtt{r}^t_u)$, $\mathtt{s} = \mathtt{s}^{\ell_u}_{t,\mathtt{s},u} \in \mathsf{Der}(\mathtt{r}^t_u)$ and $u \in S_0$ as desired. If $\mathtt{s}^{i+1}_{t,\mathtt{s},u} \notin \mathsf{Der}(\mathtt{r}^t_u)$, $u \in S(\mathtt{s}^{i+1}_{t,\mathtt{s},u}, t-1)$, and by assumption all edges in $\{(\mathtt{s}^j_{t,\mathtt{s},u}, \mathtt{s}^{j+1}_{t,\mathtt{s},u})\}_{j \geq i+1}$ belong to E^{PRF}_t or $E^{\mathtt{st}}_t$, so $u \in S(\mathtt{s}^{\ell_u}_{t,\mathtt{s},u}, t-1) = S(\mathtt{s}, t-1) = S_0$ as desired.

This concludes the proof of the first part of property (iii).

For the last part we assume that there is no distributed work, that is, there exists a user u' such that for every $u \in S(\mathtt{s}, t) \setminus (S(\mathtt{s}, t-1) \cup \{u'\})$ it holds that $\mathtt{s} \in \mathsf{Der}(\mathtt{st}^{t-1}_u, \mathtt{M}_{t-1}, \mathtt{MU}^t_{u'})$ and $\mathtt{s} \in \mathsf{Der}(\mathtt{st}^{t-1}_{u'}, \mathtt{r}^t_{u'}, \mathtt{M}_{t-1})$. Therefore we can restrict the paths $P_{t,\mathtt{s},u}$ to only contain nodes $\mathtt{s}' \in \mathsf{Der}(\mathtt{st}^{t-1}_u, \mathtt{M}_{t-1}, \mathtt{MU}^t_{u'})$ or $\mathtt{s}' \in \mathsf{Der}(\mathtt{st}^{t-1}_{u'}, \mathtt{r}^t_{u'}, \mathtt{M}_{t-1})$. With this restriction the edges in $(E^{\mathsf{Enc}}_t \setminus E^{\mathsf{Enc}}_{t-1}) \cap E_{t,\mathtt{s}}$ can only correspond to ciphertexts in $\mathtt{MU}^t_{u'}$ (i.e., $(E^{\mathsf{Enc}}_t \setminus E^{\mathsf{Enc}}_{t-1}) \cap E_{t,\mathtt{s}} \subseteq \mathtt{MU}^t_{u'}$) and if a path contains $\mathtt{r}^t_{\tilde{u}}$ for some user $\tilde{u}$, it must be $\tilde{u} = u'$.

If $S_0 = \{u\}$ for some user such that $\mathtt{s} \in \mathsf{Der}(\mathtt{r}^t_u)$, it must be the case that $u = u'$ as argued in the previous paragraph. If $S_0 = S(\mathtt{s}, t-1)$, the fact that $\mathtt{s} \in \mathsf{Der}(\mathtt{st}^{t-1}_{u'}, \mathtt{r}^t_{u'}, \mathtt{M}_{t-1})$ implies that $P_{t,\mathtt{s},u'}$ contains no edges in $E^{\mathsf{Enc}}_t \setminus E^{\mathsf{Enc}}_{t-1}$ which, in turn, shows $u' \in S_0$. In both cases $u' \in S_0$ which completes the proof.

□

Remark 3. *Lemma 5 requires that* CGKA *not use nested encryption, i.e., not generate encryptions of ciphertexts. On a technical level, this restriction guarantees that for the graph constructed in the lemma's proof for every edge* $(\mathtt{s}_1, \mathtt{s}_2)$ *we have that knowledge of secret* $\mathtt{s}_1$ *implies knowledge of* $\mathtt{s}_2$*. On a more intuitive level, allowing ciphertexts of the form* $\mathtt{c} = \mathsf{Enc}(\mathtt{pk}_2, \mathsf{Enc}(\mathtt{pk}_1, \mathtt{m}))$ *would enable users to send ciphertext c in one round but release message* $\mathtt{m}$ *in a later round by at this point in time sending* $\mathtt{sk}_2$ *in the plain, at cost of no additional ciphertexts. While this does not seem to help with the total communication cost, it could in principle enable users to distribute their workload over several rounds. An analogous statement holds, if one allows the use of dual PRFs (as considered in the symbolic model of [BDR20]).*

4.2 Lower Bounds on the Update Cost in the Symbolic Model

We now show that the worst-case lower bound on the communication cost of CGKA schemes in the combinatorial model (Theorem 3) carries over to the symbolic model for OW-k-PCS$_{\neg\mathrm{RC}}$-secure schemes. By worst-case we mean that we rely on an adversarially chosen sequences of updates. Concretely, the adversary will exploit the fact that users in the set U_t of concurrently updating users are not aware of the other members of U_t and choose it in a way that essentially forces many users to produce large update messages.

The proof follows the idea already outlined in Remark 2. As a consequence of Lemma 5, if we impose certain restrictions on the kind of CGKA schemes we consider, we can reduce the problem of showing lower bounds on the communication cost in the symbolic model to the problem of giving lower bounds in the combinatorial model for the cost function $\mathrm{Cost}(u, t) = |\mathtt{MU}^t_u|$ that counts the number of symbolic variables sent by u in round t.

Our bound requires CGKA to not use nested encryption and assumes CGKA does not allow users to distribute work. The former is needed in order to be able to use Lemma 5, whereas the latter guarantees that in property (iii) of Lemma 5 we obtain the same expression as in Equation 4 of the combinatorial model. Additionally, if CGKA has publicly-computable update cost, the sequence of adversarially chosen updates can be computed as the security game is played and if CGKA has offline publicly-computable update cost, the sequence of updates can be computed after the initialization phase.

The result without any of these additional properties, that is, when CGKA does not have publicly-computable update cost, guarantees that for any choice of non-symbolic randomness there exists a sequence of updates for which the total communication cost is at least roughly $n^{1+1/k} \cdot k/\log(k)$. However, it may not be possible to determine such a sequence using only public information which is the reason why we introduce the notion of publicly-computable update cost. If CGKA has this property, then it is possible to find it online, while if CGKA has offline publicly-computable update cost, it can be pre-computed.

Corollary 6. *Let $k, n, t_c \in \mathbb{N}$ and let CGKA be a correct and OW-k-PCS$_{\neg\mathrm{RC}}$-secure CGKA scheme that does not use nested encryption, and does not allow users to distribute work. Let $0 < \varepsilon < 2/5$ be a constant such that $t_{\max} = t_c + (1+\varepsilon)k \in \mathbb{N}$ and set $\alpha_\varepsilon = \frac{\varepsilon - 5\varepsilon^2/2 + \varepsilon^3}{8(1+\varepsilon)} > 0$.*

If $3 \le k \le \ln(\alpha_\varepsilon n)$, then for an arbitrary setup phase of the group $G_0 = G_t = [n]$ and an arbitrary phase of t_c rounds of updates $(U_t)_{t=1}^{t_c}$ and any choice of non-symbolic randomness in the security game OW-k-PCS$_{\neg\mathrm{RC}}$, there exist a sequence of sets U_{t_c+t} of updating users for $(1+\varepsilon)k$ rounds such that the total communication cost satisfies

$$\sum_{t=1}^{(1+\varepsilon)k} \mathrm{Cost}(U_{t_c+t}) \ge \frac{(k-1)}{4} \cdot \left\lfloor \frac{2\varepsilon n}{5(1+\varepsilon)\lceil \log(k) \rceil} \right\rfloor \left((\alpha_\varepsilon n)^{\frac{1}{(1+\varepsilon)k-1}} - 1 \right) .$$

If CGKA has publicly-computable update cost (Definition 3), the sequence of sets $(U_{t_c+t})_{t=1}^{(1+\varepsilon)k}$ can be computed online, i.e., U_{t_c+t} can be computed using public information from the previous rounds. Furthermore, if CGKA has offline publicly-computable update cost, the sequence of updates $(U_{t_c+t})_{t=1}^{(1+\varepsilon)k}$ can be computed after round t_c and is independent of the non-symbolic randomness.

Proof. Let $\vec{r} = (r_{tn,u})_{t \in [(1+\varepsilon)k], u \in [n]}$ denote the non-symbolic randomness used in rounds $t = t_c+1, \dots, t_{\max}$, namely, $r_{tn,u}$ denotes the non-symbolic randomness used by algorithm Update in round t if run by user u, i.e., $u \in U_t$.

Consider for a fixed $\vec{r}$, the function $\mathrm{Cost}(u,t) = |\mathtt{MU}_u^t|$ counting the number of symbolic variables sent by u in round t and the set system $\mathrm{Sec}(\mathcal{S}_t)$ (Definition 6). Then, as observed in Remark 2, $\mathrm{Cost}(u,t)$ and $\mathrm{Sec}(\mathcal{S}_t)$ satisfy all the properties of the combinatorial model described in Section 3.1. It follows from Theorem 3 that there exist a set $C \subseteq [n]$ of size $\lceil \alpha_\varepsilon n \rceil$ and a sequence $(U_t)_{t=t_c+1}^{t_{\max}}$ such that the instantiation of the combinatorial model with respect to $(n, k, t_{\max}, \emptyset)$ and $(U_t, C_t)_{t=1}^{t_{\max}}$, where $C_{t_c} = C$ and $C_t = \emptyset$ if $t \neq t_c$, satisfies

$$\sum_{t=1}^{(1+\varepsilon)k} \mathrm{Cost}(U_{t_c+t}) \ge \frac{(k-1)}{4} \cdot \left\lfloor \frac{2\varepsilon n}{5(1+\varepsilon)\lceil \log(k) \rceil} \right\rfloor \left((\alpha_\varepsilon n)^{\frac{1}{(1+\varepsilon)k-1}} - 1 \right)$$

as desired.

By definition of symbolic security (Definition 4), security must hold for any sequence of corrupted parties and therefore the bound holds for the aforementioned sequence of updates independently of the choice of corrupted parties.

If CGKA has publicly-computable update cost, $\mathrm{Cost}(u,t)$ can be computed from public information at the end of round $t-1$. By construction of the sets $(U_t)_{t=1}^{t_{\max}}$ in the proof of Theorem 3, they are determined by the function $\mathrm{Cost}(u,t)$. This implies that for every t the set U_t can be computed from public information at the end of round $t-1$ and therefore the sequence $(U_t)_{t=1}^{t_{\max}}$ can be computed online. We observe that by the argument in the previous paragraph it is not necessary to be able to compute the sequence of corrupted users online. And if CGKA has offline publicly-computable update cost, the sequence of updates $(U_{t_c+t})_{t=1}^{(1+\varepsilon)k}$ can be computed after round t_c by definition. □

Analogously to the combinatorial setting it is also possible to give a statement with multiple rounds of corruptions instead of just considering the communication cost of recovering from a single round of corruptions. The proof follows the same arguments to the one above and is a consequence of Corollary 4.

Corollary 7. *Let k, n, t_c, ε, and α_ε be as in Corollary 6 and let* CGKA *be a correct and* OW-k-PCS$_{\neg\mathrm{RC}}$-*secure CGKA scheme that does not use nested encryption, and does not allow users to distribute work. Let $z_{\max} \in \mathbb{N}$ and for every integer $0 \leq z < z_{\max}$ set $t_{c,z} = t_c + z \cdot (t_c + (1+\varepsilon)k)$ and $t_{\max} = z_{\max} \cdot (t_c + (1+\varepsilon)k)$.*

If $3 \leq k \leq \ln(\alpha_\varepsilon n)$, then for an arbitrary setup phase of the group $G_0 = G_t = [n]$ and $z_{\max}$ arbitrary phases of t_c rounds of updates $(U_t)_{t=t_{c,z}-t_c+1}^{t_{c,z}}$ with $0 \leq z < z_{\max}$ and any choice of non-symbolic randomness in the security game OW-k-PCS$_{\neg\mathrm{RC}}$*, there exist sequences of updates $(U_t)_{t=t_{c,z}+1}^{t_{c,z}+(1+\varepsilon)k}$ and sets of corrupted users $C_z \subseteq [n]$ each of cardinality $\lceil \alpha_\varepsilon n \rceil$ such that the total communication cost satisfies*

$$\sum_{t=t_{c,z}+1}^{t_{c,z}+(1+\varepsilon)k} \mathrm{Cost}(U_{t_{c,z}+t}) \geq \frac{(k-1)}{4} \cdot \left\lfloor \frac{2\varepsilon n}{5(1+\varepsilon)\lceil \log(k) \rceil} \right\rfloor \left((\alpha_\varepsilon n)^{\frac{1}{(1+\varepsilon)k-1}} - 1 \right)$$

for every $0 \leq z < z_{\max}$.

If CGKA *has publicly-computable update cost (Definition 3), the sequences of sets $(U_t)_{t=t_{c,z}+1}^{t_{c,z}+(1+\varepsilon)k}$ can be computed online, i.e., $U_{t_{c,z}+t}$ can be computed using public information from the previous rounds. Furthermore, if* CGKA *has offline publicly-computable update cost, the sequence of updates $(U_t)_{t=t_{c,z}+1}^{t_{c,z}+(1+\varepsilon)k}$ can be computed after round $t_{c,z}$ and is independent of the non-symbolic randomness.*

5 Upper Bound on the Update Cost

In this section we describe a simple CGKA protocol, inspired by CoCoA [AAN+22a], but both more general and, for certain values of k, with a lower total upload communication cost. Accordingly, we termed it CoCoALight. In particular, it can recover from an arbitrary number of corruptions in k rounds and with a total communication cost in the order of $nk \sqrt[k/2]{n}$ ciphertexts, without any user coordination. While CoCoA's communication complexity is lower for low values of k, CoCoALight's improves for values of k closer to $\log(n)$. We discuss this in more detail in Section 5.5.This improvement comes at the drawback of non-immediate forward secrecy, which requires at least $k/2$ updates from each user prior to their corruption. Likewise, we not prove it secure against any type of active adversary and, indeed, only describe a simple protocol satisfying IND-k-PCS$_{\mathrm{RC}}$ security. Nevertheless, it shows that the lower bound on PCS from the previous section is only $\log(k)/\sqrt[k/2]{n}$ from being tight, for $k \in [4, 2\lceil \log(n) \rceil + 1]$. Concretely, for the case $k = \log(n)$, the gap is of order just $\log(\log(n))$.

We will start the section by recalling the general ideas behind the CoCoAprotocol in Section 5.1, to then describe a simple generalization of it in Section 5.2. This generalization allows for PCS in any $k \in [2, \lceil \log(n) \rceil + 1]$ rounds, as opposed to the original protocol, which only allowed $k = \lceil \log(n) \rceil + 1$. This protocol will, in fact, exactly match CoCoA when $k = \lceil \log(n) \rceil + 1$. Accordingly, its communication cost is proportional to that of CoCoA; in particular, in the order of $nk^2 \sqrt[k]{n}$. We will only provide an informal definition and security intuition of it, as we will just use this protocol as a stepping stone to better understand the more efficient CoCoALight, which is formally described in Section 5.3. Section 5.4 contains the security proof for CoCoALight, and Section 5.5 discusses its efficiency, as well as its disadvantages in terms of forward secrecy.

5.1 Preliminaries and CoCoA

Ratchet trees. All protocols discussed in this section (as well as the original protocol they all trace back to, TreeKEM [BBR18, BBR+23], and variations of it [ACDT20, Mat19, AAN+22a, AAN+22b]) are defined with the help of a data structure called a ratchet tree. The ratchet tree of in-degree ℓ of a group of users G is a left-balanced ℓ-ary tree where each leaf node is associated to a user $u \in G$. Further, each node in the tree has an associated PKE key-pair. The general principle, often called the *tree invariant* is that users will know the public values associated to all nodes in the tree, but the secret values only of the nodes on their path to the root. This allows for an efficient rotation of the secret key material in a user's state: they sample new keys for nodes on their path, and encrypt the new secrets to the nodes on the co-path. In particular, this allows for key updates to have cost that is linear in both the in-degree and depth of the tree (i.e., in $\ell \cdot \log_\ell n$), instead of linear in the number n of users in the group. In order to reflect this hierarchical relation of node secrets, we consider the edges in the ratchet tree as pointing from the leaves to the root, so that knowledge of a node's secret implies knowledge of the secret associated to its child.

Concurrent healing in TreeKEM and CoCoA. Initial versions of TreeKEM required the mentioned key updates to be performed sequentially by users, each of them having processed all previously sent ones before sending their own. In particular, healing required an amount of communication rounds linear in the number of updating users. Recent versions of TreeKEM switched to the *Propose-and-Commit* (P&C) framework, which separated proposals (updates, adds, removes) from commits, which could execute several proposals at once. The upside of this approach is that PCS can be achieved in just two communication rounds, the second one "committing" all update proposals of the first one. The downside is that applying those concurrent proposals involves *blanking*, i.e. deleting the key-pair associated to, the nodes on the updating-user's path. This implies that both this "commit", as well as future group operations, will be more expensive (linear cost in the worst case), as the graph devolves away from its optimal binary-tree structure.

The main observation behind CoCoA is that the updates from the original TreeKEM versions can be performed concurrently by users, in a way that does not increase the cost of subsequent operations but accelerates PCS. This is achieved by means of an agreed-upon ordering of the users which establishes which updates take precedence in the case that two or more concurrent updates attempt to refresh the key material of the same node. In such a case, processing such a set of updates will result in that node having the key-pair set by the update that corresponds to the minimal user, with respect to the aforementioned ordering. The protocol makes use of a central server, which collects updates and forwards the relevant parts of them (the ones corresponding to the winning update for each node) at the end of each round. This rather simple modification to the original TreeKEM protocol has the striking effect that the number of rounds of communication rounds needed to heal a corruption is no longer linear (in the number of corrupted parties), but logarithmic. While this is longer than the 2 rounds in which P&C TreeKEM can heal, the total communication is lower, thus circumventing the lower bound by [BDR20]. We note that CoCoA diverges substantially from other existing CGKAs by having users only store a partial view of the ratchet tree, thus allowing them to substantially reduce their download cost. We ignore this here for simplicity, as this does not affect the overall sender communication, nor the healing time.

5.2 Generalized CoCoA healing in k rounds

As mentioned above, CoCoA allows group members to concurrently achieve PCS with less rounds of communication than if updates were done one after the other. This is achieved by progressively healing up the tree upwards, a healing that is effectively captured in [AAN$^+$22a] in the proof of Lemma 2. With regards to PCS, this lemma roughly states that after each party has performed $\lceil \log(n) \rceil + 1$ updates since their last corruption, no key in the resulting ratchet tree can have leaked to the adversary through a corruption. This is analogous to Lemma 2 in [KPPW$^+$21] (which, in turn, requires each user to have performed a *non-concurrent* update since the last corruption), and thus allows for the argument of the latter to carry over. Its main argument, in a rough simplification, proceeds as follows:

Intuition behind CoCoA's security proof. The security proof in [AAN$^+$22a] uses previous results from that of [KPPW$^+$21], which reduces the security of the Tainted TreeKEM protocol to that of a pebbling game on graphs. In particular, it consider the so-called *challenge graph* associated to a given epoch, made up of all nodes whose keys allow the recovery of the challenge key. Then, a reduction is given from the

the CGKA security of the protocol to that of the fairly well-understood *Generalized Selective Decryption* (GSD) [JKK+17] game, played on this graph.[3]

The argument is as follows: consider some key-pair (sk_v, pk_v) associated to node v in the challenge graph and generated at time t. Consider now the key-pairs associated to (p_1, p_2) (in the challenge graph), the parents of v and which allow recovery of sk_v, i.e., those such that either sk_v was encrypted under its public key, or was derived from the same seed by hash evaluations. Then (assuming certain consistency guarantees between users' views), for each p_i, its key-pair must be the current one or, if not, must have been overwritten at time t by another key-pair. Either way, for each p_i, at time t they are either the most recent key-pair anyone sampled for that node, or the ones inmediately before that. Through a similar argument, one can argue that the similarly defined keys associated to the parents of each p_i must be either the most recent ones at those nodes, or among the last two before those. And so on. In particular, if we consider the key-pair (sk, pk) at the root node at time t, we can deduce that the key-pairs associated to leaf nodes that can recover sk must be within the last $\kappa = \lceil \log(n) \rceil + 1$ key-pairs sampled for each of those leaves (at time t), since the depth of the tree (it is not too hard to see the depth of the challenge graph – a tree, in fact – is bounded by the maximum depth of the ratchet tree throughout the execution) is $\lceil \log(n) \rceil + 1$. Thus, if each user updated κ times since their last corruption, we have the guarantee that any key from which sk is recoverable was sampled in one of those κ updates and is therefore safe (and one can actually show that exactly κ updates from each party are actually needed to ensure this).

Generalization to healing in k rounds. The conclusion from the arguments above is that the number of rounds required for healing is exactly determined by the depth of the tree, i.e. by the length of the longest path, plus 1. Thus, the generalization to a protocol healing in k rounds for any arbitrary $k \in [2, \lceil \log(n) \rceil + 1]$ can be obtained by considering the use of ℓ_k-ary ratchet trees of depth $k-1$, $\ell_k = \lceil \sqrt[k-1]{n} \rceil$. Note that the extreme cases for $k = 2$ and $k = \lceil \log(n) \rceil + 1$ correspond exactly to the flat tree where every leaf is a parent of the root, and to CoCoA, respectively.

The algorithms for this more general protocol would just work as in CoCoA, now resolving conflicts of concurrent updates by picking a winner between a set of potentially more than 2 users. The only difference would be that seeds for any newly sampled node in an update would need to be encrypted to a higher number of nodes, to $\ell_k - 1$ in particular, making the overall cost of n users updating k times each to $nk(k-1)(\ell_k - 1)$. Last, the security of the protocol could be argued exactly as above.

5.3 CoCoALight

In this section we will describe a modification of the protocol above with a reduced overall communication complexity. The protocol uses trees of higher arity $\lceil \sqrt[k/2-1]{n} \rceil$, as we will require that the length of a user's path is $\leq k/2$. As in CoCoA, we assume users communicate in rounds, at the end of which the server collects the sent messages and delivers them to the users.

The version described here is a simplification of a fully-fledged CGKA protocol, as we mainly aim to illustrate an upper bound in a simple way. In particular:

- We assume implicit authentication, and thus ignore the use of any signatures.
- We consider only static groups, where the set of group members stays unchanged throughout the execution.[4]
- We disallow any active attacks, and assume the delivery server behaves honestly, not ever sending inconsistent messages to different users. This allows us to not introduce checks for redundant, malformed, or otherwise invalid messages.

[3] This is the case for their proof in the standard model. The proof in the Random Oracle Model proceeds directly without any reduction to GSD, but uses similar ideas.

[4] In practice, dynamic operations could be implemented similar to as in CoCoA, with the slight modification that the user performing it needs to sample a new seed for v_{root} (without affecting that user's $st.\mathtt{counter}$), as an update in CoCoALight is not guaranteed to generate a new key for the root.

As a result of the last point, we can assume total consistency between users views of the group (list of processed operations, ratchet tree, etc.). In particular, we assume the following robustness guarantees:

> Any user $u \in G$, at time t, will only (accept and) process messages that have been issued by some user $v \in G$ at time t', such that u at t and v at t' had the same view of the group.[5]

We stress that our protocol could be enhanced to achieve this guarantees under stronger adversaries, with control of the delivery server and allowed to send arbitrary packets, using standard techniques, such as signatures, a key schedule, transcript and parent hashes, etc.[AAN+22a, BBR18, AJM22].

Apart from this, there are mainly two differences to (generalized) CoCoA sketched above. On the one hand, and in line with the effort to streamline and simplify the protocol to the minimum, we do not consider partial states here, as mentioned above. On the other hand, and the key feature of the protocol, users will not rotate the keys for all nodes in their path in each update, but instead just rotate the key of a single node. Users keep track, by means of a counter, of which node they last refreshed, and will, in the following update, sample a new key for its child, increasing the counter by 1. In the case that two users send ciphertexts corresponding to the same node in the same round the server will decide a winner, as in CoCoA, and thus whose key will be the next one associated to said node, according to any agreed-upon (potentially deterministic) rule. In the case of such a collision, the user losing will still "make progress" and increase their counter, and so, in the following update will attempt to rotate the key at the next node in their path.

A consequence of rotating a single key per update is that knowledge of parts of the old state might allow the recovery of this new key. In particular, the knowledge of the secret key of the parent key of v, when v's key is being refreshed, allows the recovery of the latter (as its seed will be encrypted under the former). Thus, informally, what ensures healing is the progressive rotation of all the path's keys after corruption, and *starting* from the leaf. Note that we have no guarantee as to the state of the user's counter at the moment of corruption: in the worst case the user could have been corrupted right after updating their leaf key. The leakage of this through the corruption would allow the adversary to recover the key of all subsequent updates, up until the point the user updates their leaf again. Thus, in order to guarantee healing in k rounds, CoCoALight uses trees of depth $\approx k/2$, to ensure a rotation of all keys in the path *starting at the leaf* happens within that period.

The protocol is described in detail in Figure 9. Each user in the group has an internal state of the form $st = (u, \mathcal{T}, i, \gamma)$. Here, u is the identifier of the user, which is in the range $[n]$. $\mathcal{T}$ is a ratchet tree, capturing the user's view of the group; if the user has not yet joined a group, it will simply have the setup PKE key-pair from that user (one can see $\mathcal{T}$ in this case as just being the ratchet tree consisting of a single leaf, that of u). The counter i is an integer in the range $(0, d)$, where d is the length of $\mathsf{path}(\mathcal{T}, u)$, and keeps track of the user's progress along their path. Finally, λ is the seed sampled in the last not-yet-processed update, which is equal to $\perp$ if no such update exists. Even though we do not consider dynamic operations, nodes in the tree can be *blank*, meaning they have no associated keys (this occurs in many protocols as an effect of a removal). This will be the case during the setup of the group, which will start with a tree whose nodes are all blank, except for the root and the leaves (this is common to most ratchet-tree-based CGKAs). Blank nodes stop being so once a key is assigned to them by an updating user. The drawback of a node being blank, is that no secret can be encrypted to it - instead one needs to encrypt to its two parents. The *resolution* $\mathsf{resol}(v)$ of a node v is the set of nodes under whose keys one has to encrypt to, in order to communicate a secret to all users whose leaves are ancestors of v. That is, if v is not blank, $\mathsf{resol}(v) = \{v\}$. Else, $\mathsf{resol}(v) = \cup_{p \in \mathsf{parents}(v)} \mathsf{resol}(p)$, where $\mathsf{parents}$ simply returns the two parent nodes of v in the ratchet tree.

For simplicity, we assume the (leaf) public keys of each user are known to everyone else; we formalize them by including them in the public parameters *pub*. Additionally, we assume for simplicity that the messages included in the vector M, input to $\mathsf{Process}$, are ordered "bottom-up", and only a single message per node is relayed by the server in case of concurrent updates (that of the winner). That is, those corresponding to nodes lower in the tree appear first. This allows users to process them in order without the off-chance that

[5] Actually, our security model gives us the stronger guarantee $t = t'$.

st_u.user	Returns the leaf associated to user u
$st_u.\mathcal{T}$	Ratchet tree in the state of user u
st_u.ctr	Returns the counter in the state of user u
st_u.pend	Returns either $\perp$ or seed from last non-processed update
$v.pk$	Returns the public key in the state of node v
$v.sk$	Returns the secret key in the state of node v
v.blank	Returns 1 is v is blank, 0 else.

Table 2: Notation for state and node attributes.

parents(v)	Returns the pair of parents of node v
child(v)	Returns the child of node v
path(v)	Returns the set of nodes in the path of v to v_{root}
resol(v)	Returns the resolution of node v
leaf(u)	Returns the leaf node associated to user u

Table 3: Helper functions for ratchet trees, given implicit tree $\mathcal{T}$.

they overwrite keys they might later, while still processing M, need. Further, we use the notation in Table 2 to interact with particular keys or elements of users' states.

When referring to nodes we assume an implicit ratchet tree they belong to, since the topology of the ratchet tree stays unchanged throughout the execution (as our protocol is static). We use the helper functions from Table 3.

The protocol makes use of the following functions:

- $\mathsf{GenTree}_k(pub, (pk, sk), r)$ – Takes as input the public parameters pub, which includes the public keys of n users, a key-pair corresponding to one of those users, and a secret r. Outputs an all blank ratchet tree of depth $\lfloor k/2 \rfloor$ with n leaves, except for the root node, that has the seed r associated to its root; and the leaves, which have the n public keys associated to them (also the secret key sk for the corresponding leaf).
- $\mathsf{PathNode}(u, i)$ – Takes as input a user u (implicitly part of a group with associated ratchet tree $\mathcal{T}$) and and integer i. Outputs the i^{th} node in $\mathsf{path}(u)$, with the 0^{th} node being $\mathsf{leaf}(u)$.

5.4 Post Compromise Security proof

In this section we show that the protocol described above achieves PCS after k rounds of communication for any $k \in [4, 2(\lfloor \log(n) \rfloor + 1)]$.[6] This is formalized through the security game in Figure 4.

To prove security of CoCoALight, we follow the approach of [KPPW+21] and consider the graph structure that is generated throughout the security experiment. A node v in the so-called *CGKA graph* is associated with seed r, and a key-pair $(v.pk, v.sk) = \mathsf{PKE.Gen}(r)$. The edges of the graph, on the other hand, are induced by dependencies via (public-key) encryptions. To be more precise, an edge (v, w) corresponds to a ciphertext of the form $\mathsf{PKE.Enc}_{v.pk}(r_w)$. The structure of the CGKA graph depends on the sequence of calls to oracles INIT and ROUND made by the adversary. To argue security of a group key K, we consider the subgraph of the CGKA graph that consists of all ancestors of the node associated to said group key – the so-called *recovery graph of K*. We will be particularly interested in the the recovery graph of the challenge group key, which we will refer to as the *challenge graph*, following terminology from previous works. By correctness, the challenge group key can be derived from any secret key/seed associated to a node in the challenge graph. To argue security, none of the secret keys in the challenge graph must be leaked to the

[6]We can assume k is even, since for k odd, the scheme outlined above will actually heal in $k - 1$ rounds.

```
Algorithm Setup($n$)
$\bar{st}, pub \leftarrow \emptyset$
for $u \in [n]$:
    $r \leftarrow_\$ Rnd$
    $(pk, sk) \leftarrow$ PKE.Gen($r$)
    $st_u \leftarrow (u, (pk, sk), 0, \perp)$
    $\bar{st} \leftarrow \bar{st} \cup st_i$
    $pub \leftarrow pub \cup (u, pk)$
return $\bar{st}, pub$

Algorithm Process($st, pub, M$)
$(u, \mathcal{T}, i, \gamma) \leftarrow st$
if $M[0] =$ "init":
    $(pk, sk) \leftarrow \mathcal{T}$
    $c \leftarrow M[u]$
    $r \leftarrow \mathsf{Dec}_{sk}(c)$
    $T \leftarrow \mathsf{GenTree}_k(pub, (pk, sk), r)$
    $st \leftarrow (u, T, 0, \perp)$
else:
    for $(v, pk, c) \in M$:
        $v.pk \leftarrow pk$
        if $v =$ PathNode($u, i$) and $\gamma \neq \perp$:
            $(v, r) \leftarrow \gamma$
            $(v.pk, v.sk) \leftarrow$ PKE.Gen($r$)
            $st$.pend $\leftarrow \perp$
            if $i = |\mathsf{path}(u)|$:
                $st$.counter $\leftarrow 0$
            else $st$.counter $\leftarrow i + 1$
        elseif $v \in \mathsf{path}(u)$ and $c \neq \perp$:
            $w = \mathsf{resol}(v) \cap \mathsf{path}(u)$
            $r \leftarrow \mathsf{Dec}_{sk_w}(c)$
            $(pk, sk) \leftarrow$ PKE.Gen($r$)
            $v.sk \leftarrow sk$
return $(st, v_{root}.sk)$

Algorithm Init($st, pub, n$)
$(u, (pk, sk), i, \gamma) \leftarrow st$
$(j, pk_j)_{j \in [n]} \leftarrow pub$
assert $u \in [n]$
$MI \leftarrow$ ("init")
$r \leftarrow_\$$
for $w \in [n]$:
    $c_w \leftarrow_\$ \mathsf{PKE.Enc}_{pk_w}(r)$
    $MI \leftarrow MI \cup c_w$
return $(st, MI)$

Algorithm Update($st$)
$(u, \mathcal{T}, i, \gamma) \leftarrow st$
$v \leftarrow$ PathNode($u, i$)
$r \leftarrow_\$ Rnd$
$st$.pend $\leftarrow (v, r)$
$(v.pk, v.sk) \leftarrow$ PKE.Gen($r$)
if $i = 0$:
    $MU \leftarrow MU \cup (v, v.pk, \perp)$
else:
    for $p \in \mathsf{resol}(v)$:
        $c_p \leftarrow_\$ \mathsf{PKE.Enc}_{p.pk}(r)$
        $MU \leftarrow MU \cup (v, v.pk, c_p)$
return $(st, MU)$
```

Figure 9: Description of protocol CoCoALight$_k$, healing in k rounds.

adversary via a corruption. Rather strikingly, since k updates for each user means each user rotates every key in their path to the root only twice, we show that this is indeed the case for CoCoALight.

Theorem 8. *Let K be the challenge group key in the* IND-k-PCS$_{\text{RC}}$ *game and assume* safe$_{k\text{-PCS}}$ *returns* 1. *Then none of the seeds and secret keys in K's challenge graph are leaked via corruption.*

Proof. First, observe that from any execution of the game we can extract a unique sequence of group keys (and their corresponding recovery graphs), given by the initial call to INIT, and all subsequent calls to ROUND. We will refer to this sequence as $(K_i)_{i \in [t_{\max}]}$, and the corresponding recovery graphs as $(C_i)_{i \in [t_{\max}]}$. Additionally, each epoch has an associated ratchet tree $\mathcal{T}_i$. In the case we are considering of static groups, all the $\mathcal{T}_i$ are the same when seen as graphs, though will obviously differ in the key material associated to them. Further, some of the $\mathcal{T}_i$ will have blank nodes, as will be the case with the ratchet tree after initialization. However, when seen as graphs they will all be isomorphic, and we will simply refer to them by $\mathcal{T}$. Note that, due to the blanks, the C_i will not all be isomorphic to each other as graphs. Nevertheless, there is a clear injection from the node set of C_i to $\mathcal{T}$, as each C_i is a minor of $\mathcal{T}$, i.e. a graph resulting from applying a series of edge contractions to $\mathcal{T}$ (in particular, the contractions of exactly those edges that share a common blank node).

Throughout this proof, we will treat the C_i as maps from the set of nodes to the set of public keys, such that $C_i(v)$ is the public key associated to node v in C_i. For $v \notin C_i$ (i.e. because v is blank at time i), we say $C_i(v) = \perp$. Let $K^* = K_{t^*}$ be the challenge group key in the experiment, with C^* and $\mathcal{T}^*$ being the challenge graph and its associated ratchet tree, respectively. Then, for all nodes $v \in \mathcal{T}$, we will denote by $v.pk^j$ the

$j+1^{th}$ last public key associated to v, with respect to the time K^* was generated; e.g., $v.pk^0$ corresponds to the key associated to v in C^*, $v.pk^1$ corresponds to the one that $v.pk^0$ overwrote, and so on. Note that there exists some j', namely the number of keys every associated to v, such that for $j \geq j'$, $v.pk^j = \perp$. For the last bit of notation, we will write $C_i \lhd C_j$ if $i < j$, i.e., if C_i is the challenge graph of a key which preceded that of C_j. In a slight abuse of notation, for a node $v \in \mathcal{T}$, we will also write, $v.pk^j \lhd v.pk^i$, whenever $i < j$; i.e., whenever $v.pk^j$ precedes $v.pk^i$. Note that, while we index the graphs in order increasing with time, the ordering for the keys associated to a node is opposite, since for these we count from the time of the challenge, backwards. In both cases, if $a \lhd b$, then the intuition is the same, in that a came in time before b.

We start by defining a *crossing* between two challenge graphs C_i and C_j. We say that there is a crossing between such a pair of challenge graphs if $C_i \lhd C_j$, but there exist node v such that $C_i(v) \rhd C_j(v)$ (and $C_j(v) \neq \perp$). The proof can now be seen as the combination of proofs for the following statements, from which the theorem follows inmediately:

1. Assume for contradiction that a key in C^* leaked through a corruption during the game execution, then there exists a crossing between two challenge graphs.
2. No crossing between any two challenge graphs exists.

Proof of Statement 1. Since, by assumption, the predicate $\mathsf{safe}_{k\text{-PCS}}$ evaluates to 1, we know each user u has performed at least k updates since the last query q of the form $\mathsf{CORR}(u)$ was invoked. In, particular, this means that each node in $\mathsf{path}(u)$ has been updated twice since this time, as $\mathcal{T}$ has depth $\lfloor k/2 \rfloor$. Thus, if the seed or keys associated to some node v have leaked, it must be that $C^*(v) = v.pk^i$ for some $i \geq 2$. Otherwise, this key would not have existed at the time of corruption.

Assume that it is the seed or secret key at v that leaks. From the observation in the paragraph above, it follows that there must have been rounds t_1 and t_2, $t_1 < t_2$, ocurring after q and before the challenge query, where u updated (or tried to, concurrently to someone else) node v_{root} and v, respectively. Let $\hat{C}$ be the challenge graph associated with round t_1. We have that $\hat{C} \lhd C^*$. Now, let $\hat{pk}$ be the key associated to v at time t_2. Since we know v got updated by some user at time t_2, after $C^*(v)$ was already sampled, we have $C^*(v) \lhd \hat{pk}$. Thus, it remains to show that $\hat{pk} \lhd \hat{C}(v)$, which would imply $C^*(v) \lhd \hat{C}(v)$. This, in turn, implies there is a crossing between C^* and $\hat{C}$.

To show this last statement, note that it is a special case of the following: let C be a challenge graph such that $C(v_{root})$ was sampled by user $\hat{u}$. Then, for any vertex w and any j such that $w.pk^j$ was sampled by $\hat{u}$ before they sampled $C(v_{root})$, it holds that $w.pk^j \lhd C(w)$. This follows straight away from the fact that updates are processed by all users in the same round they are issued, and cannot be processed twice. $\square$

Proof of Statement 2. Assume for contradiction there is a crossing between C_i and C_j, $C_i \lhd C_j$, and let v be the highest node (closest one to v_{root}), such that $C_i(v) \rhd C_j(v)$. Let $w = \mathsf{child}(v)$. Note that it is not possible that $C_i(w) = C_j(w)$, since this would imply (except with negligible probability corresponding to two updates sampling the same seed) that the user who sampled this key encrypted it to two different public keys associated to the same node, v. Thus, by assumption, $C_i(w) \lhd C_j(w)$. Now, observe $C_j(v)$ preceded $C_i(v)$, by assumption, and that the latter, in turn, was generated strictly before $C_i(w)$, since both keys belong to the same challenge graph. Thus, when $C_j(w)$ was generated, in round j, $C_j(v)$ was no longer part of $\mathcal{T}_j$, which is a contradiction.

This concludes the proof.

$\square$

The security of the protocol now follows directly through standard arguments [KPPW+21]. In our case, the reduction from the security of the employed PKE scheme would just employ an adversary A against the IND-k-PCS$_{\text{RC}}$ security game by simulating the game for A, and embedding the challenge ciphertext from the PKE game in the one containing K^* in round t^*. Against adaptive adversaries, the reduction would need to guess the key on which it will get challenged, which could be done using the *piecewise guessing* framework [JKK+17], as in [KPPW+21].[7]

$\square$

[7]For technical reasons, one would need to either have the group key be a value from which no public key can be derived

5.5 Efficiency and tradeoffs

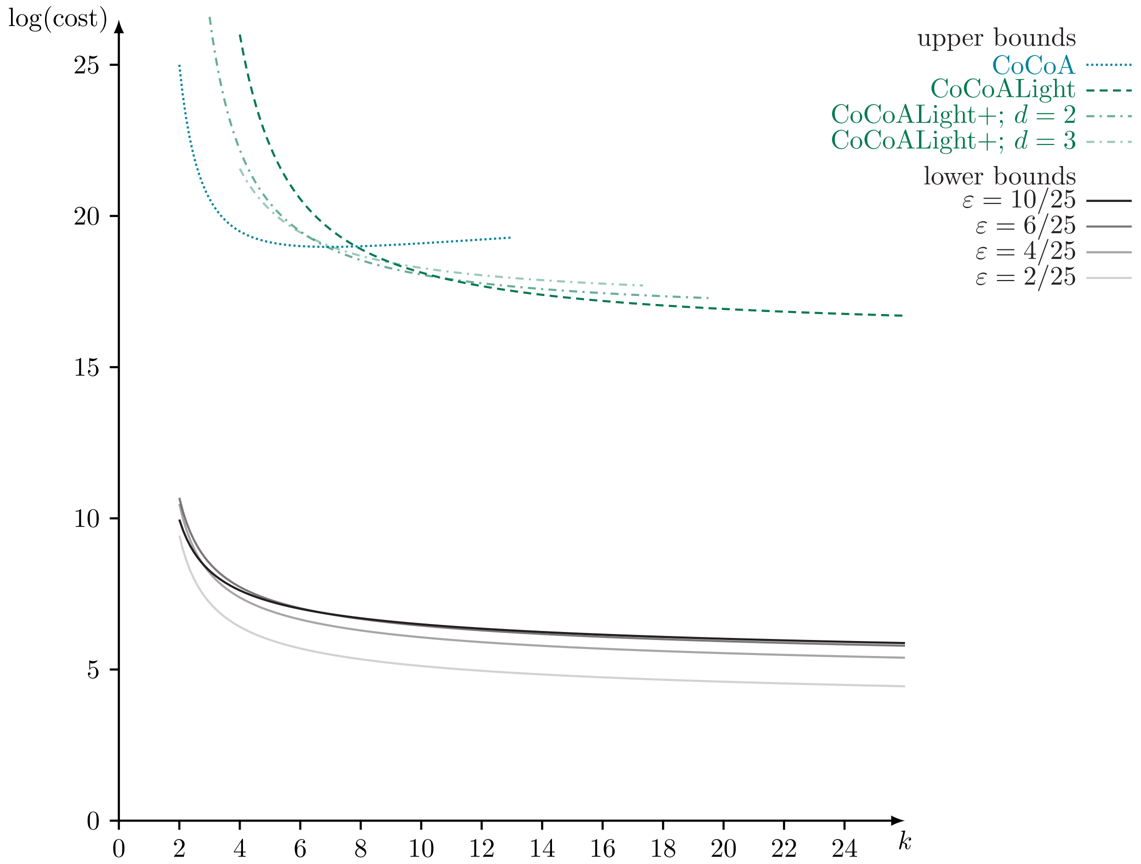

Figure 10: Shown are the communication costs for for different values of k, for (the generalization from Section 5.2 of) CoCoA [AAN+22a] and CoCoALight (Figure 9), as well as the lower-bound shown in Theorem 3, for different values of ϵ. Further, a generalization of CoCoALight, termed CoCoALight+ is shown for different values of $d \in \{2, 3\}$. The size of the group is $n = 4096$.

In this section we briefly discuss the efficiency of CoCoALight and compare it to that of CoCoA, as well as to the lowerbound shown in this paper.

The communication cost of the respective protocols and lowerbounds (recall our lowerbound depends on variable $\epsilon \in (0, 2/5)$) is shown in Figure 10. Notice that, whereas CoCoALight is more efficient for higher values of k, CoCoA is still better for lower values. Indeed, the figure shows that both protocols present quite different communication efficiency profiles, both concrete and asymptotic. In order to smooth out this difference, one could consider more a more general protocol, of which the former are special cases of, and that we briefly sketch further ahead in this section. Essentially, one can have updates rotate a number d of nodes in the user's path, instead of just 1 (as in CoCoALight) or the whole path (as in CoCoA). The cost of this generalized protocol for a couple of different values of d is also displayed in Figure 10 under the name CoCoALight+.

Recall that the generalized CoCoA protocol from Section 5.2 requires a total number of ciphertexts

(and queried to an encryption oracle). This can be achieved by, e.g., adding a key schedule through the use of a PRF, as in [AAN+22a], or by determining that the root node has no associated public key, as in [KPPW+21]. We omit this, for simplicity.

equal to $nk^2 \sqrt[k]{n}$. In order to see the cost of CoCoALight note that each update entails crafting (at most) a number of ciphertexts equal to the in-degree of the ratchet tree, which is $\lceil \sqrt[k/2-1]{n} \rceil$. Thus, if each user performs k updates, the total number of ciphertexts is in the order of $nk \sqrt[k/2-1]{n}$. The slight caveat is that $k \in [4, 2(\lceil \log(n) \rceil + 1)]$, as opposed to $k \in [2, \lceil \log(n) \rceil + 1]$ for the latter. For values of k supported by both protocols, the difference is approximately a factor of $k / \sqrt[k/2]{n}$, meaning the two curves meet at the value of k for which $k^k = n^2$.

While the lower bound shown in this paper matches the asymptotic behaviour of CoCoALight, the concrete difference is still considerable. Thus, it remains an open problem to improve the constants of either our lower bound or existing constructions that allow for concurrent healing, the latter perhaps through the use of ideas like those of [HKP+21] and [AHKM22].

Bridging the upper bounds gap. As mentioned above, one can define a more general protocol encompassing both previously discussed ones. Indeed, instead of having an update rotate the keys on a whole path (as in CoCoA) or on a single node (as in CoCoALight), an arbitrary number d of keys evenly spaced out over the user's path could be rotated. Since the number of rotated keys affects the speed with which PCS is achieved, the depth of the ratchet tree will depend on both k and d. More in detail, this algorithm would work very much like the one from Figure 9, with the difference that whenever a user u with counter i updates, they would rotate the keys of nodes $i + j \cdot |\mathsf{path}(u)|/d \mod |\mathsf{path}(u)|$ for all $j \in \{0, \ldots, d-1\}$ and increase the counter by 1. In order to determine the needed tree depth, recall that healing actually happens when all keys on the user's path are replaced one after another starting from the leaf. In CoCoALight this translates to all keys being replaced after the user's counter i is equal to 0. In turn, in this general protocol, if δ is the tree depth, healing would happen δ updates after either the counter i of the user updating satisfies $i + j \cdot \delta/d \mod \delta = 0$ for any j. That is, after $\lceil \delta \cdot (1 + 1/d) \rceil$ updates.

Thus, for the protocol to heal in k rounds, we need a tree of depth $\delta \approx k(1 + 1/d)^{-1}$, i.e., of in-degree $\approx \sqrt[\delta-1]{n}$. In particular, the cost of an update is $\approx \delta(\sqrt[\delta-1]{n} - 1)$. Since we have n parties, each updating $\delta \cdot (1 + 1/d)$ times, this translates fo a total cost of order

$$n\delta^2(1 + \frac{1}{d}) \sqrt[\delta-1]{n} = \frac{nk^2 \sqrt[\delta-1]{n}}{1 + \frac{1}{d}} \quad .$$

Note, however, the interdependence between d and δ. On the one hand, δ is a function of d, and must also belong to the interval $[2, \lceil \log(n) \rceil + 1]$. On the other, d must belong to the interval $[1, d]$. Figure 10 shows the efficiency profile of this protocol for $d = 2, 3$, starting at values $k = 3$ and $k = 4$ respectively (note that the setting $d = k$ is exactly CoCoA). Further, note that while the functions are discrete, as k can only take integer values, we plot them as continuous in order to better appreciate the interpolation-like effect that CoCoALight+ has bridging the gap between the other two protocols.

Forward Secrecy. In this paper we focus on the communication cost of PCS, and consider forward secrecy outside of its scope. This is, however, as aspect in which this protocol is worse that existing ones. Indeed, while for other protocols a user replaces all their secret key material when effecting an update, this does not need to be the case for CoCoALight. In the worst case, a user might need to perform $k/2$ updates before all secret keys in their path have been rotated. We leave a formal analysis of FS guarantees for future work.

References

[AAB$^+$21] Joël Alwen, Benedikt Auerbach, Mirza Ahad Baig, Miguel Cueto Noval, Karen Klein, Guillermo Pascual-Perez, Krzysztof Pietrzak, and Michael Walter. Grafting key trees: Efficient key management for overlapping groups. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part III*, volume 13044 of *LNCS*, pages 222–253. Springer, Heidelberg, November 2021.

[AAN$^+$22a] Joël Alwen, Benedikt Auerbach, Miguel Cueto Noval, Karen Klein, Guillermo Pascual-Perez, Krzysztof Pietrzak, and Michael Walter. CoCoA: Concurrent continuous group key agreement. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 815–844. Springer, Heidelberg, May / June 2022.

[AAN$^+$22b] Joël Alwen, Benedikt Auerbach, Miguel Cueto Noval, Karen Klein, Guillermo Pascual-Perez, and Krzysztof Pietrzak. Decaf: Decentralizable continuous group key agreement with fast healing. Cryptology ePrint Archive, Paper 2022/559, 2022. https://eprint.iacr.org/2022/559.

[ACDT20] 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.

[ACJM20] Joël Alwen, Sandro Coretti, Daniel Jost, and Marta Mularczyk. Continuous group key agreement with active security. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 261–290. Springer, Heidelberg, November 2020.

[AHKM22] Joël Alwen, Dominik Hartmann, Eike Kiltz, and Marta Mularczyk. Server-aided continuous group key agreement. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 69–82. ACM Press, November 2022.

[AJM22] Joël Alwen, Daniel Jost, and Marta Mularczyk. On the insider security of MLS. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 34–68. Springer, Heidelberg, August 2022.

[BBR18] Karthikeyan Bhargavan, Richard Barnes, and Eric Rescorla. TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups. https://mailarchive.ietf.org/arch/attach/mls/pdf1XUH6o.pdf, May 2018.

[BBR$^+$23] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. RFC 9420, July 2023.

[BCK21] Chris Brzuska, Eric Cornelissen, and Konrad Kohbrok. Cryptographic security of the MLS RFC, draft 11. Cryptology ePrint Archive, Report 2021/137, 2021. https://eprint.iacr.org/2021/137.

[BDG$^+$22] Alexander Bienstock, Yevgeniy Dodis, Sanjam Garg, Garrison Grogan, Mohammad Hajiabadi, and Paul Rösler. On the worst-case inefficiency of CGKA. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part II*, volume 13748 of *LNCS*, pages 213–243. Springer, Heidelberg, November 2022.

[BDR20] Alexander Bienstock, Yevgeniy Dodis, and Paul Rösler. On the price of concurrency in group ratcheting protocols. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 198–228. Springer, Heidelberg, November 2020.

[CCG$^+$18] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1802–1819. ACM Press, October 2018.

[CGI$^+$99] Ran Canetti, Juan A. Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast security: A taxonomy and some efficient constructions. In *IEEE INFOCOM'99*, pages 708–716, New York, NY, USA, March 21–25, 1999.

[CHK21] 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.

[DDF21] Julien Devigne, Céline Duguey, and Pierre-Alain Fouque. MLS group messaging: How zero-knowledge can secure updates. In Elisa Bertino, Haya Shulman, and Michael Waidner, editors, *ESORICS 2021, Part II*, volume 12973 of *LNCS*, pages 587–607. Springer, Heidelberg, October 2021.

[DY83] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[HKP$^+$21] 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 Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1441–1462. ACM Press, November 2021.

[HKP22] Keitaro Hashimoto, Shuichi Katsumata, and Thomas Prest. How to hide MetaData in MLS-like secure group messaging: Simple, modular, and post-quantum. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1399–1412. ACM Press, November 2022.

[JKK$^+$17] Zahra Jafargholi, Chethan Kamath, Karen Klein, Ilan Komargodski, Krzysztof Pietrzak, and Daniel Wichs. Be adaptive, avoid overcommitting. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 133–163. Springer, Heidelberg, August 2017.

[JMM19] Daniel Jost, Ueli Maurer, and Marta Mularczyk. A unified and composable take on ratcheting. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 180–210. Springer, Heidelberg, December 2019.

[KPPW$^+$21] 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.

[Mat19] Matthew A. Weidner. Group Messaging for Secure Asynchronous Collaboration. Master's thesis, University of Cambridge, June 2019.

[MP04] Daniele Micciancio and Saurabh Panjwani. Optimal communication complexity of generic multicast key distribution. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 153–170. Springer, Heidelberg, May 2004.

[WHA99] D. Wallner, E. Harder, and R. Agee. Key management for multicast: Issues and architectures. Request for Comments: 2627, Internet Engineering Task Force, 1999.

[WKHB21] Matthew Weidner, Martin Kleppmann, Daniel Hugenroth, and Alastair R. Beresford. Key agreement for decentralized secure group messaging with strong security guarantees. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2024–2045. ACM Press, November 2021.