Towards Efficient Secure Group Messaging

by

Miguel Cueto Noval

June, 2026

A thesis submitted to the
Graduate School
of the
Institute of Science and Technology Austria
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Committee in charge:

Carrie Bernecky, Chair
Krzysztof Pietrzak
Matthew Kwan
Alon Rosen

Institute of
Science and
Technology
Austria

The thesis of Miguel Cueto Noval, titled *Towards Efficient Secure Group Messaging*, is approved by:

Supervisor: Krzysztof Pietrzak, ISTA, Klosterneuburg, Austria

Signature: _____

Committee Member: Matthew Kwan, ISTA, Klosterneuburg, Austria

Signature: _____

Committee Member: Alon Rosen, Department of Computing Sciences, Bocconi University, Milan, Italy

Signature: _____

Defense Chair: Carrie Bernecky, ISTA, Klosterneuburg, Austria

Signature: _____

Signed page is on file

ISTA Thesis, ISSN: 2663-337X

ISBN: 978-3-99078-087-9

I hereby declare that this thesis is my own work and that it does not contain other people's work without this being so stated; this thesis does not contain my previous work without this being stated, and the bibliography contains all the literature that I used in writing the dissertation.

I accept full responsibility for the content and factual accuracy of this work, including the data and their analysis and presentation, and the text and citation of other work.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other university or institution.

I certify that any republication of materials presented in this thesis has been approved by the relevant publishers and co-authors.

Signature: _____

Miguel Cueto Noval
June, 2026

Signed page is on file

Abstract

The widespread adoption of apps like Whatsapp and Signal has translated into billions of people all around the world communicating on a regular basis by making use of services that offer end-to-end encryption and even provide security guarantees when a user's device is compromised.

This was made possible by the introduction of the Double Ratchet Algorithm [PMS16], which was designed for a setting where communication takes place between two parties. However, in practice, many apps offer the possibility of creating groups. The protocols they use to secure communication are inefficient for large group which has the undesireable consequence that the aforementioned apps have established limits on the group size of roughly 1000 users. This has motivated the introduction of the Messaging Layer Security (MLS) standard [BBR+23] by the IETF which is based on a primitive called Continuous Group Key Agreement (CGKA) [ACDT20].

This primitive allows a group of users to maintain a shared secret key that is frequently rotated by the group members in order to change group membership, achieve forward secrecy (FS) and post compromise security (PCS). Most protocols are based on binary trees where the nodes are associated to a pair formed by public key and a secret key. Each leaf corresponds to one of the group members and a user knows the secret keys associated to nodes along the path from their leaf to the root. When a user wants to update their key material they have to change $\log(N)$ many keys. This requires uploading $\log(N)$ many ciphertexts to communicate the new keys to the rest of the group members in a way that respects the tree structure.

In this thesis we study how much communication between group members is required in order to add and remove users from a group as well as in order to provide PCS when we consider CGKAs built using standard cryptographic primitives like pseudo-random functions and public-key encryption. Furthermore, we also consider the case of MLS and provide the first lower bound showing that its communication complexity is much worse than previously believed, i.e., it is very far from $\log(N)$. Finally, we also propose a variant of MLS which provably achieves the same security properties with a much lower communication cost.

Acknowledgements

First and foremost, I would like to thank my supervisor Krzysztof Pietrzak for giving me the opportunity of pursuing a PhD and introducing me into cryptographic research. Over the years, he has been a constant source of problems and ideas, and, without his continued support, this thesis would not have been possible.

I am also deeply grateful to Alon Rosen for hosting me in Milan for a number of research visits. I could not have possibly hoped for a more generous, welcoming mentor, who introduced me into new emerging research problems as well as helped me to grow as a person.

I would like to express my immense gratitude to Benedikt Auerbach for guiding me in the early stages of my PhD and showing me what doing research is like. I would also like to profusely thank Akin Ünal, from whom I have learned almost everything I know about algebraic cryptanalysis.

My incredible collaborators Ahad, Akin, Andrej, Alon, Benedikt, Boran, Charlotte, Chethan, Christoph, Guillermo, Ilia, Joël, Karen, Krzysztof, Margarita, Matthew, Michael A., Michael W., Michelle, Nicholas, Patrick, Simon and Stella have taught me the value of team work and also made the PhD journey an authentic pleasure. I would also like to thank Rita and Christine for guiding me through all the different administrative processes.

I consider myself very lucky for having been part of the cryptography group formed by Ahad, Akin, Anshu, Benedikt, Charlotte, Christoph, Guille, Karen, Michelle, Ray and Suvradip. To all of you for the many moments we have shared over coffee, during trips or just recovering from paper rejections. To Ahad for always coming to campus and making my days feel a bit better.

For the many hikes I would like to thank Manas, Leo, Theju, Sri and Ahad. To Manas for all the board games parties you have organized. Also a big thanks to Ahad, Ben and Chrisl for all the padel games. I am also very grateful to all my friends from Spain for supporting me and making every opportunity we have had to see one another wonderful.

I am especially grateful to my brother David; sharing the experience of being PhD students has been incredible in spite of the distance. Thanks for always having been

there and understanding all the highs and lows of this journey. Finally, my parents for instilling in me this passion for learning and having always supported me unconditionally.

About the Author

Miguel Cueto Noval completed a Bachelor's degree in Mathematics at the University of Oviedo and an M1 in Advanced Mathematics at ENS de Lyon. His reasearch interests involve cryptology broadly understood and has focused on topics such as lower bounds, group messaging and algebraic cryptanalysis. During his PhD he presented his work at Eurocrypt, TCC and Crypto and attended numerous schools and conferences in the area of cryptography. He also completed a research visit to Alon Rosen at Bocconi University.

List of Collaborators and Publications

1. Chapter 3 reuses "©IACR 2023, 10.1007/978-3-031-48621-0_10" "Benedikt Auerbach, Miguel Cueto Noval, Guillermo Pascual-Perez, and Krzysztof Pietrzak. On the cost of post-compromise security in concurrent continuous group-key agreement. In Guy Rothblum and Hoeteck Wee, editors, *TCC 2023, Part III*, volume 14371 of *LNCS*, pages 271–300. Springer, Heidelberg, Nov 2023 [ACPP23]". Mainly responsible for results in the symbolic model and one of the two upper bounds.

2. Chapter 4 reuses "©IACR 2024, 10.1007/978-3-031-78011-0_14" "Michael Anastos, Benedikt Auerbach, Mirza Ahad Baig, Miguel Cueto Noval, Matthew Kwan, Guillermo Pascual-Perez, and Krzysztof Pietrzak. The cost of maintaining keys in dynamic groups with applications to multicast encryption and group messaging. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part I*, volume 15364 of *LNCS*, pages 413–443. Springer, Cham, December 2024 [AAB$^+$24]". Mainly responsible for lower bounds in the symbolic model and discussion on the use of broadcast encryption.

3. Chapter 5 reuses "©IACR 2025, 10.1007/978-3-032-01913-4_5" "Benedikt Auerbach, Miguel Cueto Noval, Boran Erol, and Krzysztof Pietrzak. Continuous group-key agreement: Concurrent updates without pruning. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VIII*, volume 16007 of *LNCS*, pages 141–172. Springer, Cham, August 2025 [ANEP25]". Mainly responsible for the study of the complexity of the new protocol and its security and some of the lower bounds.

The following list contains other works written during the PhD period but not included in the thesis.

4. "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 [KPPW+21]"

5. "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, Cham, November 2021 [AAB+21]"

6. "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, Cham, May / June 2022 [AAN+22]"

7. "Andrej Bogdanov, Miguel Cueto Noval, Charlotte Hoffmann, and Alon Rosen. Public-key encryption from homogeneous CLWE. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part II*, volume 13748 of *LNCS*, pages 565–592. Springer, Cham, November 2022 [BNHR22]"

8. "Joël Alwen, Benedikt Auerbach, Miguel Cueto Noval, Karen Klein, Guillermo Pascual-Perez, and Krzyzstof Pietrzak. DeCAF: Decentralizable CGKA with fast healing. In Clemente Galdi and Duong Hieu Phan, editors, *SCN 24, Part II*, volume 14974 of *LNCS*, pages 294–313. Springer, Cham, September 2024 [AAN+24]"

9. "Miguel Cueto Noval, Simon-Philipp Merz, Patrick Stählin, and Akin Ünal. On the soundness of algebraic attacks against code-based assumptions. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part VI*, volume 15606 of *LNCS*, pages 385–415. Springer, Cham, May 2025 [NMSÜ25]"

10. "Nicholas Brandt, Miguel Cueto Noval, Christoph U. Günther, Akin Ünal, and Stella Wohnig. Constrained verifiable random functions without obfuscation and friends. In Benny Applebaum and Huijia (Rachel) Lin, editors, *TCC 2025, Part IV*, volume 16271 of *LNCS*, pages 478–511. Springer, Cham, December 2025 [BNG+25]"

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

The fact that more than half of the world's population (54% according to [SB25]) now has access to the internet through their own smartphone also means that an ever-increasing number of people can make use of the messaging services provided by apps like WhatsApp, iMessage, Messenger or Signal. In fact, according to WhatsApp's official website "more than three billion people in over 180 countries use WhatsApp" [Wha].

This means that whenever two people have a conversation on any of these apps they are benefiting from the security properties of the Double Ratchet Algorithm introduced by Perrin and Marlinspike [PMS16] or some variant thereof. These protocols allow the users to derive some keys that are known to both parties and use them to encrypt their messages in order to prevent anyone else from learning their content or impersonating any of the parties. This is usually referred to as *end-to-end encryption (E2EE)*. However, the key difference with previous protocols is that the Double Ratchet Algorithm instead of simply deriving one key, derives a sequence of keys through key updates that provide confidentiality even when a key is leaked in the following sense:

- the messages sent before remain secret (forward secrecy), and,
- after a new key is derived using fresh randomness, future messages are also kept secret (post-compromise security).

The aforementioned apps do not restrict communication to take place between two parties. Instead, users may create group chats. This leads to the question of how to offer participants the same security guarantees as before but now in a group setting. The most natural solution for a group of n users is to consider the $n(n-1)$ pairs of distinct users separately and for each of them execute the Double Ratchet Algorithm. However, this has a major shortcoming, namely, communication complexity scales linearly in the group size as sending a message requires encrypting it individually to

each party. This is not just a purely theoretical observation. It actually has real-world implications; WhatsApp limits the number of users in a group to 1024 users and Signal establishes a maximum group size of 1000 users.

The question of building a scheme that combines security and efficiency has motivated the creation of the IETF working group *Messaging Layer Security (MLS)* and has led to the publication of the RFC 9420 [BBR+23] standard, to which we will refer as MLS from now on. Unlike the aforementioned apps, it has the goal of supporting groups of up to 50000 users.

This thesis studies the trade-offs between efficiency and security in the context of group messaging. The problem becomes trivial when one considers heavy machinery such as succinct multiparty non-interactive key exchange [ABG+25]. However, this is very far from being practically efficient. Therefore, we only consider protocols that are built using standard cryptographic primitives such as public-key encryption or pseudo-random generators.

1.1 A Simple Approach to Building Secure Group Messaging

For now, let's consider the problem of confidentiality and ignore other properties such as authenticity. That is, we would like to guarantee that only group members can read the content of the messages. There is already a standard cryptographic primitive that solves this problem, namely, symmetric encryption. If all group members agree on a secret key and they encrypt their messages with this key, then they can be sure that only the other group members can recover the original messages. In other words, the problem of confidentiality in group messaging can be reduced to the question of group key agreement. The main challenge in its application to group messaging stems from the fact that we must deal with the problem of membership change.

For instance, removing a user from a group should mean that the remaining users preserve the ability to communicate confidentially, despite the fact that the removed user learned the key they were previously using. Analogously to the approach described earlier, this can be solved by having the remaining group members agree on a new key.

This intuition was formalized in the work of Alwen *et al.* [ACDT20, ACDT21] who introduced a primitive called Continuous Group Key Agreement (CGKA) and formally showed how to build secure group messaging from it (and other cryptographic primitives) while guaranteeing confidentiality and authenticity as well as other security properties specific to group messaging that we will discuss later. The first CGKA scheme is the ART protocol of Cohn-Gordon *et al.* [CCG+18] and actually predates the formalization of the CGKA primitive itself.

Multicast Encryption. Before we delve deeper into the study of CGKA, it is worth mentioning that a similar problem also arises in other contexts beyond that of group messaging. If one considers the case of pay TV, where customers have to pay a monthly fee in order to have access to the content of some channels, the question of being able to communicate a message (here the channel's content) to an evolving group of users (here the customers who pay) has clear similarities to the one of group messaging. There is a natural cryptographic way of solving this problem resembling the previous one; the TV signal is sent encrypted using a symmetric scheme and the company provides the secret key to the users. If a customer stops paying, everything the company has to do is to encrypt the signal under a new key and distribute the new key to those who keep paying as well as communicate it to any new customers. This solves the original problem in a way that still involves sending only one signal and its efficiency is determined by that of the key rotation procedure.

In this kind of setting we have a central authority (CA) that communicates the corresponding secret key to group members and changes this key as determined by how group membership evolves in time. This cryptographic primitive is called multicast encryption (ME). ME predates CGKA by about two decades and the existence of a central authority makes the problem simpler since there is no need to use public-key encryption.

Continuous Group Key Agreement. While in a multicast protocol the main goal of the CA is to communicate a secret key in a way that guarantees that only current group members can learn it, in CGKA this has to be achieved in the absence of a CA. Instead, in CGKA the add and remove operations are carried out by the users themselves and communication takes place over an untrusted network. Moreover, users can also perform another kind of operation that is called an update; it enables a user to refresh their key material without any membership changes taking place. Every time one of these operations takes place, the key on which users agree changes and we say that a new *epoch* starts. Therefore, each epoch has its own associated *group key*. The two security properties that we want to achieve by issuing updates are post-compromise security (PCS) and forward secrecy (FS).

At a high level, we say that a CGKA scheme provides PCS if it guarantees that users can agree on a secure key even after a compromise provided that a certain amount of communication takes place between the group members. A secure key is one that remains indistinguishable from random even when given the keys known to the corrupted users during that period. FS refers to the property that a compromise should not leak information about the group keys from past epochs. In both notions there is a *healing period* between the compromise and the secure keys. How long this period is, leads to more fine-grained notions of security as we will discuss later. Both notions can be combined into one called *post-compromise forward secrecy* (PCFS) that requires that

any key outside the healing period of any of the corrupted keys (from both previous and future epochs) looks random.

Selective vs. Adaptive Adversaries. There are also other questions to take into account when defining what constitutes a secure CGKA scheme. In the CGKA literature adversaries are usually adaptive, i.e., their actions can be adjusted as the execution of the security experiment proceeds. Here we use the term actions in an informal way that the precise definition of security should make explicit, but the reader may think of them as a sequence of operations to be carried out by the users and an epoch whose group key the adversary challenges. This is obviously a desirable property since the intended application of group messaging provides a long period of time (e.g., months or years) in which to carry out an attack and it would not be realistic to assume that an adversary commits to its actions at the beginning of a security experiment.

Passive vs. Active Adversaries. In CGKA it is also particularly relevant whether we consider a passive adversary, i.e., one that is restricted to observing communication and not allowed to tamper with the messages exchanged by the user, or an active one which can behave in arbitrary ways. Some works consider *partially active* adversaries that can control the delivery of messages and corrupt users, but not use the information obtained in order to craft messages. This is not a very realistic model and MLS is known to satisfy a stronger notion of security, namely, *insider security* [AJM22] which captures the possibility that corrupted group members behave maliciously.

Asynchronous communication. In addition to the above considerations, a CGKA is usually designed for asynchronous settings in which not all users may be online at all times and then when a user comes online, it should be able to process the messages previously exchanged and catch up with the group after said messages are relayed by an untrusted server. This is a natural scenario in applications like group messaging and most CGKA constructions (among them the first CGKA construction [CCG+18] and MLS) are designed to operate in an asynchronous setting.

1.2 Efficiency in Multicast Encryption and Continuous Group Key Agreement

The previously described approach to secure group messaging naturally leads to the study of the efficiency of CGKA constructions. Since ME is an older primitive and it has served as a starting point for building CGKA, we first discuss the main idea behind efficient ME schemes.

Key Trees. The basic design principle uses so-called *key trees* and serves as an inspiration for many of the existing ME and CGKA constructions. Key trees were first used in the context of ME by Wallner, Harder and Agee in [WHA99] and by Wong, Gouda and Lam in [WGL00]. They are (usually binary) trees in which the leaves correspond to the users and every node in the tree has an associated secret key. The secrets that belong to any of the nodes along the path from a user's leaf to the root are exactly the secrets in the tree that are known to that user. The root's secret is therefore called the group secret. The protocol proceeds in a way that guarantees that a user can never learn any secret outside of its path to the root. This property receives the name of the *tree invariant*. If we assume that the edges are directed from the root to the leaves[1], then an edge (u, v) corresponds to a ciphertext of the form $\mathsf{Enc}(k_v, k_u)$ that allows anyone with knowledge of the key k_v to also obtain the key k_u.

The internal nodes of the tree make it possible to communicate any change to the group key in an efficient way. Adding a new user can be done by increasing the depth of the tree by one if it is a full binary tree or just making use of one of the available leaves and assigning it to the new user if the number of users N is not a power of two, and, in both cases, sampling fresh secrets along the newly added user's path. In a binary tree this results in at most $\lceil \log(n) \rceil + 1$ ciphertexts.

Eliminating a user corresponds to sampling a new group key and communicating it to the remaining users in a way that avoids using any of the keys known to the removed user. In a key tree this can be done by first deleting all the secrets along the removed user's path and then for each node along their path from their leaf to the root it suffices to sample a new secret and communicate it to all the users below it by encrypting it to its children except for the leaf of the removed user. This requires at most $2\log(n)$ ciphertexts in the case of binary trees. The communication can be reduced by a factor of 2 if one uses a pseudo-random generator (PRG) to derive the new keys along a path as shown in [CGI+99a].

In the context of CGKA, the use of key trees, which usually receive the name of ratchet trees, was first proposed by Cohn-Gordon, Cremers, Garratt, Millican and Milner in [CCG+18], which uses binary trees whose nodes have associated ElGamal secrets. Namely, leaves have a secret x_i and a public value g^{x_i} associated to them and each leaf corresponds to one party who is the only one who knows the corresponding secret. Internal nodes are defined in a recursive way where a node with children whose secret values are a and b is assigned a public value $g^{f(g^{ab})}$ and a secret g^{ab} for some map f to the integers. Most CGKA constructions take the approach of TreeKEM [BBR+23] and use key trees where the public/secret key pair associated to the nodes correspond to an arbitrary public-key encryption.

[1] This convention is not always adopted in the ME and CGKA literature and for example [AAB+21] considers edges pointing towards the root.

However, a crucial difference between these protocols as well as other CGKA constructions like [Wei19, KPPW+21, ACDT20, AAN+22, AAN+24] and the aforementioned multicast schemes is that their efficiency is not well understood. The lack of a central authority implies that adds and removals do not always respect the binary tree structure and, instead, they destroy it. In all of the aforementioned CGKA protocols there exist sequences of adds and removals that result in future operations requiring that a linear amount of ciphertexts be sent. That is, their worst-case communication complexity is linear rather than logarithmic. Moreover, in some protocols like TreeKEM even if we restrict ourselves to considering update operations there are sequences which result in a linear amount of ciphertexts being sent. This defeats the purpose of using binary trees in the first place.

1.2.1 Efficiency of Replacements

When studying the communication complexity of the add and remove operations we are usually interested in asymptotic results as a function of the group size. Therefore we sometimes decide to keep the group size invariant and do so by considering the action of removing a user and adding a user at the same time. We refer to this as a replacement. From the previous description on how key trees work, it follows that we can do a replacement in a way that involves $\log(n) + 1$ many ciphertexts; we remove a group member and use the available leaf for the user we would like to add. Moreover, this only requires the use of symmetric encryption and PRGs.

Batched Replacements in ME. In a large group, it may very well be the case that the CA wants to remove multiple users at the same time. If done sequentially, this would require $d(\log(n) + 1)$ many ciphertexts where d denotes the number of users to be removed. However, if we choose d many leaves in a binary tree as the one used in the protocol outlined in the previous paragraphs we would expect to see their paths intersect. Thus, proceeding to remove the users sequentially results in a number of nodes being repeatedly assigned new secrets. In order to avoid this redundancy we can try to batch those replacements in a way that each node only has to be assigned a new key once. Indeed, if we choose any d leaves in a binary tree, we would need to replace at most $d(1 + \log(n/d))$ nodes [NNL01]. By following this approach one obtains protocols with improved communication cost as shown implicitly by [NNL01] and explicitly for ME in [LYGL01, SM03] with hashing and symmetric-key encryption as building blocks; d users can be replaced in a single round with a communication cost of $d(1 + \log(n/d))$ without increasing the cost of future operations.

(Batched) Replacements in CGKA. Whether the previous approach works in CGKA heavily depends on the security notion considered. Even if the size of the

batch is just one! If we do not consider concurrency and restrict ourselves to a very weak security model where all users are honest and no randomness leakage when sampling new keys is considered and, in particular, all users delete any information as determined by the protocol, this is indeed the case. By using Tainted TreeKEM without taints [KPPW+21] the resulting effect on the tree is essentially the same as in the case of ME though it requires the use of public-key encryption instead.

However, if we do not trust that a removed user has actually deleted any previous keys it may have learned as part of the protocol execution, then Tainted TreeKEM (now in its version with taints for security) has worst-case linear communication. In fact, this is inherent to any CGKA scheme built in a black-box way from PKE that provides PCS, and is, therefore, secure against a non-adaptive adversary that learns (through corruption) old secrets of a user that is later removed. Specifically, a worst-case linear bound was given in [BDG+22].

Previously Known Lower Bounds for Replacements. The first lower bounds on the communication needed to replace one user in ME are due to Canetti, Malkin and Nissim [CMN99] and Snoeyink, Suri and Varghese [SSV01]. Both consider ME schemes built only using symmetric-key encryption and, in particular, do not allow the use of pseudorandom generators. In [CMN99] it is proven that there exists a user whose removal requires that the central authority broadcasts $\log(n)$ ciphertexts if one only considers so-called *structure preserving protocols*. In [SSV01] this additional requirement is dropped, but instead of proving that there is one user whose removal induces such a cost, the authors prove a weaker statement, namely, that there exists a sequence of $2n$ replacements such that the total communication is $\Omega(n \log(n))$.

The first worst-case lower bound to apply to ME protocols built using symmetric-key encryption, PRGs and secret sharing is due to Micciancio and Panjwani [MP04] who defined security by making use of a symbolic model in the style of Dolev and Yao [DY83]. They showed the existence of sequences of (non-batched) replacements that induce a $\log(n) - o(1)$ communication cost per replacement proving that the protocol of [CGI+99a] has optimal worst-case complexity.

When considering CGKA, one should also take into account public-key encryption. Alwen *et al.* [AAB+21] lift the bound of [MP04] to an average-case bound meaning that instead of showing the existence of a sequence that induces a logarithmic cost, they prove that a randomly chosen sequence of replacements also has $\log(n) - o(1)$ communication cost per replacement. Their result applies to both primitives, CGKA and ME.

The work of Bienstock *et al.* [BDG+22] considers CGKA built in a black-box way from public-key encryption and give a worst-case sequence of operations that result in a *sustained* lower bound of $\Omega(n)$. The sequence is built by first having a user u_1 add

a set of users S to the group where $|S| = \Omega(n)$. Next, a previous group member, u_2, proceeds to replace u_1 and since the new group key cannot be learned by u_1, this requires that u_2 addresses the users in S separately because any keys they agree on were communicated to them by u_1 in the previous round. In the next round another user u_3 replaces u_2 and by repeating this process, we obtain a sequence of replacements of a single user per round where the communication cost in each round is $\Omega(n)$.

The approach used to prove the bound clearly relies on not trusting user u_1 to have deleted any key material it communicated to the users in S when adding them. This is a security property not satisfied by Tainted TreeKEM when used without taints and explains why this scheme can achieve a $O(\log(n))$ communication cost for non-batched replacements. The argument used to prove the bound does not extend to the case of ME schemes either.

1.2.2 Efficiency of Updates in CGKA

Concurrency in CGKA. The absence of a CA also introduces an element that has to be considered when studying large groups; multiple users may want to perform operations at the same time. One approach would be to simply require that all operations be done sequentially, i.e., if two users want to do, for example, an update, then one has to do it first and the other user has to first process said update and then proceed to generate their own. This is clearly undesirable for large group sizes as it means that PCS may only be achieved after a linear number of rounds (in the number of concurrently updating parties).[2] This is indeed the case for a number of constructions such as rTreeKEM [ACDT20], Tainted TreeKEM [KPPW+21], Causal TreeKEM [Wei19] and early versions of the CGKA underlying MLS, more specifically, versions 7 and older of TreeKEM.

The Propose-and-Commit Paradigm. In order to handle concurrency, TreeKEM introduces two different kinds of operations called *proposals* and *commits*. A user can propose to add a new user, remove a group member or an update, but proposal messages do not establish new group keys. As part of an update proposal, a user includes a new public leaf key and stores the corresponding secret. Later on a user can issue a commit to a (possibly empty) set of proposals. As part of the commit the user samples new keys for each node along their path. All the non-leaf nodes on the paths of users who issued an update proposal and all the nodes on the path of the users being removed that do not belong to the committing user's path become *blank*, i.e., all their key pairs are deleted and no new key pair is assigned to them. As a

[2] This is assuming that priority is given to parties which have not had a successful update for the longest period of time.

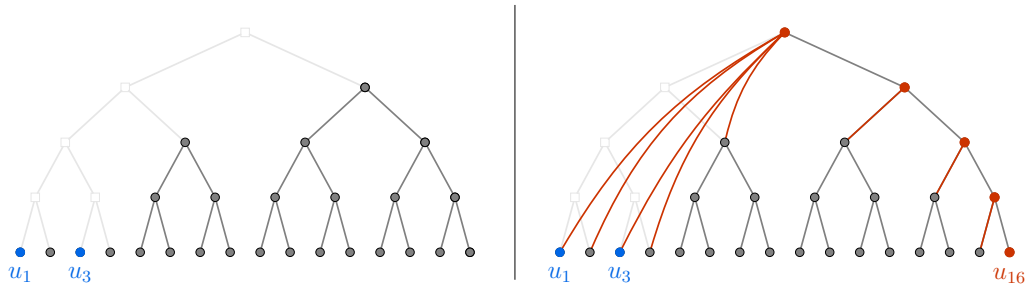

Figure 1.1: Illustration of two concurrent update proposals in TreeKEM by users u_1 and u_3 (left) and a commit to their proposals by user u_{16} (right). As part of their update proposals users u_1 and u_3 sample new leaf keys (colored in blue) and user u_{16} samples new keys for all the nodes along their path to the root (colored in orange). The ciphertexts sent by u_{16} correspond to the edges colored in orange. This sequence of operations results in a total of four blank nodes (light gray nodes).

result, the committing user's co-path[3] may include some blank nodes and therefore the committing user may have to generate additional ciphertexts in order to communicate the new secret keys. This allows to handle concurrent proposals, but commits do not support concurrency. For an example see Figure 1.1.

The current version of TreeKEM based on the *Propose-and-Commit* framework achieves PCS in just two rounds in the sense that if all corrupted parties issue a proposal and these are included in a commit, the resulting group key is secure. Protocols like CoCoA [AAN+22] and DeCAF [AAN+24] allow to process concurrent operations without relying on the Propose-and-Commit framework, though it comes at the cost of a relaxation of the notion of PCS considered, which can take a logarithmic number of rounds to achieve. Additionally, in CoCoA the server crafts individualized packages for each user instead of simply forwarding the messages it receives. This makes it possible to reduce the number of ciphertexts each user downloads. DeCAF achieves PCS in $\log(c)$ many rounds where c is the number of corrupted parties. This value is not known, but, in practice, one may expect it to be much smaller than n. However, in DeCAF users must receive all the messages sent by other users and therefore the recipient communication is higher than in CoCoA.

Previously Known Lower Bounds for Updates. The work of Bienstock, Dodis and Rösler [BDR20] was the first to prove a lower bound on the communication needed in a PCS secure CGKA scheme. They consider a symbolic model similar to the one of [MP04] in which CGKA schemes are built from PRFs, public key encryption and

[3]The co-path of a user u is the set of children of the nodes in u's path that do not belong to u's path.

broadcast encryption. They show that if all the n members of the group can do updates concurrently and 2 rounds of (possibly concurrent) communication suffice to agree on a secure key after all the users get corrupted, then a total number of at least $n(n-1)$ many messages have to be exchanged. The argument proceeds by observing that if the adversary corrupts all the group members, then after a first round of concurrent updates the users do not agree on any secure keys. This means that in the second round of concurrent updates every user has to send a message to the remaining $n-1$ with the new group key. This yields the lower bound of $n(n-1)$, i.e., a linear overhead per updating party.

The previous argument relies on the assumption that there is no coordination between users. If we allow users to coordinate with those doing concurrent updates, then the lower bound is just linear since the first round still requires that users publish a new secret while in the second one it would be enough for one user to send the new group key.

1.3 Our Contributions and Outline

1.3.1 On the Cost of PCS in CGKA

The lower bound of [BDR20] shows that if a CGKA is built from standard primitives and achieves PCS in just two rounds, then it requires linear communication per user. This motivates the study of protocols that may need more rounds to guarantee that a secure key is obtained but do so with a lower communication cost. In CoCoA [AAN+22] each update is of logarithmic size while PCS is achieved after all previously corrupted users perform a logarithmic number of updates even if this is done concurrently.

In Chapter 3 we study upper and lower bounds on the communication needed when one considers such a delayed form of PCS. This notion is parametrized by an integer k between 2 and $\log(n)$ that denotes how many rounds of updates by the corrupted parties must suffice to derive a secure key even if communication is done concurrently.

We obtain a lower bound of order $kn^{1+1/(k-1)}/\log(k)$. When $k=2$, this recovers the bound of [BDR20]. Furthermore we introduce two variants of CoCoA which achieve PCS in k many rounds and result in nearly matching upper bounds with communication cost of $k^2n^{1+1/(k-1)}$ and $kn^{1+2/(k-1)}$.

1.3.2 The Cost of Maintaining Keys in Dynamic Groups

Multicast encryption protocols that allow a CA to replace d many users by sending just $d(1+\log(n/d))$ ciphertexts have been around for more than two decades. In the case when $d=1$ this was proven to be optimal in [MP04, AAB+21] (among protocols built

from standard primitives). However, in the case where $d > 1$ we only know the trivial lower bound of d shown in [BDR20].

In Chapter 4 we close this gap by showing that any ME construction based on standard primitives requires that the CA sends at least $\Omega(d \log(n/d))$ ciphertexts. This is an average case lower bound in the sense that it holds when the parties to be replaced are chosen randomly. We also show that this lower bound holds for CGKA schemes. The result covers all the standard primitives considered in the symbolic models of [MP04, BDR20, AAB$^+$21] with the exception of broadcast encryption (BE) which was also taken into account in [BDR20]. This is, however, justified because we show in Appendix A.1.4 that this primitive can be used to build ME which supports batched user replacements with a cost of just $d + 1$ messages.[4]

1.3.3 Concurrent Updates without Pruning

The introduction of the propose-and-commit framework together with its blanking mechanism made it possible for TreeKEM [BBR$^+$23] to handle concurrency. However, blanks, as shown in Figure 1.1, do not preserve the binary tree structure and, in particular, the in-degree of some nodes increases. This makes future operations more costly.

In Chapter 5 we initialize the formal study of the actual communication cost of the MLS protocol in a way that introduces the number of users who make proposals or commits as parameters p and c, respectively. In our model p users issue a proposal, then a user commits to all said proposals and this is followed by $c - 1$ rounds in which one user commits to an empty set of proposals. We show that if the users doing proposals and commits are chosen at random, the expected size of a commit message is $\Omega(n^{\log_2(1+\frac{p}{p+c})})$. If we consider the case when $p = 1 = c$, this is already worse than $\Omega(\sqrt{n})$. This means that MLS' communication is not just far from logarithmic in the worst-case but also on average.

On the positive side we give a new version of MLS that provably guarantees that commit messages are of size $O(\log(n))$ when proposals and commits are chosen at random and p and c are constant. We also show that it satisfies the same notion of security as MLS does.

[4] Basically, d messages are needed to communicate to the added parties their secret key in the BE scheme and one additional BE ciphertext is used to communicate the new group key to those who remain in the group.

1.4 Other Related Work

There are questions in group messaging that are not considered in this work. For instance, in the real world users are often members of multiple groups with partially overlapping membership. This makes some efficiency gains possible as studied in [AAB+21], but it can also have an impact on security as studied in [CHK21]. We focus our attention on the setting where only one group is taken into consideration.

Even if it is true that the most common approach to secure group messaging follows the modular approach of [ACDT21] and is therefore based on CGKA, it should also be observed that there is a line of work that is not based on TreeKEM [WKHB21, BCG23, CEST24] or not even on a CGKA (and focuses on other security properties) [LZPR24, LZPR25].

This thesis does not take into account the external operations of MLS since these are not part of the usual definition of CGKA, but the recent work of Cremers, Günsay, Wesselkamp and Zhao [CGWZ25] has introduced the notion of external CGKA to account for them. Moreover, the MLS working group is still active at the time of writing this thesis and further additions to the specification are to be expected.

When it comes to the study of the security of MLS or some of its components there is also a line of work whose approach is based on the use of automated tools to obtain machine-checked symbolic proofs [CJL23, WPBB23, WPB25].

A feature that apps usually offer is the possibility of restricting certain group operations such as adds or removals to a subset of members called administrators. This leads to the notion of *administrated CGKA* which was introduced in [BCV23] by Balbás, Collins and Vaudenay.

Another topic that is practically relevant, but not covered by the security notions studied in this work is that of metadata protection. This can be achieved in a generic way as shown in the work of Hashimoto, Katsumata and Prest [HKP22].

Alternatively, one could also study the applications of CGKA (or some of the techniques used when constructing it) to other problems beyond group messaging such as encrypted cloud storage as done by Backendal, Balbás and Haller in [BBH25].

Finally, it should be noted that we focus our attention in ME and CGKA and how to build them from standard primitives (e.g., public-key encryption or signatures) as well as the efficiency of the resulting schemes without discussing specific instantiations of said primitives. This becomes particularly relevant if one wants to implement these protocols and see wide-spread adoption of the MLS standard, and is especially important in the post-quantum setting. In what concerns authentication this has been studied in the work of Hashimoto, Katsumata and Pascual-Perez [HKPP25].

CHAPTER 2

Preliminaries

This chapter presents some of the notation that we use throughout the thesis, basic mathematical results and elementary cryptographic concepts. More advanced notions like that of CGKA will be defined in slightly different ways in each chapter and are therefore not included here.

2.1 Mathematical Preliminaries

For an integer $n \in \mathbb{N}$, we denote the set $\{1, \ldots, n\}$ by $[n]$. We use the notation $\mathbb{R}_{>0}$ to denote the set of all positive real numbers and $\mathbb{R}_{\geq 0}$ for the set of non-negative real numbers. The set of all finite binary strings is denoted by $\{0,1\}^*$.

Definition 2.1.1 (Minimal set cover)**.** *Let $n \in \mathbb{N}$, $\mathcal{S} \subseteq 2^{[n]}$ and $X \subseteq [n]$. A* cover *of X with respect to $\mathcal{S}$ is a set $\mathcal{T} \subseteq \mathcal{S}$ satisfying $X \subseteq \bigcup_{T \in \mathcal{T}} T$. A cover of X with respect to $\mathcal{S}$ of minimal cardinality is referred to as a* minimum cover*. We use the notation $\mathrm{minCover}_{\supseteq}(X, \mathcal{S})$ for any such cover while $\mathrm{SizeMinCov}_{\supseteq}(X, \mathcal{S})$ denotes the cardinality of a minimum cover of X with respect to $\mathcal{S}$. If only covers formed by subsets of X are taken into consideration, we use the notation $\mathrm{minCover}_{=}(X, \mathcal{S})$ and $\mathrm{SizeMinCov}_{=}(X, \mathcal{S})$.*

Note that there may exist multiple minimal set covers of X with respect to $\mathcal{S}$.

Definition 2.1.2 (Negligible function)**.** *A function $f : \mathbb{N} \longrightarrow \mathbb{R}_{\geq 0}$ is said to be* negligible *if for every integer $c \in \mathbb{N}$ there exists $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ it holds that $f(n) < n^{-c}$.*

Proposition 2.1.3 (Inequality of arithmetic and geometric means)**.** *For $k \in \mathbb{N}$, let $x_1, \ldots, x_k \in \mathbb{R}$ be non-negative. Then*

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

Lemma 2.1.4 (Bollobás Set Pairs Inequality [Bol65])**.** *Let $m \in \mathbb{N}$ and consider families of finite sets $\mathcal{X} = \{X_1, X_2, \ldots, X_m\}$ and $\mathcal{Y} = \{Y_1, Y_2, \ldots, Y_m\}$ such that $X_i \cap Y_j = \emptyset$ if and only if $i, j \in [m]$ are equal. Then*

$$\sum_{i=1}^{m} \binom{|X_i| + |Y_i|}{|X_i|}^{-1} \leq 1.$$

Lemma 2.1.5 (Bernoulli's Inequality)**.** *For all $x \geq -1$ and $r \in \mathbb{R} \setminus (0, 1)$ it holds that*

$$(1 + x)^r \geq 1 + rx.$$

2.2 Basic Cryptographic Definitions

Here we provide definitions for some of the fundamental cryptographic primitives that are used later on. We adapt them from [KL14], but can also be found in any standard cryptography textbook.

2.2.1 Public Key Encryption

Definition 2.2.1. *A public-key encryption (PKE) scheme is a tuple of efficient algorithms* $\mathsf{PKE} = (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *formed by*

- *a probabilistic key generation algorithm* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$ *that outputs a pair formed by a public key and a secret key,*
- *a probabilistic encryption algorithm that on input a public key* pk *and a message m outputs a ciphertext* $c \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, m)$*,*
- *a deterministic decryption algorithm that on input a ciphertext c and a secret key* sk *outputs a message* $m \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, c)$ *or an error value $\perp$.*

We say it is correct if for all $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$ *and all messages m it holds that*

$$\mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{PKE.Enc}(\mathsf{pk}, m)) = m.$$

Now we proceed to define security. For an adversary $\mathcal{A}$ and a PKE scheme PKE we consider the following experiment $\mathsf{Exp}^{\mathcal{A}}_{\mathsf{PKE}}(\lambda)$:

1. sample $b \in \leftarrow_{\$} \{0,1\}$,
2. sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{PKE.Gen}(1^{\lambda})$ and give 1^{λ} and pk to $\mathcal{A}$,
3. $\mathcal{A}$ is allowed to make decryption queries in which it sends a ciphertext c and receives $\mathsf{PKE.Dec}(\mathsf{sk}, c)$,
4. $\mathcal{A}$ sends a pair of messages (m_0, m_1) and receives a challenge ciphertext $c^* \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, m_b)$
5. $\mathcal{A}$ is allowed to make decryption queries in which it sends a ciphertext c with the condition that $c \neq c^*$ and receives $\mathsf{PKE.Dec}(\mathsf{sk}, c)$,
6. $\mathcal{A}$ outputs a bit b',
7. the output of the experiment $\mathsf{Exp}^{\mathcal{A}}_{\mathsf{PKE}}(\lambda)$ is 1 if $b = b'$ and 0 otherwise.

Definition 2.2.2. *We say that a PKE scheme* PKE *is IND-CCA2 secure if for all efficient adversaries* $\mathcal{A}$

$$|\Pr[\mathsf{Exp}^{\mathcal{A}}_{\mathsf{PKE}}(\lambda) = 1] - \frac{1}{2}|$$

is negligible.

2.2.2 Digital Signatures

Definition 2.2.3. *A digital signature is a tuple of efficient algorithms* $\mathsf{SIG} = (\mathsf{GenS}, \mathsf{Sign}, \mathsf{Vrfy})$ *formed by*

- *a probabilistic key generation algorithm* $(\mathsf{svk}, \mathsf{ssk}) \leftarrow \mathsf{GenS}(1^{\lambda})$ *that outputs a pair formed by a public verification key and a secret signing key,*
- *a probabilistic encryption algorithm that on input a secret signing key* ssk *and a message* m *outputs a signature* $\sigma \leftarrow \mathsf{Sign}(\mathsf{ssk}, m)$*,*
- *a deterministic verification algorithm that on input a message signature pair* (m, σ) *and a verification key* svk *outputs a boolean value* $b \leftarrow \mathsf{Vrfy}(\mathsf{svk}, m, \sigma)$*.*

We say it is correct if for all $(\mathsf{svk}, \mathsf{ssk}) \leftarrow \mathsf{GenS}(1^{\lambda})$ *and all messages* m *it holds that*

$$\mathsf{Vrfy}(\mathsf{svk}, \mathsf{Sign}(\mathsf{ssk}, m)) = 1.$$

For an adversary $\mathcal{A}$ and a signature scheme SIG we consider the following experiment $\mathsf{Exp}^{\mathcal{A}}_{\mathsf{SIG}}(\lambda)$:

1. sample $(\mathsf{svk}, \mathsf{ssk}) \leftarrow \mathsf{GenS}(1^\lambda)$ and give 1^λ and svk to $\mathcal{A}$,
2. $\mathcal{A}$ is allowed to make signing queries in which it sends a message m_i and receives $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{ssk}, m_i)$,
3. $\mathcal{A}$ outputs a message-signature pair (m^*, σ^*),
4. the output of $\mathsf{Exp}^{\mathcal{A}}_{\mathsf{SIG}}(\lambda)$ is 1 if for all queries i it holds that $(m^*, \sigma^*) \neq (m_i, \sigma_i)$ and $\mathsf{Vrfy}(\mathsf{svk}, m^*, \sigma^*) = 1$. Otherwise the output is 0.

Definition 2.2.4. *We say that a signature scheme* SIG *is SEUF-CMA secure if for all efficient adversaries* $\mathcal{A}$

$$|\Pr[\mathsf{Exp}^{\mathcal{A}}_{\mathsf{SIG}}(\lambda) = 1] - \frac{1}{2}|$$

is negligible.

2.2.3 Random Oracle Model

As is often the case, we will consider the random oracle model (ROM). Therefore, a hash function $H\colon \{0,1\}^* \longrightarrow \{0,1\}^n$ will be seen as a randomly chosen function from the set of all functions with domain $\{0,1\}^*$ and codomain $\{0,1\}^n$, and we will assume that the output of H on some input $x \in \{0,1\}^*$ can only be learned by querying an oracle on said input x.

CHAPTER 3

On the Cost of PCS in CGKA

This chapter essentially replicates the results that appear in our publication [ACPP23].

3.1 Introduction

Updating keys in a ratchet tree as illustrated in Figure 3.1 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, i.e., they 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+22] 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 3.2. 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+22] 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 earlier non-concurrent versions of TreeKEM.

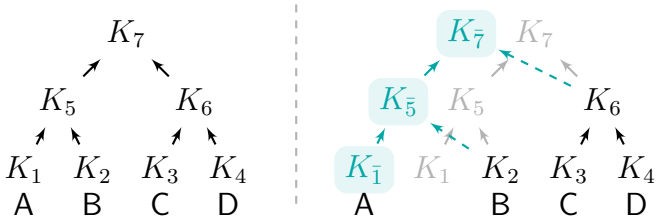

Figure 3.1: Left: Illustration of a ratchet tree with $n = 4$ users $\{A, B, C, D\}$ where each key $K_i = (pk_i, sk_i)$ is a public/secret key tuple. Right: To update and achieve PCS Alice rotates keys $\{K_1, K_5, K_7\}$ by sampling new keys $\{K_{\bar{1}}, K_{\bar{5}}, K_{\bar{7}}\}$ (blue, shaded background) and encrypting each secret key under the public key of their parent (blue, dashed arrows), e.g. $K_2 \to K_{\bar{5}}$ 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+99b], we can save the ciphertexts for the solid blue arrows and only need $\lceil\log(n)\rceil = 2$ ciphertexts.

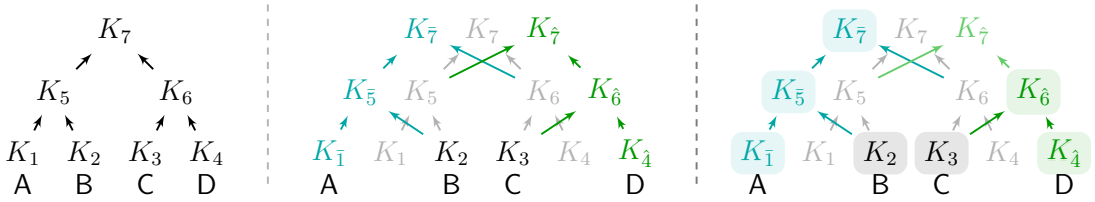

Figure 3.2: Illustration of CoCoA where Alice and Dave concurrently rotate their keys. Left: state of the ratchet tree before the updates. Middle: Keys $\{K_{\bar{1}}, K_{\bar{5}}, K_{\bar{7}}\}$ and $\{K_{\hat{4}}, K_{\hat{6}}, K_{\hat{7}}\}$ generated by Alice and Dave's updates respectively. There's 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 once all corrupted parties updated $\log(n)$ times PCS will be achieved (in particular, if Dave updates once more PCS is achieved).

In this chapter 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. 3.1 (where users are denoted $\{A, B, C, D\}$ not $\{1, 2, 3, 4\}$), the sets corresponding to the keys $K_1, .., K_7$ are

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

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. 3.1). 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 3.3.1 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 3.3.2 we use it 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 3.4.1.

3.1.1 Our Bounds.

In this chapter we prove lower bounds on the communication cost of CGKA protocols achieving PCS. Moreover, in Section 3.5 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" ($\neg\mathrm{RC}$ 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 $\neg\mathrm{RC}$ 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. Our security game is parameterized by the

[1] It is possible, as in [AAN+22, 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	$\neg\mathrm{RC}$	[BDR20]
CoCoA on $\sqrt[k-1]{n}$-ary trees	$n\,k^2\,\sqrt[k-1]{n}$	k	RC	Sec. 3.5.2
CoCoA on 2-ary trees	$n\log(n)^2$	$\log(n)$	RC	[AAN+22]
CoCoALight on $\sqrt[(k-1)/2]{n}$-ary trees	$n\,k\,\sqrt[(k-1)/2]{n}$	k	RC	Sec. 3.5.3

Lower bounds

Restrictions	Communication	Rounds	Rand. corr.	See
None	n^2	2	$\neg\mathrm{RC}$	[BDR20]
NDW, NNE, PCU^*	$n\log(n)/\log(\log(n))$	$\log(n)$	$\neg\mathrm{RC}$	Cor. 3.4.12
NDW, NNE, PCU^*	$\varepsilon\cdot n\cdot\sqrt[(1+\varepsilon)k-1]{\alpha_\varepsilon n}\cdot k/\log(k)$	k	$\neg\mathrm{RC}$	Cor. 3.4.12

Table 3.1: 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 3.1.3 below for more details. Here, $\alpha_\varepsilon \approx \varepsilon$ is some constant depending on ε.

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 provided every party who was corrupted in the past was asked to update at least k times (since their last corruption). Table 3.1 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 $n^{1/k}$.

Upper bound. We introduce in Section 3.5 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 3.5.

3.1.2 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 any key satisfying the safe predicate trivially leaked as a result of a user corruption during the execution. We do this in Theorem 3.5.1, 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+22].

3.1.3 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+24]. 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 that 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 $O(n \log(n))$.

Finally, two recent works [HKP+21, AHKM22], explore the use of multi-recipient multi-message PKE (mmPKE), which allows for 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 $\mathsf{Enc}(\mathsf{pk}_1, \mathsf{Enc}(\mathsf{pk}_2, m))$. 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.

3.2 Preliminaries

We now establish syntax for continuous group-key agreement (CGKA) schemes as needed for this chapter. 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 CGKA.Setup, CGKA.Init, CGKA.Upd, CGKA.Proc, and CGKA.Key. Algorithms CGKA.Setup and CGKA.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, CGKA.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. CGKA.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 CGKA.Upd. 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 CGKA.Key.

More formally,

- CGKA.Setup$(n; \mathsf{r})$ on input the group size n and random coins r outputs public information pub, as well as an initial state st_u for every user $u \in G = [n]$.
- CGKA.Init$(\mathsf{st}_u, \mathsf{pub}; \mathsf{r})$ receives as input a user's (initial) state, the public information pub, and random coins r. Its output $(\mathsf{st}'_u, \mathsf{MI}_u)$ consists of the initializing user's updated state and a control message MI_u.
- CGKA.Upd$(\mathsf{st}_u, \mathsf{pub}; \mathsf{r})$ 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_u^t.
- Deterministic algorithm CGKA.Proc$(\mathsf{st}_u, \mathsf{pub}, \mathsf{M})$ 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 $(\mathsf{MU}_v)_v$. Its output is the processing user's updated state st'_u.
- Deterministic algorithm CGKA.Key(st_u) on input a user's state returns u's view of current group key k_G belonging to key space $\mathcal{K}$.

When comparing to the syntax of [BDR20], one can find two differences. On the one hand, we chose not to merge algorithms CGKA.Setup and CGKA.Init, as in [BDR20], although this would be possible since we consider the simple setting where a single static

Game $\mathrm{CORRECT}^{\mathsf{CGKA}}(n, u_0, (U_t)_{t=1}^{t_{\max}})$

$t \leftarrow 0$
INIT(n, u_0)
while $t \leq t_{\max}$:
  $t \leftarrow t + 1$
  ROUND(U_t)
return 0

Oracle INIT(n, u_0)

$(\mathsf{pub}, (\mathsf{st}_u)_{u\in[n]}) \leftarrow \mathsf{CGKA.Setup}(n)$
$(\mathsf{st}_{u_0}, \mathsf{MI}^0_{u_0}) \leftarrow \mathsf{CGKA.Init}(\mathsf{st}_{u_0}, \mathsf{pub}, n; \mathsf{r})$
for $u \in [n]$:
  $\mathsf{st}_u \leftarrow \mathsf{CGKA.Proc}(\mathsf{st}_u, \mathsf{pub}, \mathsf{MI}^0_{u_0})$
  $\mathsf{k}^u_G \leftarrow \mathsf{CGKA.Key}(\mathsf{st}_u)$
if $\exists u, v \in [n] : \mathsf{k}^u_G \neq \mathsf{k}^v_G$:
  return 1

Oracle ROUND(U_t)

$\mathsf{MU} \leftarrow \emptyset$
for $u \in U_t$:
  $(\mathsf{st}_u, \mathsf{MU}^t_u) \leftarrow \mathsf{Update}(\mathsf{st}_u, \mathsf{pub})$
  $\mathsf{MU} \leftarrow_\cup \mathsf{MU}^t_u$
for $u \in [n]$:
  $\mathsf{st}_u \leftarrow \mathsf{CGKA.Proc}(\mathsf{st}_u, \mathsf{pub}, \mathsf{MU})$
  $\mathsf{k}^u_G \leftarrow \mathsf{CGKA.Key}(\mathsf{st}_u)$
if $\exists u, v \in [n] : \mathsf{k}^u_G \neq \mathsf{k}^v_G$: \\ disagreement on group key
  return 1

Figure 3.3: Correctness game for continuous group-key agreement scheme CGKA.

group is created. On the other hand, we include the algorithm CGKA.Key, 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 3.3. 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-$\mathrm{PCS}_{\mathrm{mode}}$), and one-wayness ($\mathrm{OW}$-$k$-$\mathrm{PCS}_{\mathrm{mode}}$). Here, $\mathrm{mode} \in \{\neg\mathrm{RC}, \mathrm{RC}\}$ 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-$\mathrm{PCS}_{\neg\mathrm{RC}}$,

is used for our lower bounds in Section 3.3.2 and the strongest, $\mathrm{IND}\text{-}k\text{-}\mathrm{PCS}_{\mathrm{RC}}$, for our new upper bound of Section 3.5.

The security games are formally defined in Figure 3.4. 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 CORR(u) to reveal user u's current internal state st_u, and, in the case that $\mathrm{mode} = \mathrm{RC}$, 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 $\mathrm{IND}\text{-}k\text{-}\mathrm{PCS}^{\mathsf{CGKA}}_{\mathrm{mode}}(\mathsf{A})$, 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 $\mathrm{safe}\text{-}k\text{-}\mathrm{PCS}$ holds. In Game $\mathrm{OW}\text{-}k\text{-}\mathrm{PCS}^{\mathsf{CGKA}}_{\mathrm{mode}}(\mathsf{A})$, 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 $\mathrm{safe}\text{-}k\text{-}\mathrm{PCS}$ holds. The predicate $\mathrm{safe}\text{-}k\text{-}\mathrm{PCS}$ 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 3.2.1 (k-PCS security)**.** *Let* CGKA *be a continuous group-key agreement scheme,* $k \in \mathbb{N}$*, and* $\mathrm{mode} \in \{\mathrm{RC}, \neg\mathrm{RC}\}$*. Then* CGKA *is* $\mathrm{IND}\text{-}k\text{-}\mathrm{PCS}_{\mathrm{mode}}$ secure*, if for every PPT adversary* A *the advantage function*

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

is negligible.

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

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

is negligible.

Remark 3.2.2. *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 35 *and* 36 *of the safe predicate, which disallow corrupting users after the challenged round* t^**.*

Game $\text{IND-}k\text{-PCS}^{\mathsf{CGKA}}_{\text{mode}}(\mathsf{A})$

00 $b^* \leftarrow_\$ \{0,1\}$
01 $b \leftarrow \mathsf{A}^{\mathsf{INIT},\mathsf{ROUND},\mathsf{CORR},\mathsf{CHALL}}$
02 return $[b = b^*] \wedge \text{safe}_{k\text{-PCS}}$

Game $\text{OW-}k\text{-PCS}^{\mathsf{CGKA}}_{\text{mode}}(\mathsf{A})$

03 $\mathsf{k} \leftarrow \mathsf{A}^{\mathsf{INIT},\mathsf{ROUND},\mathsf{CORR},\mathsf{CHALL}}$
04 return $[\mathsf{k}^* = \mathsf{k}] \wedge \text{safe}_{k\text{-PCS}}$

Oracle $\mathsf{INIT}(n, u_0)$ \\ one call; called first

05 $t \leftarrow 0$
06 $\text{Cor}[u] \leftarrow \emptyset$, $\text{Upd}[u] \leftarrow \emptyset$ for all $u \in [n]$
07 $\text{Coins}[u,t] \leftarrow \emptyset$ for all $u \in [n]$, $\forall t$
08 $\mathsf{r}_{\text{setup}} \leftarrow_\$ \mathcal{R}$
09 $(\mathsf{pub}, (\mathsf{st}_u)_{u\in[n]}) \leftarrow \mathsf{CGKA.Setup}(n; \mathsf{r}_{\text{setup}})$
10 $\mathsf{r} \leftarrow_\$ \mathcal{R}$; $\text{Coins}[u_0, t] \leftarrow_\cup \{\mathsf{r}\}$
11 $(\mathsf{st}_{u_0}, \mathsf{MI}^0_{u_0}) \leftarrow \mathsf{CGKA.Init}(\mathsf{st}_{u_0}, \mathsf{pub}, n; \mathsf{r})$
12 for $u \in [n]$:
13 $\quad \mathsf{st}_u \leftarrow \mathsf{CGKA.Proc}(\mathsf{st}_u, \mathsf{pub}, \mathsf{MI}^0_{u_0})$
14 $\quad \mathsf{k}_G \leftarrow \mathsf{CGKA.Key}(\mathsf{st}_u)$
15 return $\mathsf{pub}, \mathsf{MI}^0_{u_0}$

Oracle CHALL
\\ one call, Game $\text{IND-}k\text{-PCS}^{\mathsf{CGKA}}_{\text{mode}}(\mathsf{A})$

16 $t^* \leftarrow t$
17 $\mathsf{k}_0 \leftarrow_\$ \mathcal{K}$; $\mathsf{k}_1 \leftarrow \mathsf{k}_G$
18 return k_{b^*}

Oracle CHALL
\\ one call, Game $\text{OW-}k\text{-PCS}^{\mathsf{CGKA}}_{\text{mode}}(\mathsf{A})$

19 $t^* \leftarrow t$
20 $\mathsf{k}^* \leftarrow \mathsf{k}_G$

Oracle $\mathsf{ROUND}(U_t)$

21 $t \leftarrow t + 1$
22 $\mathsf{MU} \leftarrow \emptyset$
23 for $u \in U_t$:
24 $\quad$ require $u \in [n]$
25 $\quad \text{Upd}[u] \leftarrow_\cup \{t\}$
26 $\quad \mathsf{r} \leftarrow_\$ \mathcal{R}$; $\text{Coins}[u,t] \leftarrow_\cup \{\mathsf{r}\}$
27 $\quad (\mathsf{st}_u, \mathsf{MU}^t_u) \leftarrow \mathsf{Update}(\mathsf{st}_u, \mathsf{pub}; \mathsf{r})$
28 $\quad \mathsf{MU} \leftarrow_\cup \mathsf{MU}^t_u$
29 for $u \in [n]$:
30 $\quad \mathsf{st}_u \leftarrow \mathsf{CGKA.Proc}(\mathsf{st}_u, \mathsf{pub}, \mathsf{MU})$
31 $\quad \mathsf{k}_G \leftarrow \mathsf{CGKA.Key}(\mathsf{st}_u)$
32 return MU

Predicate $\text{safe}_{k\text{-PCS}}$

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

Oracle $\mathsf{CORR}(u)$

41 $\text{Cor}[u] \leftarrow_\cup \{t\}$
42 return $(\mathsf{st}_u, \text{Coins}[u,t])$ \\ if mode = RC
43 return st_u \\ if mode = ¬RC

Figure 3.4: Security games $\text{IND-}k\text{-PCS}_{\text{mode}}$ for indistinguishability and $\text{OW-}k\text{-PCS}_{\text{mode}}$ for one-wayness of group keys with respect to mode of randomness-corruption $\text{mode} \in \{\text{RC}, \neg\text{RC}\}$. 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.

(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 3.5, 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+22, 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 $\mathsf{Enc}(\mathsf{pk}_1, \mathsf{Enc}(\mathsf{pk}_2, m))$.
- CGKA does not distribute work (NDW). This means that, if in any round a user gets 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 3.4.1 (Def. 3.4.2 and Def. 3.4.4), 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 $|\mathsf{MU}_u|$ of an update that a user u would produce if asked to 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 3.2.3 (Publicly-computable update cost)**.** *Let* CGKA *be a CGKA scheme. Consider an execution of game* IND-k-$\mathrm{PCS}_{\mathrm{mode}}$ *(or* OW-k-$\mathrm{PCS}_{\mathrm{mode}}$*). 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_u^t *and public information* pub*, it is possible to efficiently compute* $|\mathsf{MU}|$*, where* $(\mathsf{st}', \mathsf{MU}) \leftarrow \mathsf{Update}(\mathsf{st}_u^t, \mathsf{pub}; r)$ *would be the output of calling the update procedure, from public information at the end of round* $t-1$ *(i.e., all messages* $\mathsf{MI}_{u_0}^0$ *and* $\mathsf{MU}_u^{t'}$ *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* $\mathsf{MI}^0_{u_0}$*, the sequence* $(U_{t'})_{t' \leq t-1}$*,* n*,* k*,* pub *and* u_0*.*

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.

3.3 Results 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 3.3.1, the bound in Section 3.3.2.

3.3.1 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. This connection will be made formal later in Section 3.4.1. The interaction proceeds in rounds, during which users schedule update 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.

The 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

$$u \in U_{t_i} \text{ for all } i \in \{1, \ldots, k\} \quad \text{and} \tag{3.1}$$

$$\max\{t_c \in \{0, \ldots, t_{\max}\} : u \in C_{t_c}\} < t_1 < \cdots < t_k \leq t \ . \tag{3.2}$$

Recall that since we only want to argue about post-compromise security but not forward-secrecy the condition also excludes the corruption of users *after* round t. This has the effect of requiring fewer rounds to be safe and therefore it only strengthens our lower bounds.

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 internal state. [2] Here 'secure key' refers to keys that were established by update operations and cannot be trivially recovered by 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 the key in plain, if it contains an encryption of the key under a key the adversary has access to, or if the key is the result of a PRF evaluation under a key the adversary has access to. In every round the sets $S(\mathsf{sk})$ 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 3.5 (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

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

[2] More precisely, in our translation to the symbolic model in Section 3.4.1 we will use the set of users that at some point between the last time they were corrupted and round t had access to the key.

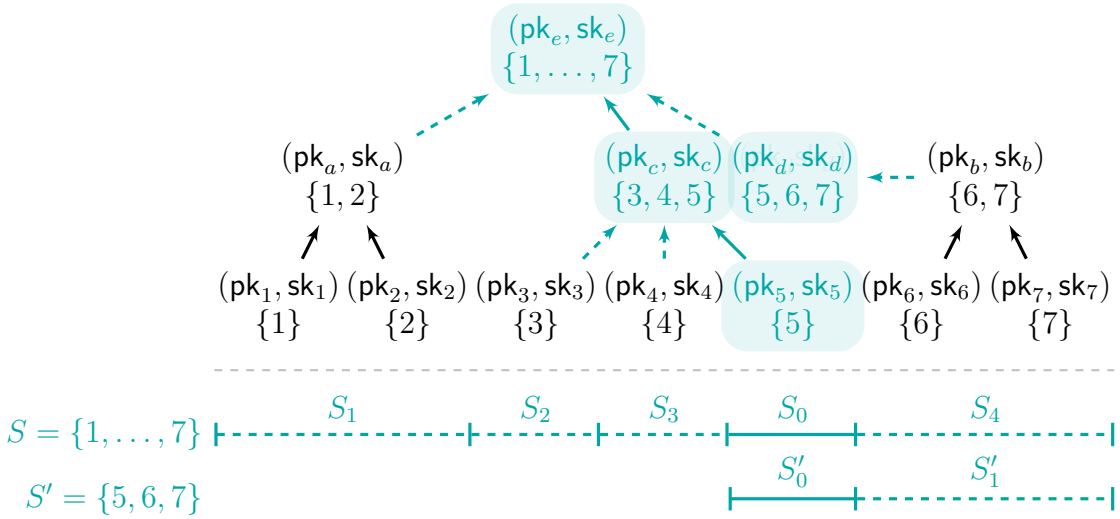

Figure 3.5: Illustration of a ratchet tree and its associated set system $\mathcal{S}_t$. Vertices contain key-pairs (above) and the associated set (below). Keys already present in the system at time $t-1$ are depicted in black and keys added by user 5 in round t in blue with shaded background. Edges indicate that knowledge secret key of the source implies knowledge of the one of the sink. Dashed, blue edges correspond to ciphertexts $\mathsf{Enc}_{\mathsf{pk}_{\text{source}}}(\mathsf{sk}_{\text{sink}})$ 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 $(\mathsf{pk}_5, \mathsf{sk}_5)$ and $(\mathsf{pk}_d, \mathsf{sk}_d)$ using fresh randomness, and $(\mathsf{pk}_c, \mathsf{sk}_c)$ and $(\mathsf{pk}_e, \mathsf{sk}_e)$ using a PRF. Bottom: Depiction of the sets required to exist by property (iii) using the examples $S = \{1, \ldots, 7\} = \bigcup_{i=0}^{k} S_i$ 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).

Its output is an integer $\mathrm{Cost}(u, t, M_t)$. For better legibility, we will simply write $\mathrm{Cost}(u, t)$ 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, users are not aware whether they are corrupted or not. However, if asked to update by the adversary, they may decide to create particular ciphertexts depending on the history of operations performed so far, as these may have impacted 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 group members share a common key. Further, by security, whenever a round is safe, the corresponding shared key 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 under a public key pk′ that must have either been present in the system at the end of round $t-1$ or been previously generated by 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 needs not 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

$$S_0 = \{u\} \text{ for some } u \in U_t \quad \text{or} \quad S_0 \in \mathcal{S}_{t-1} \ , \tag{3.3}$$

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

$$S \cap C_{\leq t} \subseteq \bigcup_{i=0}^{h} S_i \tag{3.4}$$

where $C_{\leq t} = \bigcup_{0 \leq t' \leq t} C_{t'}$ are the users that have been corrupted at least once in or before round t. And, regarding the cost function, we require that

$$\exists u \in S_0 \cap U_t \text{ such that } \mathrm{Cost}(u, t) \geq h \ . \tag{3.5}$$

Note that if in Equation 3.3 we have $S_0 = \{u\}$ then the user in Equation 3.5 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 3.5, then by Equations 3.3, 3.4, and 3.5 it holds that

$$\mathrm{Cost}(u, t) \geq |\mathrm{minCover}_{\supseteq}(S \cap C_{\leq t}, \mathcal{S}_{t-1} \cup \{u\})| - 1 \ . \tag{3.6}$$

The precise connection between the combinatorial model and the symbolic model is established later in Section 3.4.1. 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 3.2 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 $\mathrm{Cost}(u, t)$.

3.3.2 Lower Bound in the Combinatorial Model

We now give 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 3.3.1. *Let $n, k, t_c, m \in \mathbb{N}$, $C \subseteq [n]$, 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

$$\prod_{t=t_{\min}}^{m} s_t / s_{t-1} = s_m / s_{t_{\min}} = c \ . \tag{3.7}$$

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 $S \cap C \subseteq S_0 \cup \bigcup_{i=1}^{h} S_i$ 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 2.1.3). 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 3.7 that $c \leq \prod_{t=1}^{m'} x_t \leq (x/m')^{m'}$. Solving for x gives

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

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 $m - 1 \leq \ln(c)$. □

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 3.3.2. *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 $3 \leq k \leq \ln(\alpha_\varepsilon n)$, 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*

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

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

$$\sum_{u \in U_{t_c+t}} \mathrm{Cost}(u, t_c+t) \geq \max_{u \in U_{t_c+t}} \mathrm{Cost}(u, t_c+t) \cdot \frac{1}{2} \left\lfloor \frac{\varepsilon n'}{2\lceil \log(k) \rceil} \right\rfloor . \tag{3.8}$$

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 $U_{k\text{-Update}}$ 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 $C = U_{k\text{-Update}}$, then the round $t_{\max} = t_c + (1+\varepsilon)k$ must be safe as defined in Equation 3.1 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 3.3.1 for $m = (1+\varepsilon)k$ and $c = \lceil n\alpha_\varepsilon\rceil$ and obtain

$$\begin{aligned}\sum_{t=1}^{(1+\varepsilon)k} \max_{u\in U_{t_c+t}} \mathrm{Cost}(u, t_c+t) &\geq ((1+\varepsilon)k-1)\left(\sqrt[(1+\varepsilon)k-1]{\alpha_\varepsilon n}-1\right) \\ &\geq (k-1)\left(\sqrt[(1+\varepsilon)k-1]{\alpha_\varepsilon n}-1\right)\ .\end{aligned} \tag{3.9}$$

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 $\max_{u\in U_{t_c+t}} c_u^t < \delta = (k-1)/(2(1+\varepsilon)k)\cdot\left(\sqrt[(1+\varepsilon)k-1]{\alpha_\varepsilon n}-1\right)$. Thus, if we denote the set of such rounds in $\{t_c+1,\ldots,t_{\max}\}$ by $T_{(b)}$ and the set of rounds in which case (c) occurs by $T_{(c)}$, then we obtain $\sum_{t\in T_{(b)}} \max_{u\in U_{t_c+t}} c_u^t \leq (k-1)/2\cdot\left(\sqrt[(1+\varepsilon)k-1]{\alpha_\varepsilon n}-1\right)$. Plugging this in Equation 3.9 yields $\sum_{t\in T_{(c)}} \max_{u\in U_{t_c+t}} c_u^t \geq (k-1)/2\cdot\left(\sqrt[(1+\varepsilon)k-1]{\alpha_\varepsilon n}-1\right)$. This by Equation 3.8 implies the theorem's statement, as

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

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 (3.8) exists, and (B) that $|U_{k\text{-Update}}| \geq \lceil\alpha_\varepsilon n\rceil$.

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., $c_1^t \leq c_2^t \leq \cdots \leq c_n^t$. 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 3.8. For an illustration of this see Figure 3.6.

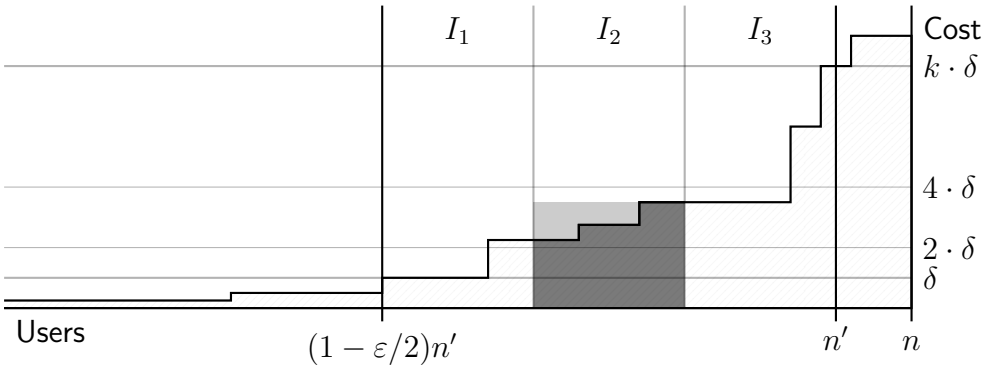

Figure 3.6: Existence of U_t satisfying the property of case (c) for $k = 8$. The interval $[(1-\varepsilon/2)n', n']$, of users with update cost between δ and $k \cdot \delta$ is split into $\log(k)$ subintervals I_1, I_2, I_3 of length $n'\varepsilon/(2\log(k))$. While within the left and right subintervals 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).

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

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

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

$$\begin{aligned}(1+\varepsilon)k\alpha n + (k-1)\cdot(1-\alpha)n &\geq (1+\varepsilon)\left\lceil(1-\varepsilon/2)n'\right\rceil k \\ &\geq (1+\varepsilon)(1-\varepsilon/2)(1-\varepsilon/(4\log(k)))nk \ ,\end{aligned}$$

which by solving for α yields

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

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 3.3.2 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 3.3.2 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 3.3.3. *Let k, n, t_c, ε, and α_ε be as in Theorem 3.3.2. 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 $0\leq z<z_{\max}$.

3.4 Results in the Symbolic Model

In this section we 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 3.3.1, which allows us to conclude that the bounds derived in the combinatorial model also hold with respect to the symbolic model.

3.4.1 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) A pseudorandom function PRF takes as input a key $\mathtt{k}$ and a message $\mathtt{m}$ and returns a key $\mathtt{k}' = \mathsf{PRF}(\mathtt{k}, \mathtt{m})$.

(ii) A public-key encryption scheme consists of a tuple of algorithms $(\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$, where PKE.Gen on input of secret key $\mathtt{sk}$ returns a public key $\mathtt{pk}$. PKE.Enc takes as input a public key $\mathtt{pk}$ and a message $\mathtt{m}$, and outputs a ciphertext $\mathtt{c}$ with message data type. $\mathsf{PKE.Dec}(\mathtt{sk}, \mathtt{c})$ takes as input a secret key $\mathtt{sk}$ and a ciphertext $\mathtt{c}$, and outputs a message $\mathtt{m}$. 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 allows dual PRFs see Remark 3.4.11 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 3.2. 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 3.3 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 $(\mathtt{ST}'_u, \mathtt{MU}_u) \leftarrow \mathsf{CGKA.Upd}(\mathtt{ST}_u, \mathtt{pub}; \mathtt{r}, r)$, then we require that $\mathtt{ST}'_u, \mathtt{MU}_u \in \mathsf{Der}(\{\mathtt{ST}_u, \mathtt{pub}, \mathtt{r}\})$, and similarly if $\mathtt{ST}'_u \leftarrow \mathsf{CGKA.Proc}(\mathtt{ST}_u, \mathtt{pub}, \mathtt{M})$, 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-$\mathrm{PCS}_{\neg\mathrm{RC}}$ of Definition 3.2.1. As 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 $t \in \{1, \ldots, t_{\max}\}$, if it was either the t-th query to ROUND, or, for calls to CHALL or CORR, if it was made after the t-th 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-$\mathrm{PCS}_{\neg\mathrm{RC}}$ security game in the symbolic model can be found in Figure 3.8.

Definition 3.4.1 (Symbolic k-PCS security)**.** *Let* CGKA *be a continuous group-key agreement scheme,* $k \in \mathbb{N}$*. Then* CGKA *is* OW-k-$\mathrm{PCS}_{\neg\mathrm{RC}}$ secure*, if for all* $(n, u_0, C_0, (U_t, C_t)_{t=1}^{t_{\max}}, t^*)$ *it holds that*

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

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

This notion of security, in which for any sequence $(n, u_0, C_0, (U_t, C_t)_{t=1}^{t_{\max}}, t^*)$ the game is lost, 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 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 3.4.11 after the proof of this section's main result.

Definition 3.4.2 (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 3.3.1 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 $\mathrm{OW}\text{-}k\text{-}\mathrm{PCS}_{\neg\mathrm{RC}}$. 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 $\mathrm{OW}\text{-}k\text{-}\mathrm{PCS}_{\neg\mathrm{RC}}$, 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 in round t associate to them the set of users which had access to $\mathtt{s}$ at some point between their last corruption and round t.

Definition 3.4.3 (Useful secrets and associated sets)**.** *Consider adversary* A *playing game* $\mathrm{OW}\text{-}k\text{-}\mathrm{PCS}_{\neg\mathrm{RC}}$ *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* $\mathtt{s} \notin \mathsf{Der}(\{\mathtt{M}_t, \mathtt{COR}_t\})$*. We define the associated set of a secret* $\mathtt{s}$ *in round* t *as*

$$S(\mathtt{s}, t) := \{u \in [n] \mid \mathtt{s} \in \mathsf{Der}(\mathtt{ST}_u^{t_{c,u}}, (\mathtt{R}_u^{t'})_{t_{c,u}+1 \leq t' \leq t}, \mathtt{M}_t)\} \subseteq [n]$$

where $t_{c,u} := \max\{\tilde{t} \mid u \in C_{\tilde{t}} \text{ and } \tilde{t} \leq t\}$ *(*$t_{c,u} = -1$ *if* u *has never been corrupted). We define the associated set of* $\mathtt{s}$ *after the setup as*

$$S(\mathtt{s}, -1) := \{u \in [n] \mid \mathtt{s} \in \mathsf{Der}(\mathtt{ST}_u^{-1}, \mathtt{pub})\} \subseteq [n] \ .$$

We define the set system *in round* t *as*

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

The intuition behind the definition of $S(\mathtt{s}, t)$ is that any user who can derive the secret $\mathtt{s}$ in a round t' such that $t_{c,u} + 1 \leq t' \leq t$ (i.e., $\mathtt{s} \in \mathsf{Der}(\mathtt{ST}_u^{t'}, \mathtt{M}_{t'})$) should belong to the set $S(\mathtt{s}, t)$. This is indeed the case since $\mathtt{ST}_u^{t'} \subseteq \mathsf{Der}(\mathtt{ST}_u^{t'-1}, \mathtt{R}_u^{t'}, \mathtt{M}_{t'})$ because symbolic outputs of algorithms can always be derived symbolically from their symbolic inputs.

The definition of $t_{c,u}$ depends on the user u and the time t, but not the secret $\mathtt{s}$. It is usually clear from context what round t we are considering, but in some cases we may write $t_{c,u,t}$ to make it explicit. Observe that for any $t \geq 0$, $t_{c,u} \leq t$. The following property holds; if $\mathtt{s}$ is useful at time t, then for every $u \in S(\mathtt{s}, t)$ it must be the case that $t_{c,u} < t$ (if for some $u \in S(\mathtt{s}, t)$ we have $t_{c,u} = t$, then $\mathtt{s} \in \mathsf{Der}(\mathtt{ST}_u^t, \mathtt{M}_t) \subseteq \mathsf{Der}(\mathtt{M}_t, \mathtt{COR}_t)$ and we get a contradiction with the definition of $\mathtt{s}$ being useful in round t). The notation $\mathtt{ST}_u^{-1}$ refers to the state that u is assigned by the CGKA.Setup algorithm.

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, 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 3.7 for an illustration of a scheme that *does* make use of distributed work.

Definition 3.4.4 (No distributed work)**.** *Consider a scheme* CGKA *and an execution of game* $\mathrm{OW}\text{-}k\text{-}\mathrm{PCS}_{\neg\mathrm{RC}}$ *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* $\mathtt{s}$*, 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,

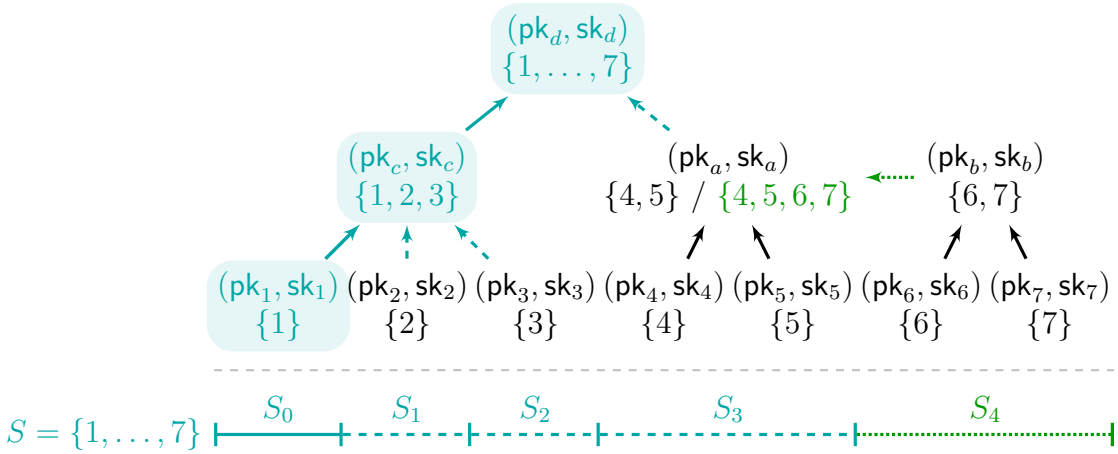

Figure 3.7: Top: Example of a ratchet tree and its associated set system $\mathcal{S}_t$ making use of distributed work. Vertices contain key-pairs (above) and the associated set (below). Edges indicate that knowledge of the secret key of the source implies knowledge of the one of the sink. Dashed edges correspond to ciphertexts $\mathsf{Enc}_{\mathsf{pk}_{\text{source}}}(\mathsf{sk}_{\text{sink}})$ 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 $(\mathsf{pk}_a, \mathsf{sk}_a)$ 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 Theorem 3.4.9 using the example $S = \{1,\dots,7\} = \bigcup_{i=0}^{k} S_i$ 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 Theorem 3.4.9; (iii)).

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 $S \in \mathcal{S}_t$ 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 from which it can be derived 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 $\mathcal{S}_{t-1}$ 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 3.5 from the combinatorial model.

Lemma 3.4.5. *If $\mathtt{s}$ is useful at time $t \geq 0$ and $u \in S(\mathtt{s}, t)$, one of the following statements is true:*

1. *there exist a secret $\mathtt{s}'$ and a message ad such that $\mathsf{PRF}(\mathtt{s}', ad) = \mathtt{s}$, $\mathtt{s}'$ is useful at time t and $\mathtt{s}' \in \mathsf{Der}(\mathtt{ST}_u^{t_{c,u}}, (\mathtt{R}_u^{t'})_{t_{c,u}+1 \leq t' \leq t}, \mathtt{M}_t)$,*
2. *$\mathtt{s} \in \mathtt{ST}_u^{t_{c,u}}$,*
3. *$\mathtt{s} \in \mathtt{r}_u^{t'}$ for some $t_{c,u} + 1 \leq t' \leq t$,*
4. *there exists $\mathtt{c} = \mathsf{PKE.Enc}(\mathtt{pk}, \mathtt{s}) \in \mathtt{ST}_u^{t_{c,u}} \cup (\mathtt{r}_u^{t'})_{t_{c,u}+1 \leq t' \leq t} \cup \mathtt{M}_t$ such that $\mathtt{sk} \in \mathsf{Der}(\mathtt{ST}_u^{t_{c,u}}, (\mathtt{r}_u^{t'})_{t_{c,u}+1 \leq t' \leq t}, \mathtt{M}_t)$ and $\mathtt{pk} = \mathsf{PKE.Gen}(\mathtt{sk})$.*

Observe that case 2 is only possible if $t_{c,u} = -1$, and case 4 does not consider the possibility of nested encryption as we assume that our scheme does not use it.

Proof. If $\mathtt{s}$ is a PRF image, i.e., there exist a secret $\mathtt{s}'$ and a message ad such that $\mathsf{PRF}(\mathtt{s}', ad) = \mathtt{s}$, $\mathtt{s}'$ must be useful at time t and we have two possibilities depending on whether $\mathtt{s}' \in \mathsf{Der}(\mathtt{ST}_u^{t_{c,u}}, (\mathtt{r}_u^{t'})_{t_{c,u}+1 \leq t' \leq t}, \mathtt{M}_t)$. If so, we are in the first case. Else, either $\mathsf{PRF}(\mathtt{s}', ad) = \mathtt{s} \in \mathtt{ST}_u^{t_{c,u}} \cup (\mathtt{r}_u^{t'})_{t_{c,u}+1 \leq t' \leq t} \cup \mathtt{M}_t$ or there exist a ciphertext with the properties stated in case 4. By assumption $\mathtt{s}$ is useful at time t, so the case when $\mathtt{s} \in \mathtt{ST}_u^{t_{c,u}} \cup (\mathtt{r}_u^{t'})_{t_{c,u}+1 \leq t' \leq t} \cup \mathtt{M}_t$ reduces to only two options, namely, $\mathtt{s} \in \mathtt{ST}_u^{t_{c,u}}$ or $\mathtt{s} \in \mathtt{r}_u^{t'}$ for some $t_{c,u} + 1 \leq t' \leq t$.

If $\mathtt{s}$ is not a PRF image, we get a similar case division. Indeed, either $\mathtt{s} \in \mathtt{ST}_u^{t_{c,u}} \cup (\mathtt{r}_u^{t'})_{t_{c,u}+1 \leq t' \leq t} \cup \mathtt{M}_t$ or there exists a ciphertext with the properties stated in case 4.

The observation that case 2 is only possible if $t_{c,u} = -1$ follows from the fact that $\mathtt{ST}_u^{t_{c,u}} \in \mathtt{COR}_{t_{c,u}} \subseteq \mathtt{COR}_t$ when $t_{c,u} \geq 0$. □

We now proceed to study case 4 in greater detail.

Lemma 3.4.6. *If* $\mathtt{s}$ *is useful at time* $t \geq 0$ *and* $u \in S(\mathtt{s}, t)$ *and there exists* $\mathtt{c} = \mathsf{PKE.Enc}(\mathtt{pk}, \mathtt{s}) \in \mathtt{ST}_u^{t_{c,u}} \cup (\mathtt{r}_u^{t'})_{t_{c,u}+1 \leq t' \leq t} \cup \mathtt{M}_t$ *such that* $\mathtt{sk} \in \mathsf{Der}(\mathtt{ST}_u^{t_{c,u}}, (\mathtt{r}_u^{t'})_{t_{c,u}+1 \leq t' \leq t}, \mathtt{M}_t)$ *and* $\mathtt{pk} = \mathsf{PKE.Gen}(\mathtt{sk})$*, one of the following is true:*

4.1. $\mathtt{c} \in \mathtt{M}_t$ *and* $\mathtt{sk}$ *is useful at time* t*,*

4.2. $t_{c,u} \geq 0$*,* $\mathtt{c} \in \mathtt{ST}_u^{t_{c,u}} \setminus \mathtt{M}_t$ *and* $\mathtt{sk}$ *is useful at time* t*,*

4.3. $t_{c,u} = -1$*,* $\mathtt{c} \in \mathtt{ST}_u^{-1} \setminus \mathtt{M}_t$ *and* $\mathtt{s} \in \mathsf{Der}(\mathtt{r}_{setup})$*.*

Observe that in case 4.3 $\mathtt{sk}$ *is not guaranteed to be useful at time* t*.*

Proof. By assumption $\mathtt{c} \in \mathtt{ST}_u^{t_{c,u}} \cup (\mathtt{r}_u^{t'})_{t_{c,u}+1 \leq t' \leq t} \cup \mathtt{M}_t$ and $\mathtt{c}$ is not of type randomness, so it must be the case that either $\mathtt{c} \in \mathtt{M}_t$ or $\mathtt{c} \in \mathtt{ST}_u^{t_{c,u}} \setminus \mathtt{M}_t$.

If $\mathtt{c} \in \mathtt{M}_t$, the fact that $\mathtt{s}$ is useful at time t implies that $\mathtt{sk}$ is useful at time t. This corresponds to case 4.1.

If $\mathtt{c} \in \mathtt{ST}_u^{t_{c,u}} \setminus \mathtt{M}_t$ and $t_{c,u} \geq 0$, then $\mathtt{c} \in \mathtt{COR}_{t_{c,u}} \subseteq \mathtt{COR}_t$. The fact that $\mathtt{s}$ is useful at time t implies that so must $\mathtt{sk}$ be. This is case 4.2.

If $\mathtt{c} \in \mathtt{ST}_u^{t_{c,u}} \setminus \mathtt{M}_t$ and $t_{c,u} = -1$, then $\mathtt{c} = \mathsf{PKE.Enc}(\mathtt{pk}, \mathtt{s}) \in \mathtt{ST}_u^{-1} \subseteq \mathsf{Der}(\mathtt{r}_{\mathrm{setup}})$ because symbolic outputs of algorithms can always be derived symbolically from their symbolic inputs. Therefore we have to consider three cases: $\mathtt{pk}, \mathtt{s} \in \mathsf{Der}(\mathtt{r}_{\mathrm{setup}})$ or $\mathtt{c} \in \mathtt{r}_{\mathrm{setup}}$ or there exists an encryption of $\mathtt{c}$ in $\mathtt{r}_{\mathrm{setup}}$. Only the first of the three options is possible as $\mathtt{c}$ is not of type randomness. □

We introduce some notation:

$$\mathsf{PRF}^{-1}(\mathtt{s}) := \{\mathtt{s}' \mid \exists k \in \mathbb{N}_{\geq 0} \exists ad_1, \ldots, ad_k \colon \mathtt{s} = \mathsf{PRF}(\cdot, ad_k) \circ \ldots \circ \mathsf{PRF}(\cdot, ad_1)(\mathtt{s}')\}$$

Cases 1 in Lemma 3.4.5 and 4.1 and 4.2 in Lemma 3.4.6 yield a new useful secret $\mathtt{sk}$ that satisfies the same conditions as $\mathtt{s}$ in Lemma 3.4.5. This makes it possible to repeatedly apply these two results until we reach cases 2 or 3 or 4.3. This is captured in the following result.

Corollary 3.4.7. *If* $\mathtt{s}$ *is useful at time* $t \geq 0$ *and* $u \in S(\mathtt{s}, t)$*, there exists a sequence* $\{\mathtt{s}_{1,u}, \ldots, \mathtt{s}_{\ell_u,u}\}$ *such that*

(i) every secret $\mathtt{s}_{i,u}$ *is useful at time* t *and* $\mathtt{s}_{i,u} \in \mathsf{Der}(\mathtt{ST}_u^{t_{c,u}}, (\mathtt{r}_u^{t'})_{t_{c,u}+1 \leq t' \leq t}, \mathtt{M}_t)$*,*

(ii) $\mathtt{s}_{\ell_u,u} \in \mathsf{PRF}^{-1}(\mathtt{s})$*,*

(iii) if $t_{c,u} \geq 0$*,* $\mathtt{s}_{1,u} \in \mathtt{R}_u^{t'}$ *for some* $t_{c,u} + 1 \leq t' \leq t$*,*

(iv) *if* $t_{c,u} = -1$, $\mathtt{s}_{1,u}$ *satisfies that* $\mathtt{s}_{1,u} \in \mathtt{ST}_u^{t_{c,u}}$ *or* $\mathtt{s}_{1,u} \in \mathtt{R}_u^{t'}$ *for some* $t_{c,u}+1 \leq t' \leq t$ *or* $\mathtt{s}_{1,u} \in \mathsf{Der}(\mathtt{r}_{setup})$,

(v) *for every* $i \in \{1, \ldots, \ell_u - 1\}$, *there exist* $\mathtt{s}'_{i,u}$ *such that* $\mathtt{s}_{i,u} \in \mathsf{PRF}^{-1}(\mathtt{s}'_{i,u})$ *and a ciphertext* $\mathtt{c}_{i,u} = \mathsf{PKE.Enc}(\mathtt{pk}'_{i,u}, \mathtt{s}_{i+1,u}) \in \mathtt{ST}_u^{t_{c,u}} \cup \mathtt{M}_t$ *where* $\mathtt{pk}'_{i,u} = \mathsf{PKE.Gen}(\mathtt{s}'_{i,u})$,

(vi) *for any* t^* *such that* $t_{c,u,t} \leq t^* \leq t$ *we have that if* $\mathtt{c}_{i,u} \in \mathtt{ST}_u^{t_{c,u,t}} \cup \mathtt{M}_{t^*}$ *and* $u \in S(\mathtt{s}_{i,u}, t^*)$ *then* $u \in S(\mathtt{s}_{i+1,u}, t^*)$,

(vii) *If* $\mathtt{s}_{i,u} \in \mathsf{Der}(\mathtt{R}_u^{t'})$ *for some* $t_{c,u} < t' \leq t$, *then* $\mathtt{s}_{i,u} \in \mathtt{R}_u^{t'}$ *and* $i = 1$. *Moreover if* $t_{c,u} \geq 0$, *for every* $i \geq 2$ *it holds that* $\mathtt{s}_{i,u} \notin \mathtt{ST}_u^{t_{c,u}} \cup \bigcup_{t_{c,u}+1 \leq t' \leq t} \mathsf{Der}(\mathtt{R}_u^{t'})$.

Proof. We set $X = \emptyset$ and $\mathtt{x} = \mathtt{s}$. Now we start a recursive process by considering the following case division that follows from Lemma 3.4.5 and Lemma 3.4.6 for secret $\mathtt{x}$:

- If we are in cases 2 or 3, we do $X \leftarrow \{\mathtt{x}\} \cup X$ and stop.
- If we are in case 1, the sequence X remains unchanged and repeat the process with $\mathtt{x} \leftarrow \mathtt{s}'$.
- If we are in cases 4.1 or 4.2 of Lemma 3.4.6, we do $X \leftarrow \{\mathtt{x}\} \cup X$ and repeat the process with $\mathtt{x} \leftarrow \mathtt{sk}$.
- If we are in case 4.3, we do $X \leftarrow \{\mathtt{x}\} \cup X$ and stop.

Procedure always stops as argued in the paragraph above Corollary 3.4.7. The first five properties follow directly from Lemma 3.4.5, Lemma 3.4.6 and the construction of the sequence X. Property (vi) is a consequence of (v) as we show below.

By assumption, $t_{c,u,t} \leq t^* \leq t$, therefore $t_{c,u,t} = t_{c,u,t^*}$ The fact that $u \in S(\mathtt{s}_{i,u}, t^*)$ implies that $\mathtt{s}_{i,u} \in \mathsf{Der}(\mathtt{ST}_u^{t_{c,u,t^*}}, (\mathtt{R}_u^{t'})_{t_{c,u,t^*}+1 \leq t' \leq t^*}, \mathtt{M}_{t^*})$. Since $\mathtt{c}_{i,u} \in \mathtt{ST}_u^{t_{c,u,t}} \cup \mathtt{M}_{t^*} = \mathtt{ST}_u^{t_{c,u,t^*}} \cup \mathtt{M}_{t^*}$ we get that $u \in S(\mathtt{s}_{i+1,u}, t^*)$.

In order to show property (vii) we observe that if $\mathtt{s}_{i,u} \in \mathsf{Der}(\mathtt{R}_u^{t'})$, then either there exists $\mathtt{s}'_{i,u}$ such that $\mathtt{s}'_{i,u} \in \mathsf{PRF}^{-1}(\mathtt{s}_{i,u})$ or $\mathtt{s}_{i,u} \in \mathtt{R}_u^{t'}$. In the first case $\mathtt{s}_{i,u}$ would not have been included in the sequence, so it must be the case that $\mathtt{s}_{i,u} \in \mathtt{R}_u^{t'}$, which also implies that the recursive process stops, i.e., $i = 1$. It also follows from the way the sequence is defined that $\mathtt{s}_{i,u} \notin \mathtt{ST}_u^{t_{c,u}} \cup \bigcup_{t_{c,u}+1 \leq t' \leq t} \mathtt{R}_u^{t'}$. From the combination of both properties it follows that $\mathtt{s}_{i,u} \notin \mathtt{ST}_u^{t_{c,u}} \cup \bigcup_{t_{c,u}+1 \leq t' \leq t} \mathsf{Der}(\mathtt{R}_u^{t'})$ for all $\mathtt{s}_{i,u}$ with $i \geq 2$. □

We obtain a corollary of property (vi) that is a consequence of applying it repeatedly.

Corollary 3.4.8. *If* $\mathtt{s}$ *is useful at time* $t \geq 0$, $u \in S(\mathtt{s}, t)$ *and there exists an index* $i \in \{1, \ldots, \ell_u\}$ *such that* $u \in S(\mathtt{s}_{i,u}, t-1)$, *then either* $u \in S(\mathtt{s}_{\ell_u,u}, t-1) \subseteq S(\mathtt{s}, t-1)$ *or there is an index* $k < \ell_u$ *such that* $u \in S(\mathtt{s}_{k,u}, t-1)$ *and* $\mathsf{c}_{k,u} \in \mathtt{M}_t \setminus \mathtt{M}_{t-1}$.

Proof. We start by recalling the fact that $\mathtt{s}$ is useful at time t implies that for every $u \in S(\mathtt{s}, t)$ it must be the case that $t_{c,u} < t$.

If $\ell_u = i$, then property (ii) of Corollary 3.4.7 guarantees that $S(\mathtt{s}_{i,u}, t-1) \subseteq S(\mathtt{s}, t-1)$ and we would get $u \in S(\mathtt{s}, t-1)$ as desired. Otherwise, $\ell_u > i$ and we consider the following two cases:

- If $\mathsf{c}_{i,u} \in \mathtt{M}_t \setminus \mathtt{M}_{t-1}$, then it suffices to take $k = i$ since $u \in S(\mathtt{s}_{i,u}, t-1)$ by assumption.
- Else $\mathsf{c}_{i,u} \in \mathtt{ST}_u^{t_{c,u,t}} \cup \mathtt{M}_{t-1}$. We have $t_{c,u,t} \leq t-1 \leq t$ (where the first inequality follows from the previous observation that $t_{c,u,t} < t$) and by (vi) of Corollary 3.4.7 for $t^* = t-1$, we obtain that $u \in S(\mathtt{s}_{i+1,u}, t-1)$. We now repeat the same argument depending on whether $\ell_u = i+1$ or $\ell_u > i+1$ and if $\ell_u > i+1$, we would consider whether $\mathsf{c}_{i+1,u} \in \mathtt{M}_t \setminus \mathtt{M}_{t-1}$ (and take $k = i+1$) or $\mathsf{c}_{i+1,u} \in \mathtt{ST}_u^{t_{c,u,t}} \cup \mathtt{M}_{t-1}$.

By repeating this argument we either get that $u \in S(\mathtt{s}_{\ell_u,u}, t-1)$ and from property (ii) of Corollary 3.4.7 deduce that $S(\mathtt{s}_{\ell_u,u}, t-1) \subseteq S(\mathtt{s}, t-1)$ and $u \in S(\mathtt{s}, t-1)$, or find an index $k < \ell_u$ such that $u \in S(\mathtt{s}_{k,u}, t-1)$ with $\mathsf{c}_{k,u} \in \mathtt{M}_t \setminus \mathtt{M}_{t-1}$. □

With these results we can prove the main result of this section, a theorem that connects the symbolic model and the combinatorial model.

Theorem 3.4.9. *Let* CGKA *be a perfectly correct continuous group-key agreement scheme that is* OW-k-$\mathrm{PCS}_{\neg\mathrm{RC}}$*-secure and does not use nested encryption. Consider an adversary playing game* OW-k-$\mathrm{PCS}_{\neg\mathrm{RC}}$ *of Figure 3.8 in the symbolic model.*

(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* $S \in \mathcal{S}_t$ *it holds that* $u \notin S$.

(iii) Let $t \geq 1$. *Recall that* $U_t \subseteq [n]$ *indicate the users that updated in round* t *and for* $u \in U_t$ *the set* $\mathtt{MU}_u^t$ *corresponds 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

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

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

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

where $C_{\leq t} = \bigcup_{0 \leq t' \leq t} C_{t'}$ are the users that have been corrupted at least once in round t or before. If CGKA *does not allow users to distribute work, then the last statement can be replaced by the following stronger expression:*

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

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

Remark 3.4.10. *Consider an execution of game OW-k-$\mathrm{PCS}_{\neg\mathrm{RC}}$ in the symbolic model, where* CGKA *is a perfectly correct, OW-k-$\mathrm{PCS}_{\neg\mathrm{RC}}$-secure CGKA scheme which does not use nested encryption and does not allow users to distribute work, and set $\mathrm{Cost}(u,t) = |\mathtt{MU}_u^t|$ to be the number of symbolic variables sent by u in round t. Then, by Theorem 3.4.9 the associated set system $\mathcal{S}_t$ (Definition 3.4.3) and Cost satisfy all properties of the combinatorial model described in Section 3.3.1. 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 Theorem 3.4.9. In Figure 3.8 we recall the security game for OW-k-$\mathrm{PCS}_{\neg\mathrm{RC}}$, 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, $\mathtt{ST}_u^t$. Therefore $\mathtt{k}_G \in \mathsf{Der}(\mathtt{ST}_u^t) \subseteq \mathsf{Der}(\mathtt{ST}_u^{t_{c,u}}, (\mathtt{R}_u^{t'})_{t_{c,u}+1 \leq t' \leq t}, \mathtt{M}_t)$. 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 $u \in C_t$, then $t_{c,u} = t$ and $\mathsf{Der}(\mathtt{ST}_u^{t_{c,u}}, (\mathtt{R}_u^{t'})_{t_{c,u}+1 \leq t' \leq t}, \mathtt{M}_t) = \mathsf{Der}(\mathtt{ST}_u^t, \mathtt{M}_t) \subseteq \mathsf{Der}(\{\mathtt{M}_t, \mathtt{COR}_t\})$. Therefore for any secret $\mathtt{s}$ that is useful at time t, $u \notin S(\mathtt{s}, t)$.

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

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

Oracle INIT(n, u_0) \\ one call; called first

08 $t \leftarrow 0$
09 $\mathtt{M} \leftarrow \emptyset$, $\mathtt{COR}_0 \leftarrow \emptyset$
10 $\mathrm{Cor}[u] \leftarrow \emptyset$, $\mathrm{Upd}[u] \leftarrow \emptyset$ for all $u \in [n]$
11 $\mathrm{Coins}[u, t] \leftarrow \emptyset$ for all $u \in [n]$, $\forall t$
12 $\mathtt{r}_{\mathrm{setup}} \leftarrow_{\$} \mathcal{R}$
13 $(\mathtt{pub}, (\mathtt{ST}_u)_{u\in[n]}) \leftarrow \mathsf{CGKA.Setup}(n; \mathtt{r}_{\mathrm{setup}})$
14 $\mathtt{r} \leftarrow_{\$} \mathcal{R}$; $\mathrm{Coins}[u_0, t] \leftarrow_{\cup} \{\mathtt{r}\}$
15 $(\mathtt{ST}_{u_0}, \mathtt{MI}_{u_0}^0) \leftarrow \mathsf{CGKA.Init}(\mathtt{ST}_{u_0}, \mathtt{pub}, n; \mathtt{r})$
16 for $u \in [n]$:
17 $\mathtt{ST}_u \leftarrow \mathsf{CGKA.Proc}(\mathtt{ST}_u, \mathtt{pub}, \mathtt{MI}_{u_0}^0)$
18 $\mathtt{k}_G \leftarrow \mathsf{CGKA.Key}(\mathtt{ST}_u)$
19 $\mathtt{M} \leftarrow_{\cup} \{\mathtt{pub}, \mathtt{MI}_{u_0}^0\}$

Oracle CHALL \\ one call

20 $\mathtt{k}^* \leftarrow \mathtt{k}_G$

Oracle ROUND(U_t)

21 $\mathtt{MU} \leftarrow \emptyset$
22 for $u \in U_t$:
23 $\mathrm{Upd}[u] \leftarrow_{\cup} \{t\}$
24 $\mathtt{r} \leftarrow_{\$} \mathcal{R}$; $\mathrm{Coins}[u, t] \leftarrow_{\cup} \{\mathtt{r}\}$
25 $(\mathtt{ST}_u, \mathtt{MU}_u^t) \leftarrow \mathsf{Update}(\mathtt{ST}_u, \mathtt{pub}; \mathtt{r})$
26 $\mathtt{MU} \leftarrow_{\cup} \{\mathtt{MU}_u^t\}$
27 for $u \in [n]$:
28 $\mathtt{ST}_u \leftarrow \mathsf{CGKA.Proc}(\mathtt{ST}_u, \mathtt{pub}, \mathtt{MU})$
29 $\mathtt{k}_G \leftarrow \mathsf{CGKA.Key}(\mathtt{ST}_u)$
30 $\mathtt{M} \leftarrow_{\cup} \mathtt{MU}$

Predicate $\mathrm{safe}_{k\text{-PCS}}$

31 for $u \in [n]$:
32 if $\mathrm{Cor}[u] \neq \emptyset$
33 if $\exists t \in \mathrm{Cor}[u] : t \geq t^*$: \\ corruption after challenge
34 return 0
35 $t_u^c \leftarrow \max\{t \in \mathrm{Cor}[u]\}$
36 if $|\{t \in \mathrm{Upd}[u] : t_u^c < t \leq t^*\}| < k$: \\ too few updates
37 return 0
38 return 1

Oracle CORR(u)

39 $\mathrm{Cor}[u] \leftarrow_{\cup} \{t\}$
40 $\mathtt{COR}_t \leftarrow_{\cup} \{\mathtt{ST}_u\}$

Figure 3.8: Security game $\mathrm{OW}\text{-}k\text{-}\mathrm{PCS}_{\neg\mathrm{RC}}$ 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.

(iii) We consider the sequences $\{\mathtt{s}_{1,u}, \ldots, \mathtt{s}_{\ell_u,u}\}$ that exist for each $u \in S(\mathtt{s}, t)$ by Corollary 3.4.7 and define the set $\mathcal{I}_t := \bigcup_{u\in S(\mathtt{s},t)}\{(\mathtt{s}'_{i,u}, \mathtt{s}_{i+1,u}) \colon i \in [\ell_u - 1] \text{ and } \mathtt{c}_{i,u} \in \mathtt{M}_t \backslash \mathtt{M}_{t-1}\}$. If $(\mathtt{s}'_{i,u}, \mathtt{s}_{i+1,u})$ and $(\mathtt{s}'_{j,v}, \mathtt{s}_{j+1,v})$ have the same associated ciphertexts, i.e., $\mathtt{c}_{i,u} = \mathtt{c}_{j,v}$, then $\mathtt{s}_{i+1,u} = \mathtt{s}_{j+1,v}$ and $\mathtt{s}'_{i,u} = \mathtt{s}'_{j,v}$. This shows that $|\mathcal{I}_t| \leq |\mathtt{M}_t \setminus \mathtt{M}_{t-1}|$.

For any element $(\mathtt{s}_1, \mathtt{s}_2) \in \mathcal{I}_t$, we introduce the associated set $S_{(\mathtt{s}_1,\mathtt{s}_2)} = S(\mathtt{s}_1, t-1)$. It holds that $S_{(\mathtt{s}_1,\mathtt{s}_2)} \in \mathcal{S}_{t-1}$ since $\mathtt{s}_1$ is useful at time t.

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 $\mathtt{s} \in \mathsf{Der}(\mathtt{R}_u^t)$, because randomness is generated independently. If $S_0 = S(\mathtt{s}, t-1)$, it holds that $S_0 \in \mathcal{S}_{t-1}$ since $\mathtt{s}$ is useful at time t. If $\mathtt{s} \in \mathsf{Der}(\mathtt{R}_u^t)$ for some user u, then either $t_{c,u} = t$ (i.e., $u \in C_t$) and $u \notin S(\mathtt{s}, t)$ or $t_{c,u} < t$ and $u \in S(\mathtt{s}, t)$.

Let $u \in S(\mathtt{s}, t)$ and assume that $t_{c,u} \geq 0$. By assumption, $\mathtt{s}$ is useful at time t, so $t_{c,u,t} < t$. From $t_{c,u} \geq 0$, it follows that $\mathtt{s}_{1,u} \in \mathtt{R}_u^{t'}$ for some $t_{c,u} + 1 \leq t' \leq t$ by property (iii) of Corollary 3.4.7. Our goal is to prove that $u \in S_0 \cup \bigcup_{(\mathtt{s}_1,\mathtt{s}_2)\in\mathcal{I}_t} S_{(\mathtt{s}_1,\mathtt{s}_2)}$ We consider two cases depending on whether $t' = t$ or not:

- if $t' < t$, then $u \in S(\mathtt{s}_{1,u}, t-1)$ (since $\mathtt{s}_{1,u} \in \mathtt{r}_u^{t'}$). By Corollary 3.4.8 for $i = 1$ we either get that $u \in S(\mathtt{s}, t-1) = S_0$ or find an index $k < \ell_u$ such that $u \in S_{(\mathtt{s}'_{k,u}, \mathtt{s}_{k+1,u})}$ with $(\mathtt{s}'_{k,u}, \mathtt{s}_{k+1,u}) \in \mathcal{I}_t$.
- if $t' = t$, we distinguish between the case when $\ell_u = 1$ and $\ell_u > 1$. If $\ell_u = 1$, then $u \in S_0$ (since $\mathtt{s}_{1,u} \in \mathtt{r}_u^t$). If $\ell_u > 1$ we claim that the ciphertext $\mathsf{c}_{1,u}$ must belong to $\mathtt{M}_t \backslash \mathtt{M}_{t-1}$. By (v) of Corollary 3.4.7, $\mathsf{c}_{1,u} \in \mathtt{ST}_u^{t_{c,u,t}} \cup \mathtt{M}_t$ for some $\mathtt{s}'_{1,u}$ such that $\mathtt{s}_{1,u} \in \mathsf{PRF}^{-1}(\mathtt{s}'_{1,u})$ and $\mathtt{pk}'_{1,u} = \mathsf{Gen}(\mathtt{s}'_{1,u})$, but $\mathtt{s}_{1,u} \in \mathtt{R}_u^t$ (this follows from $t' = t$) and this implies that the public key $\mathtt{pk}'_{1,u}$ is only available after round t. Therefore $\mathsf{c}_{1,u} \notin \mathtt{ST}_u^{t_{c,u,t}}$ (where we use $t_{c,u,t} < t$) and $\mathsf{c}_{1,u} \notin \mathtt{M}_{t-1}$, which implies $\mathsf{c}_{1,u} \in \mathtt{M}_t \setminus \mathtt{M}_{t-1}$. Moreover, the fact that u is the only user that knows $\mathtt{pk}'_{1,u}$ at the time of generating the messages in round t, implies $\mathsf{c}_{1,u} = \mathsf{Enc}(\mathtt{pk}'_{1,u}, \mathtt{s}_{2,u}) \in \mathtt{MU}_u^t \subseteq \mathsf{Der}(\mathtt{ST}_u^{t-1}, \mathsf{pub}, \mathtt{R}_u^t)$.

  We claim that $u \in S(\mathtt{s}_{2,u}, t-1)$. Since $\mathsf{c}_{1,u} \in \mathtt{MU}_u^t \subseteq \mathsf{Der}(\mathtt{ST}_u^{t-1}, \mathsf{pub}, \mathtt{R}_u^t)$, we have either $\mathtt{pk}'_{1,u}, \mathtt{s}_{2,u} \in \mathsf{Der}(\mathtt{ST}_u^{t-1}, \mathsf{pub}, \mathtt{R}_u^t)$ or $\mathsf{c}_{1,u} \in \mathtt{ST}_u^{t-1} \cup \mathsf{pub} \cup \mathtt{R}_u^t$ (we do not consider the possibility that there is an encryption of $\mathsf{c}_{1,u}$ in $\mathtt{ST}_u^{t-1} \cup \mathsf{pub} \cup \mathtt{R}_u^t$ since we assume that nested encryption is not used). The latter case is actually impossible since $\mathsf{c}_{1,u} \notin \mathtt{R}_u^t$ because of the type and it cannot be that $\mathsf{c}_{1,u} \in \mathtt{ST}_u^{t-1} \cup \mathsf{pub}$ because $\mathtt{pk}'_{1,u}$ is not available in any round before round t. In the first case we know that there is no element in $\mathsf{Der}(\mathtt{ST}_u^{t-1}, \mathsf{pub}, \mathtt{R}_u^t) \cap \mathsf{PRF}^{-1}(\mathtt{s}_{2,u})$ by construction of the sequence, so it must be that $\mathtt{s}_{2,u} \in \mathtt{ST}_u^{t-1} \cup \mathsf{pub} \cup \mathtt{R}_u^t$ or there exists a ciphertext $\mathsf{c} = \mathsf{Enc}(\mathtt{pk}, \mathtt{s}_{2,u}) \in \mathtt{ST}_u^{t-1} \cup \mathsf{pub} \cup \mathtt{R}_u^t$ such that $\mathtt{sk} \in \mathsf{Der}(\mathtt{ST}_u^{t-1}, \mathsf{pub}, \mathtt{R}_u^t)$. We analyze these two sub-cases:

  – if $\mathtt{s}_{2,u} \in \mathtt{ST}_u^{t-1} \cup \mathsf{pub} \cup \mathtt{R}_u^t$, it must be

  $$\mathtt{s}_{2,u} \in \mathtt{ST}_u^{t-1} \subseteq \mathsf{Der}(\mathtt{ST}_u^{t_{c,u,t-1}}, (\mathtt{r}_u^{\tilde{t}})_{t_{c,u,t-1}+1 \leq \tilde{t} \leq t-1}, \mathtt{M}_{t-1})$$

  because of properties (i) and (vii). Therefore $u \in S(\mathtt{s}_{2,u}, t-1)$.

  – If there exists $\mathsf{c} = \mathsf{Enc}(\mathtt{pk}, \mathtt{s}_{2,u}) \in \mathtt{ST}_u^{t-1} \cup \mathsf{pub} \cup \mathtt{R}_u^t$ such that $\mathtt{sk} \in \mathsf{Der}(\mathtt{ST}_u^{t-1}, \mathsf{pub}, \mathtt{R}_u^t)$, then $\mathsf{c} = \mathsf{Enc}(\mathtt{pk}, \mathtt{s}_{2,u}) \in \mathtt{ST}_u^{t-1} \cup \mathsf{pub}$ because c

is not of type randomness. The fact that $\mathsf{c} \in \mathtt{ST}_u^{t-1} \cup \mathtt{pub}$ implies that $\mathtt{sk} \in \mathsf{Der}(\mathtt{ST}_u^{t-1}, \mathtt{pub})$ not just $\mathtt{sk} \in \mathsf{Der}(\mathtt{ST}_u^{t-1}, \mathtt{pub}, \mathtt{R}_u^t)$. Therefore $u \in S(\mathtt{s}_{2,u}, t-1)$.

This proves the claim that $u \in S(\mathtt{s}_{2,u}, t-1)$.

By Corollary 3.4.8 for $i = 2$ we either get that $u \in S(\mathtt{s}, t-1) = S_0$ or find an index $k < \ell_u$ such that $u \in S_{(\mathtt{s}'_{k,u}, \mathtt{s}_{k+1,u})}$ with $(\mathtt{s}'_{k,u}, \mathtt{s}_{k+1,u}) \in \mathcal{I}_t$.

This proves that $S(\mathtt{s}, t) \cap C_{\leq t} \subseteq S_0 \cup \bigcup_{(\mathtt{s}_1, \mathtt{s}_2) \in \mathcal{I}_t} S_{(\mathtt{s}_1, \mathtt{s}_2)}$, i.e, $S(\mathtt{s}, t) \cap C_{\leq t}$ can be covered with sets in the family $\mathcal{F} = \{S_0\} \cup \{S_{(\mathtt{s}_1, \mathtt{s}_2)} \colon (\mathtt{s}_1, \mathtt{s}_2) \in \mathcal{I}_t\}$. We use the inequality $|\mathcal{I}_t| \leq |\mathtt{M}_t \setminus \mathtt{M}_{t-1}|$ to obtain that

$$\sum_{u \in U_t} |\mathtt{MU}_u^t| = |\mathtt{M}_t \setminus \mathtt{M}_{t-1}| \geq |\mathcal{I}_t| \geq |\mathcal{F}| - 1 \ .$$

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 $\mathtt{s} \in \mathsf{Der}(\mathtt{ST}_u^{t-1}, \mathtt{M}_{t-1}, \mathtt{MU}_{u'}^t)$ and $\mathtt{s} \in \mathsf{Der}(\mathtt{ST}_{u'}^{t-1}, \mathtt{R}_{u'}^t, \mathtt{M}_{t-1})$. With this additional property we can choose the ciphertexts $\mathsf{c}_{i,u}$ in a way such that every $\mathsf{c}_{i,u} \in \mathtt{M}_t \setminus \mathtt{M}_{t-1}$ is actually in $\mathtt{MU}_{u'}^t$. Therefore $|\mathcal{I}_t| \leq |\mathtt{MU}_{u'}^t|$.

Moreover, if $\mathtt{s} \in \mathsf{Der}(\mathtt{R}_u^t)$ it must be $u = u'$ and then $S_0 = \{u'\}$. If $\mathtt{s} \notin \mathsf{Der}(\mathtt{R}_u^t)$, the fact that $\mathtt{s} \in \mathsf{Der}(\mathtt{ST}_{u'}^{t-1}, \mathtt{R}_{u'}^t, \mathtt{M}_{t-1})$ implies that $\mathtt{s} \in \mathsf{Der}(\mathtt{ST}_{u'}^{t-1}, \mathtt{M}_{t-1})$ and this shows that $u' \in S(\mathtt{s}, t-1) = S_0$. Thus in both cases $u' \in S_0$ which completes the proof.

□

Remark 3.4.11. *Theorem 3.4.9 requires that* CGKA *not use nested encryption, i.e., not generate encryptions of ciphertexts. On a technical level, this restriction guarantees that for every pair* $(\mathtt{s}_1, \mathtt{s}_2) \in \mathcal{I}_t$ *constructed in the lemma's proof 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* $\mathsf{c} = \mathsf{Enc}(\mathtt{pk}_2, \mathsf{Enc}(\mathtt{pk}_1, \mathtt{m}))$ *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]).*

3.4.2 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 3.3.2) carries over to the symbolic model for OW-k-$\mathrm{PCS}_{\neg\mathrm{RC}}$-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 3.4.10. As a consequence of Theorem 3.4.9, 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 $\mathrm{Cost}(u,t) = |\mathtt{MU}_u^t|$ that counts the number of symbolic variables sent by u in round t.

Our bound requires CGKA not to use nested encryption and assumes CGKA does not allow users to distribute work. The former is needed in order to be able to use Theorem 3.4.9, whereas the latter guarantees that in property (iii) of Theorem 3.4.9 we obtain the same expression as in Equation 3.5 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 3.4.12. *Let $k, n, t_c \in \mathbb{N}$ and let* CGKA *be a correct and OW-k-$\mathrm{PCS}_{\neg\mathrm{RC}}$-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 $3 \leq k \leq \ln(\alpha_\varepsilon n)$, 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 $\mathrm{OW}\text{-}k\text{-}\mathrm{PCS}_{\neg\mathrm{RC}}$*, there exists a sequence of sets* U_{t_c+t} *of updating users for* $(1+\varepsilon)k$ *rounds such that the total communication cost satisfies*

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

If CGKA *has publicly-computable update cost (Definition 3.2.3), 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 $\vec{r} = (r_{t,u})_{t\in[(1+\varepsilon)k],u\in[n]}$ denote the non-symbolic randomness used in rounds $t = t_c + 1, \ldots, t_{\max}$, namely, $r_{t,u}$ 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 3.4.3). Then, as observed in Remark 3.4.10, $\mathrm{Cost}(u,t)$ and $\mathrm{Sec}(\mathcal{S}_t)$ satisfy all the properties of the combinatorial model described in Section 3.3.1. It follows from Theorem 3.3.2 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

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

as desired.

By definition of symbolic security (Definition 3.4.1), 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 3.3.2, 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 3.3.3.

Corollary 3.4.13. *Let k, n, t_c, ε, and α_ε be as in Corollary 3.4.12 and let* CGKA *be a correct and* OW-k-$\mathrm{PCS}_{\neg\mathrm{RC}}$*-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-$\mathrm{PCS}_{\neg\mathrm{RC}}$*, 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 3.2.3), 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.*

3.5 Almost-Matching Upper Bound

In this section we describe a simple CGKA protocol, inspired by CoCoA [AAN+22], 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 $\log(n)$.

This improvement comes at the expense of non-immediate forward secrecy, which requires at least $k/2$ updates from each user prior to their corruption. Likewise, we do not prove it secure against any type of active adversary and, indeed, only describe a simple protocol satisfying IND-k-$\mathrm{PCS}_{\mathrm{RC}}$ 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 CoCoA protocol in Section 3.5.1, to then describe a simple generalization of it in Section 3.5.2. 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 3.5.3. Section 3.5.4 contains the security proof for CoCoALight, and Section 3.5.5 discusses its efficiency, as well as its disadvantages in terms of forward secrecy.

3.5.1 Preliminaries and CoCoA

Ratchet trees. All protocols discussed in this section (as well as the original protocol they all trace back to, TreeKEM [BBR18], and variations of it [ACDT20, Wei19, AAN+22, AAN+24]) 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, in this section 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. We study the consequences of this phenomenon in Chapter 5.

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.

3.5.2 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+22] 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+22] 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 $(\mathsf{sk}_v, \mathsf{pk}_v)$ 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 immediately 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 $(\mathsf{sk}, \mathsf{pk})$ 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 (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

[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.

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.

3.5.3 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.

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+22, AJM22, BBR+23].

[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.`counter`), as an update in CoCoALight is not guaranteed to generate a new key for the root.

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

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 3.9. Each user in the group has an internal state of the form $\mathsf{st} = (u, \mathsf{T}, i, \gamma)$. Here, u is the identifier of the user, which is in the range $[n]$. 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 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 $\mathsf{path}(\mathsf{T}, u)$, and keeps track of the user's progress along their path. Finally, γ 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*

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

Table 3.2: Notation for state and node attributes.

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

Table 3.3: Helper functions for ratchet trees, given implicit tree T.

$\mathsf{resol}(v)$ 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 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 CGKA.Proc, 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 they overwrite keys they might later, while still processing M, need. Further, we use the notation in Table 3.2 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 3.3.

The protocol makes use of the following functions:

- $\mathsf{GenTree}_k(\mathsf{pub}, (\mathsf{pk}, \mathsf{sk}), r)$ takes as input the public parameters pub, which includes the public keys of n users, a key-pair corresponding to one of those users,

Algorithm CGKA.Setup(n)

00 $\bar{\mathsf{st}}, \mathsf{pub} \leftarrow \emptyset$
01 for $u \in [n]$:
02 $r \leftarrow_{\$} \mathcal{R}$
03 $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Gen}(r)$
04 $\mathsf{st}_u \leftarrow (u, (\mathsf{pk}, \mathsf{sk}), 0, \perp)$
05 $\bar{\mathsf{st}} \leftarrow \bar{\mathsf{st}} \cup \mathsf{st}_u$
06 $\mathsf{pub} \leftarrow \mathsf{pub} \cup (u, \mathsf{pk})$
07 return $\bar{\mathsf{st}}, \mathsf{pub}$

Algorithm CGKA.Proc(st, pub, M)

08 $(u, \mathsf{T}, i, \gamma) \leftarrow \mathsf{st}$
09 if $\mathsf{M}[0] =$ "$init$":
10 $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{T}$
11 $c \leftarrow \mathsf{M}[u]$
12 $r \leftarrow \mathsf{Dec}_{\mathsf{sk}}(c)$
13 $T \leftarrow \mathsf{GenTree}_k(\mathsf{pub}, (\mathsf{pk}, \mathsf{sk}), r)$
14 $\mathsf{st} \leftarrow (u, T, 0, \perp)$
15 else:
16 for $(v, \mathsf{pk}, c) \in \mathsf{M}$:
17 $v.\mathsf{pk} \leftarrow \mathsf{pk}$
18 if $v = \mathsf{PathNode}(u, i)$ and $\gamma \neq \perp$:
19 $(v, r) \leftarrow \gamma$
20 $(v.\mathsf{pk}, v.\mathsf{sk}) \leftarrow \mathsf{PKE.Gen}(r)$
21 $\mathsf{st.pend} \leftarrow \perp$
22 if $i = |\mathsf{path}(u)|$:
23 $\mathsf{st.counter} \leftarrow 0$
24 else $\mathsf{st.counter} \leftarrow i + 1$
25 elseif $v \in \mathsf{path}(u)$ and $c \neq \perp$:
26 $w = \mathsf{resol}(v) \cap \mathsf{path}(u)$
27 $r \leftarrow \mathsf{Dec}_{\mathsf{sk}_w}(c)$
28 $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Gen}(r)$
29 $v.\mathsf{sk} \leftarrow \mathsf{sk}$
30 return $(\mathsf{st}, v_{root}.\mathsf{sk})$

Algorithm CGKA.Init(st, pub, n)

31 $(u, (\mathsf{pk}, \mathsf{sk}), i, \gamma) \leftarrow \mathsf{st}$
32 $(j, \mathsf{pk}_j)_{j \in [n]} \leftarrow \mathsf{pub}$
33 assert $u \in [n]$
34 $\mathsf{MI} \leftarrow$ ("$init$")
35 $r \leftarrow_{\$}$
36 for $w \in [n]$:
37 $c_w \leftarrow_{\$} \mathsf{PKE.Enc}_{\mathsf{pk}_w}(r)$
38 $\mathsf{MI} \leftarrow \mathsf{MI} \cup c_w$
39 return $(\mathsf{st}, \mathsf{MI})$

Algorithm CGKA.Upd(st)

40 $(u, \mathsf{T}, i, \gamma) \leftarrow \mathsf{st}$
41 $v \leftarrow \mathsf{PathNode}(u, i)$
42 $r \leftarrow_{\$} \mathcal{R}$
43 $\mathsf{st.pend} \leftarrow (v, r)$
44 $(v.\mathsf{pk}, v.\mathsf{sk}) \leftarrow \mathsf{PKE.Gen}(r)$
45 if $i = 0$:
46 $\mathsf{MU} \leftarrow \mathsf{MU} \cup (v, v.\mathsf{pk}, \perp)$
47 else:
48 for $p \in \mathsf{resol}(v)$:
49 $c_p \leftarrow_{\$} \mathsf{PKE.Enc}_{p.\mathsf{pk}}(r)$
50 $\mathsf{MU} \leftarrow \mathsf{MU} \cup (v, v.\mathsf{pk}, c_p)$
51 return $(\mathsf{st}, \mathsf{MU})$

Figure 3.9: Description of protocol CoCoALight$_k$, healing in k rounds.

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).

- PathNode(u, i) takes as input a user u (implicitly part of a group with associated ratchet tree T) and and integer i. Outputs the i^{th} node in path(u), with the 0^{th} node being leaf(u).

3.5.4 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 3.4.

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 $(v.\mathsf{pk}, v.\mathsf{sk}) = \mathsf{PKE.Gen}(r)$. 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 $\mathsf{PKE.Enc}_{v.\mathsf{pk}}(r_w)$. 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 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 3.5.1. *Let* k *be the challenge group key in the* $\text{IND-}k\text{-PCS}_{\text{RC}}$ *game and assume* $\text{safe}_{k\text{-PCS}}$ *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 $(\mathsf{k}_i)_{i \in [t_{\max}]}$, and the corresponding recovery graphs as $(C_i)_{i \in [t_{\max}]}$. Additionally, each epoch has an associated ratchet tree T_i. In the case we are considering of static groups, all the T_i are the same when seen as graphs, though will obviously differ in the key material associated to them. Further, some of the 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 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 T, as each C_i is a minor of T, i.e. a graph resulting from applying a series of edge contractions to T (in particular, the contractions of exactly those edges that share a common blank node).

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

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 $\mathsf{k}^* = \mathsf{k}_{t^*}$ be the challenge group key in the experiment, with C^* and T^* being the challenge graph and its associated ratchet tree, respectively. Then, for all nodes $v \in \mathsf{T}$, we will denote by $v.\mathsf{pk}^j$ the $j + 1^{th}$ last public key associated to v, with respect to the time k^* was generated; e.g., $v.\mathsf{pk}^0$ corresponds to the key associated to v in C^*, $v.\mathsf{pk}^1$ corresponds to the one that $v.\mathsf{pk}^0$ overwrote, and so on. Note that there exists some j', namely the number of keys every associated to v, such that for $j \geq j'$, $v.\mathsf{pk}^j = \perp$. For the last bit of notation, we will write $C_i \triangleleft C_j$ 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 $v \in \mathsf{T}$, we will also write, $v.\mathsf{pk}^j \triangleleft v.\mathsf{pk}^i$, whenever $i < j$; i.e., whenever $v.\mathsf{pk}^j$ precedes $v.\mathsf{pk}^i$. 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 $a \triangleleft b$, 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 $C_i \triangleleft C_j$, but there exist node v such that $C_i(v) \triangleright C_j(v)$ (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 $\mathrm{safe}_{k\text{-PCS}}$ 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 T has depth $\lfloor k/2 \rfloor$. Thus, if the seed or keys associated to some node v have leaked, it must be that $C^*(v) = v.\mathsf{pk}^i$ 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 $\hat{C} \triangleleft C^*$. Now, let $\hat{\mathsf{pk}}$ 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 $C^*(v) \triangleleft \hat{\mathsf{pk}}$. Thus, it remains to show that $\hat{\mathsf{pk}} \triangleleft \hat{C}(v)$, which would imply $C^*(v) \triangleleft \hat{C}(v)$. 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 $w.\mathsf{pk}^j$ was sampled by $\hat{u}$ before they sampled $C(v_{root})$, it holds that $w.\mathsf{pk}^j \triangleleft C(w)$. 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. □

Proof of Statement 2. Assume for contradiction there is a crossing between C_i and C_j, $C_i \triangleleft C_j$, and let v be the highest node (closest one to v_{root}), such that $C_i(v) \triangleright C_j(v)$. 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, $C_i(w) \triangleleft C_j(w)$. 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 T_j, which is a contradiction.

This concludes the proof. □

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 $\mathrm{IND}\text{-}k\text{-}\mathrm{PCS}_{\mathrm{RC}}$ 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] □

3.5.5 Efficiency and tradeoffs

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 chapter.

The communication cost of the respective protocols and lowerbounds (recall our lowerbound depends on variable $\epsilon \in (0, 2/5)$) is shown in Figure 3.10. 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

[7] For technical reasons, one would need to either have the group key be a value from which no public key can be derived (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+22], or by determining that the root node has no associated public key, as in [KPPW+21]. We omit this, for simplicity.

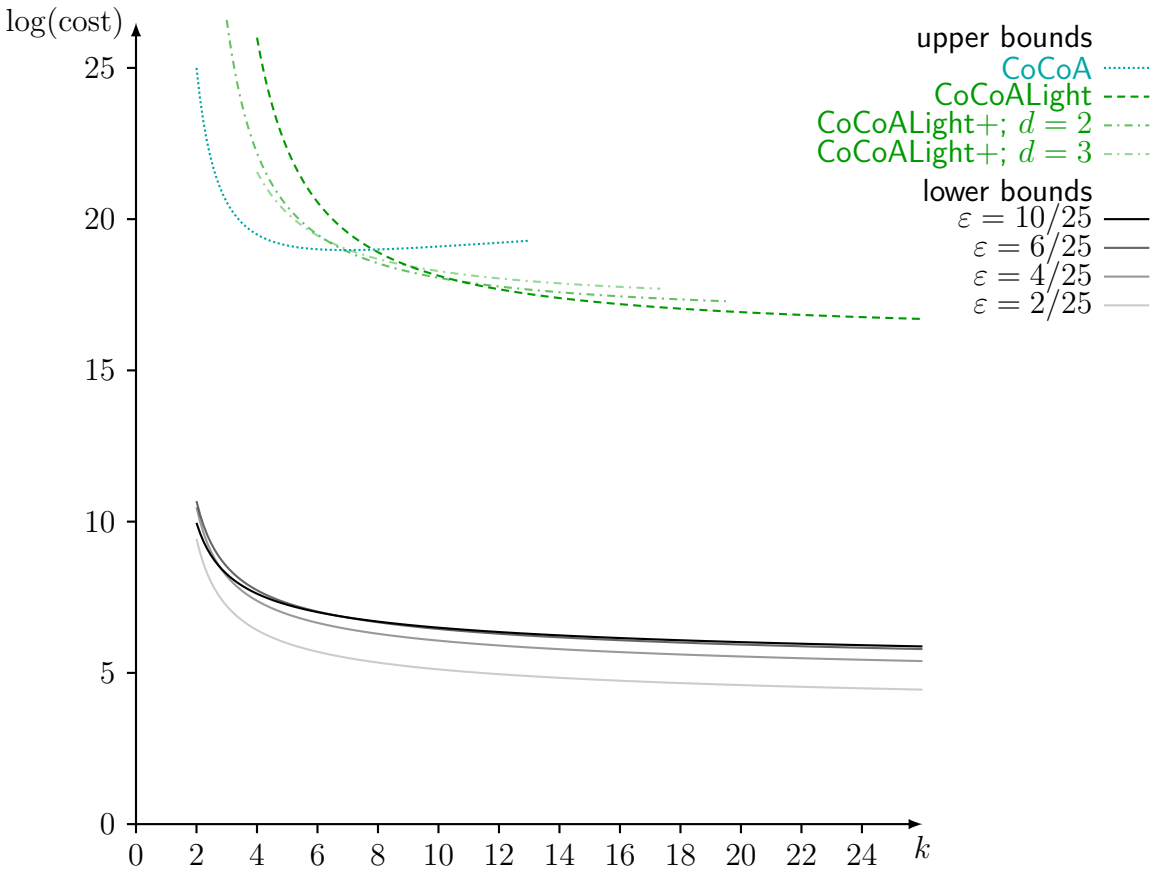

Figure 3.10: Shown are the communication costs for different values of k, for (the generalization from Section 3.5.2 of) CoCoA [AAN+22] and CoCoALight (Figure 3.9), as well as the lower-bound shown in Theorem 3.3.2, for different values of ϵ. Further, a generalization of CoCoALight, termed CoCoALight+ is shown for different values of $d \in \{2, 3\}$. The size of the group is $n = 4096$.

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 3.10 under the name CoCoALight+.

Recall that the generalized CoCoA protocol from Section 3.5.2 requires a total number of ciphertexts 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 3.9, 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

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

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 $[1, \delta]$. Figure 3.10 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 chapter 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.

CHAPTER 4

The Cost of Maintaining Keys in Dynamic Groups

This chapter essentially replicates the results that appear in our publication [AAB+24].

4.1 Introduction

For more than twenty years we have known that some of the existing multicast encryption (ME) schemes are essentially optimal when we consider non-batched replacements. However, the question remains has remained open in the case where multiple replacement operations are carried out at the same time. In this chapter we study lower bounds for batched replacements both for ME and CGKA and close the gap between the existing upper bounds in the order of $d(1 + \log(n/d))$ that one obtains from ME schemes [LYGL01, YLZL02, SM03] and the implicit lower bound of d for CGKA schemes built from standard primitives due to [BDR20]. On a high level, we prove the following statement.

> In the symbolic model, consider a secure and correct ME (or CGKA) scheme. If a set of d users chosen uniformly at random from a group with n members is replaced with different users, then the protocol messages must have contained at least $\frac{\ln(2)}{3} \cdot d \cdot \log(n/d)$ ciphertexts in expectation.

In the above we allow for symmetric encryption, pseudorandom generators, (dual) pseudorandom functions, and secret sharing as building blocks for ME, and for (key-updatable) public-key encryption, pseudorandom generators, (dual) pseudorandom functions, and secret sharing in the case of CGKA.

As there exist constructions of ME [LYGL01, YLZL02, SM03] and CGKA [BBR+23] (the latter however only with respect to fair-weather complexity, i.e., only for beneficial sequences of operations) that achieve a communication cost of $d(1 + \log(n/d))$ our bound is tight up to a small multiplicative factor.[1] Intuitively, this shows that existing ME and CGKA constructions are optimal and, in particular, suggests that the way the removal of users is handled in the MLS standard [BBR+23] cannot be improved by simple means.

We point out that our bound is an average case bound, as the set of replaced users is chosen uniformly at random (as is the case for the single user bound of [AAB+21]), as opposed to ones relying on an adversarially chosen sequences of operations [MP04, ACPP23, BDG+22]. Our technical statements regarding ME and CGKA (Corollaries 4.3.6 and A.1.7) take *amortized* communication complexity into account. I.e., we consider a game running over $t_{\max}$ rounds where, in every round t, a set of d_t group members, chosen uniformly at random from the current group, is replaced by new users. Then, if we denote the set of ciphertexts and keys sent in round t by $\mathtt{M}_t$, we prove that

$$\mathbb{E}\left[\sum_{t=0}^{t_{\max}} |\mathtt{M}_t|\right] \geq \frac{\ln(2)}{3} \sum_{t=1}^{t_{\max}} d_t \log\left(\frac{n}{d_t}\right).$$

Note that we cannot guarantee that $|\mathtt{M}_t| \geq (\ln(2)/3) \cdot d_t \cdot \log(n/d_t)$ for all t. This is necessary as, in principle, some of the communication required to replace the users in round t might already have happened in prior rounds, as will be discussed in greater detail below.

Proof overview. Conceptually, we follow the approach of Chapter 3, where we prove lower bounds on the cost incurred by CGKA schemes recovering from corruption(s) over several rounds. That is, we decouple the combinatorial problem at the core of minimizing the cost for batched user replacements from the more technical issues that arise when arguing within the confines of the symbolic model. More precisely, our proof consists of two major parts, the first of which is common to both the case of multicast encryption and CGKA simultaneously. First, in Section 4.2.1 we capture the problem of securely replacing a batch of users in ME and CGKA in a clean, self-contained combinatorial model, and prove our lower bound within this model. The second step consists of showing that bounds in the combinatorial model imply bounds in the symbolic model. This is done for ME in Section 4.3.2 and for CGKA in Appendix A.1, the proofs being very similar. We discuss these steps in greater detail below.

[1] We point out that for typical use cases we have $d \ll n$. Further, the case $\log(n/d) < 1$ in which our bound is not asymptotically optimal implies $d > n/2$. In this case the linear lower bound by Bienstock, Dodis, and Rösler [BDR20] is asymptotically optimal.

The combinatorial model. Our aim with the combinatorial model is to capture in an intuitive way how the sets of users that share a secure secret evolve over time. Here, 'secret' can be thought of as a symmetric key or secret key depending on whether we want to model ME or CGKA. More precisely, for $n_{\max}, t_{\max} \in \mathbb{N}$, we let $[n_{\max}]$ be the universe of users and, for rounds $t \in [t_{\max}]$, consider a sequence of groups $G_t \subseteq [n_{\max}]$ evolving round-by-round by replacing a set of group members in every round. We then capture the secrets shared by users in each round as a sequence of set systems $\mathcal{S}_t \subseteq 2^{[n_{\max}]}$. A set $S \subseteq [n_{\max}]$ being part of $\mathcal{S}_t$ intuitively means that there exists a secret $\mathtt{r}$ with the following two properties. On the one hand, the set of users in G_t that, in round t (or any round before), are able to recover $\mathtt{r}$ from their internal states and the protocol messages sent so far is exactly given by S; and, on the other hand, $\mathtt{r}$ is secure. The latter means that, even given the protocol messages as well as the current and all prior states of every user *not* in G_t, it is not possible to recover $\mathtt{r}$.

Correctness and security of the corresponding ME or CGKA scheme impose two restrictions on $\mathcal{S}_t$. Namely, for all t we have $G_t \in \mathcal{S}_t$, which corresponds to the existence of a secure group key shared by all the members in G_t. Further, as keys known to non-members of the group are being considered insecure, it must be the case that $S \subseteq G_t$ for all $S \in \mathcal{S}_t$.

The set system $\mathcal{S}_t$ evolves over time. Removing a user u from the group leads to all secrets they had at some point access to being considered insecure. This means that all sets in $\mathcal{S}_{t-1}$ that contained u can no longer be present in $\mathcal{S}_t$. On the other hand, by sending protocol messages, new secrets can be shared with users, meaning that sets can be added to $\mathcal{S}_t$. Adding sets to $\mathcal{S}_t$, however, comes at a communication cost, since the corresponding secrets cannot be simply sent in the clear, but instead must be encrypted under (potentially multiple) secure keys already present in the system. We capture this with a cost function $\mathrm{Cost}(t)$ that, for now, can be thought of as a lower bound on the ciphertexts needed to be sent in the rounds up to t, in order to communicate the secrets corresponding to $\mathcal{S}_t$. While the definition of $\mathcal{S}_t$ can be seen as a natural generalization of the set system introduced for static groups in Chapter 3 to the setting of dynamic groups, our definition of the cost function deviates substantially from prior lower bounds in the symbolic model, and we consider it to be one of the main conceptual contributions of this work, as we discuss below.

Defining the cost function. Prior works giving lower bounds for ME or CGKA schemes in the symbolic model follow one of two different approaches for counting the number of ciphertexts sent in order to achieve both correctness and security of the scheme. [MP04] and [AAB+21] for round t use as cost function the amount of ciphertexts that were used to communicate the group key of round $t-1$, and are no longer of use in round t as they are encryptions of keys that are known by users

removed from the group in round t. Each of these ciphertexts can be identified with a particular secret (which is one of the encryption keys used in the ciphertext), and thus in our abstraction this cost metric can be seen as giving a cover of the set $G_{t-1} \setminus D_t$ using sets in $\mathcal{S}_{t-1}$, where D_t denotes the set of users removed from the group in round t. Moreover, it also admits another interpretation, namely, these ciphertexts are encryptions of secrets that are known by users in D_t and this means that the cost metric can be seen as counting some of the sets removed from the set system in round t.

On the other hand, [BDR20] and Chapter 3 consider the number of ciphertexts sent in a particular round t that are necessary to communicate a new secret $\mathtt{r}$ to (some of) the group members. Note that in order to communicate $\mathtt{r}$, it must have been encrypted under secret keys already established by the scheme. Seen through the abstraction of set system $\mathcal{S}_t$, this means if the set S (corresponding to $\mathtt{r}$) is added to $\mathcal{S}_t$, it must have been covered by the union of a collection of sets in $\mathcal{S}_{t-1}$. Accordingly, one can essentially use as cost function the size of a minimum cover of S with respect to $\mathcal{S}_{t-1}$, i.e., the smallest amount of sets in $\mathcal{S}_{t-1}$ the union of which covers S. To be a bit more precise, the cover may also include singletons $\{u\}$ for all users $u \in [n_{\max}]$ (corresponding to the users' personal keys). In particular, this is relevant regarding users being added to the group which, by the rules imposed by correctness and security, cannot be part of any set in $\mathcal{S}_{t-1}$.

In this work we define $\mathrm{Cost}(t)$ taking into account both the number of removed sets and the size of a minimum cover of G_t using sets in $\mathcal{S}_{t-1}$ and singletons for all users $u \in [n_{\max}]$. Unlike [MP04] and [AAB+21], which only take into account *some* of the destroyed sets, we generalize their approach and count *every* set removed from the set system. This is motivated by the following observation. When considering a scheme in the combinatorial model, we would like to exploit that, in every round t, the group G_t must be an element of $\mathcal{S}_t$. Following the second cost metric described above, we can argue that the cost of adding the corresponding key to the set system must be at least the size of a minimum cover of G_t with respect to the prior set system $\mathcal{S}_{t-1}$. However, we consider a security experiment running over multiple rounds and do not want to impose unnecessary restrictions on $\mathcal{S}_{t-1}$. In particular, it could be the case that $\mathcal{S}_{t-1} = 2^{G_{t-1}}$ is the full power set of the prior group. Thus, if we denote the d users removed from and added to G_{t-1} by D_t and A_t, respectively, we have that $S = (G_{t-1} \setminus D_t) \in \mathcal{S}_{t-1}$, and obtain a minimum cover of the new group as

$$G_t = S \cup \bigcup_{u \in A_t} \{u\}.$$

This means that removing the users from the group comes essentially for free, and the only contribution to the cost function stems from adding the users in A_t to the group. As a consequence, using this cost metric we would end up with a cost that is linear in d, in turn recovering the bound already implicitly given in [BDR20].

Note, however, that in the example above the set system $\mathcal{S}_{t-1}$ contains a number of sets that is exponential in the group size n. Further, every set in $\mathcal{S}_{t-1}$ containing at least one of the removed users would no longer be considered secure after round t, and there is an exponential number of sets of this type. Thus, in the first cost metric discussed above, i.e., counting sets removed from the system, maintaining such a huge system would be prohibitively expensive. For this reason, in this paper we use the sum of the two prior approaches as cost metric and define

$$\mathrm{Cost}(t) = \underbrace{|\{S \in \mathcal{S}_{t-1} : S \cap D_t \neq \emptyset\}|}_{\text{sets removed from } \mathcal{S}_{t-1}} + \underbrace{\mathrm{SizeMinCov}_{=}(G_t, \mathcal{S}_{t-1} \cup \{\{u\} : u \in G_t\})}_{\text{size of minimum cover of } G_t}.$$

For an illustration of our cost function for a concrete key-tree see Figure 4.1. We stress that $\mathrm{Cost}(t)$ is not to be understood as the amount of ciphertexts sent in round t, but instead as a lower bound on the ciphertexts sent up to, and including, that round. Further, our precise definition of the cost function (see Definition 4.2.1) also accounts for minor potential savings in terms of ciphertexts, stemming from the following two observations. On the one hand, that adding singletons to $\mathcal{S}_t$ does not necessarily require sending a ciphertext; on the other, that one ciphertext in the second summand of $\mathrm{Cost}(t)$ can be saved by deriving new keys from the output of a pseudorandom generator evaluated on the secret corresponding to one of the sets making up the minimum cover.

Lower bounding $\mathrm{Cost}(t)$ in the combinatorial model. The example discussed above suggests a trade-off between the two terms of $\mathrm{Cost}(t)$. Intuitively, the larger the set system $\mathcal{S}_{t-1}$, the cheaper it is to add G_t to $\mathcal{S}_t$. Here, the extreme case is given by the example discussed above, which essentially corresponds to preparing a key for the removal of every possible subset of G_{t-1}, and that leads to a large cost due to the first summand of $\mathrm{Cost}(t)$. The opposite extreme would be $\mathcal{S}_{t-1} = G_{t-1} \cup \{\{u\} : u \in G_{t-1}\}$ where, except for the group key, there is only a personal key for every group member. In this case any cover of G_t with respect to the previous set system would be made up of singletons and thus of size linear in $|G_t|$. Hence, to minimize the overall cost, intuitively it makes sense to balance the two components of $\mathrm{Cost}(t)$, which turns out to also be the case for the best known constructions of ME [LYGL01, YLZL02, SM03]. In these constructions, based on balanced binary trees, replacing d uniformly random group members requires in expectation to replace $\Theta(d(1 + \log(n/d)))$ keys in the system, each of which comes at the cost of one ciphertext. Further, the expected size of a minimum cover of G_t turns out to be of the same size. Accordingly, both summands of $\mathrm{Cost}(t)$ are of order $\Theta(d(1 + \log(n/d)))$.

We show that these constructions are optimal (up to a small constant factor) by roughly proving in Theorem 4.2.4 that, for every choice of $(\mathcal{S}_t)_{t=0}^{t_{\max}}$ satisfying the requirements

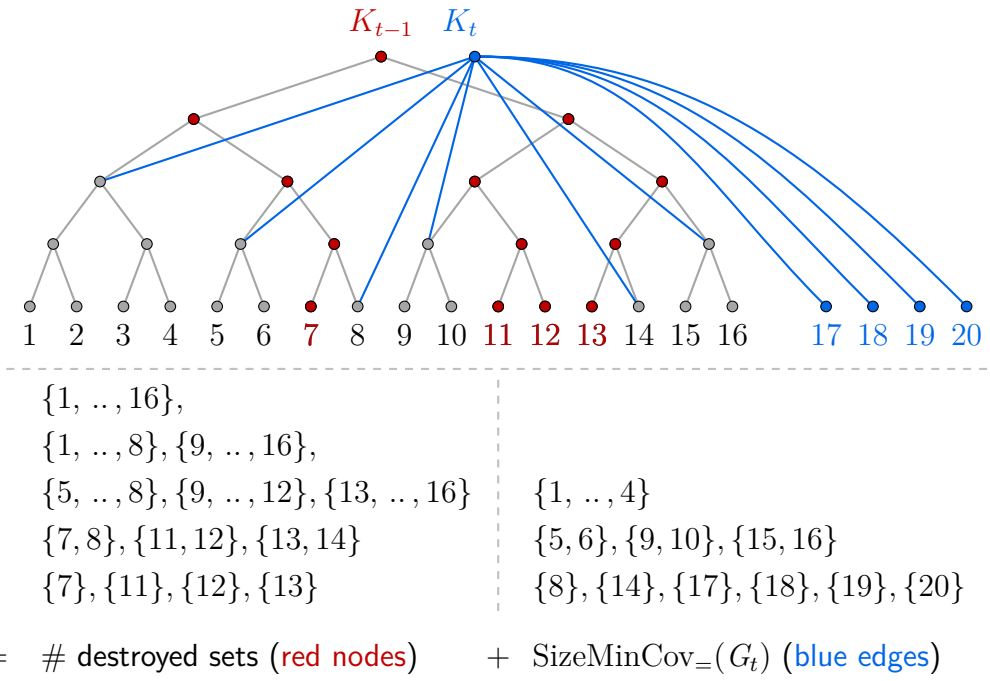

Figure 4.1: Example of our cost function on a balanced binary key-graph (top). The set associated to a node (key) is given by all leaves (users) whose path to the root contains the node in question. The figure depicts users 7, 11, 12, 13 in the group $G_{t-1} = \{1, \ldots, 16\}$ being replaced by users 17, 18, 19, and 20, producing group G_t. Gray and blue nodes were already part of the system at time $t-1$, blue nodes are added at time t. Our cost function $\mathrm{Cost}(t)$ counts the sets in $\mathcal{S}_{t-1}$ no longer secure after removing the users (bottom left, corresponding to the red nodes), as well as the size of a minimum cover of the new group with respect to the remaining secure sets and the added users' personal keys (bottom right, corresponding to the blue edges), i.e., the number of ciphertexts that have to be sent in order to establish the new group key K_t. Note that what is described here is a simplification ignoring possible optimizations and special cases (that our formal model does capture).

of the combinatorial model, it must hold that

$$\mathbb{E}\left[\sum_{t=0}^{t_{\max}} \mathrm{Cost}(t)\right] \geq \sum_{t=1}^{t_{\max}} d_t \ln\left(\frac{n}{d_t}\right),$$

where d_t denotes the amount of users replaced in round t and the set D_t of users replaced is sampled uniformly at random in every round. In the proof we consider two families $(X_{D_t})_{D_t \subseteq G_{t-1}}$ and $(Y_{D_t})_{D_t \subseteq G_{t-1}}$ of set systems $X_{D_t}, Y_{D_t} \subseteq 2^{[n_{\max}]}$, parameterized by all possible choices of D_t. These essentially correspond to the two summands

of the cost function. Accordingly, the elements of X_{D_t} capture the sets in $\mathcal{S}_{t-1}$ that are destroyed due to the removal of users in round t, and the elements of Y_{D_t} correspond to a minimum cover of the new group G_t with respect to $\mathcal{S}_{t-1}$. We then observe that the two families of set systems satisfy a disjointedness condition required for the Bollobás Set Pairs Inequality [Bol65]. Applying the inequality allows us to lower bound a term related to $\sum_{D_t \subseteq G_{t-1}} |X_{D_t}| + |Y_{D_t}|$ which, after some calculations, yields the desired bound on $\mathrm{Cost}(t)$.

Translation to the symbolic model. In the second conceptual step, we prove that lower bounds on $\sum_{t=1}^{t_{\max}} \mathrm{Cost}(t)$ in the combinatorial model imply lower bounds on the amount of ciphertexts sent by a secure and correct ME or CGKA scheme in the symbolic model, albeit at a potential loss of a factor of $1/3$. In the symbolic model [DY83], one considers ME or CGKA schemes constructed from cryptographic primitives used as building blocks that are essentially modeled to satisfy ideal security. Our lower bound for ME allows for pseudorandom generators (PRGs), pseudorandom functions (PRFs), dual PRFs (dPRFs), secret sharing, and symmetric encryption (SE), and thus in particular covers all building blocks used for the corresponding upper bounds [LYGL01, YLZL02, SM03]. Compared to prior lower bounds, it covers more building blocks than the ones considered in [MP04] and [AAB$^+$21], which do not cover dPRFs. Our lower bound for CGKA uses PRGs, PRFs, dPRFs, secret sharing, public-key encryption (PKE), and key-updatable public-key encryption (kuPKE) as building blocks, and, in particular, covers all primitives used in important schemes like MLS [BBR$^+$23].

Regarding a comparison to prior bounds, while it covers strictly more primitives than the one of [AAB$^+$21], it is incomparable to the bound of [BDR20], who do not allow for secret sharing, but additionally consider broadcast encryption (BE). We consider it an interesting open question, whether our bound can be extended to BE, but point out that it is a very powerful primitive, on the one hand, has to the best of our knowledge not been used in practical CGKA constructions, and, on the other hand, implies the existence of a multicast encryption scheme with constant communication complexity (see Appendix A.1.4). As a consequence any such bound would have to substantially differ from the techniques used in this work.

We now describe the translation of our combinatorial bound to the symbolic model in more detail. In every round t, to every secure secret $\mathtt{r}$ present in the symbolic model we associate the set of users that had access to $\mathtt{r}$ in a round up to and including t. Then, we prove that the resulting set systems satisfy the security and correctness properties imposed in the combinatorial model, and further that

$$\sum_{t=0}^{t_{\max}} |\mathtt{M}_t| \geq \frac{1}{3} \sum_{t=0}^{t_{\max}} \mathrm{Cost}(t),$$

where $\mathtt{M}_t$ denotes the protocol message sent in the symbolic model in round t. In fact, the inequality holds even if we only take into account the number of ciphertexts sent rather than all protocol messages. In combination with our lower bound in the combinatorial model, this immediately yields the desired bounds on ME and CGKA.

The loss of $1/3$ in our bound is due to the following. Consider the cost function $\mathrm{Cost}(t)$, for some t, in the combinatorial model. Intuitively, each of the two components, i.e., amount of sets removed from the system and size of a minimum cover of G_t, is justified by the requirement that a corresponding amount of ciphertexts is sent to communicate the respective secrets. However, it might be the case that a ciphertext that corresponded to a part of the minimum cover of group G_t in a later round $t' > t$, might be the one being used to justify the cost of a set being removed from $\mathcal{S}_{t'-1}$ following the removal of some users. In this case, the same ciphertext is being counted twice in $\sum_{t=1}^{t_{\max}} \mathrm{Cost}(t)$.

For a minimal example of this, consider the universe of users $\{u_i : i = 0, \ldots, n_{\max}\}$, where each user holds a personal key $\mathtt{k}_i$, and the sequence of groups is given by $G_t = \{u_0, u_t\}$. I.e., the second user of a group of size 2 is replaced in every round by encrypting the new group key to user u_t's personal key (it is possible to communicate the new group key to user 0 without the need of an additional ciphertext by making a clever use of a pseudorandom generator). Then, the ciphertext accounting for the minimum cover of G_t is the same as the one corresponding to the set $G_t = \{u_0, u_t\}$ that is being removed from the set system $\mathcal{S}_t$ when considering the cost of the following round $t + 1$. Accordingly we have that $\mathrm{Cost}(t) = 2$ for every round, while only one ciphertext is being sent per round.

However, we are able to show that this kind of double counting is essentially the only thing that can go wrong in our translation between combinatorial and symbolic models. The idea behind this is to derive separate bounds on the sum over t of each summand of $\mathrm{Cost}(t)$.

We start by studying $\sum_{t=0}^{t_{\max}} (\mathrm{SizeMinCov}_{=}(G_t, \mathcal{S}_{t-1} \cup \{\{u\} : u \in G_t\}) - 1)$. In order to find a cover for the set G_t we first cover the subset of users in G_{t-1} that are not removed from the group at time t, i.e., $G_{t-1} \setminus D_t$. Every user in $G_{t-1} \setminus D_t$ must be able to derive the group key of round $t - 1$ and do so by decrypting some ciphertexts sent in or before round $t - 1$. If for each of these ciphertexts we consider the set of users who know the secret key needed for decryption we obtain a cover $\mathcal{C}_{t,1}$ of $G_{t-1} \setminus D_t$. Moreover, these ciphertext can be chosen so that they are an encryption of a secret that is no longer useful in round t. This guarantees that the ciphertexts used for $\mathcal{C}_{t,1}$ and $\mathcal{C}_{\tilde{t},1}$ are different for $t \neq \tilde{t}$. Therefore $\sum_{t=1}^{t_{\max}} |\mathcal{C}_{t,1}| \leq \sum_{t=0}^{t_{\max}-1} |\mathtt{M}_t|$.

Next we obtain a cover for G_t by considering the singletons $\{u\}$ for each user that is added in round t. Since the users being added at time t do not share any secrets

with the users in G_{t-1}, we have $|\mathtt{M}_t| \geq |A_t| - 1$, where the subtraction comes from the possible use of PRGs. However this might introduce some double counting as these ciphertext might be used to obtain the inequality at the end of the previous paragraph. Thus

$$\sum_{t=0}^{t_{\max}} |\mathtt{M}_t| \geq \frac{1}{2} \sum_{t=0}^{t_{\max}} (\mathrm{SizeMinCov}_{=}(G_t, \mathcal{S}_{t-1} \cup \{\{u\} : u \in G_t\}) - 1).$$

Regarding our bound on the first summand of $\mathrm{Cost}(t)$, if we consider a set S associated to some secret $\mathtt{r}$ in round $t-1$ that contains at least two users, it must be the case that at least one of them had to decrypt a ciphertext in order to learn $\mathtt{r}$. Now, the additional restriction that $S \cap D_t \neq \emptyset$ guarantees that this ciphertext is not used in future rounds and therefore we obtain

$$\sum_{t=0}^{t_{\max}} |\mathtt{M}_t| \geq \sum_{t=1}^{t_{\max}} |\{S \in \mathcal{S}_{t-1} : S \cap D_t \neq \emptyset \text{ and } |S| > 1\}|.$$

Combining the bounds on both components yields our bound on $\sum_{t=1}^{t_{\max}} \mathrm{Cost}(t)$.

4.2 Lower Bound for Multicast Encryption in the Combinatorial Model

In this section we present a simple, purely combinatorial model that aims to capture the communication cost of batched replacements of users in multicast-encryption (ME) and continuous group-key agreement (CGKA) schemes. In both settings, a group of users evolving through rounds wants to agree on a sequence of group-keys by sending and processing protocol messages. In the case of ME, these are generated and sent by a central authority, whereas in the case of CGKA they are generated by the users themselves and distributed via an untrusted server. Security essentially requires that, even given access to all protocol messages and the current and prior internal states of all users not currently in the group, it is not possible to gain any information on the current group key.

4.2.1 The Combinatorial Model

Our model closely resembles the one used in Section 3.3.1 but extends it to dynamic groups, and further differs in some aspects such as, for example, the cost metric (see Remark 4.2.2). Looking ahead, in Sections 4.3.2 and Appendix A.1 we show that lower bounds in the combinatorial model imply lower bounds on the number of ciphertexts sent in ME or CGKA schemes in the symbolic model.

High-level structure and evolution of the group. An instantiation of the combinatorial model consists of two integers $n_{\max}, t_{\max} \in \mathbb{N}$, a set $G_0 \subseteq [n_{\max}]$, sequences of sets $(D_t, A_t)_{t=1}^{t_{\max}}$, and a sequence of collections of sets $(\mathcal{S}_t)_{t=0}^{t_{\max}}$. Here $[n_{\max}]$ represents the universe of users, $t_{\max}$ the number of rounds, and G_0 the initial group. For $t \in [t_{\max}]$, the sets D_t and A_t represent the users removed and added from and to the group, respectively. Accordingly, for $t \geq 1$ we inductively define the group in round t as

$$G_t := (G_{t-1} \cup A_t) \setminus D_t.$$

To make the additions and removals to and from the group meaningful we impose the requirement that $D_t \subseteq G_{t-1}$ and $A_t \subseteq [n_{\max}] \setminus G_{t-1}$ for all $t \geq 1$. Regarding the removed users, we will even impose the stronger requirement that they are never added back to the group, i.e, that for all $t \in [t_{\max}]$ we have that

$$A_t \subseteq [n_{\max}] \setminus \Big(G_{t-1} \cup \bigcup_{t'=1}^{t-1} D_{t'} \Big).$$

Looking ahead, this requirement will be necessary to formally justify our cost function. Consider the scenario where every even round user u replaces user v, and vice versa in odd rounds. Then, after the first couple of rounds, no more communication is required, as all users could simply switch between 2 previously established group keys, essentially allowing the repeated replacement of 2 users for free when considering the amortized cost over many rounds. Our restriction above, thus, allows us to get around artificial examples of this kind.

Associated set system and cost function. The final component of the combinatorial model is a sequence $(\mathcal{S}_t)_{t=0}^{t_{\max}}$, where every $\mathcal{S}_t \subseteq 2^{[n_{\max}]}$ is a collection of sets of users. We will refer to $\mathcal{S}_t$ as the *set system* at time t. The intuition behind a set S being part of $\mathcal{S}_t$ is that, in the corresponding ME or CGKA scheme, there exists a key that is secure in round t and known to exactly the users contained in S. More precisely, this means that the key is derivable from the (current or any prior) internal state of every user $u \in S$, but cannot be recovered from the sent protocol messages as well as the current and previous states of users not in S.

To capture the correctness and security of ME and CGKA schemes the set systems $\mathcal{S}_t$ in an instantiation of the combinatorial model must satisfy the following two properties:

(i) $G_t \in \mathcal{S}_t$ for all $t \in [t_{\max}]$. This corresponds to all group members in round t agreeing on a secure key.

(ii) $S \subseteq G_t$ for all sets $S \in \mathcal{S}_t$. This property represents that all keys known to users not in G_t are considered insecure in round t.

Finally, we associate a cost to each round in an instantiation of the combinatorial model. The cost of round t is given by the sum of two terms; the first essentially being the size of a minimum cover of the new group G_t with respect to set system $\mathcal{S}_{t-1}$ of round $t-1$, and the second essentially being the number of sets in $\mathcal{S}_{t-1}$ no longer present in $\mathcal{S}_t$ due to the removal of the users in D_t. Intuitively, the first summand corresponds to a lower bound on the number of ciphertexts that have to be send in round t in order to establish the group key K_t and the second summand to keys established in previous rounds that are no longer secure, but were established at the cost of sending at least one ciphertext (see Remark 4.2.2).

Definition 4.2.1. *Let* $\left((n_{\max}, t_{\max}, G_0), (D_t, A_t)_{t=1}^{t_{\max}}, (\mathcal{S}_t)_{t=0}^{t_{\max}}\right)$ *be an instantiation of the combinatorial model. We define the cost of round* 0 *as*

$$\mathrm{Cost}(0) = \mathrm{SizeMinCov}_{=}(G_0, \{\{u\} : u \in G_0\}) - 1 = |G_0| - 1,$$

and for $t \geq 1$ *we define the cost of round* t *as*

$$\begin{aligned}\mathrm{Cost}(t) = \mathrm{SizeMinCov}_{=}(G_t, \mathcal{S}_{t-1} \cup \{\{u\} : u \in G_t\}) - 1 \\ + |\{S \in \mathcal{S}_{t-1} : S \cap D_t \neq \emptyset \text{ and } |S| > 1\}|.\end{aligned}$$

Looking ahead, we will show that $\sum_{t=0}^{t_{\max}} \mathrm{Cost}(t)$ in the symbolic model corresponds to a lower bound on the number of ciphertexts sent as part of protocol messages over the whole execution of the experiment, albeit with a loss of a factor of 3. Before proving our lower bound in the combinatorial model we discuss some similarities and differences to the combinatorial model used in Chapter 3, which is used to derive lower bounds on the communication cost of recovering from state corruptions in CGKA by means of concurrent key updates.

Remark 4.2.2. *While the structure of our combinatorial model closely resembles the one of the previous chapter, it differs from it in the following aspects.*

(a) Our model allows for additions and removals of users to and from the group.

(b) We work with a different cost function. The cost metric used in Section 3.3.1 for set $S \in \mathcal{S}_t \setminus \mathcal{S}_{t-1}$ *and round* t *(following the minimum cover approach discussed in the introduction) essentially quantifies the communication cost required in round* t *to add* S *to the set system. The cost function we use in this work additionally takes the cost of sets being eliminated from* $\mathcal{S}_{t-1}$ *into account. Accordingly* $\mathrm{Cost}(t)$ *is not to be understood as the communication sent in the current round, but instead also takes communication that already occurred in prior rounds into account.*

(c) *In Section 3.3.1 we also connect the cost of adding a set S to $\mathcal{S}_t$ to a minimum cover with respect to the previous set system $\mathcal{S}_{t-1}$, However, in Section 3.3.1 we use a relaxed definition of minimum cover, which requires S to be covered by a union of sets, not necessarily to be* equal *to the union. The intuition behind this is that, in this chapter, instead of security against an adversary corrupting users, we want to protect the group key against the users themselves as soon as they have been removed from the group, even if they stored all keys they previously had access to. Phrased differently, if a secret/symmetric key is communicated to a user at any point in time, we assume it remains known to that user for the remainder of the experiment. Accordingly, now we define the set system $\mathcal{S}_t$ by associating to a (secure) key the set of users in G_t which, at any point in time until round t, had access to the key. Since (except for the user generating the key from fresh randomness) all other users in the corresponding set must have learned it from a ciphertext encrypted under a secure key already known to them, the corresponding communication cost must be at least the size of a minimum cover of the set with respect to the previous set system (in the stronger sense of Definition 2.1.1).*

On the other hand, in Section 3.3.1 we associated to a key the set of users who are able to recover the key from their states since the last corruption before round t, effectively allowing users to forget keys they knew in some prior round and leading to a relaxed definition of minimum cover.

4.2.2 Lower Bound for Batched Replacements of Users

We now prove a bound on the communication complexity of batched replacement of users in the combinatorial model. It essentially states that the prior multicast encryption schemes batching dynamic operations [LYGL01, SM03] and the MLS continuous group-key agreement standard [BBR+23] (the latter with respect to fair-weather communication complexity) are optimal up to a small constant factor.

On a technical level, we define two families of subsets of G_t, which essentially correspond to the two contributors to the cost function, i.e., the sets forming a minimum cover of the new group G_t with respect to the previous set system, and the sets containing at least one user removed in the current round, respectively. We then observe that said families satisfy the disjointedness condition required to apply the Bollobás Set Pairs Inequality. This allows us to lower bound their sizes, which after some calculations implies the desired bound.

We first prove an implication of the Bollobás Set Pairs Inequality (Lemma 2.1.4).

Lemma 4.2.3. *Let $\mathcal{X} = \{X_1, X_2, \ldots, X_m\}$ and $\mathcal{Y} = \{Y_1, Y_2, \ldots, Y_m\}$ be families of subsets of a finite set Z such that $X_i \cap Y_j \neq \emptyset$ if and only if $i, j \in [m]$ are distinct.*

Then

$$\sum_{i=1}^{m} (|X_i| + |Y_i|) \geq m \ln m.$$

Proof. It follows directly from Lemma 2.1.4 that

$$\sum_{i=1}^{m} \binom{|X_i| + |Y_i|}{|X_i|}^{-1} \leq 1,$$

which, using the bound $\binom{n}{k} \leq \left(\frac{en}{k}\right)^k$, implies

$$\begin{aligned} 1 &\geq \sum_{i=1}^{m} \binom{|X_i| + |Y_i|}{|X_i|}^{-1} \geq \sum_{i=1}^{m} \left(\frac{e\,(|X_i| + |Y_i|)}{|X_i|}\right)^{-|X_i|} \\ &= \sum_{i=1}^{m} e^{-|X_i|} \cdot \left(1 + \frac{|Y_i|}{|X_i|}\right)^{-|X_i|}. \end{aligned}$$

Now, using that $(1 + x/n)^{-n} \geq e^{-x}$ and by multiplying by $1/m$ we obtain that

$$\frac{1}{m} \geq \frac{1}{m} \sum_{i=1}^{m} e^{-|Y_i|} \cdot e^{-|X_i|} \geq \frac{1}{m} \sum_{i=1}^{m} e^{-(|Y_i| + |X_i|)}.$$

By the inequality of arithmetic and geometric means (Proposition 2.1.3) we have that

$$\frac{1}{m} \geq \left(\prod_{i=1}^{m} e^{-(|X_i| + |Y_i|)}\right)^{1/m} = \left(\prod_{i=1}^{m} e^{(|X_i| + |Y_i|)}\right)^{-1/m},$$

which by taking $\ln$ gives the desired result of $\sum_{i=1}^{m} (|X_i| + |Y_i|) \geq m \ln m$. □

We are now able to show that in the combinatorial model replacing a set of d users chosen uniformly at random in a group of size n has cost at least $d \cdot \ln(n/d)$ in expectation. The bound holds in an amortized sense, i.e., even if the experiment is repeated for several rounds. More formally, we obtain the following.

Theorem 4.2.4. *Let $n, t_{\max}, n_{\max} \in \mathbb{N}$ and $(d_t)_{t=1}^{t_{\max}}$ such that $d_t \in \mathbb{N}$ with $d_t \leq n$ for all t. Consider an instantiation of the combinatorial model with respect to $(n_{\max}, t_{\max}, G_0)$ and $(D_t, A_t)_{t=1}^{t_{\max}}$ where $|G_0| = n$ and, for all t, the set D_t of removed users is sampled uniformly at random from the set $\{D \subseteq G_{t-1} \mid |D| = d_t\}$, and A_t can be arbitrary according to the restrictions $A_t \subseteq [n_{\max}] \setminus (G_{t-1} \cup \bigcup_{t'=1}^{t-1} D_{t'})$ and $|A_t| = |D_t|$. Then it holds that*

$$\mathbb{E}\left[\sum_{t=0}^{t_{\max}} \mathrm{Cost}(t)\right] \geq \sum_{t=1}^{t_{\max}} \left(d_t \ln\left(\frac{n}{d_t}\right) - 1\right),$$

where the expectation is taken over the choice of $(D_t)_t$*. In particular, if* $d_t = d$ *for all* t*, then*

$$\mathbb{E}\left[\sum_{t=0}^{t_{\max}} \mathrm{Cost}(t)\right] \geq t_{\max} \cdot \left(d \cdot \log\left(\frac{n}{d}\right) - 1\right).$$

Proof. Since $|D_t| = |A_t| = d_t$ for all $t > 0$, we have $|G_t| = n$ for all $t \geq 0$. We first consider the cost of a single round $t \in [t_{\max}]$. By definition $G_t = (G_{t-1} \cup A_t) \setminus D_t$, where $D_t \subset G_{t-1}$, and $G_{t-1} \cap A_t = \emptyset$. Therefore we get that $G_t \cap G_{t-1} = G_{t-1} \setminus D_t = G_t \setminus A_t$ and

$$\begin{aligned}
&\mathrm{SizeMinCov}_{=}(G_t, \mathcal{S}_{t-1} \cup \{\{u\} : u \in G_t\}) \\
=&\,\mathrm{SizeMinCov}_{=}(G_t \setminus A_t, \mathcal{S}_{t-1} \cup \{\{u\} : u \in G_t \setminus A_t\}) + |\{\{u\} : u \in A_t\}| \\
=&\,\mathrm{SizeMinCov}_{=}(G_t \setminus A_t, \mathcal{S}_{t-1} \cup \{\{u\} : u \in G_t \setminus A_t\}) + d_t \\
=&\,\mathrm{SizeMinCov}_{=}(G_{t-1} \setminus D_t, \mathcal{S}_{t-1} \cup \{\{u\} : u \in G_{t-1} \setminus D_t\}) + d_t \qquad (4.1)
\end{aligned}$$

as $S \cap A_t = \emptyset$ for all $S \in \mathcal{S}_{t-1}$ (as $S \subseteq G_{t-1}$ by Property (ii) of the combinatorial model).

Now for each possible subset $D_t \subseteq G_{t-1}$ such that $|D_t| = d_t$, consider the sets

$$\begin{aligned}
X_{D_t} &= \{S \in \mathcal{S}_{t-1} \mid S \cap D_t \neq \emptyset\} \cup \{\{u\} \colon u \in D_t\} \\
&= \{S \in \mathcal{S}_{t-1} \mid S \cap D_t \neq \emptyset \text{ and } |S| > 1\} \cup \{\{u\} \colon u \in D_t\}. \qquad (4.2)
\end{aligned}$$

Further, let Y_{D_t} denote any minimum cover of $G_{t-1} \setminus D_t$ with respect to $\mathcal{S}_{t-1} \cup \{\{u\} \colon u \in G_{t-1} \setminus D_t\}$. Such a minimum cover always exists since $G_{t-1} \setminus D_t$ is actually covered by sets in $\mathcal{S}_{t-1} \cup \{\{u\} \colon u \in G_{t-1} \setminus D_t\}$. Note that Y_{D_t} also is a minimum cover of $G_{t-1} \setminus D_t$ with respect to $(\mathcal{S}_{t-1} \cup \{\{u\} \colon u \in G_{t-1} \setminus D_t\}) \setminus X_{D_t}$, and that by Equation 4.1 we have that

$$\mathrm{SizeMinCov}_{=}(G_t, \mathcal{S}_{t-1} \cup \{\{u\} : u \in G_t\}) = |Y_{D_t}| + d_t. \qquad (4.3)$$

We claim that $X_{D_t} \cap Y_{D'_t} = \emptyset$ if and only if $D_t = D'_t$. Take the case of $D_t = D'_t$; if $S \in Y_{D_t}$, then by definition of Y_{D_t}, $S \notin X_{D_t}$, thus $X_{D_t} \cap Y_{D_t} = \emptyset$. Now for the case of $D_t \neq D'_t$, there must be a user u such that $u \in D_t, u \notin D'_t$. Since $Y_{D'_t}$ covers $G_{t-1} \setminus D'_t$, there must exist $S \in Y_{D'_t}$ such that $u \in S$. Thus $S \in X_{D_t}$. Hence $X_{D_t} \cap Y_{D'_t} \neq \emptyset$.

Using Lemma 4.2.3 we obtain

$$\frac{1}{\binom{n}{d_t}} \sum_{D_t \subseteq G_{t-1}, |D_t| = d_t} |X_{D_t}| + |Y_{D_t}| \geq \frac{1}{\binom{n}{d_t}} \binom{n}{d_t} \ln \binom{n}{d_t} \geq d_t \ln \frac{n}{d_t}. \qquad (4.4)$$

Note that Equation 4.4 gives a lower bound on the expectation of $|X_{D_t}| + |Y_{D_t}|$ if the set D_t is chosen uniformly at random. To make this formal, given n, $n_{\max}$, $t_{\max}$,

$(d_t)_{t=1}^{t_{\max}}$, and G_0, we define a sequence of random variables $(\mathbf{D}_t, \mathbf{A}_t, \mathbf{G}_t)_{t=1}^{t_{\max}}$ all taking values in $2^{[n_{\max}]}$ where $\mathbf{G}_1 := (G_0 \cup \mathbf{A}_1) \setminus \mathbf{D}_1$ and for $t \geq 2$

$$\mathbf{G}_t := (\mathbf{G}_{t-1} \cup \mathbf{A}_t) \setminus \mathbf{D}_t.$$

The sequence is distributed as follows. The set $\mathbf{D}_1$ of users removed in the first round is distributed uniformly over $\{D_1 \subseteq G_0 : |D_1| = d_1\}$ and $\mathbf{A}_1$ can distributed arbitrarily over $\{A_1 \subseteq [n_{\max}] \setminus G_0 : |A_1| = d_1\}$. Now, conditioned on $\mathbf{D}_{t'} = D_{t'}$, $\mathbf{A}_{t'} = A_{t'}$, and $\mathbf{G}_{t'} = G_{t'}$ for $t' \in [t-1]$, the random variables $\mathbf{D}_t$ is distributed uniformly over $\{D_t \subseteq G_{t-1} : |D_t| = d_t\}$ and $\mathbf{A}_t$ can be distributed arbitrarily over $\{A_t \subseteq [n_{\max}] \setminus (G_{t-1} \cup \bigcup_{t'=1}^{t-1} D_{t'}) : |A_t| = d_t\}$.

If we consider the expected cost of round t (see Definition 4.2.1) with respect to the sequence of adds and removes given by $(\mathbf{D}_t, \mathbf{A}_t)_{t=1}^{t_{\max}}$ we obtain by Equations 4.2, 4.3, and 4.4 that

$$\begin{aligned}
\mathbb{E}[\mathrm{Cost}(t)] &= \mathbb{E}[|\{S \in \mathcal{S}_{t-1} : S \cap \mathbf{D}_t \neq \emptyset \text{ and } |S| > 1\}|] \\
&\quad + \mathbb{E}[\mathrm{SizeMinCov}_{=}(\mathbf{G}_t, \mathcal{S}_{t-1} \cup \{\{u\} : u \in \mathbf{G}_t\})] - 1 \\
&\geq \mathbb{E}[|X_{\mathbf{D}_t}| - d_t + |Y_{\mathbf{D}_t}| + d_t] - 1 \\
&\geq d_t \ln \frac{n}{d_t} - 1.
\end{aligned}$$

Now, the theorem's statement follows by linearity of expectation. □

4.3 Lower Bound for Multicast Encryption in the Symbolic Model

In this section we define multicast encryption in the symbolic model and show that the lower bound on batched replacement of users in the combinatorial model of Section 4.2.2 carries over. Section 4.3.1 specifies the symbolic model and provides syntax for multicast encryption, Section 4.3.2 proves the corresponding bound.

4.3.1 Multicast Encryption in the symbolic model

Considered building blocks. We now define syntax for *multicast encryption* (ME) in a symbolic model in the style of Dolev and Yao [DY83]. In models of this type, keys and ciphertexts of cryptographic primitives are seen as symbolic variables, which are generated according to grammar rules, and can be derived from sets of other symbolic variables according to an entailment relation $\vdash$, which itself models ideal security notions of the used cryptographic building blocks. Throughout this section we will denote symbolic variables in typewriter font to distinguish them from non-symbolic inputs and

outputs of algorithms. Further, single variables are depicted using lower case letters, sets of variables using upper case letters.

In our symbolic treatment of multicast encryption we consider symbolic variables of the following two types: (pseudo)random strings denoted by $\mathtt{r}$ and messages $\mathtt{m}$. The former will also serve as keys of symmetric encryption schemes and, in this context, we will often denote them by $\mathtt{k}$. Similarly, ciphertexts of symmetric encryption are of message type and we will often denote them by $\mathtt{c}$. We consider ME schemes constructed from symmetric encryption schemes (SE), pseudorandom generators (PRG), pseudorandom functions (PRF), dual pseudorandom functions (dPRF) and secret sharing defined according to the following syntax;

- A symmetric encryption scheme $\mathsf{SE} = (\mathsf{SE.Enc}, \mathsf{SE.Dec})$ specifies an encryption algorithm $\mathsf{SE.Enc}(\mathtt{k}, \mathtt{m})$ that, on input a symmetric key $\mathtt{k}$ of type $\mathtt{r}$ and a message $\mathtt{m}$, returns a ciphertext $\mathtt{c}$ that is of message type. Deterministic decryption algorithm $\mathsf{SE.Dec}(\mathtt{k}, \mathtt{c})$, on input symmetric key $\mathtt{k}$ and ciphertext $\mathtt{c}$, returns a message $\mathtt{m}$.

  We require perfect correctness, i.e., $\mathsf{SE.Dec}(\mathtt{k}, \mathsf{SE.Enc}(\mathtt{k}, \mathtt{m})) = \mathtt{m}$ for all $\mathtt{k}$ and $\mathtt{m}$.

- A pseudorandom generator $\mathsf{PRG}(\mathtt{r})$ takes a random string $\mathtt{r}$ as input and returns a value $(\mathtt{r}_1, \mathtt{r}_2)$ consisting of two pseudorandom strings. For simplicity we restrict ourselves to PRGs with stretch 2. Note that PRGs with larger stretch can easily be built from these using standard methods.

- A pseudorandom function $\mathsf{PRF}(\mathtt{r}, ad)$, on input random string $\mathtt{r}$ and non-symbolic associated data ad, returns a pseudorandom string $\mathtt{r}'$.

- A dual pseudorandom function $\mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2)$, takes two random strings $\mathtt{r}_1, \mathtt{r}_2$ as input and returns a pseudorandom string $\mathtt{r}' = \mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2) = \mathsf{dPRF}(\mathtt{r}_2, \mathtt{r}_1)$.

- A secret sharing scheme given by two algorithms S and R. On input a message $\mathtt{m}$, S outputs a set of s many shares $\mathsf{S}(\mathtt{m}) = \{\mathsf{S}_i(\mathtt{m})\}_{i \in [s]}$ of type message and the original message can be recovered given some subset of shares as determined by an access structure $\Gamma \subseteq 2^{[s]}$, namely, for every $I \in \Gamma$, $\mathsf{R}(I, \{\mathsf{S}_i(\mathtt{m})\}_{i \in I}) = \mathtt{m}$.

We now describe the grammar rules and entailment relation.

Variable type		Grammar rule
$\mathtt{r}$	$\leftarrow$	terminal type, $\mathsf{PRG}(\mathtt{r}), \mathsf{PRF}(\mathtt{r}), \mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2)$
$\mathtt{m}$	$\leftarrow$	$\mathtt{r}$, $\mathsf{SE.Enc}(\mathtt{k}, \mathtt{m}), \mathsf{S}_i(\mathtt{m})$
Entailment relation		
$\mathtt{m} \in \mathtt{M}$	$\Rightarrow$	$\mathtt{M} \vdash \mathtt{m}$
$\mathtt{M} \vdash \mathtt{r}$	$\Rightarrow$	$\mathtt{M} \vdash \mathsf{PRG}(\mathtt{r}) = (\mathtt{r}_1, \mathtt{r}_2)$
$\mathtt{M} \vdash \mathtt{r}$	$\Rightarrow$	$\forall ad\colon \mathtt{M} \vdash \mathsf{PRF}(\mathtt{r}, ad)$
$\mathtt{M} \vdash \mathtt{r}_1, \mathtt{r}_2$	$\Rightarrow$	$\mathtt{M} \vdash \mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2)$
$\mathtt{M} \vdash \mathtt{r}, \mathtt{m}$	$\Rightarrow$	$\mathtt{M} \vdash \mathsf{SE.Enc}(\mathtt{r}, \mathtt{m})$
$\mathtt{M} \vdash (\mathtt{r}, \mathtt{c}) : \mathtt{c} = \mathsf{SE.Enc}(\mathtt{r}, \mathtt{m})$	$\Rightarrow$	$\mathtt{M} \vdash \mathtt{m}$
$\exists I \in \Gamma\colon \mathtt{M} \vdash \{\mathsf{S}_i(\mathtt{m})\}_{i \in I}$	$\Rightarrow$	$\mathtt{M} \vdash \mathtt{m}$

The grammar rules state that (pseudo)random coins can either be directly sampled or generated using a PRG or be obtained as the image of PRF or dPRF; that the encryption algorithm of SE, on input a key of type $\mathtt{r}$ and message $\mathtt{m}$, generates a ciphertext; and that messages can be of arbitrary type. The entailment relation states that every symbolic variable contained in a set $\mathtt{M}$ can be recovered from the set. Further, it models ideal PRG security, stating that outputs of a PRG can only be recovered if given access to the respective input. PRF security is also modeled in the same way, which means that there is no significant difference between PRGs and PRFs in the symbolic model. The security of a dPRF is modeled by requiring that outputs of a dPRF can only be recovered given access to both inputs. Similarly, ideal SE security, i.e., that ciphertexts can only be decrypted if given access to the corresponding key. For a more detailed explanation and examples of the symbolic model we refer to [MP04]. The security of the secret sharing scheme corresponds to the requirement that the original message can be recovered from a set of shares as determined by the access structure. Given a set $\mathtt{M}$ of symbolic variables we denote the set of all variables derivable from it using the entailment relation by $\mathsf{Der}(\mathtt{M})$, i.e.,

$$\mathtt{m} \in \mathsf{Der}(\mathtt{M}) \text{ exactly if } \mathtt{M} \vdash \mathtt{m}.$$

If $\mathtt{M}_1, \mathtt{M}_2$ are two sets of symbolic variables, we use the notation $\mathsf{Der}(\mathtt{M}_1, \mathtt{M}_2) = \mathsf{Der}(\mathtt{M}_1 \cup \mathtt{M}_2)$.

Multicast encryption syntax. A multicast encryption scheme essentially allows a central authority to provide a dynamically changing group of users with a group key by sending protocol messages via a broadcast channel. The main goal being to use protocol messages that are as small as possible while still achieving correctness and security, i.e., that group members in every round agree on a group key that, however,

cannot be recovered from the sent protocol messages even if given access to all previous states of non-members of the group.

As our goal in this work is to prove lower bounds and these are easier to state by keeping the group size constant over all rounds, we work with a simplified syntax only allowing for replacements of users, but not arbitrary removes and adds. We essentially follow [MP04, AAB+21], who analyzed the communication cost of multicast encryption for replacing a single user per round in the setting of a single group and of a system of potentially overlapping groups, respectively. The main difference in this work is that we allow for batched operations, i.e, replacing a set of more than one user at a time. Note that such a replacement can always be implemented by both removing and adding parties and thus our bounds in particular also hold for schemes allowing these operations.

In the following we split the inputs and outputs of algorithms into a symbolic part, i.e., sets of symbolic variables, and a non-symbolic part containing, e.g., user identifiers. As already stated above, the former variables are depicted in typewriter font, the latter in italics.

A multicast encryption scheme ME specifies algorithms ME.Setup, ME.Init, ME.Repl, ME.Proc, ME.Key with the following syntax;[2]

- ME.Setup($n_{\max}$; R) takes as input $n_{\max}$, the universe of users, and the set of random coins R. It sets up the initial state $(\mathtt{ST}_u^{-1}, \mathsf{st}_u^{-1})$ for every user $u \in [n_{\max}]$. The symbolic part of the initial state, namely, $\mathtt{ST}_u^{-1}$ is subject to the requirements that $\mathtt{ST}_u^{-1} = \{\mathtt{k}_u^{-1}\}$ where $\mathtt{k}_u^{-1}$ is of type random string and $\mathtt{ST}_u^{-1} \cap \mathsf{Der}(\bigcup_{v \in [n_{\max}] \setminus \{u\}} \mathtt{ST}_v^{-1}) = \emptyset$. Similar assumptions are also made in [AAB+21, MP04]. For instance, in [MP04] it is assumed that each user is assigned exactly one key that cannot be derived from those assigned to other users, while in [AAB+21] users are additionally assigned a key for each subgroup they belong to with the property that they cannot be derived from keys of users that do not belong to the corresponding subgroup. Making this kind of assumption is justified. Otherwise one could consider schemes in which communication is artificially reduced. For instance, generating two keys $k_{S,1}, k_{S,2}$ for each possible subset $S \in 2^{[n_{\max}]}$ during setup and instead of giving users these keys, they would get a ciphertext $c_S = \mathsf{Enc}(k_{S,1}, k_{S,2})$ for each set they are a member of and then the key $k_{S,1}$ would be sent in the clear in a later round.[3]

[2] One can consider the possibility that some of these algorithms be randomized by also including non-symbolic randomness as an input and the results would hold for any choice of non-symbolic randomness.

[3] The restriction that $\mathtt{ST}_u^{-1}$ consists of just one element can be weakened if one requires that it only consists of random coins and for all coins $\mathtt{r} \in \mathtt{ST}_u^{-1}$ it holds that $\mathtt{r} \notin \mathsf{Der}((\mathtt{ST}_u^{-1} \setminus \{\mathtt{r}\}) \cup$

- ME.Init$(G_0; \mathtt{R})$, on input the first group G_0 and a set of random coins $\mathtt{R}$, outputs a control message $(\mathtt{M}, \mathsf{M})$. Further, it implicitly sets up the initial group key $\mathtt{k}^0$.

- ME.Repl$(A_t, D_t; \mathtt{R})$ allows replacing a set D_t of group members by a set A_t of new users. In round t it takes as input the set $A_t \in [n_{\max}] \setminus G_{t-1}$ of users to be added to the group, $D_t \subseteq G_{t-1}$, the set of users to be removed, and a set of random coins $\mathtt{R}$. We require that $|A_t| = |D_t|$. The output of the algorithm is a control message $(\mathtt{M}, \mathsf{M})$. Further, the algorithm implicitly sets up the t^{th} group key $\mathtt{k}_t$.

- Deterministic algorithm ME.Proc$((\mathtt{ST}_u^{t-1}, \mathsf{st}_u^{t-1}), (\mathtt{M}, \mathsf{M}))$ takes as input, in round t, a user's internal state $(\mathtt{ST}_u^{t-1}, \mathsf{st}_u^{t-1})$ as well as a control message $(\mathtt{M}, \mathsf{M})$ (either output by ME.Init or by ME.Repl). It returns the user's updated state $(\mathtt{ST}_u^t, \mathsf{st}_u^t)$.

- Deterministic algorithm ME.Key$(\mathtt{ST}_u^t, \mathsf{st}_u^t)$, on input user u's state at the end of round t, returns the t^{th} group key $\mathtt{k}^t$.

The algorithms ME.Setup, ME.Init, ME.Repl are run by the central authority and it is understood that they also take as input all users' states and all messages despite this not being explicitly indicated. We require that symbolic outputs of algorithms are derivable from the symbolic part of their inputs, e.g., if $(\mathtt{ST}_u^t, \mathsf{st}_u^t) \leftarrow$ ME.Proc$((\mathtt{ST}_u^{t-1}, \mathsf{st}_u^{t-1}), (\mathtt{M}, \mathsf{M}))$ then it must hold that $(\mathtt{ST}_u^{t-1}, \mathtt{M}) \vdash \mathtt{ST}_u^t$. Moreover, we also require that only a finite number of derivation steps is needed. Further and for brevity, while in the following we will make the users removed from, and added to, the group explicit, we will often drop the non-symbolic parts of protocol messages and users' states, and simply write $\mathtt{ST}_u^t \leftarrow$ ME.Proc$(\mathtt{ST}_u^{t-1}, \mathtt{M})$.

Correctness and security. We capture security and correctness of multicast-encryption schemes in the symbolic model simultaneously with the game in Figure 4.2. Similar to the experiment in the combinatorial model, the game is parameterized by a tuple $(n_{\max}, t_{\max}, G_0)$ which specifies the initialization of the group and a sequence $(A_t, D_t)_{t=1}^{t_{\max}}$ of users added to, and removed from, the group. In round 0 the states of all users in $[n_{\max}]$ are set up using ME.Setup and the group G_0 is initialized using ME.Init and ME.Proc. Then security and correctness are verified for the first round, meaning that (a) all users in G_0 have access to the (unique) group key $\mathtt{k}^0$, and (b) the non-members of G_0 are not able to derive $\mathtt{k}^0$ from their internal states and the protocol message $\mathtt{M}^0$ sent in round 0 even if colluding. If both checks succeed, the game proceeds in rounds t. In each of them the users in A_t are added to the group and the users in D_t removed

$\cup_{v \in [n_{\max}] \setminus \{u\}} \mathtt{ST}_v^{-1}$). This is done in Appendix A.1 in the case of CGKA schemes and it applies, mutatis mutandis, to the case of ME. But it comes at the cost of an additional step in the proof of the lower bound, so we leave it for the appendix.

Game $\mathrm{SEC}^{\mathsf{ME}}((n_{\max}, t_{\max}, G_0), (A_t, D_t)_{t=1}^{t_{\max}})$

00 sample $\mathtt{R}_{-1}, \mathtt{R}_0$
01 $(\mathtt{ST}_u^{-1})_{u\in[n_{\max}]} \leftarrow \mathsf{ME.Setup}(n_{\max}; \mathtt{R}_{-1})$
02 $\mathtt{M}_0 \leftarrow \mathsf{ME.Init}(G_0; \mathtt{R}_0)$
03 for $u \in G_0$:
04 $\quad \mathtt{ST}_u^0 \leftarrow \mathsf{ME.Proc}(\mathtt{ST}_u^{-1}, \mathtt{M}_0)$
05 $\quad \mathtt{k}_u^0 \leftarrow \mathsf{ME.Key}(\mathtt{ST}_u^0)$
06 $\quad \mathtt{k}^0 \leftarrow \mathtt{k}_u^0$
07 for $u \in [n_{\max}] \setminus G_0$:
08 $\quad \mathtt{ST}_u^0 \leftarrow \mathtt{ST}_u^{-1}$
09 if $\exists u \in G_0 : \mathtt{k}_u^0 \neq \mathtt{k}^0$:
10 $\quad$ return 0 \\ disagreement on key
11 if $\mathtt{k}^0 \in \mathsf{Der}(\mathtt{M}_0, ((\mathtt{ST}_u^{t'})_{t'=-1}^0)_{u\notin G_0})$:
12 $\quad$ return 0 \\ group key insecure
13 for $t = 1, \ldots, t_{\max}$:
14 $\quad \mathsf{ROUND}(A_t, D_t)$
15 return 1

Oracle $\mathsf{ROUND}(A_t, D_t)$

16 require $D_t \subseteq G_{t-1} \wedge A_t \subseteq [n_{\max}] \setminus G_{t-1}$
17 $G_t \leftarrow (G_{t-1} \cup A_t) \setminus D_t$
18 sample $\mathtt{R}_t$
19 $\mathtt{M}_t \leftarrow \mathsf{ME.Repl}(A_t, D_t; \mathtt{R}_t)$
20 for $u \in G_t$:
21 $\quad \mathtt{ST}_u^t \leftarrow \mathsf{ME.Proc}(\mathtt{ST}_u^{t-1}, \mathtt{M}_t)$
22 $\quad \mathtt{k}_u^t \leftarrow \mathsf{ME.Key}(\mathtt{ST}_u^t)$
23 $\quad \mathtt{k}^t \leftarrow \mathtt{k}_u^t$
24 for $u \in [n_{\max}] \setminus G_t$:
25 $\quad \mathtt{ST}_u^t \leftarrow \mathtt{ST}_u^{t-1}$
26 if $\exists u \in G_t : \mathtt{k}_u^t \neq \mathtt{k}^t$:
27 $\quad$ return 0 \\ disagreement on key
28 if $\mathtt{k}^t \in \mathsf{Der}((\mathtt{M}_{t'})_{t'=0}^t, ((\mathtt{ST}_u^{t'})_{t'=-1}^t)_{u\notin G_t})$:
29 $\quad$ return 0 \\ group key insecure

Figure 4.2: Symbolic security and correctness game for multicast encryption scheme ME. In Line 16 if the condition after **require** is not met the game aborts and outputs 1, meaning that the execution of the game is considered to have been secure.

from it using $\mathsf{ME.Repl}(A_t, D_t)$, and all current group members are made to process the resulting protocol message $\mathtt{M}^t$ with ME.Proc. Again it is checked that the round satisfies correctness and security. The former means that all users in G_t derive the same group key for round t, which can essentially be seen as the requirement that

$$\exists \mathtt{k}^t : \mathtt{k}^t = \mathsf{ME.Key}(\mathtt{ST}_u^t) \text{ for all } u \in G_t.$$

Note that this in particular implies $\mathtt{k}^t \in \mathsf{Der}(\mathtt{ST}_u^t)$ for all $u \in G_t$.

The latter means that, even if all non group-members never deleted their old states and collude, they are not able to recover the current group key, i.e.,

$$\mathtt{k}^t \notin \mathsf{Der}\left((\mathtt{M}_{t'})_{t'=0}^t, ((\mathtt{ST}_u^{t'})_{t'=-1}^t)_{u\in[n_{\max}]\setminus G_t}\right).$$

This notion of security only asks for post-compromise security and not forward-secrecy since we are not considering the possibility that the group key at time t can be derived from future exposures. This only strengthens our lower-bound. If one of the checks fails, the game aborts and returns 0, else it returns 1. We say that a ME scheme ME is correct and secure, if game SEC with respect to any input $(n_{\max}, t_{\max}, G_0), (A_t, D_t)_{t=1}^{t_{\max}}$ returns 1.

Useful keys and associated set system. Consider an execution of game $\mathrm{SEC}^{\mathsf{ME}}$ with respect to $(n_{\max}, t_{\max}, G_0)$ and $(A_t, D_t)_{t=1}^{t_{\max}}$. Let $t \in [t_{\max}]_0 := [t_{\max}] \cup \{0\}$ and

consider a random coin $\mathtt{r}$ that was generated in some round up to and including t. We say $\mathtt{r}$ is *useful* at time t, if

$$\mathtt{r} \notin \mathsf{Der}\left((\mathtt{M}_{t'})_{t'=0}^{t}, ((\mathtt{ST}_u^{t'})_{t'=-1}^{t})_{u\in[n_{\max}]\setminus G_t}\right),$$

which means that it cannot be derived from all protocol messages sent so far and all prior and current states of users that are not members of the group at time t. Following an approach similar to the one of Section 3.4.1, we associate to a secure coin $\mathtt{r}$ a set of users, with the important difference, however, that now the set contains all users that had access to $\mathtt{r}$ *at any point in time*.

Definition 4.3.1. *Consider an execution of security game* $\mathrm{SEC}^{\mathsf{ME}}$ *with respect to input* $(n_{\max}, t_{\max}, G_0)$ *and* $(A_t, D_t)_{t=1}^{t_{\max}}$*. Let* $t \in [t_{\max}]_0$ *and* $\mathtt{r}$ *be a random coin. We define*

$$S(t,\mathtt{r}) := \{u \in [n_{\max}] \mid \mathtt{r} \in \mathsf{Der}(\mathtt{ST}_u^{-1}, (\mathtt{M}_{t'})_{t'\le t:\ u\in G_{t'}})\}$$

It should be noted that $\mathtt{ST}_u^{t'} \subseteq \mathsf{Der}(\mathtt{ST}_u^{-1}, (\mathtt{M}_{t'})_{t'\le t:\ u\in G_{t'}})$ *for* $t' \le t$*. Further, we define the set system at time* t *as*

$$\mathcal{S}_t := \{S \subseteq [n_{\max}] \mid \exists \text{ useful coin } \mathtt{r} : S = S(t,\mathtt{r})\}.$$

We prove two Lemmas that capture how derivation works in the symbolic model and connects it to the sets defined in Definition 4.3.1.

Lemma 4.3.2. *Let* $\mathtt{r}$ *be of type random coin and useful at time* $t \in [t_{\max}]_0$*, and* u *a user such that* $u \in S(t,\mathtt{r})$*. Then (at least) one of the following cases holds.*

1. *There exist* $\mathtt{r}'$ *with* $\mathsf{PRG}(\mathtt{r}') = (\mathtt{r}_1, \mathtt{r}_2)$ *and* $i \in \{1,2\}$ *such that* $\mathtt{r} = \mathtt{r}_i$*. Further,* $\mathtt{r}'$ *is useful at time* t *and* $u \in S(t,\mathtt{r}')$*.*
2. *There exists* $\mathtt{r}'$ *and associated data* ad *such that* $\mathsf{PRF}(\mathtt{r}', ad) = \mathtt{r}$*. Further,* $\mathtt{r}'$ *is useful at time* t *and* $u \in S(t,\mathtt{r}')$*.*
3. *There exist* $\mathtt{r}_1$ *and* $\mathtt{r}_2$ *such that* $\mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2) = \mathtt{r}$*, at least one of* $\mathtt{r}_1$ *and* $\mathtt{r}_2$ *is useful at time* t*, and* $u \in S(t,\mathtt{r}_1) \cap S(t,\mathtt{r}_2)$*.*
4. $\mathtt{r} \in \mathtt{ST}_u^{-1}$
5. *There exists* $\mathtt{c} = e_0(\cdot) \circ \ldots \circ e_g(\cdot) \circ \ldots \circ e_h(\mathtt{r})$ *where* $e_i = \mathsf{SE.Enc}(\mathtt{r}_i, \cdot)$ *or* $e_i = \mathsf{S}_{i_j}(\cdot)$ *and*
   a) $\mathtt{c} \in \bigcup_{\tilde{t}\le t:\ u\in G_{\tilde{t}}} \mathtt{M}_{\tilde{t}}$*,*
   b) *if* $e_i = \mathsf{SE.Enc}(\mathtt{r}_i, \cdot)$ *and* $i \ge g+1$*,* $\mathtt{r}_i$ *is not useful at time* t*,*

c) there exists $i \in \{0, \ldots, h\}$ *such that* $e_i = \mathsf{SE.Enc}(\mathtt{r}_i, \cdot)$,

d) $e_g = \mathsf{SE.Enc}(\mathtt{r}_g, \cdot)$ *and* $\mathtt{r}_g$ *is useful at time* t*, and*

e) for all encryptions $e_i = \mathsf{SE.Enc}(\mathtt{r}_i, \cdot)$*, it holds that* $u \in S(t, \mathtt{r}_i)$.

Proof. If $\mathtt{r}$ admits a PRG pre-image $\mathtt{r}'$, $\mathtt{r}'$ must be useful at time t since $\mathtt{r}$ is. Therefore we have two possible cases depending on whether $u \in S(t, \mathtt{r}')$. If $u \in S(t, \mathtt{r}')$ we are in Case 1. If $u \notin S(t, \mathtt{r}')$, then one of the following holds:

- $\mathtt{r} \in \mathtt{ST}_u^{-1} \cup \bigcup_{t' \leq t:\, u \in G_{t'}} \mathtt{M}_{t'}$ and the fact that $\mathtt{r}$ is useful implies that $\mathtt{r} \in \mathtt{ST}_u^{-1}$.
- Or, by repeatedly applying the last two rules of the entailment relation, there exists a ciphertext $\mathtt{c} \in \mathtt{ST}_u^{-1} \cup \bigcup_{t' \leq t:\, u \in G_{t'}} \mathtt{M}_{t'}$ of the form $e_0(\cdot) \circ \ldots \circ e_g(\cdot) \circ \ldots \circ e_h(\mathtt{r})$ where each e_i is an application of S or SE.Enc such that condition *(e)* holds. By assumption $\mathtt{ST}_u^{-1}$ only contains symbols of type random coins, so $\mathtt{c} \in \bigcup_{t' \leq t:\, u \in G_{t'}} \mathtt{M}_{t'}$. Therefore there must exist at least one encryption in $\mathtt{c}$ under a useful key since $\mathtt{r}$ is useful. This shows *(c)* and *(d)*. Condition *(b)* is just a matter of choice.

The two options above correspond to Cases 4 or 5, respectively.

If $\mathtt{r}$ does not admit a PRG pre-image, we consider whether it admits a PRF pre-image $\mathtt{r}'$, which must be useful at time t since $\mathtt{r}$ is. If $u \in S(t, \mathtt{r}')$ we are in Case 2, else we are in Cases 4 or 5. If $\mathtt{r}$ does not admit a PRF pre-image, we study whether there exist $\mathtt{r}_1$ and $\mathtt{r}_2$ such that $\mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2) = \mathtt{r}$. In this case at least one of $\mathtt{r}_1$ and $\mathtt{r}_2$ must be useful at time t since $\mathtt{r}$ is. If $u \in S(t, \mathtt{r}_1) \cap S(t, \mathtt{r}_2)$, then we are in Case 3. Else we are in Cases 4 or 5. □

Repeatedly applying Lemma 4.3.2 one can obtain the following result:

Lemma 4.3.3. *Let* $\mathtt{r}$ *be of type random coin and useful at time* $t \in [t_{\max}]_0$*, and* u *a user such that* $u \in S(t, \mathtt{r})$*. Then there exists a sequence* $\{\mathtt{r}_{1,u,t}, \ldots, \mathtt{r}_{\ell_u,u,t}\}$ *such that*

6. *for all* i *the secret* $\mathtt{r}_{i,u,t}$ *is useful at time* t *and* $u \in S(t, \mathtt{r}_{i,u,t})$,
7. $\mathtt{r}_{\ell_u,u,t} = \mathtt{r}$,
8. $\mathtt{r}_{1,u,t} \in \mathtt{ST}_u^{-1}$*, and*
9. *for all* $i \in \{1, \ldots, \ell_u - 1\}$ *one of the following is true*

   a) $\mathsf{PRG}(\mathtt{r}_{i,u,t}) = (\mathtt{r}_1, \mathtt{r}_2)$ *for some* $\mathtt{r}_1$, $\mathtt{r}_2$ *such that either* $\mathtt{r}_{i+1,u,t} = \mathtt{r}_1$ *or* $\mathtt{r}_{i+1,u,t} = \mathtt{r}_2$*, or*

b) there exists ad *such that* $\mathsf{PRF}(\mathtt{r}_{i,u,t}, ad) = \mathtt{r}_{i+1,u,t}$*, or*

c) there exists $\mathtt{r}'_{i,u,t}$ *such that* $u \in S(t, \mathtt{r}'_{i,u,t})$ *and* $\mathsf{dPRF}(\mathtt{r}_{i,u,t}, \mathtt{r}'_{i,u,t}) = \mathtt{r}_{i+1,u,t}$*, or*

d) there exists a ciphertext $\mathtt{c}_{i,u,t} \in \bigcup_{\tilde{t} \le t:\ u \in G_{\tilde{t}}} \mathtt{M}_{\tilde{t}}$ *such that*

$$\mathtt{c}_{i,u,t} = e_0(\cdot) \circ \ldots \circ e_g(\cdot) \circ \ldots \circ e_h(\mathtt{r}_{i+1,u,t})$$

where all properties of Case 5 are satisfied and $\mathtt{r}_{i,u,t} = \mathtt{r}_g$ *the secret used in* $e_g = \mathsf{SE.Enc}(\mathtt{r}_g, \cdot)$*.*

Observe that ℓ_u *depends on* u, t *and* $\mathtt{r}$*, so in some cases we make this explicit and write* $\ell_{u,t,\mathtt{r}}$ *or just* $\ell_{u,t}$ *if the random coin* $\mathtt{r}$ *is clear from context.*

Proof. Let $\mathtt{r} \leftarrow \mathtt{r}$ and $\mathsf{Seq} \leftarrow \emptyset$. Repeat $(\mathtt{r}, \mathsf{Seq}) \leftarrow f(\mathtt{r}, \mathsf{Seq})$ until $\mathtt{r} = \mathsf{STOP}$ where:

$$f(\mathtt{r}, \mathsf{Seq}) = \begin{cases} \text{if we are in Case 1, do } (\mathtt{r}, \mathsf{Seq}) \leftarrow (\mathtt{r}', \{\mathtt{r}\} \cup \mathsf{Seq}), \\ \text{if we are in Case 2, do } (\mathtt{r}, \mathsf{Seq}) \leftarrow (\mathtt{r}', \{\mathtt{r}\} \cup \mathsf{Seq}), \\ \text{if we are in Case 3 and } \mathtt{r}_i \text{ is useful, do } (\mathtt{r}, \mathsf{Seq}) \leftarrow (\mathtt{r}_i, \{\mathtt{r}\} \cup \mathsf{Seq}), \\ \text{if we are in Case 4, do } (\mathtt{r}, \mathsf{Seq}) \leftarrow (\mathsf{STOP}, \{\mathtt{r}\} \cup \mathsf{Seq}), \\ \text{if we are in Case 5, do } (\mathtt{r}, \mathsf{Seq}) \leftarrow (\mathtt{r}_g, \{\mathtt{r}\} \cup \mathsf{Seq}). \end{cases}$$

By construction, Properties 6, 7, 8, as well as one of Properties 9a to 9d are clearly satisfied by Seq at every point in time. This process must end since we require that only a finite number of derivation steps is made by the ME algorithms. □

Now we follow the approach of [MP04] in order to construct a graph for each round and use it to establish a connection between the sets in $\mathcal{S}_t$ and those in $\mathcal{S}_{t-1}$ obtaining a similar result to the one in Section 3.4.1.

The sequences constructed in Lemma 4.3.3 suggest considering the following graph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ for $t \in [t_{\max} - 1]_0$. The set of nodes $\mathcal{V}_t$ is a subset of useful random coins at time t which corresponds to the elements of the sequences $\{\mathtt{k}^t_{1,u,t}, \ldots, \mathtt{k}^t_{\ell_u,u,t}\}$ associated to the group key $\mathtt{k}^t = \mathtt{k}^t_{\ell_u,u,t}$ and each user $u \in S(t, \mathtt{k}^t)$. The set of edges $\mathcal{E}_t$ consists of all pairs of the form $(\mathtt{k}^t_{i,u,t}, \mathtt{k}^t_{i+1,u,t})$.

In the case that an edge $(\mathtt{k}^t_{i,u,t}, \mathtt{k}^t_{i+1,u,t})$ is obtained from Property 9c, i.e., there exists $\mathtt{r}'_{i,u,t}$ such that $\mathsf{dPRF}(\mathtt{r}_{i,u,t}, \mathtt{r}'_{i,u,t}) = \mathtt{r}_{i+1,u,t}$ and $u \in S(t, \mathtt{r}'_{i,u,t})$, one can construct the sequence from Lemma 4.3.3 using the secret $\mathtt{r}'_{i,u,t}$ instead of $\mathtt{r}_{i,u,t}$ when both $\mathtt{r}_{i,u,t}$ and $\mathtt{r}'_{i,u,t}$ are useful at time t. If this happens, we make the same choice for all users in order to guarantee that one dPRF pre-image (Case 9c) does not result in two edges in $\mathcal{E}_t$. This is possible since $u \in S(t, \mathtt{r}_{i,u,t}) \cap S(t, \mathtt{r}'_{i,u,t})$.

If an edge $(\mathtt{k}^t_{i,u,t}, \mathtt{k}^t_{i+1,u,t})$ satisfies Properties 9a, 9b or 9c we refer to it as a trivial edge, while we refer to an edge that satisfies Property 9d as a communication edge. The graph $\mathcal{G}_t$ has some basic properties which we state in the following result.

Lemma 4.3.4. *Let* ME *be a correct and secure ME scheme. Consider an execution of game* $\mathrm{SEC}^{\mathsf{ME}}$ *on input* $(n_{\max}, t_{\max}, G_0)$ *and* $(A_t, D_t)_{t=1}^{t_{\max}}$ *such that* $D_t \subseteq G_{t-1}$ *and* $A_t \subseteq [n_{\max}] \setminus (G_{t-1} \cup \bigcup_{t'=1}^{t-1} D_t)$ *for all* $t \in [t_{\max}]$*. Let* $t \in \{0, \ldots, t_{\max} - 1\}$ *and* $\mathtt{k}^t$ *denote the group key at time* t *output by* ME.Key *in Line 22. Then the following properties of the graph* $\mathcal{G}_t$ *are true.*

10. *For every* $u \in S(t, \mathtt{k}^t)$*, the node* $\mathtt{k}^t_{1,u,t}$ *has no incoming edges and* $\mathtt{k}^t_{1,u,t} \neq \mathtt{k}^t_{1,v,t}$ *for all* $u \neq v$*. Actually it holds that* $S(t, \mathtt{k}^{-1}_u) = \{u\}$*.*

11. *For every node* $\mathtt{k}^t_{i,u,t}$ *there exists at most one node* $\mathtt{r}$ *such that* $\mathsf{PRG}(\mathtt{r}) = (\mathtt{r}'_1, \mathtt{r}'_2)$ *and* $\mathtt{r}'_j = \mathtt{k}^t_{i,u,t}$ *for some* $j \in \{1, 2\}$*, or that* $\mathsf{PRF}(\mathtt{r}, ad) = \mathtt{k}^t_{i,u,t}$ *for some* ad*, or that* $\mathsf{dPRF}(\mathtt{r}, \mathtt{r}') = \mathtt{k}^t_{i,u,t}$ *or* $\mathsf{dPRF}(\mathtt{r}', \mathtt{r}) = \mathtt{k}^t_{i,u,t}$ *for some* $\mathtt{r}'$ *(where* $\mathtt{r}'$ *may not be in* $\mathcal{V}_t$*).*

12. *There exists at most one user* u *in* $S(t, \mathtt{k}^t)$ *such that for all* $1 \leq i \leq \ell_u - 1$ *the edge* $(\mathtt{r}_{i,u,t}, \mathtt{r}_{i+1,u,t})$ *is a trivial edge.*

13. *If* $D_{t+1} \neq \emptyset$*, then for every* $u \in S(t, \mathtt{k}^t) \setminus D_{t+1}$*, there exists* $j_{u,t}$ *such that* $1 \leq j_{u,t} < \ell_{u,t}$ *and for the corresponding edge* $(\mathtt{k}^t_{j_{u,t},u,t}, \mathtt{k}^t_{j_{u,t}+1,u,t}) \in \mathcal{E}_t$ *there exists a user* $v \in D_{t+1}$ *such that* $v \in S(t, \mathtt{k}^t_{j_{u,t}+1,u,t})$ *and for all* $w \in D_{t+1}$ *we have* $w \notin S(t, \mathtt{k}^t_{j_{u,t},u,t})$*. Moreover,* $j_{u,t}$ *will denote the least integer in* $\{1, \ldots, \ell_{u,t} - 1\}$ *with this property.*

Proof. Since $\mathtt{ST}^{-1}_u = \{\mathtt{k}^{-1}_u\}$, it follows from Property 8 that $\mathtt{k}^t_{1,u,t} = \mathtt{k}^{-1}_u$. If there exists $(\mathtt{r}, \mathtt{k}^{-1}_u) \in \mathcal{E}_t$, then there exists a user $v \in S(t, \mathtt{k}^t)$ such that $u \neq v$ and $\mathtt{k}^t_{i,v,t} = \mathtt{r}$ and $\mathtt{k}^t_{i+1,v,t} = \mathtt{k}^{-1}_u$ by definition of $\mathcal{G}_t$. By Property 6 applied to $\mathtt{k}^t_{i+1,v,t}$, we obtain $v \in S(t, \mathtt{k}^{-1}_u)$. This would imply that it would not be secure to remove user v in round $t+1$ while maintaining u in the group. Indeed,

$$\mathtt{k}^{t+1} \in \mathsf{Der}(\mathtt{ST}^{-1}_u, (\mathtt{M}_{t'})_{t' \leq t+1 \colon u \in G_{t'}}) \subseteq \mathsf{Der}\left((\mathtt{M}_{t'})^{t+1}_{t'=0}, ((\mathtt{ST}^{t'}_u)^{t+1}_{t'=-1})_{u \in [n_{\max}] \setminus G_{t+1}}\right).$$

We have actually shown that $v \in S(t, \mathtt{k}^{-1}_u)$ implies $v = u$. Therefore $S(t, \mathtt{k}^{-1}_u) = \{u\}$. This completes the proof of Property 10.

Property 11 follows directly from the properties of the symbolic model and the fact that when constructing $\mathcal{G}_t$ we choose only one edge of the two possible for dPRF evaluations.

Property 12 is a direct consequence of the two previous properties.

Property 13 follows from the observation that the node $\mathtt{k}_{\ell_u,u,t} = \mathtt{k}^t$ satisfies the first condition for all users in D_{t+1} and the node $\mathtt{k}_{1,u,t} = \mathtt{k}_u^{-1}$ satisfies the second condition (by Property 10). Since $D_{t+1} \neq \emptyset$ by assumption, there must exist an edge with the required property. □

4.3.2 Lower Bound on Batched Replacements

We now show that a subset $\tilde{\mathcal{S}}_t$ of the set system $\mathcal{S}_t$ defined above satisfies the properties of the combinatorial model regarding correctness and security and, additionally, that the amount of ciphertexts sent in the symbolic model matches the cost function of Section 4.2.1 (with respect to the set system $\tilde{\mathcal{S}}_t$) up to a multiplicative loss of 3. As a consequence, the lower bound derived in Section 4.2.2 applies to batched replacements in multicast in the symbolic model.

Lemma 4.3.5. *Let $n_{\max}$ and $t_{\max}$ be in $\mathbb{N}$ and $(d_t)_{t=1}^{t_{\max}}$ such that $d_t \leq n_{\max}$ for all t. Let* ME *be a correct and secure ME scheme. Consider an execution of game $\mathrm{SEC}^{\mathsf{ME}}$ on input $(n_{\max}, t_{\max}, G_0)$ and $(A_t, D_t)_{t=1}^{t_{\max}}$ such that $D_t \subseteq G_{t-1}$ and $A_t \subseteq [n_{\max}] \setminus (G_{t-1} \cup \bigcup_{t'=1}^{t-1} D_{t'})$ for all $t \in [t_{\max}]$. Let $(\mathcal{S}_t)_{t=0}^{t_{\max}}$ be the associated set system as defined in Definition 4.3.1. Further, for $t \in [t_{\max}]_0$ let*

$$\tilde{\mathcal{S}}_t = \left\{ S \in \mathcal{S}_t \;\middle|\; \begin{array}{l} \exists \mathtt{r} \text{ such that } S = S(t,\mathtt{r}),\ \mathtt{r} \text{ is useful at time } t \text{ and} \\ \nexists \mathtt{r}_1, \mathtt{r}_2 \text{ such that } \mathsf{dPRF}(\mathtt{r}_1,\mathtt{r}_2) = \mathtt{r} \text{ and } S(t,\mathtt{r}_1) \cap S(t,\mathtt{r}_2) = S(t,\mathtt{r}) \end{array} \right\}.$$

Then it holds that

(i) $G_t \in \tilde{\mathcal{S}}_t$ *for all* $t \in [t_{\max}]_0$,

(ii) $S \subseteq G_t$ *for all* $S \in \mathcal{S}_t$ *and, in particular,* $S \subseteq G_t$ *for all* $S \in \tilde{\mathcal{S}}_t$

(iii) $\sum_{t=0}^{t_{\max}} |\mathtt{M}_t| \geq 1/3 \cdot \sum_{t=0}^{t_{\max}} \mathrm{Cost}(t)$, *where* $\mathrm{Cost}(t)$ *is the cost function defined in Section 4.2.1 with respect to* $\tilde{\mathcal{S}}_t$, *namely:*

$$\begin{aligned} \mathrm{Cost}(t) = (&\mathrm{SizeMinCov}_{=}(G_t, \tilde{\mathcal{S}}_{t-1} \cup \{\{u\} : u \in G_t\}) - 1) \\ &+ |\{S \in \tilde{\mathcal{S}}_{t-1} : S \cap D_t \neq \emptyset \text{ and } |S| > 1\}|. \end{aligned}$$

The reason for introducing $\tilde{\mathcal{S}}_t$ is that allowing the use of dPRFs means that the original set $\mathcal{S}_t$ also contains the intersection of any pair of sets such that one of the associated secrets is useful and this does not require any additional communication. Before turning to the lemma's proof we state our bound on the communication complexity of batched replacements in multicast encryption, which follows directly by applying Theorem 4.2.4 to set system $(\tilde{\mathcal{S}}_t)_{t=0}^{t_{\max}}$ which is possible due to Lemma 4.3.5.

Corollary 4.3.6. *Let $n \leq n_{\max}$ and $t_{\max}$ be in $\mathbb{N}$ and $(d_t)_{t=1}^{t_{\max}}$ such that $d_t \leq n$ for all t. Let* ME *be a correct and secure ME scheme. Consider an execution of game $\mathrm{SEC}^{\mathsf{ME}}$ on input $(n_{\max}, t_{\max}, G_0)$ and $(A_t, D_t)_{t=1}^{t_{\max}}$ where $|G_0| = n$ and, for all t, the set D_t of removed users is sampled uniformly at random from the set $\{D \subseteq G_{t-1} \mid |D| = d_t\}$ and A_t can be arbitrary according to the restrictions $A_t \subseteq [n_{\max}] \setminus (G_{t-1} \cup \bigcup_{t'=1}^{t-1} D_{t'})$ and $|A_t| = |D_t|$. Then, it holds that*

$$\mathbb{E}\left[\sum_{t=0}^{t_{\max}} |\mathtt{M}_t|\right] \geq \frac{\ln(2)}{3} \sum_{t=1}^{t_{\max}} d_t \log\left(\frac{n}{d_t}\right),$$

where the expectation is taken over the choice of $(D_t)_t$. In particular, if $d_t = d$ for all t, then

$$\mathbb{E}\left[\sum_{t=0}^{t_{\max}} |\mathtt{M}_t|\right] \geq \frac{\ln(2)}{3} t_{\max} \cdot d \cdot \log\left(\frac{n}{d}\right).$$

Proof of Lemma 4.3.5. We start proving Property (ii). Let $S = S(t, \mathtt{r}) \in \mathcal{S}_t$ and $u \in S$. By definition of $S(t, \mathtt{r})$ we have that $\mathtt{r} \in \mathsf{Der}(\mathtt{ST}_u^{-1}, (\mathtt{M}_{t'})_{t' \leq t:\, u \in G_{t'}})$ and since $\mathtt{r}$ is useful at time t it holds that $\mathtt{r} \notin \mathsf{Der}((\mathtt{M}_{t'})_{t'=0}^{t}, ((\mathtt{ST}_u^{t'})_{t'=-1}^{t})_{u \notin G_t})$. Thus $u \in G_t$ as claimed in Property (ii).

Now we proceed to show that Property (i) is true. Recall that $\mathtt{ST}_u^t \subseteq \mathsf{Der}(\mathtt{ST}_u^{-1}, (\mathtt{M}_{t'})_{t' \leq t:\, u \in G_{t'}})$. By correctness there exists a key $\mathtt{k}^t$ such that $\mathtt{k}^t = \mathsf{ME.Key}(\mathtt{ST}_u^t)$ for all users $u \in G_t$ and by security we have that $\mathtt{k}^t \notin \mathsf{Der}((\mathtt{M}_{t'})_{t'=0}^{t}, ((\mathtt{ST}_u^{t'})_{t'=-1}^{t})_{u \notin G_t})$, so $S(t, \mathtt{k}^t) = G_t$ and $S(t, \mathtt{k}^t) \in \mathcal{S}_t$. Moreover, assume that there exist $\mathtt{r}_1, \mathtt{r}_2$ such that $\mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2) = \mathtt{k}^t$ and $S(t, \mathtt{r}_1) \cap S(t, \mathtt{r}_2) = S(t, \mathtt{k}^t) = G_t$. Since $\mathtt{k}^t$ is useful at time t, there exists $i \in \{1, 2\}$ such that $\mathtt{r}_i$ is useful at time t. By Property (ii) it must hold that $S(t, \mathtt{r}_i) \subseteq G_t$. The fact that $S(t, \mathtt{r}_1) \cap S(t, \mathtt{r}_2) = S(t, \mathtt{k}^t) = G_t$ implies that we also have $G_t \subseteq S(t, \mathtt{r}_i)$. Therefore $S(t, \mathtt{r}_i) = G_t$. By repeating this process we can find a secret $\mathtt{r}$ that is useful at time t such that $S(t, \mathtt{r}) = G_t$ and that satisfies the property that $\nexists \mathtt{r}_1, \mathtt{r}_2$ such that $\mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2) = \mathtt{r}$ and $S(t, \mathtt{r}_1) \cap S(t, \mathtt{r}_2) = S(t, \mathtt{r})$. This shows that $G_t \in \tilde{\mathcal{S}}_t$ as claimed in Property (i). Observe that we have shown $S(t, \mathtt{k}^t) = G_t$ and not just $G_t \in \tilde{\mathcal{S}}_t$.

We now proceed to prove Property (iii). We divide the proof into showing each of the following two equations separately:

$$\sum_{t=0}^{t_{\max}} |\mathtt{M}_t| \geq \frac{1}{2} \sum_{t=0}^{t_{\max}} (\mathrm{SizeMinCov}_{=}(G_t, \tilde{\mathcal{S}}_{t-1} \cup \{\{u\} : u \in G_t\}) - 1) \tag{4.5}$$

$$\sum_{t=0}^{t_{\max}} |\mathtt{M}_t| \geq \sum_{t=1}^{t_{\max}} |\{S \in \tilde{\mathcal{S}}_{t-1} : S \cap D_t \neq \emptyset \text{ and } |S| > 1\}|. \tag{4.6}$$

Let $t \in \{1, \ldots, t_{\max}\}$ and denote by $\mathtt{k}^t$ the group key of round t. If $D_t = \emptyset$, $\mathrm{SizeMinCov}_{=}(G_t, \tilde{\mathcal{S}}_{t-1} \cup \{\{u\} : u \in G_t\}) - 1 = 0$, so we assume that $D_t \neq \emptyset$. In

order to construct a cover of G_t we give a cover of $G_t \setminus A_t$ and a cover of A_t. Observe that $u \in G_t \setminus A_t$ if and only if $u \in G_{t-1} \setminus D_t$.

For each $u \in G_{t-1} \setminus D_t$, we consider the index $j_{u,t-1}$ from Property 13. We claim that

$$\mathcal{C}_{t,1} = \{S(t-1, \mathtt{k}^{t-1}_{j_{u,t-1},u,t-1}) \mid u \in G_{t-1} \setminus D_t\}$$

is a cover of $G_{t-1} \setminus D_t$ and $\mathcal{C}_{t,1} \subseteq \mathcal{S}_{t-1}$. The fact that $u \in S(t-1, \mathtt{k}^{t-1}_{j_{u,t-1},u,t-1})$ and $S(t-1, \mathtt{k}^{t-1}_{j_{u,t-1},u,t-1}) \in \mathcal{S}_{t-1}$ is a consequence of Property 6. It also holds that $S(t-1, \mathtt{k}^{t-1}_{j_{u,t-1},u,t-1}) \subseteq G_{t-1} \setminus D_t$ by Properties 13 and (ii).

Let $d_{j_{u,t-1}+1}$ denote the in-degree of the node $\mathtt{k}^{t-1}_{j_{u,t-1}+1,u,t-1}$ in $\mathcal{G}_{t-1}$ and let $u_1 = u, u_2, \ldots, u_h$ be users such that $\mathtt{k}^{t-1}_{j_{u,t-1}+1,u,t-1} = \mathtt{k}^{t-1}_{j_{u_i,t-1}+1,u_i,t-1}$ for $i = 1, \ldots, h$ and for any user v with $\mathtt{k}^{t-1}_{j_{v,t-1}+1,v,t-1} = \mathtt{k}^{t-1}_{j_{u,t-1}+1,u,t-1}$, there exists a unique $i \in \{1, \ldots, h\}$ such that $\mathtt{k}^{t-1}_{j_{v,t-1},v,t-1} = \mathtt{k}^{t-1}_{j_{u_i,t-1},u_i,t-1}$. I.e., we choose exactly one user u_i for each of the incoming edges of the node $\mathtt{k}^{t-1}_{j_{u,t-1}+1,u,t-1}$ that satisfy Property 13. Therefore $d_{j_{u,t-1}+1} \geq h$. Each node $\mathtt{k}^{t-1}_{j_{u,t-1}+1,u,t-1}$ has at most one incoming trivial edge (Property 11). All the other $d_{j_{u,t-1}+1} - 1$ incoming edges correspond to ciphertexts in $\bigcup_{t'=0}^{t-1} \mathtt{M}_{t'}$. If $d_{j_{u,t-1}+1} \geq h+1$, then the node $\mathtt{k}^{t-1}_{j_{u,t-1}+1,u,t-1}$ contributes to $\mathcal{C}_{t,1}$ with at most $h \leq d_{j_{u,t-1}+1} - 1$ sets. If $d_{j_{u,t-1}+1} = h$, then we can consider the graph we would obtain for the secret $\mathtt{k}^{t-1}_{j_{u,t-1}+1,u,t-1}$ rather than $\mathtt{k}^{t-1}$. Let's denote it $\mathcal{H}$. The set of nodes corresponds to the elements of the sequences constructed in Lemma 4.3.3 for the secret $\mathtt{k}^{t-1}_{j_{u,t-1}+1,u,t-1}$ and the set of edges consists of all pairs of consecutive elements in those sequences. Since the node $\mathtt{k}^{t-1}_{j_{u,t-1}+1,u,t-1}$ in $\mathcal{H}$ has at least one additional incoming edge that corresponds to the user $v \in D_t$ guaranteed to exist by Property 13 for $\mathcal{G}_{t-1}$, which does not contribute to $\mathcal{C}_{t,1}$, the node $\mathtt{k}^{t-1}_{j_{u,t-1}+1,u,t-1}$ contributes to $\mathcal{C}_{t,1}$ with at most $h = d_{j_{u,t-1}+1}$ sets and we have at least $d_{j_{u,t-1}+1}$ ciphertexts in $\bigcup_{t'=0}^{t-1} \mathtt{M}_{t'}$. Thus $|\mathcal{C}_{t,1}| \leq \sum_{t'=0}^{t-1} |\mathtt{M}_{t'}|$.

We observe that $S(t-1, \mathtt{k}^{t-1}_{j_{u,t-1}+1,u,t-1}) \cap D_t \neq \emptyset$. Therefore, $\mathtt{k}^{t-1}_{j_{u,t-1}+1,u,t-1}$ is not a useful random coin at time t. This guarantees that the node $\mathtt{k}^{t-1}_{j_{u,t-1}+1,u,t-1}$ will not be in $\mathcal{G}_{\tilde{t}}$ for $\tilde{t} \geq t$. Therefore $\sum_{t=1}^{t_{\max}} |\mathcal{C}_{t,1}| \leq \sum_{t=0}^{t_{\max}-1} |\mathtt{M}_t|$.

Moreover, we can find covers $\mathcal{C}'_{t,1} \subseteq \tilde{\mathcal{S}}_{t-1}$ such that $\sum_{t=1}^{t_{\max}} |\mathcal{C}'_{t,1}| \leq \sum_{t=0}^{t_{\max}-1} |\mathtt{M}_t|$. We obtain $\mathcal{C}'_{t,1}$ from $\mathcal{C}_{t,1}$ by substituting the sets that are in $\mathcal{S}_{t-1} \setminus \tilde{\mathcal{S}}_{t-1}$ for sets in $\tilde{\mathcal{S}}_{t-1}$. Let's assume that there exist $\mathtt{r}_1, \mathtt{r}_2$ such that $\mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2) = \mathtt{k}^{t-1}_{j_{u,t-1},u,t-1}$ and $S(t-1, \mathtt{r}_1) \cap S(t-1, \mathtt{r}_2) = S(t-1, \mathtt{k}^{t-1}_{j_{u,t-1},u,t-1})$. Since $\mathtt{k}_{j_{u,t-1},u,t-1}$ is useful at time $t-1$, there exists $i \in \{1,2\}$ such that $\mathtt{r}_i$ is useful at time $t-1$. From $S(t-1, \mathtt{r}_1) \cap S(t-1, \mathtt{r}_2) = S(t-1, \mathtt{k}^{t-1}_{j_{u,t-1},u,t-1})$ and the way sequences are constructed in Lemma 4.3.3, it follows that $j_{u,t-1} > 1$, $\mathtt{k}^{t-1}_{j_{u,t-1}-1,u,t-1} = \mathtt{r}_i$. From $S(t-1, \mathtt{r}_1) \cap S(t-1, \mathtt{r}_2) = S(t-1, \mathtt{k}^{t-1}_{j_{u,t-1},u,t-1})$, we obtain $S(t-1, \mathtt{k}^{t-1}_{j_{u,t-1},u,t-1}) \subseteq S(t-1, \mathtt{r}_i) = S(t-1, \mathtt{k}^{t-1}_{j_{u,t-1}-1,u,t-1})$. The minimality

condition imposed on $j_{u,t}$ by Property 13 guarantees that $S(t-1, \mathtt{k}^{t-1}_{j_{u,t-1}-1,u,t-1}) \cap D_t = \emptyset$. This shows that $\mathcal{C}'_{t,1} = (\mathcal{C}_{t,1} \setminus \{S(t-1, \mathtt{k}^{t-1}_{j_{u,t-1},u,t-1})\}) \cup \{S(t-1, \mathtt{r}_i)\}$ is also a cover of $G_{t-1} \setminus D_t$ and it has the same size as $\mathcal{C}_{t,1}$. Therefore by repeating this process we obtain a cover of $G_{t-1} \setminus D_t$ with respect to $\tilde{\mathcal{S}}_{t-1}$ and it has at most as many sets as the original $\mathcal{C}_{t,1}$. We denote it $\mathcal{C}'_{t,1}$ and it holds that

$$\sum_{t=1}^{t_{\max}} |\mathcal{C}'_{t,1}| \leq \sum_{t=0}^{t_{\max}-1} |\mathtt{M}_t|. \tag{4.7}$$

Now we give a cover of A_t. The argument also considers the case where $t = 0$ if we define $A_0 = G_0$. For each $u \in A_t$, let $i_{u,t} \in \{1, \ldots, \ell_{u,t}\}$ be maximal such that for all $1 \leq j < i_{u,t}$, $(\mathtt{k}^t_{j,u,t}, \mathtt{k}^t_{j+1,u,t})$ is a trivial edge. By Property 12, there exists at most one user in A_t such that $i_{u,t} = \ell_{u,t}$. For every user $u \in A_t$ such that $i_{u,t} < \ell_{u,t}$, there exists a ciphertext $\mathtt{c}_{i_{u,t},u,t}$ as proven in Property 9d. All these ciphertexts must be different or else there would exist users $u, v \in A_t$ such that $\mathtt{k}^t_{i_{u,t},u,t} = \mathtt{k}^t_{i_{v,t},v,t}$, which would contradict Properties 11 and 10.

Each of the ciphertexts $\mathtt{c}_{i_{u,t},u,t}$ belongs to $\bigcup_{\tilde{t} \leq t:\, u \in G_{\tilde{t}}} \mathtt{M}_{\tilde{t}}$. From the condition $A_t \subseteq [n_{\max}] \setminus (G_{t-1} \cup \bigcup_{t'=1}^{t-1} D_t)$, it follows that $\mathtt{c}_{i_{u,t},u,t} \in \mathtt{M}_t$. Thus we have at least $|A_t| - 1$ many ciphertexts sent in round t. The cover of A_t, $\mathcal{C}_{t,2} = \{\{u\} \mid u \in A_t\}$ satisfies the inequality $|\mathtt{M}_t| \geq |\mathcal{C}_{t,2}| - 1$.

From this inequality and Equation 4.7, we obtain

$$\frac{1}{2}\sum_{t=1}^{t_{\max}} |\mathcal{C}_{t,1}| + \frac{1}{2}\sum_{t=0}^{t_{\max}} (|\mathcal{C}_{t,2}| - 1) \leq \frac{1}{2}\sum_{t=0}^{t_{\max}-1} |\mathtt{M}_t| + \frac{1}{2}\sum_{t=0}^{t_{\max}} |\mathtt{M}_t| \leq \sum_{t=0}^{t_{\max}} |\mathtt{M}_t|.$$

This shows Equation 4.5 since $|\mathcal{C}'_{t,1}| \leq |\mathcal{C}_{t,1}|$ and $\mathcal{C}'_{t,1} \cup \mathcal{C}_{t,2}$ is a cover of G_t for all $t \in [t_{\max}]_0$ (we take $\mathcal{C}'_{0,1} = \emptyset$).

Now we show Equation 4.6. Let $S = S(t-1, \mathtt{r}) \in \tilde{\mathcal{S}}_{t-1}$ such that $S \cap D_t \neq \emptyset$, $|S| > 1$, and $\nexists \mathtt{r}_1, \mathtt{r}_2$ that satisfy the following two properties: $\mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2) = \mathtt{r}$ and $S(t-1, \mathtt{r}_1) \cap S(t-1, \mathtt{r}_2) = S(t-1, \mathtt{r})$. We proceed to consider the graph $\mathcal{G}_{t-1,\mathtt{r}} = (\mathcal{V}_{t-1,\mathtt{r}}, \mathcal{E}_{t-1,\mathtt{r}})$ where $\mathcal{V}_{t-1,\mathtt{r}}$ is a subset of useful random coins at time $t-1$ which corresponds to the elements of the sequences $\{\mathtt{r}_{1,u,t-1}, \ldots, \mathtt{r}_{\ell_u,u,t-1}\}$ constructed in Lemma 4.3.3 for each user $u \in S(t-1, \mathtt{r})$. The set of edges $\mathcal{E}_{t-1,\mathtt{r}}$ consists of all pairs $(\mathtt{r}_{i,u,t-1}, \mathtt{r}_{i+1,u,t-1})$. From Property 12[4] and the fact that $|S| > 1$, it follows that not all edges in $\mathcal{E}_{t-1,\mathtt{r}}$ are trivial edges. If the node $\mathtt{r}$ only has only one incoming edge of the form $(\mathtt{r}_{\ell_u-1,u,t-1}, \mathtt{r}_{\ell_u,u,t-1} = \mathtt{r})$ for some $u \in S(t-1, \mathtt{r})$ and it is a trivial edge, then $S(t-1, \mathtt{r}_{\ell_u-1,u,t-1}) = S(t-1, \mathtt{r})$. In order to show this we consider two cases:

[4]Property 12 was shown for the graph $\mathcal{G}_t$ and the same argument shows that this property also holds for $\mathcal{G}_{t-1,\mathtt{r}}$.

- if $\nexists \mathtt{r}_1, \mathtt{r}_2$ such that $\mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2) = \mathtt{r}$, then $(\mathtt{r}_{\ell_u - 1, u, t-1}, \mathtt{r}_{\ell_u, u, t-1} = \mathtt{r})$ must correspond to a PRG or a PRF and $S(t-1, \mathtt{r}_{\ell_u - 1, u, t-1}) = S(t-1, \mathtt{r})$,
- if $\exists \mathtt{r}_1, \mathtt{r}_2$ such that $\mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2) = \mathtt{r}$, but $S(t-1, \mathtt{r}_1) \cap S(t-1, \mathtt{r}_2) \subsetneq S(t-1, \mathtt{r})$, then there exists a user $v \in S(t-1, \mathtt{r}) \setminus S(t-1, \mathtt{r}_1) \cap S(t-1, \mathtt{r}_2)$ and by Property 9c $\mathtt{r}_{\ell_v - 1, v, t-1} \neq \mathtt{r}_1$ and $\mathtt{r}_{\ell_v - 1, v, t-1} \neq \mathtt{r}_2$. This contradicts the assumption that $\mathtt{r}$ had only one incoming edge.

As argued in the previous paragraph we may assume without loss of generality that for some user $u \in S(t-1, \mathtt{r}) = S$ the edge $(\mathtt{r}_{\ell_u - 1, u, t-1}, \mathtt{r}_{\ell_u, u, t-1} = \mathtt{r})$ corresponds to a ciphertext $\mathsf{c}_{\ell_u - 1, u, t} \in \bigcup_{t'=0}^{t-1} \mathtt{M}_{t'}$. This shows that $\sum_{t'=0}^{t-1} |\mathtt{M}_{t'}| \geq |\{S \in \tilde{\mathcal{S}}_{t-1} : S \cap D_t \neq \emptyset \text{ and } |S| > 1\}|$. Moreover, the fact that $S \cap D_t \neq \emptyset$ guarantees that $\mathtt{r}$ does not appear in $\mathcal{G}_{\tilde{t}-1, \tilde{\mathtt{r}}}$ for any $\tilde{t} > t$ and useful $\tilde{\mathtt{r}}$ at time $\tilde{t}$ by Property (ii). Thus,

$$\sum_{t=0}^{t_{\max}-1} |\mathtt{M}_t| \geq \sum_{t=1}^{t_{\max}} |\{S \in \tilde{\mathcal{S}}_{t-1} : S \cap D_t \neq \emptyset \text{ and } |S| > 1\}|.$$

Finally we multiply Equation 4.5 by $2/3$ and Equation 4.6 by $1/3$ and add them together to obtain $\sum_{t=0}^{t_{\max}} |\mathtt{M}_t| \geq 1/3 \cdot \sum_{t=0}^{t_{\max}} \mathrm{Cost}(t)$, as desired. □

CHAPTER 5

CGKA: Concurrent Updates without Pruning

This chapter essentially replicates the results that appear in our publication [ANEP25].

5.1 Introduction

The CGKA scheme at the heart of the MLS standard, TreeKEM, is a public-key analog of Logical Key Hierarchies [WHA99] that in order to be able to handle the possibility that parties issue operations concurrently, which is a consequence of the lack of a central authority, makes use of the propose-and-commit paradigm. It is easy to see that TreeKEM suffers from a worst case communication complexity of order $\Omega(N)$ because there are sequences of operations that lead to the whole ratchet tree becoming blank. Nevertheless, this is known to be essentially inherent for CGKA schemes built from standard primitives that provide PCS within a single propose-commit round as shown in [BDR20, BDG+22] and Chapter 3.

While the worst case performance of TreeKEM is a far shot from the advertised $\log(N)$, the known sequences causing it are very artificial. Thus, one might still hope that TreeKEM and in turn MLS perform well for 'natural' random sequences of operations, which would be sufficient for practical applications. However, up to our knowledge there has been very little analysis in this setting so far.

5.1.1 Our Results

The contribution of this work is twofold. We initialize the formal study of MLS's communication complexity for random sequences of operations, which models the lack

of coordination between users given the asynchronous nature of CGKA schemes. To this end, we derive lower bounds on the size of its commit messages in an easy to understand model letting randomly chosen users issue update proposals and commits. Our results indicate that, unfortunately, even very small amounts of update proposals per commit operation have a substantial negative impact on MLS's communication complexity.

On the positive side, we propose an alternative way of handling update proposals that provably achieves a logarithmic communication complexity in our model. The changes required to handling proposals are not too complex, which in our view is a very positive feature, as our proposed method is compatible with the complex mechanisms employed by MLS to ensure authenticity and consistency.

Lower bounds on MLS's Average Commit Size. Our main goal is to gain an understanding of the impact that blanking nodes in the ratchet tree due to update proposals has on the size of commit messages, for randomly generated sequences of operations. Ideally, one would analyze the communication complexity of MLS using real-world user data. As such data is unfortunately not available, we opt for a simple model of communication. The advantage of this being that it, on the one hand, makes the problem of deriving formal bounds approachable, and, on the other, leads to results that are understandable on an intuitive level.

Our experiment first initializes a group of size N. Then it repeatedly starts issuing update proposals and commits for rounds t as follows.

1. A set U_P consisting of $P \in \mathbb{N}$ group members is sampled uniformly at random.
2. All users in U_P issue an update proposal.
3. A user sampled uniformly at random issues a commit to all update proposals.
4. The size $\mathrm{Cost}(t)$ of the commit is recorded.

To also be able to handle the setting where update proposals are only issued every other round, the experiment is additionally parameterized by $C \in \mathbb{N}$. In the case $C > 1$, the commit to the P proposals is followed by $C-1$ rounds of commits not implementing any proposals. We are interested in the expectation of $\mathrm{Cost}(t)$ after running the experiment for sufficiently many rounds. Informally, our main result regarding this is the following.

Theorem 5.4.2 (informal). *Let P, C be constant. Then the expected size of a commit generated according to the sequence described above is*

of order

$$\mathbb{E}[\mathrm{Cost}(t)] \geq \Omega(N^{\log_2(1+\frac{P}{P+C})}).$$

Moreover, if $P = 1 = C$, it holds that

$$\mathbb{E}[\mathrm{Cost}(t)] \geq \log^2(N)/4 - \log(N)/4,$$

which is a more relevant bound for small values of N.

We provide some approximate values of the exponent $e = \log_2(1 + \frac{P}{P+C})$ in the table below. We use $C = 1$ for $P \geq 1$ and for ease of notation write $P = 1/c$ in the case $P = 1$, $C = c$.

P	$1/5$	$1/2$	1	2	5	10	50
e	0.22	0.42	0.58	0.74	0.87	0.93	0.99

Note that already if a single update proposal is issued per commit the MLS's communication complexity devolves to about $\sqrt{N}$, a far shot from $\log(N)$. Further, if the number of proposals increases the cost of commits approaches the worst case of a linear commit cost quite fast, with an exponent of 0.99 for $P = 50$.

This means that one either must perform almost all updates sequentially, or commits become almost linear in N. As MLS aims to support groups containing up to $N = 50000$ members, neither of these options is very appealing.

Before turning to our alternative proposal of handling update proposals, we give a bit of intuition on the lower bound's proof. As the cost of a commit is mainly determined by the blanks in the ratchet tree we first investigate the expected probability of a fixed node v being blank. We notice that the transition of v's state between blank and populated forms a Markov chain, allowing us to determine its stationary distribution. We obtain that the probability of v being blank, assuming v is of sufficient depth in the tree, is roughly $P/(P + C)$.

In MLS if a user's copath contains a node v, then how much v contributes to the communication complexity of a commit by that user depends on whether v is blank. If v is populated, then encrypting to it only requires one ciphertext. If it is blank, a commit in MLS would include ciphertexts encrypted under the keys of its children and, due to the way MLS works, at least one child must be blank as well meaning that those ciphertexts would have to be encrypted under some of v's grandchildren and so on. Intuitively, this means that the number of ciphertexts needed to encrypt to a blank node v is $1 + P/(P + C)$ times the number of ciphertexts needed to encrypt to a child of v provided that both v and its child are blank, where 1 corresponds to

the child that must be blank and $P/(P+C)$ to the probability that the other child is blank. This argument assumes independence (between the events that the left child of v is blank and the right child of v is blank), which is not justified. However, we show that in expectation the number of ciphertexts does indeed increase by a factor of $1 + P/(P+C)$.

An alternate way of performing update proposals. It is known that if too many users, say, for example, linearly many in the group size, issue concurrent update operations there is no possibility for efficient CGKA. In fact, Bienstock *et al.* [BDR20] prove that in this case the communication complexity must be linear in the group size. While the lower bound discussed above indicates that even a small number of update proposals per commit operation has a devastating effect on MLS's communication complexity, it leaves open the possibility of an alternative CGKA being more viable in this setting.

Recall that update proposals introduce blanks in the tree, as all keys on the updating users' update paths are removed. A natural idea to prevent introducing blanks in the tree would be to not only have the user issuing a commit, but also updating users to re-key their complete update path. In fact this is the approach taken by the continuous group-key agreement scheme CoCoA [AAN+22] which in this setting has logarithmic communication complexity. However, proposals can be issued concurrently, meaning users make concurrent modifications to the ratchet tree. This leads to users encrypting new secrets to outdated keys, i.e., keys concurrently being replaced, and, as a second point, is not compatible with the mechanisms used by MLS to ensure integrity of the ratchet tree. As a consequence, CoCoA does achieve PCS slower than MLS, and does not achieve the same strong security notion of insider security.

The second main contribution of this work is a CGKA we denote MLS-Cutoff with the following features.

- For a constant number P of update proposals per commit operation and random operations as described above, the size of its proposal and commit messages is bounded by $O(\log(N))$
- It achieves the same strong security notion as the MLS protocol, namely insider security [AJM22] for the same safety predicate. In a nutshell, the latter says that if all compromised parties issue update proposals that are in turn committed to the group achieves PCS.
- It requires only small modifications to the MLS design. More precisely, it only modifies how the ratchet tree's nodes are blanked or populated by applying update proposals, but keeps other components required, for example, to ensure consistency and authenticity unchanged.

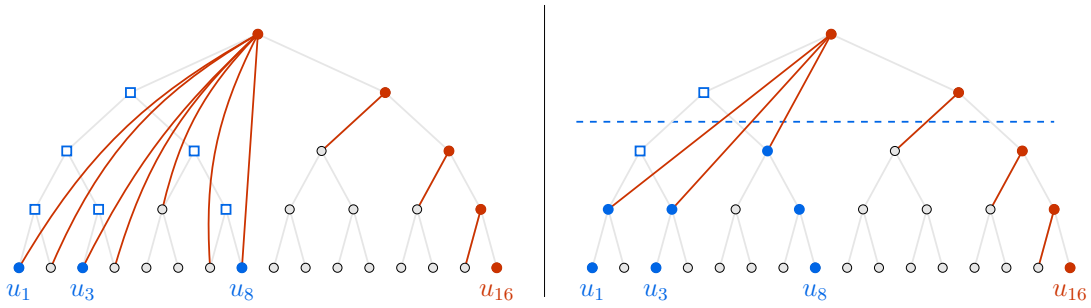

Figure 5.1: Working principle of update proposals and commits in MLS (left) and MLS-Cutoff (right). In both cases for a fully populated tree users u_1, u_3, u_8 issue an update proposal (depicted in blue) followed by a commit by user u_{16} (depicted in orange). Nodes being blanked are depicted as (empty) squares, and ciphertexts created as part of the commit with orange edges. In the depiction of MLS-Cutoff the blue dashed line marks the cutoff bound.

We provide some intuition on our design. Our main observation is that if the number of update proposals is not too large and the updating users are randomly distributed, then updating users' paths merge far up in the tree. Thus, while in MLS-Cutoff update proposals do rekey the issuing user's path, we stop the re-key operation i_{cut} steps before reaching the root, where i_{cut} is the cutoff parameter of the scheme, that is chosen as $\log(\log(N))$. If no update paths collide before the cutoff point, no secrets are encrypted to keys that are concurrently being changed. This on one hand, implies PCS at the same speed as MLS, and on the other is compatible with the tree-integrity mechanisms of tree-hash and parent-hash. In the case that we are unlucky and two updating user's paths collide before the cutoff point, the paths are blanked from (and including) the point where the paths meet. For a depiction on how MLS-Cutoff handles proposals and commits, see Figure 5.1 (right) and compare it to how MLS does it (left).

We point out that the worst-case communication complexity of MLS-Cutoff is $\Omega(N)$, as is the case for MLS. This, however, holds for any CGKA achieving fast PCS [BDR20] constructed from standard primitives. We consider MLS-Cutoff to be an attractive alternative to MLS in settings where while update proposals are required their number is not too large. This is a realistic scenario since MLS leaves the update policy to the implementation and, therefore, a rule which simply triggers an update with some probability for users that are online (or enforces it if too much time or communication happened since the last update) would lead to a fairly random update pattern if there is no user-to-user coordination and thus good performance of MLS-Cutoff. We consider it an interesting question for future work to analyze its performance in more involved settings.

5.2 Preliminaries

We recall the definition of CGKA in the Propose-Commit framework. As we aim for the security notion of insider security our syntax follows [AJM22].

Public-key infrastructure. We model the public-key infrastructure (PKI) as consisting of two services: an authentication service (AS) and a key service (KS). The former allows parties to register their signing keys and retrieve the signature verification keys of other users, while the latter allows users to register the key packages that are used by other users to add them to a group.

The authentication service stores pairs of users and signing verification keys, (u, svk), if the key svk has been registered under user identity u. Any user u can register a new key and request to receive the secret signing keys that correspond to verification keys registered under them. Moreover, a user can also check if a pair (u, svk) has already been registered and a user can also delete their secrets. We consider the AS as not trusted and therefore in our security model we will give the adversary the possibility of registering keys as well as exposing keys registered by the users (provided that they have not been deleted).

The key service stores pairs of users and key packages, (u, kp), which are used by other parties to add them to a group. Similarly to the case of AS, users are able to register a new key package as well as retrieve the secret keys sk associated to their key packages and delete their secret keys. A user can also request a key package for another party. In our model this is done by forwarding the request to the adversary which chooses the package received by the user. This reflects that our security model does not trust the KS. The adversary can also expose the secret keys of a user that have not been deleted.

Syntax. A CGKA scheme allows a group of users to agree on a shared secret evolving over epochs. To this end, the users exchange proposal messages, proposing changes to the group's state (as for example adding or removing users to or from the group), and commit messages, which implement proposals and advance the group's state to the next epoch by establishing a new shared secret.

Definition 5.2.1. *An asynchronous continuous group-key agreement protocol consists of a tuple of algorithms* $\mathsf{CGKA} = (\mathsf{CGKA.Create}, \mathsf{CGKA.Prop}, \mathsf{CGKA.Com}, \mathsf{CGKA.Proc}, \mathsf{CGKA.Join}, \mathsf{CGKA.Key})$*, which can interact with a PKI. Additionally, each user maintains an internal state which we denote by* st *and is assumed to be an implicit input of all algorithms.*

$\mathsf{CGKA.Create}(\mathsf{svk}_u)$ *is run by user* u *with signing verification key* svk_u *and creates a group with* u *as its only member.*

CGKA.Prop(op, ad) $\rightarrow$ pmsg *is run by a member who wants to propose an operation* op*, where* op $\in \{$*'add', 'rem', 'upd'*$\}$*,* ad *is some additional data that may depend on the operation. It outputs a proposal message* pmsg*.*

CGKA.Com(PMSG, ad) $\rightarrow$ (cmsg, wmsg) *is run by a member* u *to commit to a list of proposals* PMSG $= (\text{pmsg}_i)_i$ *and it also takes as input some additional data* ad*. It outputs a commit message* cmsg*, and a (potentially empty) welcome message* wmsg*.*

CGKA.Proc(cmsg, PMSG) $\rightarrow$ (PropSemantics, v) *is run by a client* u *to process a commit message* cmsg *and the corresponding proposal messages* PMSG*. It outputs some information about the proposals (which we denote* PropSemantics*) and the party* v *that generated the commit* c*.*

CGKA.Join(wmsg) $\rightarrow$ (roster, v) *is run by user* u *to process a welcome message and join the respective group. It outputs the set of group members and their signing verification keys (which we refer to as* roster*) and the party that generated the commit* c

CGKA.Key $\rightarrow$ k *is run by a member* u *to obtain the group key associated to their state.*

Correctness and security in the UC framework. We follow the work of Alwen et al. ([AJM22] and [ACJM20]). They define a notion of security based on the UC (universal composability) framework, which was first introduced by Canetti in [Can01]. We defer the formal description of the ideal CGKA functionality and the PKI functionalities to Appendix B.1. In the UC framework one formalizes security of a protocol as the indistinguishability between the 'real world' and an 'ideal world'. We will refer to the distinguisher as the environment.

In the real world the environment $\mathcal{Z}$ gets to choose the inputs (e.g., an instruction to Alice to add Bob) of the machines (e.g., the parties in an execution of a CGKA protocol) of the protocol π and receives their intermediate subroutine-outputs (e.g., a commit generated by Alice). The environment also interacts freely with an adversary $\mathcal{A}$. $\mathcal{A}$ also interacts with the machines of π and $\mathcal{A}$ has control over the network in the communication between the machines of π. The interaction between $\mathcal{A}$ and π also captures information leakage from the protocol execution (e.g., $\mathcal{A}$ may have the possibility of corrupting the parties).

In the ideal world the protocol π is substituted by an ideal functionality $\mathcal{F}$ and a collection of dummy machines that relay their inputs to $\mathcal{F}$ and the outputs of $\mathcal{F}$ to the corresponding machines. The adversary in the ideal world is denoted by $\mathcal{S}$ because it plays the role of a simulator (of the interaction between $\mathcal{A}$ and $\mathcal{Z}$ in the real world)

and we refer to it as an ideal adversary. The adversary $\mathcal{S}$ now communicates with $\mathcal{F}$ and does not interact with the dummy machines. The environment $\mathcal{Z}$ interacts with the dummy machines and the adversary $\mathcal{S}$ just as in the real world.

In the UC framework one says that a protocol π securely realizes a functionality $\mathcal{F}$ if for all efficient adversaries $\mathcal{A}$ there exists an efficient ideal adversary $\mathcal{S}$ such that the output distribution of any efficient environment $\mathcal{Z}$ after interacting with π and $\mathcal{A}$ is indistinguishable from the output distribution of $\mathcal{Z}$ after interacting with $\mathcal{F}$ and $\mathcal{S}$. This can be understood as representing the idea that an adversary $\mathcal{A}$ attacking the protocol π does not learn more than the ideal adversary (sometimes called simulator) $\mathcal{S}$ learns from interacting with the ideal functionality.

In the work of Alwen et al. [AJM22], whose security notion we use, one has to consider some additional notions since parties have access to a PKI and certain primitives are modeled as random oracles. We refer to these as setup functionalities. This is done by considering the so called hybrid models (first introduced in [CDPW07]). If $\mathcal{G}$ is a setup functionality, then the environment $\mathcal{Z}$ (as well as π, $\mathcal{A}$ and $\mathcal{S}$) is allowed to interact with $\mathcal{G}$. And one says that the a protocol π securely realizes a functionality $\mathcal{F}$ in the $\mathcal{G}$-hybrid model if the real world and ideal world are indistinguishable even if $\mathcal{Z}$ is allowed to interact with $\mathcal{G}$.

Ideal functionality for CGKA. The definition of the ideal functionality $\mathcal{F}_{\mathsf{CGKA}}$ already takes correctness into account. In order to do this certain checks are included in $\mathcal{F}_{\mathsf{CGKA}}$ and the environment rather than the adversary is given control over the network. Now we focus our attention on defining security for CGKA following [AJM22]. They consider two more constrains on the quantification of the notion of security described in the previous paragraph. We restrict ourselves to admissible environments, i.e., those that can only corrupt parties at certain times. This is defined as part of the ideal functionality which specifies some conditions that cannot be violated by the environment except with negligible probability. The second condition consists in not considering all possible adversaries but instead only corruption preserving adversaries, i.e., adversaries that trigger a corruption if and only if prompted by the environment.[1] This is done to prevent the ideal adversary from triggering a disallowed corruption since this would make security trivial in the sense that all environments would be disqualified.

In order to define the ideal functionality $\mathcal{F}_{\mathsf{CGKA}}$ we consider the history graph which keeps track of the evolution of the group. Intuitively, the nodes of the graph correspond to the operations done by the users and the edges capture the order in which they are issued. The history graph [ACDT21] is an acyclic directed graph which consists of two kinds of nodes, proposal nodes and commit nodes. The edges of the graph capture the

[1] Recall that an arbitrary adversary can choose when to corrupt a party without this being determined by the environment.

notion of how new epochs are derived. Additionally, each node is associated with a collection of attributes (Node$[c]$ for commit nodes c and Prop$[p]$ for proposal nodes p) related to the operations of a CGKA protocol. We list these attributes in the following table divided into three groups: those that all nodes have, those that only proposal nodes have and those that only commit nodes have.

The ideal functionality just keeps track of the evolution of the history graph. It starts building the history graph with just one node $root_0$ when the group is created. Additionally $\mathcal{F}_{\mathsf{CGKA}}$ stores a value called pointer Ptr$[u]$ for each party u that maps u to the last commit it processed. When a party u makes a proposal p, $\mathcal{F}_{\mathsf{CGKA}}$ creates a proposal node Prop$[p]$ whose parent in the history graph is Node[Ptr$[u]$] and u is the origin of the proposal Prop$[p]$.orig. The functionality also stores information about the kind of operation associated to p in Prop$[p]$.act. If u makes a commit c to some proposals P, $\mathcal{F}_{\mathsf{CGKA}}$ creates a commit node Node$[c]$ whose parent in the history graph is Node[Ptr$[u]$], Node$[c]$.pro is set to P and Node$[c]$.orig is set to u. And if a party v processes c, the functionality just moves v's pointer to c.

.orig	the party that triggers the creation of the node
.par	the parent commit node
.stat	good for secure nodes, bad for nodes created with adversarially chosen randomness (the adversary knows the secrets associated to this node) or adv for nodes that correspond to messages injected by the adversary
Prop$[p]$.act	indicates the kind of action and some additional information $(\mathsf{upd}, \mathsf{svk})$ if it is an update performed by a party with key svk $(\mathsf{add}, v, \mathsf{svk}_v)$ if party v with key svk_v is added (rem, v) if party v is being removed
Node$[c]$.pro	the list of proposals associated to commit c
Node$[c]$.mem	list of pairs of members and their signing verification keys
Node$[c]$.chall	true if the corresponding application secret is chosen at random and false if chosen adversarially
Node$[c]$.exp	pairs $(u, \mathrm{true/false})$ of corrupted parties u and a boolean value expressing whether the application secret also leaked to adversary

As mentioned before, the ideal functionality imposes certain restrictions on the behaviour of the environment. This is done by including two predicates, namely, safe(c) and inj-allowed(c, u) that regulate when an adversary can corrupt a user and when it can inject a commit on behalf of a user, respectively.

In order to guarantee that the only application secrets that can be learned by an adversary in the real world are the ones trivially derived from secrets of exposed users, we let the simulator choose these 'unsafe' secrets in the ideal world while the remaining ones are chosen at random by the ideal functionality. Thus, predicate safe(c) evaluates to false if and only if at least one of the following is true:

(a) Node[c].exp $\neq \emptyset$ and the application secret has been leaked to the adversary, or

(b) the adversary can process c using the secrets of exposed users and the exposed signing keys (in particular this captures the secret keys associated to key packages generated with exposed signing keys or corrupted randomness) of its ancestors.

Analogously, predicate inj-allowed(c, u) guarantees that in the real world authenticity is only broken when the adversary can trivially forge messages on behalf of a user by restricting when injections are possible in the ideal world. Predicate inj-allowed(c, u) evaluates to true if and only if u's signing key in Node[c] has been exposed (i.e., Node[c].mem[u] $\in$ Exposed) and at least one of the following two holds:

(a) Node[c].exp $\neq \emptyset$,

(b) the adversary can process c using the secrets of exposed users and the exposed signing keys (in particular this captures the secret keys associated to key packages generated with exposed signing keys or corrupted randomness) of its ancestors.

The reader may find the precise definition of the predicates safe(c) and inj-allowed(c, u) in Figure B.12.

5.2.1 Ratchet Trees

We introduce some basic terminology. We use the word tree to refer to a directed acyclic graph $\mathsf{T} = (\mathsf{V}, \mathsf{E})$ with a root, i.e., a node r such that for every node v there exists a unique directed path between r and v. Said path is denoted by path(v). As in the MLS specification we use the convention that path $= (v_1, \ldots, v_\ell = r)$ is ordered ascending to the root, i.e., in reverse direction of the edges, and does not include v.

Let u and v be two different nodes of some tree $\mathsf{T} = (\mathsf{V}, \mathsf{E})$. We say that u is a parent of v if there exists an edge $(u, v) \in \mathsf{E}$. We say that u is a child of v if and only if v is a parent of u. We denote the set of children by child(v). We say that u is an ancestor of v if there exists a directed path from u to v. The set of ancestors is denoted by ancs(u). We say that v is a descendant of u if and only if u is an ancestor of v. We say that u and v are siblings if they are children of the same node. We say that a node is a leaf if it has no children. Thus we have three different kinds of nodes; the root, the leaves and the remaining elements in V which we call intermediate nodes.

In this work we consider complete binary trees with $N = 2^n$ many nodes. Thus each non-leaf node v has exactly two children which we refer to as the left child, denoted by

left(v), and the right child, right(v). Every node v other than the root has one parent, denoted by par(v), and a sibling, sib(v). The depth of a node v is the length of the path from the root to v and we denote it by depth(v).

A notion closely related to how MLS re-keys users' paths is that of co-path of a node. The co-path of a node v is defined as the set of children of the nodes in v's path that do not belong to v's path, namely,

$$\text{co-path}(v) = \cup_{u \in \text{path}(v)} \{w \mid w \in \{\text{left}(u), \text{right}(u)\} \setminus (\text{path}(v) \cup \{v\})\}.$$

Node states. CGKA schemes like MLS make use of a *ratchet tree*, i.e., a complete binary tree $\mathsf{T} = (\mathsf{V}, \mathsf{E})$ with each group member being associated to one of the tree's leaves. Every node $v \in \mathsf{E}$ has as associated data a so called public and private state. Its public state contains

- a PKE public key v.pk,
- a list v.unm of so called unmerged leaves,
- a parent-hash value v.pHash,
- a signature verification key v.svk (only if the node is a leaf), and
- a credential v.cred (only if the node is a leaf).

The private state of the node is v.sk the secret key associated to v.pk. MLS, as well as the variant defined in this work, enforce the so called tree-invariant stating that v.sk is known by exactly the user whose leaves have v as an ancestor. Further, a node can be *blank*, meaning that all associated values are empty. We use function blank(v) to blank a node, i.e., to set all its associated data to $\perp$.

Resolution and filtered path. The existence of blank nodes leads to the notion of resolution Res(v) of a node v which, intuitively, corresponds to the smallest covering of the set of descendants of v by non-blank nodes. We define it formally as follows;

- If v is a populated node, its resolution is $\text{Res}(v) = \{v\}$.
- If v is a blank leaf, its resolution is $\text{Res}(v) = \emptyset$.
- If v is a blank internal node, its resolution is defined recursively as

$$\text{Res}(v) = \cup_{u \in \text{child}(v)} \text{Res}(u).$$

If X is a set of nodes, we use the notation $\mathsf{Res}(X)$ to denote $\cup_{v \in X} \mathsf{Res}(v)$.

A leaf v's filtered path fil-path is the update path after removing all nodes the co-child of which has an empty resolution (and unmerged leafs set). I.e., if $\mathsf{path}(v) = (v_1, \ldots, v_\ell = r)$ then using the convention $v_0 = v$ we define

$$\mathsf{fil\text{-}path}(v) = (v_i \mid i \in \{1, \ldots, \ell\},\ (\mathsf{Res}(\mathsf{sib}(v_{i-1})) \cup \mathsf{sib}(v_{i-1}).\mathsf{unm}) \neq \emptyset)\,.$$

Finally, we sometimes have to identify the node in which two leaves' filtered paths meet. Consider leaves $u, v \in \mathsf{V}$ of the same depth with filtered paths $(u_1, \ldots, u_\ell = r) \leftarrow \mathsf{fil\text{-}path}(u)$ and $(v_1, \ldots, v_{\ell'} = r) \leftarrow \mathsf{fil\text{-}path}(v)$, respectively. We define their least common ancestor $\mathsf{lca}(u, v)$ as the node u_i in $\mathsf{path}(u)$ where i is minimal such that $u_i \in \mathsf{path}(v)$. Counting from the root we define the index ind-lca of their least common ancestor as

$$\mathsf{ind\text{-}lca}(u, v) = \max(a \mid u_{\ell - a} = v_{\ell' - a}, a \in \{0, \ldots, \ell - 1\}).$$

5.3 The Messaging Layer Security Protocol

We defer the formal description of the MLS protocol[2] [BBR+23] to Appendix B.2.1. In this section we provide an overview on its working principle. As the goal of this work is to investigate the protocol's communication complexity which is largely determined by the distribution of blank nodes in the underlying ratchet tree, our exposition mainly focuses on the mechanisms leading to the addition of blanks to the tree, or removal of blanks, respectively. For ease of exposition in this section we treat the key-packages kp (consisting of the corresponding user's PKE pk, signature verification key svk, and leaf credential cred) assigned to leaves in the ratchet tree as public key pk, and, similarly, assign the tree's root a key pair which is not the case in the full protocol (see Figure B.14).

Protocol state. The cryptographic state st each user of MLS keeps track of is conceptually divided as follows.

The underlying ratchet tree T. In this simplified exposition we identify users u with their associated leaf, simply writing v_u. As discussed in Section 5.2 every user u has a complete view of the tree's public part, but only has access to the secret keys $v.\mathsf{sk}$ of nodes that lie on their update path v_u. Sometimes when operating on ratchet trees we require the identity of the user running the algorithm, which we denote by $u.\mathrm{own}$.

[2] Formally, in this work we consider the CGKA underlying MLS. Thus, our exposition ignores the secret tree scheduling keys used to encrypt and authenticate payload messages.

The key schedule collects a number of secrets used for different purposes. These include, for example, the shared group secret referred to as the application secret, the initialization secret that is combined with the randomness obtained from re-keying a path to advance the key schedule to the next epoch, and a membership key used in combination with the user's signature key to authenticate protocol messages.

Further, the protocol employs consistency mechanism ensuring that all users have the same view of the protocol's state. This includes the transcript hash that essentially encodes the complete history of the group including all operations processed so far, as well as so-called tree-hash and parent-hash values that ensure consistency of the ratchet tree. We will discuss the latter two in a bit more detail at the end of this section.

Generating proposal and commit messages. Users issue proposal messages to add or remove a user to or from the group, respectively, as well as to update the personal key pair located at their leaf. In more detail, proposal messages

$$\mathsf{pmsg} = (\text{`type'}, \mathsf{ad})$$

consist of `type'}$ indicating the type of proposal, as well as some associated data. In case of an update proposal $\text{`upd' the latter contains a new public key to be included at the issuing user's leaf, in case of an addition `add'}$ the associated data $\mathsf{ad} = \mathsf{pk}$ is the added party's personal key, and finally in case of a removal $\text{`rem' it indicates the removed user's leaf. In case of an update proposal the issuing user stores the secret key sk corresponding to pk as a pending update in a list pendUpd.

To prevent unauthorized parties from issuing proposals pmsg has to be *framed*. Essentially, this means the proposal message and group context are signed by the issuing user as well as MACed using a key only available to group members. When processing proposal messages, users unframe the message by verifying the authenticity of pmsg before recovering $((\text{`type'}, \mathsf{ad}), u\text{-snd})$. Here $u\text{-snd}$ denotes the user issuing the message.

A user u can commit to a (potentially empty) list PMSG of proposals, implementing the proposed changes, advancing the group to the next epoch, and establishing a new application secret. This is done by executing the following steps.

1. Verify every proposal pmsg $\in$ PMSG.
2. Create a copy T′ of ratchet tree st.T and
   a) call apply-props(T′, PMSG) to apply the proposed changes to T′,
   b) call rekey-path(T′, $v_{u\text{-own}}$, 0) to rekey u-own's update path in T′ and generate an UpdPath object used to communicate the changes to the other group members.
3. Compute the new key schedule based on T′.
4. Authenticate UpdPath and bind it to PMSG.
5. Construct a welcome message if necessary.
6. Store the resulting state st′ (containing T′) as a pending commit and return the corresponding, framed, commit message cmsg, which, in particular, contains UpdPath.

The largest component of cmsg is UpdPath which itself is comprised of public keys and ciphertexts. The number of ciphertexts that have to be generated depends on the blanks present in T′.

To process a commit message issued by u, users essentially follow the same sequence of operations. More precisely, they recover T′ by first applying the proposals PMSG to their current view of the tree, and then recovering the new keys on the committing user's update path from UpdPath. They update the key schedule and verify that all changes were properly authenticated. If the latter is the case, the new key schedule and ratchet tree replace the ones stored in the user's state. If the processing user is the one issuing the commit operation, they can simply update their state to the one stored as a pending commit.

Applying proposals and path rekeying. When issuing (or processing) a commit operation, blanks are introduced in the ratchet tree by calling apply-props in step 2a and removed from it when calling rekey-path (or applying the re-key operation) in step 2b. For an overview of the two functions see Figures 5.2 and 5.3.

Calling apply-props on input a copy T′ of the current ratchet tree and list PMSG of proposals applies changes to the ratchet tree as follows. For pmsg $\in$ PMSG of type 'upd' the issuing user u-snd's leaf key is replaced by the one contained in pmsg. Then u-snd's update path is blanked. If pmsg proposes removing user u-rem, leaf $v_{u\text{-rem}}$ as well as the removed user's update path are blanked. If the removal results in the right half of the tree not having any populated leaves its size is halved using function truncate. If pmsg

```
Algorithm apply-props(T′, PMSG)
00 for pmsg ∈ PMSG:
01   parse (('type', ad), u-snd) ← pmsg
02   if ('type', ad) = ('upd', pk):
03     assign-kp(T′, v_{u-snd}, pk)          \\ replace u-snd's leaf pk
04     for v ∈ fil-path(v_{u-snd})           \\ blank u-snd's path
05       blank(v)
06     if u-own = u-snd                      \\ own update proposal
07       fetch sk ← pendUpd(pmsg)
08       v_{u-own}.sk ← sk
09   else if ('type', ad) = ('rem', u-rem):
10     blank-leaf(T′, v_{u-rem})             \\ blank u-rem's leaf
11     for v ∈ path(v_{u-rem})               \\ blank u-rem's path
12       blank(v)
13     T′ ← truncate(T′)                     \\ truncate tree if necessary
14   else if ('type', ad) = ('add', (u-add, pk)):
15     add-leaf(T′, u-add, pk)               \\ assign u-add a leaf
16     for v ∈ path(v_{u-add})               \\ set v_{u-add} as unmerged
17       v.unm ←∪ {v_{u-add}}
18 return T′
```

Figure 5.2: High-level overview on function apply-props for MLS. The function's formal description is in Figure B.18.

proposes adding user $u\text{-add}$ to the group, they get assigned the leftmost unpopulated leaf in the tree, doubling the tree size in the process (by adding blank nodes) if no such leaf exists. The leaf is assigned the public key contained in ad, and added to the set of unmerged leaves for every node in its update path.

To replace the keys on their path the committing user u calls rekey-path. It takes as input T′, u's leaf v_u, and, in our presentation, an additional index i_{cut} that allows stopping the rekeying i_{cut} steps early. While MLS uses $i_{\text{cut}} = 0$, looking ahead, it will be used in the proposed alternative method of issuing update proposals. Denoting $v_0 = v_u$ and $(v_1, \ldots, v_\ell) = \text{fil-path}(v_0)$ the function samples seed s_0 and iteratively generates seed s_i using a key-derivation function, in our presentation written as hash function H_1. It then generates key pairs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{PKE.Gen}(\mathsf{H}_2(s_i))$, H_2 being another hash function, to replace the keys associated to v_i.

In order to enable the other group members to recover the secret keys they should have access to, for all $j \in [\ell]$ the co-child v_{sib} with respect to v_u's update path is computed. Then the seed s_j is encrypted under all public keys associated to nodes contained in $\mathsf{Res}(v_{\mathsf{sib}})$, as well as the ones of unmerged leaves resulting in a collection C_j of

Algorithm rekey-path$(\mathsf{T}', v_u, i_{\text{cut}})$

$v_0 \leftarrow v_u$
$\mathrm{s}_0 \leftarrow_{\$} \{0,1\}^\lambda$ \\ generate leaf seed
$(v_0.\mathsf{pk}, v_0.\mathsf{sk}) \leftarrow \mathsf{PKE.Gen}(\mathsf{H}_2(s_0))$ \\ sample leaf key pair
$(v_1, \ldots, v_\ell) \leftarrow \mathsf{fil\text{-}path}(v_u)$
for $j = 1, \ldots, \ell - i_{\text{cut}}$:
  $\mathrm{s}_j \leftarrow \mathsf{H}_1(\mathrm{s}_{j-1})$ \\ generate seed
  $(v_j.\mathsf{pk}, v_j.\mathsf{sk}) \leftarrow \mathsf{PKE.Gen}(\mathsf{H}_2(\mathrm{s}_j))$ \\ sample key pair
  $C_j \leftarrow ()$
  $v_{\mathsf{sib}} \leftarrow \mathsf{sib}(v_{j-1})$ \\ determine co-child
  for $w \in \mathsf{Res}(v_{\mathsf{sib}}) \cup v_{\mathsf{sib}}.\mathsf{unm}$: \\ encrypt seed
    $C_j \leftarrow_{\cup} \mathsf{PKE.Enc}(w.\mathsf{pk}, \mathrm{s}_j)$
$\mathsf{UpdPath} \leftarrow (\mathsf{pk}_0, (\mathsf{pk}_1, C_1), \ldots, (\mathsf{pk}_\ell, C_\ell))$
for $v \in \mathsf{path}(v_u)$ \\ merge leaves
  $v.\mathsf{unm} \leftarrow \emptyset$
return $(\mathsf{T}', \mathsf{UpdPath})$

Figure 5.3: High-level overview on function rekey-path for MLS. The function's formal description is in Figure B.17.

ciphertexts. The update path object consists of

$$\mathsf{UpdPath} \leftarrow (\mathsf{pk}_0, (\mathsf{pk}_1, C_1), \ldots, (\mathsf{pk}_\ell, C_\ell)).$$

A user processing a commit message issued by u-snd replaces the public keys on u-snd's path by the ones contained in UpdPath. Then, by computing where their own and u-snd's update paths meet (say for index j) they are able to determine node v_{sib} relevant to them, and using the secret key in $\mathsf{Res}(v_{\mathsf{sib}}) \cup v_{\mathsf{sib}}.\mathsf{unm}$ they have access to decrypt one of the ciphertexts contained in C_j to recover s_j. In turn they are able to recover the secret keys $\mathsf{sk}_j, \ldots, \mathsf{sk}_\ell$ that they should have access to according to the tree invariant by applying H_1, H_2, and PKE.Gen.

Consistency mechanisms w.r.t the ratchet tree. MLS employs tree-hashes and parent-hashes (see Figure B.16) to ensure consistent views of the tree and provide improved security guarantees for users joining an existing group. The former essentially is a commitment to the entire public state of the ratchet tree, and is computed as a Merkle commitment. I.e., a node's tree hash corresponds to a hash of the node's public values and the tree-hashes of its children.

The parent hash captures information on how the tree's nodes were populated during the protocol execution. Parent hash values are computed as part of a user's commit, along the user's update path, however in reverse order, i.e., from root to leaf. Essentially, a node's parent hash attests to its parent's public key, as well as the tree hash of

the sub-tree rooted at its co-child. It plays an important role in preventing malicious insiders from being able to successfully invite users to malformed groups.

5.4 The Communication Cost of MLS for Random Sequences of Operations

We investigate the communication complexity of MLS. Section 5.4.1 defines a simple experiment used to generate random sequences of operations. We then proceed to compute bounds on the amount of blank nodes the MLS ratchet tree will have on average for such sequences (in Section 5.4.2) before computing a lower bound on the communication cost of commit operations (in Section 5.4.3).

5.4.1 Considered Scenario

As discussed in the introduction MLS exhibits a communication complexity that is logarithmic (in the number of users) in the best case, but can devolve to being linear in the worst case. In fact, the latter must be the case for any protocol that is built from standard primitives and achieves PCS as fast as MLS [BDR20, ANPPP23, BDG+22]. Beyond this, very little is known about MLS's performance in practice.

We initiate the study of MLS's communication cost under randomized sequences of operations. While ideally, one would evaluate the performance for sequences of operations obtained from real-world user data, such data is unfortunately, to the best of our knowledge, not publicly available. As a consequence, in this work we take a pragmatic approach and consider sequences determined by a simple experiment. Essentially, we generate a fixed number of update proposals per commit operation, the users performing the operations being sampled uniformly at random. While we do not claim this realistically reflects user behavior, on the one hand, it makes the problem of computing lower bounds approachable, and, on the other, provides simple intuition on the impact that blanking has that is not obscured by technical details of a more complicated model.

We define experiment Exp-Prop-Com in Figure 5.4. It is parameterized by the group size $N = 2^n \in \mathbb{N}$, integers P, C, and the number of rounds t. Essentially, it initializes a group of N users and then for t rounds generates update-proposal and commit operations as follows. First, in every round P users U_P are chosen uniformly at random from the group $[N]$ and issue an update proposal. This is followed by an uniformly random user U_C issuing a commit, which leads to the group moving to the next epoch and ends the round. To allow for the more general scenario that proposals are only issued every couple epochs, we allow for a number of epochs in between sending update proposal in which a commit, but no proposals, is issued. This is done by instead of

Experiment Exp-Prop-Com(n, P, C, t)

00 $N \leftarrow 2^n$
01 CGKA.Create$_{u_1}$(svk) \\ initialize group
02 PMSG $\leftarrow \emptyset$
03 for u in $\{u_2, \ldots, u_N\}$:
04 pmsg $\leftarrow$ CGKA.Prop$_{u_1}$('add', u)
05 PMSG $\leftarrow_\cup$ pmsg
06 (cmsg, wmsg) $\leftarrow$ CGKA.Com$_{u_1}$(PMSG)
07 CGKA.Proc$_{u_1}$(cmsg, PMSG)
08 for u in $\{u_2, \ldots, u_N\}$:
09 CGKA.Join$_u$(wmsg)
10 initialize $\mathrm{Cost} = ()$
11 for i from 1 to t: \\ issue rounds
12 $U_P \leftarrow_\$ \{S \subseteq [N] : |U_P| = P\}$
13 $U_C \leftarrow_\$ \{S \subseteq [N] : |U_C| = C\}$
14 $\mathrm{Cost} \leftarrow_\cup$ Round(U_P, U_C)
15 return Cost

Round(U_P, U_C)

16 cost $\leftarrow ()$
17 PMSG $\leftarrow \emptyset$
18 for $k \in U_P$:
19 pmsg $\leftarrow$ CGKA.Prop$_{u_k}$('upd', $\perp$)
20 PMSG $\leftarrow_\cup$ pmsg
21 for $k \in U_C$:
22 (cmsg, $\cdot$) $\leftarrow$ CGKA.Com$_{u_k}$(PMSG)
23 cost $\leftarrow_\cup$ |cmsg|
24 for $m \in [N]$:
25 CGKA.Proc$_{u_m}$(cmsg, PMSG)
26 PMSG $\leftarrow \emptyset$
27 return cost

Figure 5.4: Experiment Exp-Prop-Com with respect to group size $N = 2^n$ and numbers of proposals P and commits C executed over t rounds. We use subscripts u to denote that an algorithm is executed by user u, and omit algorithm outputs not relevant to the experiment.

sampling a single committer to end the round, sampling C committers uniformly at random. To analyze the communication cost of operation sequences generated in this way we record the size of each commit message generated during the experiment in an array Cost.

Our goal is to compute a lower bound on the expected value of $\mathrm{Cost}(t)$ after running the experiment for sufficiently many rounds. Recall that the amount of MACs, signatures, and public keys contained in commit messages is at most logarithmic in N. Thus, $\mathrm{Cost}(t)$ is dominated by the number of ciphertexts that have to be generated when calling rekey-path (see Figure 5.2) to communicate the new secret keys on the committer u-snd's path to the other users. Accordingly, we can bound the communication cost from below by

$$\mathrm{Cost}(t) \geq \sum_{v_i \in \text{co-path}(v_{u\text{-snd}})} |\mathsf{Res}(v_i)|,$$

the sum of resolution sizes on the committer's co-path, and in our analysis it suffices to derive a bound for this expression.

5.4.2 Expected Number of Blanks

Recall that the resolution of a node is determined by which of its descendants are blank. In this section we compute the expected probability of nodes at a certain depth being blank in the ratchet tree generated by experiment Exp-Prop-Com.

Expressing blanks as a Markov chain. Consider the sequence $(\mathsf{T}_i)_{i=1}^t$ of ratchet trees generated by experiment Exp-Prop-Com, where T_i is the tree all users store as part of their state after the ith call to round function Round. For a fixed intermediate node v we can define a random variable $(B_i(v))_{i=0}^t$ taking values in $\{0,1\}$ where 0 corresponds to being a populated node and 1 to v being blank. We write B_i instead of $B_i(v)$ when there is no ambiguity. We observe that $(B_i)_{i=0}^t$ forms a Markov chain. Indeed, v transitioning from blank to populated or vice versa only depends on the choice of the users U_P and U_C issuing proposals or commits in the current round. Since the chain is clearly irreducible, it has a unique stationary distribution. Denoting by $p_{b\to b'}$ the probability of the node switching from state $B_i = b$ to $B_{i+1} = b'$, the transition matrix of $(B_i)_{i=0}^t$ with respect to state vector $(0,1)^\top$ is given by

$$M = \begin{pmatrix} p_{0\to 0} & p_{1\to 0} \\ p_{0\to 1} & p_{1\to 1} \end{pmatrix}.$$

The eigenvalues of M are given by $p_{0\to 0} - p_{1\to 0}$ and 1 and an eigenvector of one-norm 1 with eigenvalue 1, corresponding to the stationary distribution, is given by

$$\left(\frac{p_{1\to 0}}{p_{1\to 0} + p_{0\to 1}}, \frac{p_{0\to 1}}{p_{1\to 0} + p_{0\to 1}} \right)^\top. \tag{5.1}$$

Analyzing the stationary distribution of $(B_i)_{i=0}^t$. Consider the state B_i of an intermediate node v at the end of round i. If $B_i = 1$ (i.e., v is blank at the end of round i), then after the proposals U_P and commits U_C of round $i+1$ have happened B_{i+1} will remain blank exactly if none of the users in U_C have v as an ancestor. Further, if $B_i = 0$ (i.e., v is a populated node at the end of round i), then B_{i+1} will remain non-blank unless at least one of the users in U_P has v as an ancestor while none of the users in U_C do. Thus, if we denote

$$p_P := \Pr[v \in \mathsf{ancs}(U_P)] \quad \text{and} \quad p_C := \Pr[v \in \mathsf{ancs}(U_C)]$$

we have that $p_{1\to 0} = p_C$ and by independence of the choice of proposer and committer that $p_{0\to 1} = (1-p_C)p_P$. By Equation 5.1 we obtain that the stationary distribution of a node being blank is given by

$$\mathrm{blanks}_{\mathrm{MLS}} := \left(\frac{p_C}{p_C + p_P(1-p_C)}, \frac{p_P(1-p_C)}{p_C + (1-p_C)p_P} \right)^\top \in [0,1]^2. \tag{5.2}$$

We now compute the probability of v being an ancestor of a proposing or committing leaf, respectively. Assume v is at depth $\mathsf{depth}(v) = d$. Then T has $N2^{-d}$ many leaves that are in $\mathsf{ancs}(v)$. Therefore there are $N(1-2^{-d})$ many leaves that are not in $\mathsf{ancs}(v)$ which implies that

$$p_P = \Pr[v \in \mathsf{ancs}(U_P)] = 1 - \Pr[v \notin \mathsf{ancs}(U_P)] = 1 - \frac{\binom{N(1-2^{-d})}{P}}{\binom{N}{P}}.$$

Analogously,

$$p_C = \Pr[v \in \mathsf{ancs}(U_C)] = 1 - \Pr[v \notin \mathsf{ancs}(U_C)] = 1 - \frac{\binom{N(1-2^{-d})}{C}}{\binom{N}{C}}.$$

Using Equation 5.2 we have shown the following.

Lemma 5.4.1. *For an execution of experiment* Exp-Prop-Com *and internal node v of depth* $\mathsf{depth}(v) = d$ *let $(B_i)_i = (B_i(v))_i$ be defined as above. Then the probability that v is blank in the stationary distribution $B = \mathrm{blanks}_{\mathrm{MLS}}$ is*

$$\mathbb{E}[B(v) = 1] = \frac{\left(1 - \frac{\binom{N(1-2^{-d})}{P}}{\binom{N}{P}}\right)\frac{\binom{N(1-2^{-d})}{C}}{\binom{N}{C}}}{1 - \frac{\binom{N(1-2^{-d})}{C}}{\binom{N}{C}}\frac{\binom{N(1-2^{-d})}{P}}{\binom{N}{P}}}. \tag{5.3}$$

Before turning to computing a lower bound on the expected commit size in experiment Exp-Prop-Com we look at some simple cases:

When $P = 1 = C$, we have that $p_P = 2^{-d} = p_C$. Therefore,

$$\mathbb{E}[B(v) = 1] = (1 - 2^{-d})/(2 - 2^{-d}) \approx 1/2 \text{ for } d \text{ large enough.}$$

If $P + C = O(1)$,

$$\begin{aligned}\mathbb{E}[B(v) = 1] &\approx \frac{(1 - (1 - 2^{-d})^P)(1 - 2^{-d})^C}{1 - (1 - 2^{-d})^P(1 - 2^{-d})^C} \\ &\approx \frac{(1 - (1 - P2^{-d}))(1 - C2^{-d})}{1 - (1 - (P + C)2^{-d})} \\ &\approx \frac{P}{P + C} \text{ for } d \text{ large enough.}\end{aligned} \tag{5.4}$$

In both examples we see a constant probability that a node is blank which, intuitively, means that a commit should have a high upload cost. We formalize this in the next section.

5.4.3 Lower Bound on Sent Ciphertexts

In this section we prove a lower bound on MLS's communication cost for sequences of operations generated by Exp-Prop-Com. As it turns out, already a constant number of proposals per commit leads to commits being exponential in $n = \log(N)$, a far shot from the best case communication complexity. More precisely we will prove the following.

Theorem 5.4.2. *If $P+C = O(1)$ and $t > 2N^2 \log(N)$, then there exists a constant $\varepsilon > 0$ such that the expected cost of a commit after Experiment* Exp-Prop-Com(n, P, C, t) *(Figure 5.4) satisfies the following inequality:*

$$\mathbb{E}[\mathrm{Cost}(t)] \geq \Omega(N^{\log_2(1+(1-\varepsilon)\frac{P}{P+C})}).$$

Before proving this result, we show a weaker result that already gives some of the main ideas used in the proof of Theorem 5.4.2. We start by studying the size of the resolution of a node v_d at depth $d \in \{1, \ldots, \log(N) - 1\}$:

$$\begin{aligned}
\mathbb{E}[|\mathsf{Res}(v_d)|] &\geq \mathbb{E}[|\mathsf{Res}(v_d)| \mid B(v_d) = 0] \Pr[B(v_d) = 0] \\
&\quad + \mathbb{E}[|\mathsf{Res}(v_d)| \mid B(v_d) = 1] \Pr[B(v_d) = 1] \\
&= \Pr[B(v_d) = 0] \\
&\quad + \mathbb{E}[|\mathsf{Res}(\mathsf{right}(v_d))| + |\mathsf{Res}(\mathsf{left}(v_d))| \mid B(v_d) = 1] \Pr[B(v_d) = 1] \\
&= \Pr[B(v_d) = 0] \\
&\quad + 2\mathbb{E}[|\mathsf{Res}(\mathsf{left}(v_d))| \mid B(v_d) = 1] \Pr[B(v_d) = 1].
\end{aligned}$$

One can show that

$$\mathbb{E}[|\mathsf{Res}(\mathsf{left}(v_d))| \mid B(v_d) = 1] \geq \mathbb{E}[|\mathsf{Res}(\mathsf{left}(v_d))|]. \tag{5.5}$$

This implies that for a node v_d at depth $d \in \{1, \ldots, \log(N) - 1\}$,

$$\mathbb{E}[|\mathsf{Res}(v_d)|] \geq \Pr[B(v_d) = 0] + 2\mathbb{E}[|\mathsf{Res}(\mathsf{left}(v_d))|] \Pr[B(v_d) = 1].$$

A recursive application of this inequality yields

$$\mathbb{E}[|\mathsf{Res}(v_d)|] \geq 1 + \sum_{i=0}^{\log(N)-d-1} 2^i \prod_{j=0}^{i} \Pr[B(v_{d+j}) = 1].$$

Now let's assume that $P = 1 = C$. We know that for $d \geq 3$, $\mathbb{E}[B(v) = 1] \approx 1/2$. Therefore we obtain that

$$\begin{aligned}\mathbb{E}[\mathrm{Cost}(t)] &\geq \sum_{d=3}^{\log(N)} (1 + \sum_{i=0}^{\log(N)-d-1} 1/2) \\ &= \log(N) - 2 + \sum_{d=3}^{\log(N)} (\log(N) - d)/2 \\ &= \log(N) - 2 + (\log(N) - 3)(\log(N) - 2)/4 \\ &\approxeq \log^2(N)/4 - \log(N)/4.\end{aligned}$$

A similar computation when $P + C = O(1)$ and $P > C$ shows $\mathbb{E}[\mathrm{Cost}(t)] = \Omega(N^{\log_2(\frac{2P}{P+C})})$.

The gap between these bounds and the one stated in Theorem 5.4.2 is due to Equation 5.5 being non-tight. This is solved by studying how the size of the resolution of a node grows when several of its ancestors are blank.

Proof of Theorem 5.4.2. In order to prove the theorem we first establish some helpful results. Let's consider the random variable $|\mathsf{Res}(v)|$ that denotes the size of the resolution of a node v. This corresponds to the number of ciphertexts that have to be uploaded when a commit includes a fresh key for v's parent. The central idea of the lower bound is to relate the expected resolution size of a node to the expected resolution size of its children, and iterate this expectation in order to get a lower bound that's exponential in $\log(N)$. In the proof, we keep conditioning on an increasingly complicated event, and only consider the probability of a node being blank conditioned on this event. This allows us to argue using a local property whereas the resolution size of a node depends on the entire subtree of the node.

We now introduce some notation. Let $\text{Anc-bl}_t(d, v)$ be the event that all ancestors of v up to depth d are blank after the experiment ends. We usually drop the subindex t as we do with the other random variables considered in this section. When the node v is clear from context, we write $\text{Anc-bl}(d)$. Let v_a denote the ancestor of v such that $\mathsf{depth}(v_a) = d$. We use $\mathsf{T}_{\mathrm{sub}}$ to denote the full binary subtree with root v_a and leaves the nodes in T that have v_a as an ascendant and have the same depth as v. Let n' be the number of leaves in $\mathsf{T}_{\mathrm{sub}}$. Notice that $n' = 2^{\mathsf{depth}(v)-d}$.

Lemma 5.4.3. *Let $X \sim \mathrm{Geo}(q)$ be distributed according to the geometric distribution with $q \in (0, 1)$ and let $Z = X \mod (P + C)$. Let $i, j \in \{1, 2, \ldots, P + C\}$. Let s be an integer multiple of $P + C$, i.e. $s = m(P + C)$ for some $m > 0$. Then,*

$$\Pr[Z = i \mid X \leq s] \geq (1 - q)^{P+C-1} \Pr[Z = j \mid X \leq s].$$

Proof. This is an almost immediate consequence of some elementary computation with finite geometric series. Let $i, j \in \{1, 2, \ldots, P + C\}$. Notice that

$$\begin{aligned}\Pr[Z = i \mid X \leq s] &= \frac{1}{1-(1-q)^s} \sum_{h=0}^{m-1} (1-q)^{(P+C)h+i-1} q \\ &= \frac{1}{1-(1-q)^s} (1-q)^{i-1} q \sum_{h=0}^{m-1} (1-q)^{(P+C)h}.\end{aligned}$$

Now, notice that the only factor that depends on i is $(1-q)^i$. The result follows from the observation that $(1-q)^{i-j} \geq (1-q)^{P+C-1}$. □

We now introduce the event $\mathrm{All\text{-}act}$. Intuitively, $\mathrm{All\text{-}act}$ simply states that all users in $\mathsf{T}_{\mathrm{sub}}$ acted relatively recently. We'll later show that $\mathrm{All\text{-}act}$ happens with overwhelming probability. More precisely, $\mathrm{All\text{-}act}$ simply states that if we look at the last $2N^2 \log N$ actions in T, ordered from the most recent to the least recent, there's a new user from $\mathsf{T}_{\mathrm{sub}}$ in every at most $2N \log N$ steps.

In other words, we're conditioning on the following event: Let X_i be the random variable denoting how many actions it takes for the ith user to appear after the $(i-1)$ previous users already did. All these users must be different. We are assuming that $X_i \leq 2N \log N$ for every $i \in \{1, 2, \ldots, n'\}$. More formally, we will condition on X_i being less than some $s \geq 2N \log N$ that is a multiple of $P + C$, but we omit this since it does not affect the proof and introduces unnecessary notation.

Now, we analyze that the probability of v being blank conditioned on all of its ancestors up to depth d being blank and $\mathrm{All\text{-}act}$.

Lemma 5.4.4. *Let $\varepsilon > 0$ and $d \in \mathbb{N}$ such that $\varepsilon \geq 2^{\mathsf{depth}(v)-d+1}(P+C-1)/N$ where v is a node at depth $\mathsf{depth}(v) > d$. Then,*

$$\Pr[B(v) = 1 \mid \mathrm{All\text{-}act}, \mathrm{Anc\text{-}bl}(d)] \geq \frac{1}{2}(1 + (1-\varepsilon)\frac{P}{P+C}).$$

Proof. We describe an experiment that will help us determine the probabilities of possible states of $\mathsf{T}_{\mathrm{sub}}$.

For $i \in [n']$, let $X_i \sim \mathrm{Geo}(q_i)$ where $q_i = \frac{n'-i+1}{N} \leq 2^{\mathsf{depth}(v)-d}/N$. Intuitively, X_i corresponds to the number of actions that occurred after the action corresponding to X_{i-1} until a "new" member acted on the tree.

Define $Y_i := \sum_{k=1}^{i} X_k \mod (P+C)$. By linearity, we have that $Y_i = X_i + Y_{i-1} \mod (P+C)$. Intuitively, the Y_i's help us track whether this action was a commit or a proposal. More formally, notice that if $Y_i \leq C$, then the user that acted ith from last

committed, and if $C+1 \leq Y_i \leq P+C$ the user proposed. We also have that the Y_i's form a Markov chain, i.e. Y_i conditioned on Y_{i-1} is independent of the previous Y_k's.

Moreover, we sample a uniformly random string of length n' elements in order to assign each leaf node to a position. For example, $\sigma = 312$ ($\sigma(1) = 3, \sigma(2) = 1, \sigma(3) = 2$) would mean that the leftmost user was the first to act, the 'central' user was last to act and the third user acted second. Notice that $\sigma(1)$ tells us the position in which v acted and $\sigma(2)$ tells us the position in which sib(v) acted. For example, $\sigma(1) = 2$ would imply that the action on node v was second to last on $\mathsf{T}_{\mathrm{sub}}$. Notice that σ cannot repeat elements. Now we study the probability that v is blank conditioned on the events All-act and $\text{Anc-bl}(d)$:

$$\begin{aligned}
&\Pr[B(v) = 1 \mid \text{All-act}, \text{Anc-bl}(d)] \\
&= \Pr[B(v) = 1 \mid \text{All-act}, \text{Anc-bl}(d), \sigma(1) < \sigma(2)] \Pr[\sigma(1) < \sigma(2) \mid \text{All-act}, \text{Anc-bl}(d)] \\
&+ \Pr[B(v) = 1 \mid \text{All-act}, \text{Anc-bl}(d), \sigma(1) > \sigma(2)] \Pr[\sigma(1) > \sigma(2) \mid \text{All-act}, \text{Anc-bl}(d)].
\end{aligned}$$

We have that $\Pr[\sigma(1) < \sigma(2) \mid \text{All-act}, \text{Anc-bl}(d)] = \frac{1}{2}$, as which node comes later is clearly independent of $\text{All-act}, \text{Anc-bl}(d)$ due to symmetry. Moreover, if $\sigma(1) < \sigma(2)$, i.e. v acts later than sib(v), we have that v is blank since par(v) is blank as we're conditioning on $\text{Anc-bl}(d)$. Thus, we have that

$$\begin{aligned}
&\Pr[B(v) = 1 \mid \text{All-act}, \text{Anc-bl}(d)] \\
&\quad = \frac{1}{2}(1 + \Pr[B(v) = 1 \mid \text{All-act}, \text{Anc-bl}(d), \sigma(1) > \sigma(2)]).
\end{aligned}$$

For the rest of the proof, we'll prove that

$$\Pr[B(v) = 1 \mid \text{All-act}, \text{Anc-bl}(d), \sigma(1) > \sigma(2)] \geq (1 - \varepsilon)\frac{P}{P+C}$$

Before we get into the details, let's explain the intuition behind the proof. The proof rests on the observation that if $X_k \sim \mathrm{Geo}(q)$ for small q, $X_k \mod (P+C)$ is close to a uniform random variable. Moreover, X_k and Y_{k-1} are independent, so $Y_k = X_k + Y_{k-1} \mod (P+C)$ is also close to being a uniform distribution. Thus, no matter what complicated event we're conditioning on, the probability that $1 + C \leq Y_k \leq P + C$ will be approximately $\frac{P}{P+C}$.

Let $f(y_1, y_2, \ldots, y_{n'}, \sigma)$ be a predicate that returns 1 if the assignment is compatible with $\text{Anc-bl}(d), \sigma(1) > \sigma(2)$ and 0 otherwise. Let $Z = \Pr[\text{All-act}, \text{Anc-bl}(d), \sigma(1) > \sigma(2)]$. For notational simplicity, we'll write $\sum$ instead of

$$\sum_{k=1}^{n'} \sum_{\substack{\sigma \in S_{n'} \\ \sigma(1)=k}} \sum_{y_1=1}^{P+C} \sum_{y_2=1}^{P+C} \cdots \sum_{y_{k-1}=1}^{P+C} \sum_{y_{k+1}=1}^{P+C} \cdots \sum_{y_{n'}=1}^{P+C}$$

in what follows. Also for notational convenience, we omit the conditioning on All-act, even though it's implicit in every probability we write. Then, we have that $\Pr[B(v) = 1 \mid \text{All-act}, \text{Anc-bl}(d), \sigma(1) > \sigma(2)]$ is equal to

$$\frac{1}{Z} \sum \sum_{y_k=1+C}^{P+C} f(y_1, y_2, \ldots, y_{n'}, \sigma) \frac{1}{(n'-1)!} \Pr[Y_1 = y_1] \prod_{l=2}^{n'} \Pr[Y_l = y_l \mid Y_{l-1} = y_{l-1}], \tag{5.6}$$

where we used the Markov property of the Y_k's. Also notice that since we're interested in the probability of v being blank, we're only summing up the values in the support of Y_k that start from $C+1$, so we are ignoring the first C values. Again, for notation simplicity, we introduce a new variable

$$\rho := \Pr[Y_1 = y_1] \prod_{l>1, l\neq k, l\neq k+1} \Pr[Y_l = y_l \mid Y_{l-1} = y_{l-1}].$$

Rearranging Equation 5.6, we get

$$\begin{aligned} &\frac{1}{Z} \sum \frac{1}{(n'-1)!} \rho f(y_1, y_2, \ldots, y_{n'}, \sigma) \\ &\quad \cdot \sum_{y_k=1+C}^{C+P} \Pr[Y_{k+1} = y_{k+1} \mid Y_k = y_k] \Pr[Y_k = y_k \mid Y_{k-1} = y_{k-1}] \end{aligned}$$

Let's now argue why it was possible to take $f(y_1, y_2, \ldots, y_{n'}, \sigma)$ out of the sum, i.e. why it's independent of $Y_{\sigma(1)}$. If $\sigma(1) < \sigma(2)$, $f(y_1, y_2, \ldots, y_{n'}, \sigma) = 0$ no matter what Y_k is. If $\sigma(1) > \sigma(2)$, $\sigma(1)$ has no effect on its ancestors, so Y_k doesn't matter for $\text{Anc-bl}(d)$. Now, notice that if we instead sum all values in the support of Y_k, we have by the law of total probability that

$$\begin{aligned} 1 = &\frac{1}{Z} \sum \frac{1}{(n'-1)!} \rho f(y_1, y_2, \ldots, y_{n'}, \sigma) \\ &\quad \cdot \sum_{y_k=1}^{P+C} \Pr[Y_{k+1} = y_{k+1} \mid Y_k = y_k] \Pr[Y_k = y_k \mid Y_{k-1} = y_{k-1}] \end{aligned}$$

Moreover, using the Markov property of the Y_i, we have that

$$\begin{aligned} &\Pr[Y_{k+1} = y_{k+1} \mid Y_{k-1} = y_{k-1}] \\ &\quad = \sum_{y_k=1}^{P+C} \Pr[Y_{k+1} = y_{k+1} \mid Y_k = y_k, Y_{k-1} = y_{k-1}] \Pr[Y_k = y_k \mid Y_{k-1} = y_{k-1}] \\ &\quad = \sum_{y_k=1}^{P+C} \Pr[Y_{k+1} = y_{k+1} \mid Y_k = y_k] \Pr[Y_k = y_k \mid Y_{k-1} = y_{k-1}] \end{aligned} \tag{5.7}$$

Thus, it suffices to show that

$$\sum_{y_k=1+C}^{P+C} \Pr[Y_{k+1} = y_{k+1} \mid Y_k = y_k] \Pr[Y_k = y_k \mid Y_{k-1} = y_{k-1}]$$
$$\geq (1-\varepsilon)\frac{P}{P+C} \Pr[Y_{k+1} = y_{k+1} \mid Y_{k-1} = y_{k-1}].$$

Let $t_{max} := \arg\max_t \Pr[Y_{k+1} = y_{k+1} \mid Y_k = t] \Pr[Y_k = t \mid Y_{k-1} = y_{k-1}]$. Notice that

$$\Pr[Y_{k+1} = y_{k+1} \mid Y_k = t_{max}] \Pr[Y_k = t_{max} \mid Y_{k-1} = y_{k-1}]$$
$$\geq \frac{1}{P+C} \Pr[Y_{k+1} = y_{k+1} \mid Y_{k-1} = y_{k-1}] \tag{5.8}$$

which follows from Equation 5.7. Then, by Lemma 5.4.3 (observe that X_k is bounded because of the event $\mathrm{All\text{-}act}$), for any $1 + C \leq t \leq P + C$, we have that

$$\begin{aligned}
&\Pr[Y_{k+1} = y_{k+1} \mid Y_k = t] \Pr[Y_k = t \mid Y_{k-1} = y_{k-1}] \\
&\geq \Pr[Y_{k+1} = y_{k+1} \mid Y_k = t_{max}] \Pr[Y_k = t_{max} \mid Y_{k-1} = y_{k-1}](1-q)^{2(P+C-1)} \\
&\geq (1 - 2q(P+C-1))\frac{1}{P+C} \Pr[Y_{k+1} = y_{k+1} \mid Y_{k-1} = y_{k-1}] \\
&\geq (1-\varepsilon)\frac{1}{P+C} \Pr[Y_{k+1} = y_{k+1} \mid Y_{k-1} = y_{k-1}]
\end{aligned}$$

where the second inequality follows from Lemma 2.1.5 and Equation 5.8. The last inequality follows from the assumption that $\varepsilon \geq 2^{\mathsf{depth}(v)-d+1}(P+C-1)/N$ and the fact that $q \leq 2^{\mathsf{depth}(v)-d}/N$. Thus,

$$\sum_{t=1+C}^{P+C} \Pr[Y_{k+1} = y_{k+1} \mid Y_k = t] \Pr[Y_k = t \mid Y_{k-1} = y_{k-1}]$$
$$\geq (1-\varepsilon)\frac{P}{P+C} \Pr[Y_{k+1} = y_{k+1} \mid Y_{k-1} = y_{k-1}].$$

We thus conclude the proof. □

We have two more lemmata left to prove before moving onto combining our results.

Lemma 5.4.5. *Assuming at least* $2N^2 \log(N)$ *actions took place,* $\Pr[\mathrm{All\text{-}act}] \geq 1 - \frac{1}{N}$.

Proof. Notice that for any $i \in \{1, \ldots, N\}$,

$$\Pr[X_i > 2N \log N] \leq (1 - \frac{1}{N})^{2N \log N} \leq \exp(-2\log(N)) = N^{-2}.$$

Then, $\Pr[\mathrm{All\text{-}act}] \geq 1 - \frac{1}{N}$ by the union bound over all the X_i . □

Lemma 5.4.6. *Let $\varepsilon > 0$ and $d \in \mathbb{N}$ such that $\varepsilon \geq 2^{-d}(P + C - 1)$ and consider a node v such that $\log(N) - 1 \geq \mathsf{depth}(v) > d$. Then, we have that*

$$\begin{aligned}&\mathbb{E}[|\mathsf{Res}(v)| \mid \text{All-act}, \text{Anc-bl}(d)] \\ &\quad \geq (1 + (1-\varepsilon)\frac{P}{P+C})\mathbb{E}[|\mathsf{Res}(\mathsf{left}(v))| \mid \text{All-act}, \text{Anc-bl}(d, \mathsf{left}(v))].\end{aligned}$$

Proof. Observe that $\varepsilon \geq 2^{\mathsf{depth}(v)-d+1}(P + C - 1)/N$ since $\log(N) - 1 \geq \mathsf{depth}(v)$. By Lemma 5.4.4, we have that

$$\begin{aligned}&\mathbb{E}[|\mathsf{Res}(v)| \mid \text{All-act}, \text{Anc-bl}(d, v)] \\ &\quad \geq \mathbb{E}[|\mathsf{Res}(v)| \mid \text{All-act}, \text{Anc-bl}(d, v), B(v) = 1] \cdot \\ &\qquad\qquad \cdot \Pr[B(v) = 1 \mid \text{All-act}, \text{Anc-bl}(d, v)] \\ &\quad \geq \mathbb{E}[|\mathsf{Res}(v)| \mid \text{All-act}, \text{Anc-bl}(d, v), B(v) = 1]\frac{1}{2}(1 + (1-\varepsilon)\frac{P}{P+C}) \\ &\quad = 2\mathbb{E}[|\mathsf{Res}(\mathsf{left}(v))| \mid \text{All-act}, \text{Anc-bl}(d, v), B(v) = 1]\frac{1}{2}(1 + (1-\varepsilon)\frac{P}{P+C}) \\ &\quad = \mathbb{E}[|\mathsf{Res}(\mathsf{left}(v))| \mid \text{All-act}, \text{Anc-bl}(d, \mathsf{left}(v))](1 + (1-\varepsilon)\frac{P}{P+C}).\end{aligned}$$

□

We are now ready to prove the section's main theorem.

Proof of Theorem 5.4.2. Let v be any node at depth $d(v) = d$ and let $\varepsilon > 0$ be a small constant such that $\varepsilon \geq 2^{-d}(P + C - 1)$. The fact that such d and ε exist follows from the assumption that $P + C = O(1)$. We show that the expected resolution size of v is $\Omega(N^{\log_2(1+(1-\varepsilon)\frac{P}{P+C})})$. Since any commit will have a node of said depth in its copath, this is sufficient for the proof. We start with the simple observation that

$$\begin{aligned}\mathbb{E}[|\mathsf{Res}(v)|] &\geq \mathbb{E}[|\mathsf{Res}(v)| \mid \text{All-act}, B(v) = 1] \Pr[\text{All-act}, B(v) = 1] \\ &= 2\mathbb{E}[|\mathsf{Res}(\mathsf{left}(v))| \mid \text{All-act}, \text{Anc-bl}(d, \mathsf{left}(v))] \Pr[\text{All-act}, B(v) = 1].\end{aligned}$$

We use Lemma 5.4.5 in order to obtain that

$$\begin{aligned}\Pr[\text{All-act}, B(v) = 1] &\geq \Pr[B(v) = 1] - \Pr[\neg\text{All-act}] \\ &\geq \Pr[B(v) = 1] - \frac{1}{N}.\end{aligned}$$

It follows from repeatedly applying Lemma 5.4.6 that

$$\begin{aligned}
&\mathbb{E}[|\mathsf{Res}(v)|] \\
&\quad \geq 2(\Pr[B(v)=1] - \frac{1}{N})\mathbb{E}[|\mathsf{Res}(\mathsf{left}(v))| \mid \text{All-act}, \text{Anc-bl}(d, \mathsf{left}(v))] \\
&\quad \geq 2(\Pr[B(v)=1] - \frac{1}{N})(1 + (1-\varepsilon)\frac{P}{P+C})^{\log_2 N - d(\mathsf{left}(v))}.
\end{aligned}$$

From Lemma 5.4.1 we obtain that

$$\begin{aligned}
&\mathbb{E}[|\mathsf{Res}(v)|] \\
&\quad \geq \Omega\left(\left(\frac{\left(1 - \frac{\binom{N(1-2^{-d})}{P}}{\binom{N}{P}}\right)\frac{\binom{N(1-2^{-d})}{C}}{\binom{N}{C}}}{1 - \frac{\binom{N(1-2^{-d})}{C}}{\binom{N}{C}}\frac{\binom{N(1-2^{-d})}{P}}{\binom{N}{P}}} - \frac{1}{N}\right)\left(1 + (1-\varepsilon)\frac{P}{P+C}\right)^{\log_2 N - d}\right).
\end{aligned}$$

We make use of Equation 5.4 and the expression above simplifies to $\mathbb{E}[\mathrm{Cost}(t)] \geq \Omega(N^{\log_2(1+(1-\varepsilon)\frac{P}{P+C})})$, as desired. □

The proof above does not explicitly state the constant that appears in the lower bound of $\mathbb{E}[\mathrm{Cost}(t)] = \Omega(N^{\log_2(1+(1-\varepsilon)\frac{P}{P+C})})$. However, one can obtain an explicit constant from the proof. For instance if $P = 1 = C$, one can take $\varepsilon = 1/8$ and then for a node v_3 at depth 3 and a node v_4 at depth 4, the proof shows that

$$\begin{aligned}
\mathbb{E}[|\mathsf{Res}(v_3)|] &\geq 2 \cdot 1/2 \cdot (1 + (1 - 1/8) \cdot 1/2)^{\log_2 N - 3 - 1} \\
\mathbb{E}[|\mathsf{Res}(v_4)|] &\geq 2 \cdot 1/2 \cdot (1 + (1 - 1/8) \cdot 1/2)^{\log_2 N - 4 - 1}.
\end{aligned}$$

This implies that

$$\begin{aligned}
\mathbb{E}[\mathrm{Cost}(t)] &\geq \mathbb{E}[|\mathsf{Res}(v_3)|] + \mathbb{E}[|\mathsf{Res}(v_4)|] \\
&\approxeq 0.39 \cdot N^{0.52}.
\end{aligned}$$

For small values of N the previously shown bound of $\log^2(N)/4 - \log(N)/4$ is more relevant.

5.5 MLS-Cutoff: an Alternative Method for Update Proposals

We introduce MLS-Cutoff a protocol variant of MLS that achieves better communication complexity under random sequences of operations as defined in experiment Exp-Prop-Com. As discussed in the previous section, blanking the path of users

issuing an update proposals already for a small constant number of proposals leads to a constant fraction of the tree's node being blank on expectation, which in turn substantially negatively impacts the size of commit messages.

To prevent introducing blanks in the tree, one could have users perform a rekey-path operation with every update proposal. In fact, this is the approach taken by the continuous group-key agreement scheme CoCoA [AAN+22]. However, taking this approach has two downsides, both due to the fact that proposals can be issued concurrently, meaning users make concurrent modifications to the ratchet tree.

1. If two users u_1, u_2 rekey their path concurrently, u_1 will encrypt a path secret s_j under a public known to u_2 that is supposed to be replaced by u_2's rekey operation. As a consequence CoCoA suffers from slower PCS. In MLS, if every compromised user issues an update proposal and another user commits to the proposals, then the new application secret will be secure. In CoCoA, on the other hand, it can take up to depth(T) rounds to recover from compromise.

2. Concurrently rekeying paths prevents users to compute and sign parent hashes, as they cannot anticipate which keys will be placed in the sub-trees rooted at their co-path nodes. (Recall that the parent-hash value of node v is computed with respect to the tree-hash of its sibling sib(v).) As a consequence, CoCoA only achieves semi-active but neither active [ACJM20] nor insider security [AJM22].

We observe that the issues above only occur from the point on where u_1 and u_2's update paths meet. Further, if the number of issued update proposals is small, with high probability the users' update paths merge high up in the tree. Our protocol MLS-Cutoff makes use of this observation by having update proposals only re-key the path up to a certain point. Looking ahead, proposals do not affect the $\log(\text{depth}(\mathsf{T}))$ topmost levels of the tree. Still, when applying the proposals it can be the case that a node is affected by more than one rekey operation if the issuing users are situated close enough in the tree. In this case we have to resort to blanking the node. This turns out to be sufficient to prevent the two issues outlined above.

5.5.1 Protocol Description

In this section we discuss the modifications MLS-Cutoff makes to the MLS protocol and defer its formal description to Appendix B.2.2. MLS-Cutoff differs in two aspects from MLS, namely how update proposal messages are generated, and how proposals are applied to the ratchet tree using apply-props.

The protocol is parameterized by a cutoff parameter i_{cut}. CGKA.Prop works the same as before for removals or additions of users. When issuing an update, however, the

user creates a copy T′ of the ratchet tree and calls the path rekeying function (see Figure 5.3) to obtain

$$(\mathsf{T}', \mathsf{UpdPath}) \leftarrow \mathsf{rekey\text{-}path}(v_u, i_{\mathrm{cut}}).$$

The proposal message $\mathsf{pmsg} = (\text{'upd'}, \mathsf{UpdPath})$ is framed and communicated to the rest of the group, and T′ stored as a pending update.

Commit operations work in the same way as in MLS, except for using a modified subroutine apply-props in step 2a. For an overview on apply-props see Figure 5.5. The function modifies the copied tree T′ and processes removals and additions in the same way as MLS, i.e., by blanking the leaf and path of removed users and adding added users as unmerged leaves. Update proposals $(\text{'upd'}, \mathsf{UpdPath})$ from issuing user $u\text{-snd}$ are processed by first updating the keys of affected nodes. More precisely, the public keys $\mathsf{pk}_0, \ldots, \mathsf{pk}_{\ell - i_{\mathrm{cut}}}$ contained in UpdPath are used to replace the ones on $u\text{-snd}$'s update path $(v_1, \ldots, v_\ell) = \mathsf{path}(v_0)$, where $v_0 = v_{u\text{-snd}}$. Further, the processing user u recovers all secret keys they should have access to, i.e., the ones from the least common ancestor v_j of v_u and $v_{u\text{-snd}}$ up to $v_{\ell - i_{\mathrm{cut}}}$ (if the v_j is above $v_{\ell - i_{\mathrm{cut}}}$ no secret keys are recovered), and assigns them to the corresponding nodes. This is abstracted as helper function recover-sks in Figure 5.5 that takes as input T′ and ciphertext collection C_j and returns the updated view of T′ with replaced secret keys.

After the keys of nodes affected by pmsg have been updated, apply-props blanks nodes if necessary. I.e., all nodes $(v_{\ell - i_{\mathrm{cut}}+1}, \ldots, v_\ell)$ on $u\text{-snd}$'s update path after the cutoff point are blanked. Further, nodes affected by at least two update proposals are blanked as well. To this end the function keeps track of a set V_{col} containing all nodes the keys of which have been replaced so far, that is updated at the end of processing pmsg.

After having executed the alternative version of apply-props in step 2a the commit algorithm CGKA.Com proceeds in the same way as MLS. In particular, in step 2b the committing user issues a *full* path rekey operation $\mathsf{rekey\text{-}path}(u, 0)$, which establishes a new root secret.

Commits to a collection of proposals are processed in an analogous way to MLS, i.e, the only difference being the use of the modified apply-props function.

5.5.2 Security

MLS-Cutoff achieves the same level of security as MLS. More formally in Appendix B.2.2 we prove the following theorem

Theorem 5.5.1. *If* PKE *is IND-CCA2 secure,* SIG *is SEUF-CMA secure and calls to* $\mathsf{H}_1, \mathsf{H}_2$, H *and* MAC *are replaced by calls to a random oracle* $\mathcal{G}$*, then* MLS-Cutoff *securely realizes* $(\mathcal{F}_{\mathsf{AS}}^{\mathsf{I}}, \mathcal{F}_{\mathsf{KS}}^{\mathsf{I}}, \mathcal{F}_{\mathsf{CGKA}})$ *in the* $(\mathcal{F}_{\mathsf{AS}}, \mathcal{F}_{\mathsf{KS}}, \mathcal{G})$*-hybrid model.*

Algorithm apply-props(T', PMSG)

```
00 if exists ((‘upd’, ad), u-own) = pmsg ∈ PMSG        \\ own proposal
01   T′ ← pendUpd(pmsg)                                \\ recover corresponding tree
02 V_col ← ∅                                           \\ set collecting potential collisions
03 for pmsg ∈ PMSG:
04   parse ((‘type’, ad), u-snd) ← pmsg
05   if (‘type’, ad) = (‘upd’, UpdPath)
06     if u-snd = u-own                                \\ own proposal already handled
07       skip to next pmsg
08   \\ rekey nodes
09     parse (pk_0, (pk_1, C_1), ..., (pk_{ℓ−i_cut}, C_{ℓ−i_cut})) ← UpdPath
10     v_0 ← v_{u-snd}
11     (v_1, ..., v_ℓ) ← path(v_0)
12     for j = 0, ..., ℓ − i_cut
13       v_j.pk ← pk_j
14       if v_j = lca(v_{u-own}, v_{u-snd})            \\ least common ancestor
15         T′ ← recover-sks(T′, C_j)                   \\ recover secret keys
16   \\ blank nodes
17     for j ∈ {1, ..., ℓ − i_cut}
18       if v_j ∈ V_col                                \\ blank collisions
19         blank(v_j)
20     for j ∈ {ℓ − i_cut + 1, ..., ℓ}                 \\ blank remainder of path
21       blank(v_j)
22     V_col ← {v_1, ..., v_ℓ}                         \\ update collision set
23   \\ process removes, adds as before
24   else if (‘type’, ad) = (‘rem’, u-rem):
25     blank-leaf(T′, v_{u-rem})                       \\ blank u-rem's leaf
26     for v ∈ path(v_{u-rem})                         \\ blank u-rem's path
27       blank(v)
28     T′ ← truncate(T′)                               \\ truncate tree if necessary
29   else if (‘type’, ad) = (‘add’, (u-add, pk)):
30     add-leaf(T′, u-add, pk)                         \\ assign u-rem leaf
31     for v ∈ path(v_{u-add})                         \\ set v_{u-add} as unmerged
32       v.unm ←∪ {v_{u-add}}
33 return T′
```

Figure 5.5: Applying proposal messages in MLS-Cutoff. The function's formal description is in Figures B.18. We assume PMSG is ordered with update proposals followed by removals, followed by adds.

We provide some intuition on why the result holds. As MLS-Cutoff closely resembles MLS we are able to largely rely on the security proof of [AJM22]. The recent work of Cremers et al. [CGWZ25] shows that the signature scheme used must be SEUF-CMA secure rather than just EUF-CMA secure as originally stated in [AJM22], though their security proof already made use of SEUF-CMA security implicitly. We note that while in MLS-Cutoff update proposals lead to modifications to ratchet tree nodes beyond

users' personal leaves, it preserves the following property. For an update issuing user u denote their leaf by v_0, consider their update path $(v_1, \ldots, v_\ell)$, and assume that the keys of nodes $v_1, \ldots, v_k$ have been replaced after processing a commit to the update proposal and $v_{k+1}, \ldots, v_\ell$ have been blanked. Then all sub-trees rooted at the nodes $\mathsf{sib}(v_0), ..., \mathsf{sib}(v_k)$ must be in the same state as they were when the update proposal was issued. Indeed, if this were not the case for one of the nodes, then it would have caused a collision of update proposals and been blanked. This has the consequence that, on the one hand, keys being added to the tree cannot be encrypted under a public key that was concurrently being replaced, and, on the other hand, that the added nodes are compatible with the parent hash mechanism.

5.5.3 Upper Bounds on the Update Cost

We compute an upper bound on the communication complexity of MLS-Cutoff for sequences of operations as produced in experiment Exp-Prop-Com. We focus on the case $C = 1$, as the performance for $C \geq 1$ is even better.

Expected number of blanks in MLS-Cutoff**'s ratchet tree.** Let $N = 2^n$, $t \in \mathbb{N}$, $C = 1$, and let $P \in \mathbb{N}$ be constant. Consider an execution of Exp-Prop-Com(n, P, C, t) with respect to CGKA MLS-Cutoff. As in Section 5.4.2 consider the sequence $(\mathsf{T}_i)_{i=1}^t$ of ratchet trees generated after the i-th application of Exp-Prop-Com's round function. For a fixed internal node v we can define a random variable $(B_i(v))_{i=0}^t$ taking values in $\{0, 1\}$ where 0 corresponds to being a populated node and 1 to v being blank. We write B_i instead of $B_i(v)$ when there is no ambiguity. We note that all users have filtered paths of length n.

Now consider a node v below the cutoff bound i_{cut}, meaning that v's depth $d = \mathsf{depth}(v) \geq i_{\text{cut}}$, which implies that v is re-keyed if one of its descendant leaves issues an update proposal. We observe that, again, $(B_i)_i$ forms a Markov chain, since the node switching from blank or vice versa only depends on the question, whether it is affected by update proposals and/or commits.

More precisely v moves from blank to populated if it is affected by the commit or by exactly one update proposal, and from populated to blank if it is not affected by the commit and at least two update proposals.

$$p_{1\to 0} = \Pr[v \in \mathsf{ancs}(U_C) \vee |\{u \in U_P : v \in \mathsf{ancs}(v_u)\}| = 1]$$
$$p_{0\to 1} = \Pr[v \notin \mathsf{ancs}(U_C) \wedge |\{u \in U_P : v \in \mathsf{ancs}(v_u)\}| \geq 2].$$

Thus, we have

$$\begin{aligned}
p_{1\to 0} &= \Pr[v \in \mathsf{ancs}(U_C)] + \Pr[|\{u \in U_P : v \in \mathsf{ancs}(v_u)\}| = 1] \\
&\quad - \Pr[v \in \mathsf{ancs}(U_C) \wedge |\{u \in U_P : v \in \mathsf{ancs}(v_u)\}| = 1] \\
&= 1 - \frac{\binom{N(1-2^{-d})}{C}}{\binom{N}{C}} + P(1-2^{-d})^{P-1}2^{-d} - (1 - \frac{\binom{N(1-2^{-d})}{C}}{\binom{N}{C}})P(1-2^{-d})^{P-1}2^{-d} \\
&= 1 + (P(1-2^{-d})^{P-1}2^{-d} - 1)\frac{\binom{N(1-2^{-d})}{C}}{\binom{N}{C}},
\end{aligned}$$

and

$$\begin{aligned}
p_{0\to 1} &= \Pr[v \notin \mathsf{ancs}(U_C)]\Pr[|\{u \in U_P : v \in \mathsf{ancs}(v_u)\}| \geq 2] \\
&= \frac{\binom{N(1-2^{-d})}{C}}{\binom{N}{C}}(1 - (1-2^{-d})^P - P2^{-d}(1-2^{-d})^{P-1}).
\end{aligned}$$

Observe that for $C = 1$ and d sufficiently large this simplifies to

$$p_{1\to 0} \approxeq 1 - (1 - 2^{-d}) = 2^{-d}$$

and

$$p_{0\to 1} \approxeq 1 - (1-2^{-d})^P - P2^{-d}(1-2^{-d})^{P-1} \approxeq P(P-1)2^{-2d}$$

By Equation 5.1 we obtain that the probability of v being blank in the stationary distribution (when $d = \mathsf{depth}(v) \geq i_{\mathrm{cut}}$) is given by

$$\mathbb{E}[B(v) = 1] = \frac{p_{0\to 1}}{p_{1\to 0} + p_{0\to 1}} \approxeq \frac{P^2 2^{-2d}}{2^{-d} + P^2 2^{-2d}} \approxeq \frac{P^2 2^{-d}}{1 + P^2 2^{-d}} \in O\left(\frac{P^2}{2^d}\right). \tag{5.9}$$

As a consequence, the ratchet tree exhibits on expectation a constant amount of about P^2 blank nodes on level d.

Now consider a node v above the cutoff bound i_{cut}, i.e., $d = \mathsf{depth}(v) < i_{\mathrm{cut}}$. This implies that v changes from populated to blank if and only if (at least) one of its descendant leaves issues an update proposal and none of them issue a commit, whereas v changes from populated to blank if and only if at least one of them issues a commit. Using similar arguments to previous ones, we obtain that

$$p_{1\to 0} = \Pr[v \in \mathsf{ancs}(U_C)] = 1 - \frac{\binom{N(1-2^{-d})}{C}}{\binom{N}{C}}$$

and

$$p_{0\to 1} = \Pr[v \notin \mathsf{ancs}(U_C) \wedge v \in \mathsf{ancs}(U_P)] = \frac{\binom{N(1-2^{-d})}{C}}{\binom{N}{C}}\left(1 - \frac{\binom{N(1-2^{-d})}{P}}{\binom{N}{P}}\right).$$

If $C = 1$ this simplifies to

$$\begin{aligned} p_{1\to 0} &= 2^{-d}, \\ p_{0\to 1} &\approxeq (1-2^{-d})(1-(1-2^{-d})^P) \approxeq (1-2^{-d})P2^{-d}. \end{aligned}$$

By Equation 5.1 we obtain that the probability of v being blank in the stationary distribution (when $d = \mathsf{depth}(v) < i_{\mathrm{cut}}$) is given by

$$\mathbb{E}[B(v) = 1] = \frac{p_{0\to 1}}{p_{1\to 0} + p_{0\to 1}} \approxeq \frac{(1-2^{-d})P2^{-d}}{2^{-d} + (1-2^{-d})P2^{-d}} \in \left[0, \frac{P}{1+P}\right]. \tag{5.10}$$

Bounding $|\mathsf{Res}(v)|$. The size of the resolution of a node v can be bounded by the number of blank nodes in the subtree rooted at v plus one. If $C = 1$, we use this simple fact and Equations 5.9 and 5.10 in order to obtain that for nodes of $d = \mathsf{depth}(v) \geq i_{\mathrm{cut}}$

$$\begin{aligned} \mathbb{E}[|\mathsf{Res}(v)|] &\leq 1 + \sum_{d'=\log(N)}^{d+1} 2^{d'-d} \frac{P(P-1)2^{-d'}}{1+P(P-1)2^{-d'}} \\ &\leq 1 + P(P-1)2^{-d}(\log(N) - d) \end{aligned}$$

and for nodes of $d = \mathsf{depth}(v) < i_{\mathrm{cut}}$

$$\begin{aligned} \mathbb{E}[|\mathsf{Res}(v)|] &\leq 1 + \sum_{d'=\log(N)}^{i_{\mathrm{cut}}} 2^{d'-d} \frac{P(P-1)2^{-d'}}{1+P(P-1)2^{-d'}} + \sum_{d'=i_{\mathrm{cut}}-1}^{d+1} 2^{d'-d} \frac{P}{1+P} \\ &\leq 1 + P(P-1)2^{-d}(\log(N) - i_{\mathrm{cut}}) + 2^{-d}\frac{2^d - 2^{i_{\mathrm{cut}}}}{1-2} \\ &\leq P(P-1)2^{-d}(\log(N) - i_{\mathrm{cut}}) + 2^{i_{\mathrm{cut}}-d} \end{aligned}$$

Bounding $|\mathsf{pmsg}|$ and $|\mathsf{cmsg}|$. To compute a bound on the communication complexity of MLS-Cutoff for random sequences of operations, recall that by definition of $\mathsf{rekey\text{-}path}(\mathsf{T}', \mathsf{id}, i)$, if the $v_0 = v_{\mathsf{id}}$ and $(v_1, \ldots, v_\ell) = \mathsf{path}(v_0)$, the amount of ciphertexts is given by

$$|\mathsf{CTXT}| = \sum_{j=0}^{\ell-1} |\mathsf{Res}(\mathsf{sib}(v_i))|.$$

We make use of the previous bounds on the size of the resolution of a node and obtain that

$$\begin{aligned}\mathbb{E}[|\mathsf{CTXT}|] &= \sum_{d=1}^{\log(N)} \mathbb{E}[|\mathsf{Res}(v_d)|] \\ &\leq \sum_{d=1}^{i_{\mathrm{cut}}-1} (P(P-1)2^{-d}(\log(N) - i_{\mathrm{cut}}) + 2^{i_{\mathrm{cut}}-d}) \\ &+ \sum_{d=i_{\mathrm{cut}}}^{\log(N)} (1 + P(P-1)2^{-d}(\log(N) - d)) \\ &\leq P(P-1)(\log(N) - i_{\mathrm{cut}}) + 2^{i_{\mathrm{cut}}} + \log(N) - i_{\mathrm{cut}} + 1.\end{aligned}$$

If we set $i_{\mathrm{cut}} = \log(n) = \log(\log(N))$, we obtain that

$$\mathbb{E}[|\mathsf{CTXT}|] \leq \log(N) + (P(P-1)+1)\log(N/\log(N)) + 1.$$

We have shown the following.

Lemma 5.5.2. *Let $C = 1$, and let $P \in \mathbb{N}$ be constant. Let t be sufficiently large. For $N = 2^n$ set $i_{\mathrm{cut}} = \log(n)$. Then, on expectation, proposal messages* pmsg *and commit messages* cmsg *output by* MLS-Cutoff *for sequences of operations as defined by* Exp-Prop-Com *satisfy*

$$\mathbb{E}[|\mathsf{pmsg}|] \leq \mathbb{E}[|\mathsf{cmsg}|] \in O(\log(N)).$$

We point out that the amount of proposals per commit needs to be small compared to the group size N in order to guarantee a logarithmic communication complexity. Observe that Lemma 5.5.2 gives a bound that is asymptotic in N while P is assumed to be constant. If P becomes too large compared to the group size, the probability of blanks being introduced in the ratchet tree grows. In the extreme case of all users issuing update proposals, for example, the whole tree is blanked just as in MLS. We point out, that a linear communication complexity is, however, inherent for this type of 'massive' concurrency when only standard cryptographic primitives are taken into consideration (compare, e.g., the lower bounds of [BDR20]). As a second point, experiment Exp-Prop-Com does not remove users from the group, which would introduce blanks as well. However, observe that these blanks would be removed at a faster pace in MLS-Cutoff compared to MLS due to the additional path rekeying in update proposals.

Summing up, we see MLS-Cutoff as an attractive alternative to how update proposals are handled in MLS; it provides the same security properties, but has an improved

communication complexity for moderate numbers of concurrent update proposals. We consider it an interesting open question to evaluate its performance in more realistic models.

CHAPTER 6

Conclusion

In this thesis, we have made substantial progress towards constructing more efficient CGKA schemes as well as determining the limits of what can be achieved if we only consider standard cryptographic primitives as building blocks. However, very little work has been done when it comes to studying what can be achieved from more advanced primitives.

Recently, Alwen *et al.* [ABG+25] have shown that the lower bound given in Chapter 4 can be circumvented if one uses succint multiparty non-interactive key exchange (SMNIKE). Namely, they prove that using such a primitive, it is possible to build a CGKA for which the cost of replacing d many users is $O(d)$. This means that for $d \in o(n)$, their construction beats our lower bound of $\Omega(d \log(n/d))$. They also give a candidate construction of SMNIKE based on iO. It remains an open question whether it is possible to achieve a similar complexity from weaker assumptions.

Even in the case where one only considers standard primitives, the bound given in Chapter 4 does not give a complete picture of the complexity of batched replacements for CGKA since it allows for coordination between users. I.e., it remains open to explore what happens when users operate in a setting more similar to the one considered in Chapter 3 where they are not aware of what other users may be doing.

In Chapter 5 we showed that MLS does not just have worst-case linear communication cost, but rather that its communication cost may be very far from logarithmic on average. In the past it used to be assumed that this high communication cost was reserved to particularly ill-chosen sequences of operations and unlikely to be seen in more realistic scenarios. We also gave a protocol, MLS-Cutoff, to fix this issue. However, it is not entirely clear what a "realistic sequence" is. This leads to the following two open questions:

1. how do MLS and MLS-Cutoff perform when other distributions of sequences are considered?

2. Do other CGKA constructions also suffer from a similar problem?

Bringing secure group messaging based on CGKAs to the real world requires not just giving more efficient constructions, but also implementing them. This has been done in the case of MLS (see, for instance, the following list [Tho]), but we have not tried to do it with MLS-Cutoff. This would open the door to an experimental analysis of their complexity.

Finally, the use of CGKA beyond the context of group messaging remains largely unexplored with the exception of encrypted cloud storage [BBH25] and, therefore, finding more use cases for CGKA remains an open problem.

Bibliography

[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, Cham, November 2021.

[AAB+24] Michael Anastos, Benedikt Auerbach, Mirza Ahad Baig, Miguel Cueto Noval, Matthew Kwan, Guillermo Pascual-Perez, and Krzysztof Pietrzak. The cost of maintaining keys in dynamic groups with applications to multicast encryption and group messaging. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part I*, volume 15364 of *LNCS*, pages 413–443. Springer, Cham, December 2024.

[AAN+22] 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, Cham, May / June 2022.

[AAN+24] Joël Alwen, Benedikt Auerbach, Miguel Cueto Noval, Karen Klein, Guillermo Pascual-Perez, and Krzyzstof Pietrzak. DeCAF: Decentralizable CGKA with fast healing. In Clemente Galdi and Duong Hieu Phan, editors, *SCN 24, Part II*, volume 14974 of *LNCS*, pages 294–313. Springer, Cham, September 2024.

[ABG+25] Joël Alwen, Chris Brzuska, Jérôme Govinden, Patrick Harasser, and Stefano Tessaro. Succinct PPRFs via memory-tight reductions. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part V*, volume 16004 of *LNCS*, pages 628–662. Springer, Cham, August 2025.

[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, Cham, August 2020.

[ACDT21] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Modular design of secure group messaging protocols and the security of MLS. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1463–1483. ACM Press, November 2021.

[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, Cham, November 2020.

[ACPP23] Benedikt Auerbach, Miguel Cueto Noval, Guillermo Pascual-Perez, and Krzysztof Pietrzak. On the cost of post-compromise security in concurrent continuous group-key agreement. In Guy Rothblum and Hoeteck Wee, editors, *TCC 2023, Part III*, volume 14371 of *LNCS*, pages 271–300. Springer, Heidelberg, Nov 2023.

[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, Cham, August 2022.

[ANEP25] Benedikt Auerbach, Miguel Cueto Noval, Boran Erol, and Krzysztof Pietrzak. Continuous group-key agreement: Concurrent updates without pruning. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VIII*, volume 16007 of *LNCS*, pages 141–172. Springer, Cham, August 2025.

[ANPPP23] Benedikt Auerbach, Miguel Cueto Noval, Guillermo Pascual-Perez, and Krzysztof Pietrzak. On the cost of post-compromise security in concurrent continuous group-key agreement. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part III*, volume 14371 of *LNCS*, pages 271–300. Springer, Cham, November / December 2023.

[BBH25] Matilda Backendal, David Balbás, and Miro Haller. Group key progression: Strong security for shared persistent data. Cryptology ePrint Archive, Report 2025/1028, 2025.

[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.

[BCG23] David Balbás, Daniel Collins, and Phillip Gajland. WhatsUpp with sender keys? Analysis, improvements and security proofs. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part V*, volume 14442 of *LNCS*, pages 307–341. Springer, Singapore, December 2023.

[BCV23] David Balbás, Daniel Collins, and Serge Vaudenay. Cryptographic administration for secure group messaging. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 1253–1270. USENIX Association, August 2023.

[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, Cham, 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, Cham, November 2020.

[BNG+25] Nicholas Brandt, Miguel Cueto Noval, Christoph U. Günther, Akin Ünal, and Stella Wohnig. Constrained verifiable random functions without obfuscation and friends. In Benny Applebaum and Huijia (Rachel) Lin, editors, *TCC 2025, Part IV*, volume 16271 of *LNCS*, pages 478–511. Springer, Cham, December 2025.

[BNHR22] Andrej Bogdanov, Miguel Cueto Noval, Charlotte Hoffmann, and Alon Rosen. Public-key encryption from homogeneous CLWE. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part II*, volume 13748 of *LNCS*, pages 565–592. Springer, Cham, November 2022.

[Bol65] Béla Bollobás. On generalized graphs. *Acta Mathematica Academiae Scientiarum Hungarica*, 16(3):447–452, 1965.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[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.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 61–85. Springer, Berlin, Heidelberg, February 2007.

[CEST24] Kelong Cong, Karim Eldefrawy, Nigel P. Smart, and Ben Terner. The key lattice framework for concurrent group messaging. In Christina Pöpper and Lejla Batina, editors, *ACNS 2024, Part II*, volume 14584 of *LNCS*, pages 133–162. Springer, Cham, March 2024.

[CGI+99a] Ran Canetti, Juan Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast security: A taxonomy and some efficient constructions. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, volume 2, pages 708–716. IEEE, 1999.

[CGI+99b] 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.

[CGWZ25] Cas Cremers, Esra Günsay, Vera Wesselkamp, and Mang Zhao. ETK: External-operations TreeKEM and the security of MLS in RFC 9420. Cryptology ePrint Archive, Report 2025/229, 2025.

[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.

[CJL23] Cas Cremers, Charlie Jacomme, and Philip Lukert. Subterm-based proof techniques for improving the automation and scope of security protocol analysis. In *CSF 2023 Computer Security Foundations Symposium*, pages 200–213. IEEE Computer Society Press, July 2023.

[CMN99] Ran Canetti, Tal Malkin, and Kobbi Nissim. Efficient communication-storage tradeoffs for multicast encryption. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 459–474. Springer, Berlin, Heidelberg, May 1999.

[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.

[HKPP25] Keitaro Hashimoto, Shuichi Katsumata, and Guillermo Pascual-Perez. Exploring how to authenticate application messages in MLS: More efficient, post-quantum, and anonymous blocklistable. In Lujo Bauer and Giancarlo Pellegrino, editors, *USENIX Security 2025*, pages 6699–6716. USENIX Association, August 2025.

[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, Cham, 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, Cham, December 2019.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall, CRC Press, third edition, 2014.

[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.

[LYGL01] Xiaozhou Steve Li, Yang Richard Yang, Mohamed G. Gouda, and Simon S. Lam. Batch rekeying for secure group communications. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, page 525–534, New York, NY, USA, 2001. Association for Computing Machinery.

[LZPR24] Chen-Da Liu-Zhang, Christopher Portmann, and Guilherme Rito. Stateful communication with malicious parties. Cryptology ePrint Archive, Report 2024/1593, 2024.

[LZPR25] Chen-Da Liu-Zhang, Christopher Portmann, and Guilherme Rito. Simpler and stronger models for deniable authentication. Cryptology ePrint Archive, Report 2025/204, 2025.

[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, Berlin, Heidelberg, May 2004.

[NMSÜ25] Miguel Cueto Noval, Simon-Philipp Merz, Patrick Stählin, and Akin Ünal. On the soundness of algebraic attacks against code-based assumptions. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part VI*, volume 15606 of *LNCS*, pages 385–415. Springer, Cham, May 2025.

[NNL01] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 41–62. Springer, Berlin, Heidelberg, August 2001.

[PMS16] Trevor Perrin (editor), Moxie Marlinspike, and Rolfe Schmidt (revision 3+). The double ratchet algorithm. `https://signal.org/docs/specifications/doubleratchet/`, November 2016. [Online; accessed 05-May-2026; current version is from November 2025].

[SB25] Matthew Shanahan and Kalvin Bahia. The state of mobile internet connectivity. `https://www.`

gsma.com/somic/wp-content/uploads/2025/09/The-State-of-Mobile-Internet-Connectivity-2025-Overview-Report.pdf, September 2025. [Online; accessed 25-March-2026].

[SM03] Alan T Sherman and David A McGrew. Key establishment in large dynamic groups using one-way function trees. *IEEE transactions on Software Engineering*, 29(5):444–458, 2003.

[SSV01] J. Snoeyink, S. Suri, and G. Varghese. A lower bound for multicast key distribution. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 1, pages 422–431 vol.1, 2001.

[Tho] Vilte Thomas. List of mls implementations. https://github.com/mlswg/mls-implementations/blob/main/implementation_list.md. [Online; accessed 10-May-2026].

[Wei19] Matthew A. Weidner. Group Messaging for Secure Asynchronous Collaboration. Master's thesis, University of Cambridge, June 2019.

[WGL00] Chung Kei Wong, Mohamed Gouda, and Simon S Lam. Secure group communications using key graphs. *IEEE/ACM transactions on networking*, 8(1):16–30, 2000.

[Wha] Whatsapp about. https://www.whatsapp.com/about. [Online; accessed 25-March-2026].

[WHA99] Debby Wallner, Eric Harder, and Ryan Agee. Key management for multicast: Issues and architectures. Request for Comments 2627, Internet Engineering Task Force, June 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.

[WPB25] Théophile Wallez, Jonathan Protzenko, and Karthikeyan Bhargavan. TreeKEM: A modular machine-checked symbolic security analysis of group key agreement in messaging layer security. In Marina Blanton, William Enck, and Cristina Nita-Rotaru, editors, *2025 IEEE Symposium on Security and Privacy*, pages 4375–4390. IEEE Computer Society Press, May 2025.

[WPBB23] Théophile Wallez, Jonathan Protzenko, Benjamin Beurdouche, and Karthikeyan Bhargavan. TreeSync: Authenticated group management for messaging layer security. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 1217–1233. USENIX Association, August 2023.

[YLZL02] Yang Yang, X. Li, X. Zhang, and Simon Lam. Reliable group rekeying: A performance analysis. *ACM SIGCOMM Computer Communication Review*, 01 2002.

APPENDIX A

Supplementary material for Chapter 4

A.1 Lower Bounds for Continuous Group Key Agreement

In this section we show that the lower bound of Section 4.2.2 in the combinatorial model carries over to the setting of batched replacements of users in continuous group-key agreement schemes in the symbolic model. In Section A.1.1 we define the symbolic model and provide syntax for continuous group-key agreement, in Section A.1.2 we prove our lower bound.

A.1.1 Continuous Group-key Agreement in the symbolic model

Building blocks and entailment relation. We now give syntax for continuous group-key agreement (CGKA) in the symbolic model, focusing on batched replacement of users. As in Section 4.3.1, we first discuss the covered building blocks, i.e., public-key encryption (PKE), pseudorandom generators (PRG), pseudorandom functions (PRF) and dual pseudorandom functions (dPRF). Further, our bound also extends to the setting allowing for key-updatable public-key encryption (kuPKE) as an additional building block. As it makes for a cleaner presentation, instead of adding kuPKE to the considered building blocks in this section we prove this in Section A.1.3. We consider symbolic variables of the following three types; (pseudo)random coins denoted by `r`, public keys `pk`, and messages `m`. Random coins of type `r` also serve as secret keys for PKE schemes, and we will typically denote them by `sk` if they are used in that context. Similarly, ciphertexts are considered to be of message type `m` and we will denote them

by $\mathtt{c}$ if used in this context. Symbolic variables are depicted using typewriter font to distinguish them from their non-symbolic counterparts. Upper case typewriter letters are used for sets of symbolic variables.

The only building block whose syntax was not discussed in Section 4.3.1 is public-key encryption. A public-key encryption scheme $\mathsf{PKE} = (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ specifies the following. Key generation algorithm $\mathsf{PKE.Gen}(\mathtt{sk})$ receives as input a secret key $\mathtt{sk}$ (of type $\mathtt{r}$) and returns the corresponding public key $\mathtt{pk}$. Encryption algorithm $\mathsf{PKE.Enc}(\mathtt{pk}, \mathtt{m})$, on input public key $\mathtt{pk}$ and message $\mathtt{m}$, returns a ciphertext $\mathtt{c}$ (of message type). Decryption algorithm $\mathsf{PKE.Dec}(\mathtt{sk}, \mathtt{c})$, on input secret key $\mathtt{sk}$ and ciphertext $\mathtt{c}$, returns a message $\mathtt{m}$. We require perfect correctness, i.e., $\mathsf{PKE.Dec}(\mathtt{sk}, \mathsf{PKE.Enc}(\mathtt{pk}, \mathtt{m})) = \mathtt{m}$ for all $\mathtt{m}$ and all $\mathtt{sk}, \mathtt{pk}$ with $\mathtt{pk} = \mathsf{PKE.Gen}(\mathtt{sk})$.

The grammar rules, and the rules for the entailment relation $\vdash$ for PKE, PRGs, PRFs, dPRFs and the secret sharing scheme are very similar to the ones used in Section 4.3.1.

variable type		grammar rule
$\mathtt{r}$	$\leftarrow$	terminal type, $\mathsf{PRG}(\mathtt{r}), \mathsf{PRF}(\mathtt{r}), \mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2)$
$\mathtt{pk}$	$\leftarrow$	$\mathsf{PKE.Gen}(\mathtt{r})$
$\mathtt{m}$	$\leftarrow$	$\mathtt{r}$, $\mathtt{pk}$, $\mathsf{PKE.Enc}(\mathtt{r}, \mathtt{m})$
entailment relation		
$\mathtt{m} \in \mathtt{M}$	$\Rightarrow$	$\mathtt{M} \vdash \mathtt{m}$
$\mathtt{M} \vdash \mathtt{r}$	$\Rightarrow$	$\mathtt{M} \vdash \mathsf{PRG}(\mathtt{r}) = (\mathtt{r}_1, \mathtt{r}_2)$
$\mathtt{M} \vdash \mathtt{r}$	$\Rightarrow$	$\forall ad\colon \mathtt{M} \vdash \mathsf{PRF}(\mathtt{r}, ad)$
$\mathtt{M} \vdash \mathtt{r}_1, \mathtt{r}_2$	$\Rightarrow$	$\mathtt{M} \vdash \mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2)$
$\mathtt{M} \vdash \mathtt{pk}, \mathtt{m}$	$\Rightarrow$	$\mathtt{M} \vdash \mathsf{PKE.Enc}(\mathtt{pk}, \mathtt{m})$
$\mathtt{M} \vdash (\mathtt{r}, \mathtt{c})\colon \mathtt{c} = \mathsf{PKE.Enc}(\mathtt{pk}, \mathtt{m})$ and $\mathtt{pk} = \mathsf{PKE.Gen}(\mathtt{r})$	$\Rightarrow$	$\mathtt{M} \vdash \mathtt{m}$
$\exists I \in \Gamma\colon \mathtt{M} \vdash \{\mathsf{S}_i(\mathtt{m})\}_{i \in I}$	$\Rightarrow$	$\mathtt{M} \vdash \mathtt{m}$

Thus, (pseudo)random coins can be sampled freshly or derived with a PRG, a PRF or a dPRF. Public keys are constructed from random coins using PKE.Gen, and messages can be of arbitrary type and, in particular, include encryptions of other messages under public keys. To a set $\mathtt{M}$ of symbolic variables we associate $\mathsf{Der}(\mathtt{M})$, the set of variables derivable from it, i.e., for $\mathtt{m}$ we have

$$\mathtt{m} \in \mathsf{Der}(\mathtt{M}) \text{ exactly if } \mathtt{M} \vdash \mathtt{m}.$$

CGKA in the symbolic model. A *continuous group-key agreement scheme* allows a group of users to maintain a shared group key evolving over rounds. Opposed to the setting of multicast encryption, however, CGKA does not rely on a trusted central authority that has access to all secrets. Instead, every user keeps track of a personal state and can send protocol messages that are distributed to the other users via an untrusted server. The security goals being (a) that the scheme provides

post-compromise security (PCS) by users issuing update operations, which roughly corresponds to the group being able to achieve security even in the presence of past exposures of users' states[1]; and (b) enabling group members to both add new users and remove group members from the group, while maintaining the expected confidentiality guarantees with respect to these added or removed users. The goal of this section is to prove lower bounds on the communication complexity of operations of the latter kind, essentially showing that it is not possible to improve over the "fair-weather" complexity of the add/remove mechanism employed by the MLS standard. Again, for an easier presentation of our bound, we keep the group size constant over the experiment by working with a simplified syntax that, similarly to Section 4.3.2, instead of allowing for arbitrary updates, adds, and removes, uses a replacement algorithm that can be called to replace a batch of group members with a set of non-members of the same size.

Formally, a CGKA scheme specifies 5 algorithms CGKA.Setup, CGKA.Init, CGKA.Repl, CGKA.Proc, and CGKA.Key.[2] The scheme proceeds in rounds t, each of which establishes a group $G_t \subseteq [n_{\max}]$ and corresponding group key $\mathtt{k}^t$ of type $\mathtt{r}$. Here $[n_{\max}]$, for $n_{\max} \in \mathbb{N}$, is the universe of users. After initialization of the group G_0, in every round, some user u replaces a set of group members with a set of new users of the same size. Afterwards, all users process these operations, resulting in a new group and group key. More precisely, for $t \geq 1$, let A_t and D_t be the sets describing the users added to and removed from the group round t, respectively. Then, the group G_t established at the end of round t is computed from the one of the previous round as

$$G_t = (G_{t-1} \cup A_t) \setminus D_t. \tag{A.1}$$

We now describe the syntax of the algorithms more formally.

- CGKA.Setup$(n_{\max}; \mathtt{R})$, on input the universe of users and random coins $\mathtt{R}$, outputs public information $(\mathtt{pub}, \mathsf{pub})$ (e.g., containing every user's initial public key), as well as an initial state $(\mathtt{ST}_u, \mathsf{st}_u)$ for every user $u \in [n_{\max}]$. Note that the public information, as well as the users' states, consist of a symbolic and a non-symbolic part.

- CGKA.Init$((\mathtt{ST}_u, \mathsf{st}_u), (\mathtt{pub}, \mathsf{pub}), G_0; \mathtt{R})$ receives as input the user's (initial) state, the public information, the description of a group $G_0 \subseteq [n_{\max}]$, and random

[1] One could also consider other notions of security like post-compromise forward secrecy (PCFS) that require security even in the presence of both past and future exposures of users' states. By only asking for PCS we obtain a stronger result since we are proving a lower bound.

[2] Our syntax is an adaptation of the one from [ACPP23] to account for dynamic operations. Compared to that of [BDR20], ours separates the setup from group creation (needed to extend the syntax to the dynamic setting), and includes an explicit algorithm outputting the group key, as opposed to it be part of the output of CGKA.Proc.

coins $\mathtt{R}$. Its output $((\mathtt{ST}'_u, \mathsf{st}'_u), (\mathtt{M}_0, \mathsf{M}_0))$ consists of the updated state and a control message $(\mathtt{M}_0, \mathsf{M}_0)$.

- CGKA.Repl$((\mathtt{ST}_u, \mathsf{st}_u), (\mathtt{pub}, \mathsf{pub}), A_t, D_t; \mathtt{R})$ in round t takes as input a user u's state, where $u \in G_{t-1}$, the public information, a set $A_t \subseteq G_{t-1}$ of users to be added to the group, a set $D_t \subseteq [n_{\max}] \setminus G_{t-1}$ of the same size that collects the users to be added to the group, and random coins $\mathtt{R}$. It returns u's updated state $(\mathtt{ST}'_u, \mathsf{st}'_u)$ and a control message $(\mathtt{M}_t, \mathsf{M}_t)$.[3]

- Deterministic algorithm CGKA.Proc$((\mathtt{ST}_u, \mathsf{st}_u), (\mathtt{pub}, \mathsf{pub}), (\mathtt{M}, \mathsf{M}))$ gets as input a user u's state, the public information, and a protocol message. Its output is an updated state $(\mathtt{ST}'_u, \mathsf{st}'_u)$ and a group description $G \subseteq [n_{\max}]$.

- Deterministic algorithm CGKA.Key$(\mathtt{ST}_u, \mathsf{st}_u)$, on input a user's state, returns u's view of current group key $\mathtt{k}$.

Analogously to the setting of multicast encryption, for brevity in the following we will typically drop the non-symbolic parts of the public information $(\mathtt{pub}, \mathsf{pub})$, control messages $(\mathtt{M}, \mathsf{M})$, and users' states $(\mathtt{ST}_u, \mathsf{st}_u)$; and simply write $\mathtt{pub}$, $\mathtt{M}$, and $\mathtt{ST}_u$. We also impose the requirement that symbolic outputs of the algorithms are derivable from their symbolic inputs and that only a finite number of symbolic derivations is done.

Correctness and security. We define correctness and security of CGKA schemes in the symbolic model according to the game $\mathrm{SEC}^{\mathsf{CGKA}}$ in Figure A.1. The game is defined with respect to $(n_{\max}, t_{\max}, G_0, u_0)$ and $(u_t, A_t, D_t)_{t=1}^{t_{\max}}$. Similarly to the corresponding game for ME, $[n_{\max}]$ is the universe of users, $t_{\max}$ the number of rounds the game is run for, and G_0 the initial group. Following the initial set-up, user u_0 generates G_0 using algorithm CGKA.Init. Afterwards the game proceeds in rounds t determined by (u_t, A_t, D_t), where user u_t uses CGKA.Repl$(\mathtt{ST}_{u_t}^{t-1}, A_t, D_t)$ to replace the set $A_t \subseteq G_{t-1}$ by the set $D_t \subseteq [n_{\max}] \setminus G_{t-1}$. Here we require that $u_t \in G_{t-1} \setminus D_t$, i.e., users executing the replacement must be group members and users are not able to remove themselves from the group. In every round the game verifies that (a) all group members have access to the current group key, which essentially means

$$\exists \mathtt{k}^t : \mathtt{k}^t = \mathsf{CGKA.Key}(\mathtt{ST}_u^t) \text{ for all } u \in G_t;$$

[3] As it makes for a cleaner presentation, in our syntax a single user generates the control message replacing the users in A_t. However, our lower bound also applies in straightforward manner to CGKA in which users in any given round are able to concurrently issue the replacement of sets of users, and the resulting control messages are in turn processed by all group members. In this setting our lower bound holds with respect to the overall number of users removed per round.

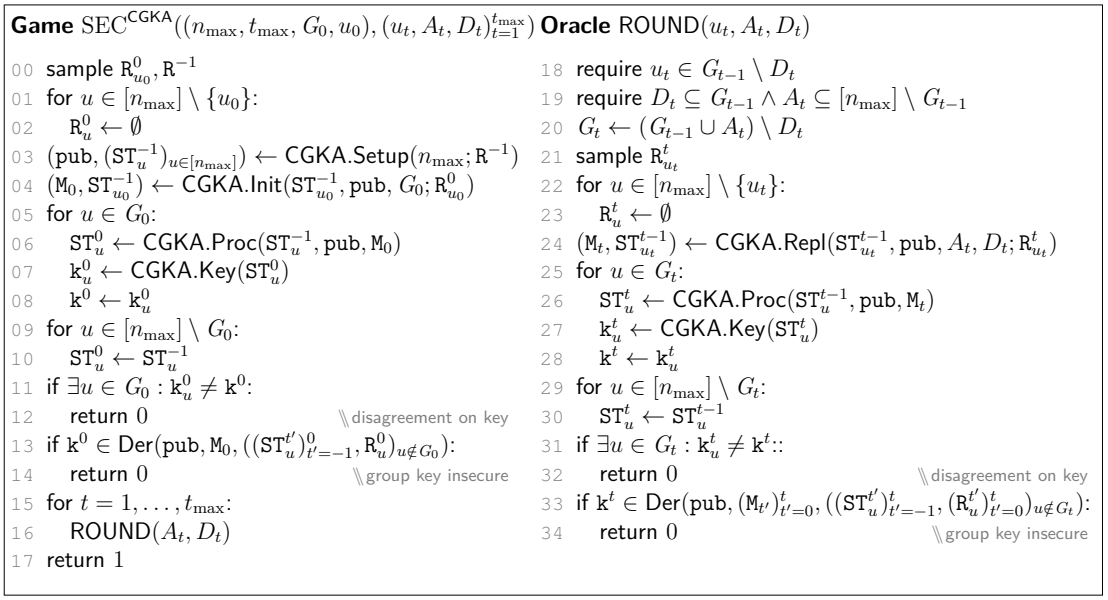

Figure A.1: Symbolic security and correctness game for continuous group-key agreement scheme CGKA.

and that (b) non-members of the group, even when colluding and having stored all their previous states $\mathtt{ST}_u^{t'}$ as well as all random coins $\mathtt{R}_u^{t'}$ sampled while issuing replacement operations, are not able to derive the current group key, i.e.,

$$\mathtt{k}^t \notin \mathsf{Der}\left(\mathtt{pub}, (\mathtt{M}_{t'})_{t'=0}^t, ((\mathtt{ST}_u^{t'})_{t'=-1}^t, (\mathtt{R}_u^{t'})_{t'=0}^t)_{u\in[n_{\max}]\setminus G_t}\right).$$

Similarly to the case of multicast encryption, we assume that the symbolic part of the initial state, namely, $\mathtt{ST}_u^{-1}$ only contains symbols of type random coins. We also require that for all $\mathtt{r} \in \mathtt{ST}_u^{-1} \bigcup_{0\leq t'\leq t_{\max}} \mathtt{R}_u^{t'}$:

$$\mathtt{r} \notin \mathsf{Der}\left(((\mathtt{ST}_u^{-1} \cup \cup_{0\leq t'\leq t_{\max}} \mathtt{R}_u^{t'}) \setminus \{\mathtt{r}\}) \cup \cup_{v\in[n_{\max}]\setminus\{u\}} (\mathtt{ST}_v^{-1} \cup \cup_{0\leq t'\leq t_{\max}} \mathtt{R}_v^{t'})\right). \tag{A.2}$$

Useful secrets and associated set system. Analogously to Section 4.3.2 we first define useful secrets generated during the execution of the security game. Thus, consider an execution of $\mathrm{SEC}^{\mathsf{CGKA}}$ with respect to CGKA scheme CGKA, $(n_{\max}, t_{\max}, G_0, u_0)$, and $(u_t, A_t, D_t)_{t=1}^{t_{\max}}$. For $t \in [t_{\max}]_0$ let $\mathtt{r}$ be a symbolic random coin (recall that $\mathtt{r}$ can also serve as secret key of a PKE scheme) generated up to and including round t. We say that $\mathtt{r}$ is an *useful secret in round* t, if

$$\mathtt{r} \notin \mathsf{Der}\left(\mathtt{pub}, (\mathtt{M}_{t'})_{t'=0}^t, ((\mathtt{ST}_u^{t'})_{t'=-1}^t, (\mathtt{R}_u^{t'})_{t'=0}^t)_{u\in[n_{\max}]\setminus G_t}\right),$$

i.e., if it cannot be derived even if all non-group members at time t are colluding. To useful secrets we associate a set of users as follows.

Definition A.1.1. *Consider an execution of security game* $\mathrm{SEC}^{\mathsf{CGKA}}$ *with respect to CGKA scheme* CGKA, *setup* $(n_{\max}, t_{\max}, G_0, u_0)$, *and sequence of operations* $(u_t, A_t, D_t)_{t=1}^{t_{\max}}$. *Let* $t \in [t_{\max}]_0$ *and* $\mathtt{r}$ *be a random coin. We associate to* $(t, \mathtt{r})$ *the set* $S(t, \mathtt{r})$ *of users that at any point in time up to round* t, *had access to* $\mathtt{r}$, *i.e.,*

$$S(t, \mathtt{r}) := \left\{ u \in [n_{\max}] \mid \mathtt{r} \in \mathsf{Der}(\mathtt{pub}, \mathtt{ST}_u^{-1}, (\mathtt{M}_{t'})_{0 \le t' \le t:\, u \in G_{t'}}, (\mathtt{R}_u^{t'})_{t'=0}^{t}) \right\}.$$

Further, we define the associated set system $\mathcal{S}_t$ *in round* t *as*

$$\mathcal{S}_t := \{ S \subseteq [n_{\max}] \mid \textit{there exists useful coin } \mathtt{r} : S = S(t, \mathtt{r}) \}.$$

In the following we will show that $(\mathcal{S}_t)_t$ (or rather a subset that we will define later) satisfies the properties required in the combinatorial model of Section 4.2.1. Before, though, we give a brief comparison to the associated set system used in Section 3.4.1.

Remark A.1.2. *In Section 3.4.1 we also consider a symbolic model for CGKA and associate a set system to useful secrets, with the aim of proving lower bounds on the communication complexity of the group recovering from corruption over a certain number of rounds. In that work, a set* $S(t, \mathtt{r})$ *associated to some useful secret* $\mathtt{r}$ *intuitively correspond to the users who have access to* $\mathtt{r}$ *in round* t. *In this work, in turn, we work with* $S(t, \mathtt{r})$ *corresponding to the set of users who in any round up to* t *had access to* $\mathtt{r}$, *be it from their internal state or because it was one of the random coins they sampled while generating a replacement operation.*

Our definitional choice has two effects. On the one hand, we are no longer able to connect the evolution of the set system from $\mathcal{S}_{t-1}$ *to* $\mathcal{S}_t$ *to the number of ciphertexts that were sent in round* t, *but have to work with the weaker concept of ciphertexts sent from the beginning of the experiment until round* t *(which, however, is still strong enough to obtain meaningful bounds). On the other hand, while the approach of Section 3.4.1 was only able to guarantee that a set* S *added to* $\mathcal{S}_t$ *in round* t *must have been covered by a union of sets in* $\mathcal{S}_{t-1}$ *and that the cost of this operation in terms of ciphertexts essentially matched the size of the union, in this work we can guarantee the existence of sets in* $\mathcal{S}_{t-1}$ *whose union exactly matches* S. *This allows us to connect the number of ciphertexts sent in order to add a set to the set system, to a cover with respect to the set system of the previous round. The latter is necessary to be able to apply the Bollobás Set Pairs Inequality.*

The connection between the derivation rules of the symbolic model and the sets introduced in Definition A.1.1 is similar to the one we proved for the case of multicast encryption in Lemmas 4.3.2 and 4.3.3. In the case of CGKA we state the results and defer the proofs to Appendix A.2.

Lemma A.1.3. *Let* $\mathtt{r}$ *be of type random coin and useful at time* $t \in [t_{\max}]_0$*, and* u *a user such that* $u \in S(t, \mathtt{r})$*. Then*

1. *there exist* $\mathtt{r}'$ *and* $i \in \{1, 2\}$ *such that* $\mathsf{PRG}(\mathtt{r}')_i = \mathtt{r}$*,* $\mathtt{r}'$ *is useful at time* t *and* $u \in S(t, \mathtt{r}')$*, or*
2. *there exist* $\mathtt{r}'$ *and associated data* ad *such that* $\mathsf{PRF}(\mathtt{r}', ad) = \mathtt{r}$*,* $\mathtt{r}'$ *is useful at time* t *and* $u \in S(t, \mathtt{r}')$*, or*
3. *there exist* $\mathtt{r}_1$ *and* $\mathtt{r}_2$ *such that* $\mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2) = \mathtt{r}$*, at least one of* $\mathtt{r}_1$ *and* $\mathtt{r}_2$ *is useful at time* t*, and* $u \in S(t, \mathtt{r}_1) \cap S(t, \mathtt{r}_2)$*, or*
4. $\mathtt{r} \in \mathtt{ST}_u^{-1} \cup \bigcup_{t'=0}^{t} \mathtt{R}_u^{t'}$*, or*
5. *there exists* $\mathtt{c} = e_0(\cdot) \circ \ldots \circ e_g(\cdot) \circ \ldots \circ e_h(\mathtt{r})$ *where* $e_i = \mathsf{PKE.Enc}(\mathtt{r}_i, \cdot)$ *or* $e_i = \mathsf{S}_{i_j}(\cdot)$ *such that*
   a) $\mathtt{c} \in \mathtt{pub} \cup \bigcup_{\tilde{t} \le t:\, u \in G_{\tilde{t}}} \mathtt{M}_{\tilde{t}}$*,*
   b) *for all* i *such that* $e_i = \mathsf{PKE.Enc}(\mathtt{pk}_i, \cdot)$ *there exist random coins* $\mathtt{r}_i$ *such that* $\mathtt{pk}_i = \mathsf{PKE.Gen}(\mathtt{r}_i)$*,*
   c) *if* $e_i = \mathsf{PKE.Enc}(\mathtt{pk}_i, \cdot)$ *and* $i \ge g + 1$*, then* $\mathtt{r}_i$ *is not useful at time* t*,*
   d) *there exists* $i \in \{0, \ldots, h\}$ *such that* $e_i = \mathsf{PKE.Enc}(\mathtt{pk}_i, \cdot)$*,*
   e) $e_g = \mathsf{PKE.Enc}(\mathtt{pk}_g, \cdot)$ *and* $\mathtt{r}_g$ *is useful at time* t*, and*
   f) *for all* i *such that* $e_i = \mathsf{PKE.Enc}(\mathtt{pk}_i, \cdot)$ *it holds that* $u \in S(t, \mathtt{r}_i)$*.*

Lemma A.1.4. *Let* $\mathtt{r}$ *be of type random coin and useful at time* $t \in [t_{\max}]_0$*, and* u *a user such that* $u \in S(t, \mathtt{r})$*. Then there exists a sequence* $\{\mathtt{r}_{1,u,t}, \ldots, \mathtt{r}_{\ell_u,u,t}\}$ *such that*

6. *for all* i *the secret* $\mathtt{r}_{i,u,t}$ *is useful at time* t *and* $u \in S(t, \mathtt{r}_{i,u,t})$*,*
7. $\mathtt{r}_{\ell_u,u,t} = \mathtt{r}$*,*
8. $\mathtt{r}_{1,u,t} \in \mathtt{ST}_u^{-1} \cup \bigcup_{t'=0}^{t} \mathtt{R}_u^{t'}$*, and*
9. *for all* $i \in \{1, \ldots, \ell_u - 1\}$ *one of the following is true*
   a) $\mathsf{PRG}(\mathtt{r}_{i,u,t}) = (\mathtt{r}_1, \mathtt{r}_2)$ *for some* $\mathtt{r}_1$*,* $\mathtt{r}_2$ *such that either* $\mathtt{r}_{i+1,u,t} = \mathtt{r}_1$ *or* $\mathtt{r}_{i+1,u,t} = \mathtt{r}_2$*, or*
   b) *there exists* ad *such that* $\mathsf{PRF}(\mathtt{r}_{i,u,t}, ad) = \mathtt{r}_{i+1,u,t}$*, or*
   c) *there exists* $\mathtt{r}'_{i,u,t}$ *such that* $u \in S(t, \mathtt{r}'_{i,u,t})$ *and* $\mathsf{dPRF}(\mathtt{r}_{i,u,t}, \mathtt{r}'_{i,u,t}) = \mathtt{r}_{i+1,u,t}$*, or*

d) there exists a ciphertext $\mathtt{c}_{i,u,t} \in \mathtt{pub} \cup \bigcup_{\tilde{t} \le t:\, u \in G_{\tilde{t}}} \mathtt{M}_{\tilde{t}}$ *such that*

$$\mathtt{c}_{i,u,t} = e_0(\cdot) \circ \ldots \circ e_g(\cdot) \circ \ldots \circ e_h(\mathtt{r}_{i+1,u,t})$$

where all properties of Case 5 are satisfied and $\mathtt{r}_{i,u,t} = \mathtt{r}_g$ *satisfies that* $e_g = \mathsf{PKE.Enc}(\mathtt{pk}_g, \cdot)$ *and* $\mathtt{pk}_g = \mathsf{PKE.Gen}(\mathtt{r}_g)$.

Observe that ℓ_u *depends on* u, t *and* $\mathtt{r}$*, so in some cases we make this explicit and write* $\ell_{u,t,\mathtt{r}}$ *or just* $\ell_{u,t}$ *if* $\mathtt{r}$ *is clear from context.*

Following the same approach as in Section 4.3.1, we define a graph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ for $t \in [t_{\max} - 1]_0$ using the sequences from Lemma A.1.4 for the group key $\mathtt{k}^t$. Namely, the set of nodes $\mathcal{V}_t$ corresponds to the elements of the sequences $\{\mathtt{k}^t_{1,u,t}, \ldots, \mathtt{k}^t_{\ell_u,u,t}\}$ associated to the group key $\mathtt{k}^t$ and each user $u \in S(t, \mathtt{k}^t)$, and the set of edges $\mathcal{E}_t$ consists of all pairs of the form $(\mathtt{k}^t_{i,u,t}, \mathtt{k}^t_{i+1,u,t})$.

Property 9c can lead to two edges in the graph $\mathcal{G}_t$ if both inputs $\mathtt{r}_{i,u,t}$ and $\mathtt{r}'_{i,u,t}$ are useful at time t. If this happens, then we make the same choice for all users' sequences in order to make sure that one dPRF pre-image does not result in two edges in $\mathcal{E}_t$. This is possible since $u \in S(t, \mathtt{r}_{i,u,t}) \cap S(t, \mathtt{r}'_{i,u,t})$.

Before studying the properties of this graph, we introduce a modification of any CGKA protocol such that the graph associated to the modified protocol has properties analogous to the ones of Lemma 4.3.4. This approach was already used in [AAB+21]. If CGKA is a CGKA scheme we define a new CGKA scheme CGKA′ by substituting any random coin in $\mathtt{r} \in \bigcup_{u \in [n_{\max}]} (\mathtt{ST}_u^{-1} \cup \bigcup_{0 \le t' \le t_{\max}} \mathtt{R}_u^{t'})$ by $\mathtt{r}' = \mathsf{PRG}_1(\tilde{\mathtt{r}})$ for some new random coin $\tilde{\mathtt{r}}$. The modified protocol CGKA′ preserves the communication cost of CGKA. If $\mathtt{r}$ is useful at time t in an execution of CGKA, then $\mathtt{r}'$ is useful at time t with respect to the same execution of CGKA′. Therefore CGKA′ preserves correctness and security. Moreover, the additional random coins $\tilde{\mathtt{r}}$ used in CGKA′ satisfy the property that there exist unique $u \in [n_{\max}]$ and $-1 \le t \le t_{\max}$ such that for all $\tilde{t} \ge t$ it holds $S(\tilde{t}, \tilde{\mathtt{r}}) = \{u\}$. This shows that the value of the function $\mathrm{Cost}(t)$ defined in Section 4.2.1 is also preserved by CGKA′.

We refer to the edges $(\mathtt{k}^t_{i,u,t}, \mathtt{k}^t_{i+1,u,t})$ that satisfy Properties 9a, 9b or 9c as trivial edges, whereas those that satisfy Property 9d are called communication edges. Now we discuss the properties of the graph $\mathcal{G}_t$ captured in the result below. For a proof we refer the reader to Appendix A.2.

Lemma A.1.5. *Let* CGKA *be a correct and secure CGKA scheme. Consider an execution of game* $\mathrm{SEC}^{\mathsf{CGKA}}$ *on input* $(n_{\max}, t_{\max}, G_0, u_0)$ *and* $(u_t, A_t, D_t)_{t=1}^{t_{\max}}$ *such that* $D_t \subseteq G_{t-1}$*,* $u_t \in G_{t-1} \setminus D_t$*, and* $A_t \subseteq [n_{\max}] \setminus (G_{t-1} \cup \bigcup_{t'=1}^{t-1} D_t)$ *for all* $t \in [t_{\max}]$*. Let* $t \in \{0, \ldots, t_{\max} - 1\}$ *and* $\mathtt{k}^t$ *denote the group key at time* t *output by* CGKA.

Then the following properties of the graph $\mathcal{G}'_t$ associated to the same execution of the modified protocol CGKA′ *are true:*

10. *For every $u \in S(t, \mathtt{k}^t)$, the node $\mathtt{k}^t_{1,u,t}$ has no incoming edges and $\mathtt{k}^t_{1,u,t} \neq \mathtt{k}^t_{1,v,t}$ for all $u \neq v$. Actually it holds that $S(t, \mathtt{k}^t_{1,u,t}) = \{u\}$.*

11. *For every node $\mathtt{k}^t_{i,u,t}$ there exists at most one node $\mathtt{r}$ such that $\mathsf{PRG}_j(\mathtt{r}) = \mathtt{k}^t_{i,u,t}$ for some $j \in \{1,2\}$, or that $\mathsf{PRF}(\mathtt{r}, ad) = \mathtt{k}^t_{i,u,t}$ for some ad, or that $\mathsf{dPRF}(\mathtt{r}, \mathtt{r}') = \mathtt{k}^t_{i,u,t}$ for some $\mathtt{r}'$ (where $\mathtt{r}'$ may not be in $\mathcal{V}_t$).*

12. *There exists at most one user u in $S(t, \mathtt{k}^t)$ such that for every $1 \leq i \leq \ell_u - 1$ the edge $(\mathtt{r}_{i,u,t}, \mathtt{r}_{i+1,u,t})$ is trivial.*

13. *If $D_{t+1} \neq \emptyset$, then for every $u \in S(t, \mathtt{k}^t) \setminus D_{t+1}$, there exists $j_{u,t}$ such that $1 \leq j_{u,t} < \ell_{u,t}$ and for the corresponding edge $(\mathtt{k}^t_{j_{u,t},u,t}, \mathtt{k}^t_{j_{u,t}+1,u,t}) \in \mathcal{E}_t$ there exists a user $v \in D_{t+1}$ such that $v \in S(t, \mathtt{k}^t_{j_{u,t}+1,u,t})$ and for all $w \in D_{t+1}$ we have $w \notin S(t, \mathtt{k}^t_{j_{u,t},u,t})$. Moreover, $j_{u,t}$ will denote the least integer in $\{1, \ldots, \ell_{u,t} - 1\}$ with this property.*

A.1.2 Lower Bound on Batched Replacements

Lemma A.1.6. *Consider an execution of game $\mathrm{SEC}^{\mathsf{CGKA}}$ with respect to a correct and secure continuous-group-key agreement scheme* CGKA *on input $(n_{\max}, t_{\max}, G_0, u_0)$, $(u_t, A_t, D_t)_{t=1}^{t_{\max}}$ such that $D_t \subseteq G_{t-1}$, $u_t \in G_{t-1} \setminus D_t$, and $A_t \subseteq [n_{\max}] \setminus (G_{t-1} \cup \bigcup_{t'=1}^{t-1} D_{t'})$ for all $t \in [t_{\max}]$. Let $(\mathcal{S}_t)_{t=0}^{t_{\max}}$ be the associated set system as defined in Definition A.1.1 and*

$$\tilde{\mathcal{S}}_t = \left\{ S \in \mathcal{S}_t \;\middle|\; \begin{array}{l} \exists \mathtt{r} \text{ such that } S = S(t, \mathtt{r}),\ \mathtt{r} \text{ is useful at time } t \text{ and} \\ \nexists \mathtt{r}_1, \mathtt{r}_2 \text{ such that } \mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2) = \mathtt{r} \text{ and } S(t, \mathtt{r}_1) \cap S(t, \mathtt{r}_2) = S(t, \mathtt{r}) \end{array} \right\}.$$

Then it holds that

(i) $G_t \in \tilde{\mathcal{S}}_t$ *for all* $t \in [t_{\max}]_0$,

(ii) $S \subseteq G_t$ *for all* $S \in \mathcal{S}_t$ *and, in particular,* $S \subseteq G_t$ *for all* $S \in \tilde{\mathcal{S}}_t$,

(iii) $|\mathtt{pub}| + \sum_{t=0}^{t_{\max}} |\mathtt{M}_t| \geq 1/3 \cdot \sum_{t=0}^{t_{\max}} \mathrm{Cost}(t)$, *where* $\mathrm{Cost}(t)$ *is the cost function defined in Section 4.2.1 with respect to* $\tilde{\mathcal{S}}_t$, *namely:*

$$\begin{aligned} \mathrm{Cost}(t) = \mathrm{SizeMinCov}_{=}(G_t, \tilde{\mathcal{S}}_{t-1} \cup \{\{u\} : u \in G_t\}) - 1 \\ + |\{S \in \tilde{\mathcal{S}}_{t-1} : S \cap D_t \neq \emptyset \text{ and } |S| > 1\}|. \end{aligned}$$

For the proof we refer the reader to Appendix A.2, but we observe that the lower bound in Property (iii) only relies on counting ciphertexts not public keys. Lemma A.1.6 shows that it is possible to apply Theorem 4.2.4 to set system $(\tilde{\mathcal{S}}_t)_{t=0}^{t_{\max}}$ and therefore we obtain the following bound on the communication complexity of batched replacements in CGKA schemes.

Corollary A.1.7. *Let $n \leq n_{\max}$ and $t_{\max}$ be in $\mathbb{N}$ and $(d_t)_{t=1}^{t_{\max}}$ such that $d_t < n$ for all t. Consider an execution of game $\mathrm{SEC}^{\mathsf{CGKA}}$ with respect to a correct and secure CGKA scheme* CGKA *on input $(n_{\max}, t_{\max}, G_0, u_0)$ and $(u_t, A_t, D_t)_{t=1}^{t_{\max}}$ with $|G_0| = n$, where for all t the set D_t of removed users is sampled uniformly at random from the set $\{D \subseteq G_{t-1} \mid |D| = d_t\}$, and u_t and A_t can be arbitrary according to the restrictions $u_t \in G_{t-1} \setminus D_t$ and $A_t \subseteq [n_{\max}] \setminus (G_{t-1} \cup \bigcup_{t'=1}^{t-1} D_t)$ respectively. Then it holds that*

$$\mathbb{E}\left[|\mathtt{pub}| + \sum_{t=0}^{t_{\max}} |\mathtt{M}_t|\right] \geq \frac{1}{3} \sum_{t=1}^{t_{\max}} d_t \ln\left(\frac{n}{d_t}\right),$$

where the expectation is taken over the choice of $(D_t)_t$. In particular, if $d_t = d$ for all t, then

$$\mathbb{E}\left[|\mathtt{pub}| + \sum_{t=0}^{t_{\max}} |\mathtt{M}_t|\right] \geq \frac{\ln(2)}{3} t_{\max} \cdot d \cdot \log\left(\frac{n}{d}\right).$$

A.1.3 Extending the bound to Key-updatable PKE.

In [BDR20] the authors in their symbolic model additionally include key-updatable public-key encryption (kuPKE)[4] and broadcast encryption (BE) in the considered building blocks. In the following we will argue that the lower bound for CGKA proven in Section A.1.2 also applies to schemes additionally taking kuPKE into account.

Key-Updatable PKE. A scheme kuPKE = (kuPKE.Gen, kuPKE.Enc, kuPKE.Dec, kuPKE.Upd) specifies the following. As is the case for PKE, kuPKE.Gen receives as input a secret key $\mathtt{sk}$ (of type $\mathtt{r}$) and returns the corresponding public key $\mathtt{pk}$. Encryption algorithm kuPKE.Enc$(\mathtt{pk}, \mathtt{m})$, on input a public key $\mathtt{pk}$ and message $\mathtt{m}$, returns a ciphertext $\mathtt{c}$ (of message type). Decryption algorithm kuPKE.Dec$(\mathtt{sk}, \mathtt{c})$, on input a secret key $\mathtt{sk}$ and ciphertext $\mathtt{c}$, returns a message $\mathtt{m}$. The update algorithm kuPKE.Upd receives as input associated data ad as well as either a public key $\mathtt{pk}$ or a secret key $\mathtt{sk}$ and returns an updated public key $\mathtt{pk}'$ or secret key $\mathtt{sk}'$, respectively. We require that the updated keys are distinct symbolic variables from the ones input to kuPKE.Upd.

[4] This primitive implies hierarchical identity based encryption (HIBE) in the random oracle model if we use as associated data a hash of the identities. By allowing the use of kuPKE we strengthen our lower bound.

The rules for the entailment relation with respect to kuPKE are

$$\begin{aligned} &\mathtt{M} \vdash \mathtt{pk} \Rightarrow \forall ad : \mathtt{M} \vdash \mathsf{kuPKE.Upd}(\mathtt{pk}, ad) \\ &\mathtt{M} \vdash \mathtt{sk} \Rightarrow \forall ad : \mathtt{M} \vdash \mathsf{kuPKE.Upd}(\mathtt{sk}, ad) \text{ and } \mathtt{M} \vdash \mathsf{kuPKE.Gen}(\mathtt{sk}) \end{aligned} \tag{A.3}$$

i.e., from a key one can obtain its update under every associated data, and regarding security and correctness

$$\begin{aligned} &\exists \mathtt{sk}_0, ad_o, \ldots, ad_{s-1} \text{ and } \mathtt{M} \vdash (\mathtt{c}, \mathtt{sk}_r) : \\ &\quad \mathtt{c} = \mathsf{kuPKE.Enc}(\mathtt{pk}_s, \mathtt{m}), \\ &\quad \mathtt{pk}_0 = \mathsf{kuPKE.Gen}(\mathtt{sk}_0), s \geq r \geq 0, \\ &\quad \forall i \in [s-1]_0 : \mathtt{pk}_{i+1} = \mathsf{kuPKE.Upd}(\mathtt{pk}_i, ad_i), \\ &\quad \forall i \in [r-1]_0 : \mathtt{sk}_{i+1} = \mathsf{kuPKE.Upd}(\mathtt{sk}_i, ad_i) \\ &\Rightarrow \mathtt{M} \vdash \mathtt{m}, \end{aligned} \tag{A.4}$$

saying that a message encrypted under a public key that was updated several times can be recovered from its ciphertext if one is able to derive at least one of the preceding corresponding secret keys in the update chain.

Extending the lower bound to kuPKE. In the following we will argue that the lower bound of Corollary A.1.7 also holds if one additionally allows for kuPKE to be one of the considered building blocks. The intuition behind this is that the update mechanism of secret keys in kuPKE can be emulated by applying a PRF to the secret key. Note, however, that using this update mechanism update public keys can only be derived if one also has access to the corresponding secret key. Thus our argument proceeds in two steps. In the first for a particular execution of the security game we essentially move every update operation kuPKE.Upd applied to a public key $\mathtt{pk}$ to the point in time where the public key was generated from its corresponding secret key. In the second step we are now able to replace kuPKE by PKE and PRF applications as every algorithm using kuPKE.Upd on a public key has access to the corresponding secret key. To prove that our bound carries over we have to argue that these modifications preserve correctness, security, and leave the cost, i.e., the number of ciphertexts, unchanged. Formally, we obtain the following.

Corollary A.1.8. *Let $n \leq n_{\max}$ and $t_{\max}$ be in $\mathbb{N}$ and $(d_t)_{t=1}^{t_{\max}}$ such that $d_t < n$ for all t. Let* CGKA *be a correct and secure CGKA scheme using PKE, kuPKE, PRG, PRF, dPRF, and secret sharing as building blocks. Consider an execution of game $\mathrm{SEC}^{\mathsf{CGKA}}$ with respect to* CGKA *on input $(n_{\max}, t_{\max}, G_0, u_0)$ and $(u_t, A_t, D_t)_{t=1}^{t_{\max}}$ with $|G_0| = n$, where for all t the set D_t of removed users is sampled uniformly at random from the set $\{D \subseteq G_{t-1} \mid |D| = d_t\}$, and u_t and A_t can be arbitrary according to the*

restrictions $u_t \in G_{t-1} \setminus D_t$ *and* $A_t \subseteq [n_{\max}] \setminus (G_{t-1} \cup \bigcup_{t'=1}^{t-1} D_{t'})$ *respectively. Then it holds that*

$$\mathbb{E}\left[|\mathtt{pub}| + \sum_{t=0}^{t_{\max}} |\mathtt{M}_t|\right] \geq \frac{1}{3} \sum_{t=1}^{t_{\max}} d_t \ln\left(\frac{n}{d_t}\right),$$

where the expectation is taken over the choice of $(D_t)_t$*. In particular, if* $d_t = d$ *for all* t*, then*

$$\mathbb{E}\left[|\mathtt{pub}| + \sum_{t=0}^{t_{\max}} |\mathtt{M}_t|\right] \geq \frac{\ln(2)}{3} t_{\max} \cdot d \cdot \log\left(\frac{n}{d}\right).$$

Proof. Consider the modification to the derivation of symbolic variables in security game $\mathrm{SEC}^{\mathsf{CGKA}}$ of Figure A.1 with respect to CGKA, $(n_{\max}, t_{\max}, G_0, u_0)$, and $(u_t, A_t, D_t)_{t=1}^{t_{\max}}$ defined as follows. Whenever one of the algorithms CGKA.Setup, CGKA.Init, CGKA.Repl, CGKA.Proc, or CGKA.Key makes a call to $\mathtt{pk}_0 \leftarrow \mathsf{kuPKE.Gen}(\mathtt{sk}_0)$ for some secret key $\mathtt{sk}_0$ do the following:

- Parse the remainder of the security experiment for all chains of associated data used to update $\mathtt{pk}_0$, i.e., all $ad_{0,j}, \ldots, ad_{\ell_j,j}$ such that there are symbolic operations $\mathtt{pk}_{i+1,j} \leftarrow \mathsf{kuPKE.Upd}(\mathtt{pk}_{i,j})$ for $i \in [\ell_j]_0$ where we define $\mathtt{pk}_{0,j} = \mathtt{pk}_0$ for all such j.

- In the code move the execution of all update operations $\mathsf{kuPKE.Upd}(\mathtt{pk}_{i,j}, ad_{i,j})$ to the lines immediately after the call to $\mathsf{kuPKE.Gen}(\mathtt{sk})$.

- Add all public keys to the outputs of the CGKA operation that the call to kuPKE.Gen occurred in, i.e., to $\mathtt{pub}$ (but not $\mathtt{ST}_u$) for CGKA.Setup, to $\mathtt{M}_0$ and $\mathtt{ST}_{u_0}$ for CGKA.Init, to $\mathtt{ST}_u$ for CGKA.Proc, and to $\mathtt{M}_t$ and $\mathtt{ST}_{u_t}$ for CGKA.Repl. Moreover all public keys in $(\mathtt{M}_{t'})_{t'=0}^{t-1}$ are included again in $\mathtt{M}_t$ to guarantee that users added to the group later also have access to them.

Note that this change does not affect the number of ciphertexts sent throughout the security game and that it preserves correctness. The lower bound in Property (iii) only relies on counting ciphertexts not public keys, so if it holds for the modified protocol and the number of ciphertexts is preserved. Indeed, all algorithms are still able to derive all the required symbolic variables, in particular the group keys, since they only learn about public keys at an earlier point in time, either as it was included in the public parameters, a public message, or because the particular user added it to their state at an earlier time (we may assume that public keys never get deleted from users' states since this does not affect the set of secrets one is able to derive as we will argue below). Further, as the modification did not change any secret symbolic variables, all users

must still derive the same group key as required in lines 11 and 31 of the security game. We now argue that including additional kuPKE public keys in the set

$$\overline{\mathtt{M}}_t := (\mathtt{pub}, (\mathtt{M}_{t'})_{t'=0}^{t}, ((\mathtt{ST}_u^{t'})_{t'=-1}^{t}, (\mathtt{R}_u^{t'})_{t'=0}^{t})_{u\in[n_{\max}]\setminus G_t})$$

leaves the set of symbolic secrets that can be derived from $\overline{\mathtt{M}}_t$ unchanged. This, in particular, implies that the modification preserves security, namely we have $\mathtt{k}^t \notin \mathsf{Der}(\overline{\mathtt{M}}_t)$ for all t as required in Lines 13 and 33. To see this, note that a public key $\mathtt{pk}$ can only be updated to other public keys, be used as a message and encrypted, or be used to encrypt a messag. In the latter case the encrypted message has to be derivable from $\overline{\mathtt{M}}_t$ as well, and in turn does not yield additional secrets if later decrypted (in case the corresponding secret key $\mathtt{sk}$ is derivable from $\overline{\mathtt{M}}_t$). Finally note that by definition of the change users' initial states still only contain variables of type random coins. Concluding our argument, that the modification does preserve correctness and security of the execution of the security game.

We now argue that after this modification we may see the use of kuPKE in the security as a variant that uses a modified syntax which simply sees all updates to public keys as being part of kuPKE.Gen. Accordingly, this modified kuPKE syntax uses algorithms $\mathsf{kuPKE.Enc}_{\mathrm{mod}}$ and $\mathsf{kuPKE.Dec}_{\mathrm{mod}}$ that behave as in normal kuPKE, an update algorithm $\mathsf{kuPKE.Upd}_{\mathrm{mod}}$ that on input a secret key $\mathtt{sk}$ and additional data ad returns updated secret key $\mathtt{sk}'$ (but no longer accepts public keys as input), and a key generation algorithm $\mathsf{kuPKE.Gen}_{\mathrm{mod}}$ that on input $\mathtt{sk}_0$ and chains of associated data $(ad_{i,j})_{i\in[\ell_j]_0,j\in[k]}$ returns corresponding public keys $\mathtt{pk}_0$ and $(\mathtt{pk}_{i,j})_{i\in[\ell_j+1],j\in[k]}$. Plugging his into the rules for the entailment relation given in Equations A.3 and A.4 they now read as

$$\begin{aligned}&\mathtt{M}\vdash \mathtt{sk} \Rightarrow \forall (ad_{i,j})_{i\in[\ell_j]_0,j\in[k]} : \mathtt{M}\vdash \mathsf{kuPKE.Gen}_{\mathrm{mod}}(\mathtt{sk}, (ad_{i,j})_{i\in[\ell_j]_0,j\in[k]}),\\ &\mathtt{M}\vdash \mathtt{sk} \Rightarrow \forall ad : \mathtt{M}\vdash \mathsf{kuPKE.Upd}_{\mathrm{mod}}(\mathtt{sk}, ad)\end{aligned} \tag{A.5}$$

and

$$\begin{aligned}&\exists \mathtt{sk}_0, (ad_{i,j})_{i\in[\ell_j]_0,j\in[k]} \text{ and } \mathtt{M}\vdash (\mathtt{c}, \mathtt{sk}_{r,j}) :\\ &\quad (\mathtt{pk}_0, (\mathtt{pk}_{i,j})_{i\in[\ell_j+1],j\in[k]}) = \mathsf{kuPKE.Gen}_{\mathrm{mod}}(\mathtt{sk}_0, (ad_{i,j})_{i\in[\ell_j]_0,j\in[k]}), \ell_j+1\geq s\geq r\geq 0,\\ &\quad \mathtt{c} = \mathsf{kuPKE.Enc}_{\mathrm{mod}}(\mathtt{pk}_{s,j}, \mathtt{m}) \text{ for some } j\in[k],\\ &\quad \forall i\in[r-1]_0 : \mathtt{sk}_{i+1,j} = \mathsf{kuPKE.Upd}_{\mathrm{mod}}(\mathtt{sk}_i, ad_{i,j})\\ &\Rightarrow \mathtt{M}\vdash \mathtt{m}.\end{aligned} \tag{A.6}$$

In a final step, we now argue that the kuPKE with modified syntax can be realized in the symbolic model from PKE and PRF as follows. Algorithms $\mathsf{kuPKE.Enc}_{\mathrm{mod}}$ and $\mathsf{kuPKE.Dec}_{\mathrm{mod}}$ are replaced by their counterparts PKE.Enc and PKE.Dec, secret keys

are updated as $\mathsf{kuPKE.Upd}_{\mathrm{mod}}(\mathtt{sk}, ad) = \mathsf{PRF}(\mathtt{sk}, ad)$, and $\mathsf{kuPKE.Gen}_{\mathrm{mod}}(\mathtt{sk}, (ad_{i,j})_{i\in[\ell_j]_0, j\in[k]})$ first sets $\mathtt{pk}_0 = \mathsf{PKE.Gen}(\mathtt{sk}_0)$, then for all $j \in [k]$ and all $i \in [\ell_j]_0$ iteratively computes $\mathtt{sk}_{i+1,j} = \mathsf{PRF}(\mathtt{sk}_{i,j}, ad_{i,j})$ and $\mathtt{pk}_{i+1,j} = \mathsf{PKE.Gen}(\mathtt{sk}_{i+1,j})$, where we set $\mathtt{pk}_{0,j} = \mathtt{pk}_0$ for all j. Its output is $(\mathtt{pk}_0, (\mathtt{pk}_{i,j})_{i\in[\ell_j], j\in[k]})$. Note, that due to the application of the PRF updated keys differ from their not updated counterparts, as required. Further we will now argue that the construction satisfies the three rules regarding the entailment relation. The rules in Equation A.5 follow immediately from the syntax of PKE and PRFs in the symbolic model. Regarding the third rule, assume that $\mathtt{M} \vdash (\mathtt{c}, \mathtt{sk}_{r,j})$ with the properties listed in Equation A.6. Then by construction it must be the case that

$$\mathtt{sk}_{r,j} = \mathsf{PRF}(\cdot, ad_{r-1,j}) \circ \cdots \circ \mathsf{PRF}(\cdot, ad_{0,j})(\mathtt{sk}_{0,j}),$$

and

$$\mathtt{pk}_{s,j} = \mathsf{PKE.Gen} \circ \mathsf{PRF}(\cdot, ad_{s-1,j}) \circ \cdots \circ \mathsf{PRF}(\cdot, ad_{0,j})(\mathtt{sk}_{0,j}).$$

Thus given $\mathtt{sk}_{r,j}$ one can compute $\mathtt{sk}_{s,j}$ from $\mathtt{sk}_{r,j}$ by applying $\mathsf{PRF}(\cdot, ad_{i,j})$ for $i \in [s, \ldots, r-1]$ (if $r = s$ no PRF applications are necessary) and it holds that $\mathtt{pk}_{s,j} = \mathsf{PKE.Gen}(\mathtt{sk}_{s,j})$. In turn, one obtains $\mathtt{m}$ from $\mathtt{c}$ by using the derivation rule modeling correctness and security of PKE, which concludes the argument that modified kuPKE can be constructed from PKE and PRFs.

This in particular implies that replacing the modified kuPKE scheme in the security game by PKE and PRF preserves correctness and security. Now, the corollary follows by applying Corollary A.1.7 and observing that the modification using only PKE and PRFs leaves the number of ciphertexts sent during the game unchanged. □

A.1.4 Regarding the Use of Broadcast Encryption as Additional Building Block.

The bound by Bienstock, Dodis, and Rösler [BDR20] additionally allows for the use of broadcast encryption (BE), which essentially allows to register from a master secret-key so called user secret-keys in a way that allows the creation of ciphertexts with respect to a master public-key and a set R which in turn can be decrypted by all registered users that are not in R. In this section we recall the definition of BE and then briefly argue that it allows for the construction of ME with constant communication complexity. Thus, while we consider it an interesting open question whether our bound for CGKA can be extended to also include BE as a building block, the proof would require an approach substantially differing from ours.

A broadcast encryption scheme $\mathsf{BE} = (\mathsf{BE.Gen}, \mathsf{BE.Reg}, \mathsf{BE.Enc}, \mathsf{BE.Dec})$ specifies the following. Algorithm BE.Gen on input a master secret-key $\mathtt{msk}$ of type $\mathtt{r}$ returns the corresponding master public-key $\mathtt{mpk}$ of public-key type. Registration algorithm BE.Reg can be used to register a secret key for a particular identity that is modeled by a

non-symbolic identifier $u \in \mathbb{N}$. On input $(\mathtt{msk}, u)$ it returns the user's secret key $\mathtt{sk}_u$, which we require to not be equal to $\mathtt{msk}$. The encryption algorithm $\mathsf{BE.Enc}(\mathtt{mpk}, R, \mathtt{m})$ on input the master public-key, a set of excluded users $R \subseteq \mathbb{N}$, and a message $\mathtt{m}$ returns a ciphertext $\mathtt{c}$ of message type. Finally, decryption algorithm BE.Dec on input $(\mathtt{sk}, \mathtt{c})$ returns a message $\mathtt{m}$. The rules for the entailment relation are that from a master secret key one can derive all users' secret keys, i.e.,

$$\mathtt{M} \vdash \mathtt{msk} \Rightarrow \forall u \in \mathbb{N} : \mathtt{M} \vdash \mathsf{BE.Reg}(\mathtt{msk}, u),$$

and that from a ciphertext encrypted under a master public key one can recover the underlying message given access to the secret key of at least one user that was not excluded, i.e.,

$$\begin{aligned} &\exists u \notin R \wedge \mathtt{M} \vdash (\mathtt{c}, \mathtt{sk}_u) : \\ &\qquad \mathtt{c} = \mathsf{BE.Enc}(\mathtt{mpk}, R, \mathtt{m}), \\ &\qquad \mathtt{mpk} = \mathsf{BE.Gen}(\mathtt{msk}), \\ &\qquad \mathtt{sk}_u = \mathsf{BE.Reg}(\mathtt{msk}, u) \\ &\Rightarrow \mathtt{M} \vdash \mathtt{m}. \end{aligned}$$

We now briefly describe how one can use BE to construct a multicast encryption scheme ME that allows for batched user replacement with constant per-round communication complexity. $\mathsf{ME.Setup}(n_{\max}; \mathtt{r})$ generates a master secret key $\mathtt{msk} = \mathsf{BE.Gen}(\mathtt{r})$ and then sets the initial state of each user $u \in [n_{\max}]$ to $\mathsf{sk}_u = \mathsf{BE.Reg}(\mathtt{msk}, u)$. To initialize a group G_0, algorithm ME.Init samples a random secret $\mathtt{k}_0$ and returns the control message $\mathsf{BE.Enc}(\mathtt{mpk}, [n_{\max}] \setminus G_0, \mathtt{k}_0)$, and similarly replacing a set D_t of users by users A_t is done by sampling key $\mathtt{k}_t$ and sending the control message $\mathsf{BE.Enc}(\mathtt{mpk}, [n_{\max}] \setminus G_t, \mathtt{k}_t)$. Note that all group members at time t have access to the group key, since they were not excluded from the recipient set, and that further, that even if all non-group members collide they are not able to derive $\mathtt{k}_t$, as they are not able to derive at least one of the $\mathtt{sk}_u$ for $u \in G_t$ (here we use that removed users are never added back to the group). Finally, note that each round a single ciphertext is sent.

A.2 Proofs of Section A.1

A.2.1 Proofs of Appendix A.1.1 (Lemmas A.1.3, A.1.4 and A.1.5)

The approach we follow is essentially the same as in the case of multicast encryption (Lemmas 4.3.2, 4.3.3 and 4.3.4) and account for the differences between the two primitives.

Proof of Lemma A.1.3. If $\mathtt{r}$ admits a PRG pre-image $\mathtt{r}'$, $\mathtt{r}'$ must be useful at time t since $\mathtt{r}$ is useful at time t. Therefore we have two possible cases depending on whether $u \in S(t, \mathtt{r}')$. If $u \in S(t, \mathtt{r}')$, we are in Case 1. If $u \notin S(t, \mathtt{r}')$, one of the following holds:

- $\mathtt{r} \in \mathtt{pub} \cup \mathtt{ST}_u^{-1} \cup \bigcup_{0 \leq t' \leq t:\, u \in G_{t'}} \mathtt{M}_{t'} \cup \bigcup_{t'=0}^{t} \mathtt{R}_u^{t'}$ and the fact that $\mathtt{r}$ is useful implies that $\mathtt{r} \in \mathtt{ST}_u^{-1} \cup \bigcup_{t'=0}^{t} \mathtt{R}_u^{t'}$.
- There exists a ciphertext $\mathtt{c} \in \mathtt{pub} \cup \mathtt{ST}_u^{-1} \cup \bigcup_{0 \leq t' \leq t:\, u \in G_{t'}} \mathtt{M}_{t'} \cup \bigcup_{t'=0}^{t} \mathtt{R}_u^{t'}$ of the form described in Case 5 such that condition *(f)* holds. By assumption $\mathtt{ST}_u^{-1}$ only contains symbols of type random coins, so $\mathtt{c} \in \mathtt{pub} \cup \bigcup_{0 \leq t' \leq t:\, u \in G_{t'}} \mathtt{M}_{t'}$. The usefulness of $\mathtt{r}$ implies that there must exist $i \in \{0, \ldots, h\}$ such that e_i is an encryption under a public key with an associated secret key that is useful. This shows *(d)* and *(e)*. Condition *(c)* is just a matter of choice.

If $\mathtt{r}$ does not admit a PRG pre-image, we proceed to consider whether there exist $\mathtt{r}'$ and associated data ad such that $\mathsf{PRF}(\mathtt{r}', ad) = \mathtt{r}$. If this is the case, $\mathtt{r}'$ must be useful at time t since $\mathtt{r}$ is. Depending on whether $u \in S(t, \mathtt{r}')$ we have two possible cases. If $u \in S(t, \mathtt{r}')$, we are in Case 2. If $u \notin S(t, \mathtt{r}')$, then we are in Cases 4 or 5 by the same argument as in the case of PRG pre-images. If $\mathtt{r}$ does not admit neither a PRG pre-image nor a PRF pre-image, we proceed to consider whether there exist $\mathtt{r}_1$ and $\mathtt{r}_2$ such that $\mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2) = \mathtt{r}$. In this case at least one of $\mathtt{r}_1$ and $\mathtt{r}_2$ must be useful at time t since $\mathtt{r}$ is. If $u \in S(t, \mathtt{r}_1) \cap S(t, \mathtt{r}_2)$, then we are in Case 3. Else we are in Cases 4 or 5 by arguing as in the case of PRG pre-images. □

Proof of Lemma A.1.4. Let $\mathtt{r} \leftarrow \mathtt{r}$ and $\mathsf{Seq} \leftarrow \emptyset$. Repeat $(\mathtt{r}, \mathsf{Seq}) \leftarrow f(\mathtt{r}, \mathsf{Seq})$ until $\mathtt{r} = \mathsf{STOP}$ where:

$$f(\mathtt{r}, \mathsf{Seq}) = \begin{cases} \text{if we are in Case 1, do } (\mathtt{r}, \mathsf{Seq}) \leftarrow (\mathtt{r}', \{\mathtt{r}\} \cup \mathsf{Seq}), \\ \text{if we are in Case 2, do } (\mathtt{r}, \mathsf{Seq}) \leftarrow (\mathtt{r}', \{\mathtt{r}\} \cup \mathsf{Seq}), \\ \text{if we are in Case 3 and } \mathtt{r}_i \text{ is useful, do } (\mathtt{r}, \mathsf{Seq}) \leftarrow (\mathtt{r}_i, \{\mathtt{r}\} \cup \mathsf{Seq}), \\ \text{if we are in Case 4, do} (\mathtt{r}, \mathsf{Seq}) \leftarrow (\mathsf{STOP}, \{\mathtt{r}\} \cup \mathsf{Seq}), \\ \text{if we are in Case 5, do} (\mathtt{r}, \mathsf{Seq}) \leftarrow (\mathtt{r}_g, \{\mathtt{r}\} \cup \mathsf{Seq}). \end{cases}$$

This process terminates since it is required that only a finite number of symbolic derivations is needed in order to derive the symbolic outputs of the CGKA algorithms from their symbolic inputs. By construction, all properties are clearly satisfied. □

Proof of Lemma A.1.5. First of all, we observe that in the modified protocol CGKA′ a random coin $\mathtt{r} \in \mathtt{ST}_u^{-1} \cup \bigcup_{t'=0}^{t} \mathtt{R}_u^{t'}$ sampled in some round can only be derived by

the user u and it remains a useful secret until $u \in D_t$. Moreover, it cannot have any incoming edges in the graph $\mathcal{G}_t$ by construction of the modified protocol CGKA′. By Property 8, $\mathtt{k}^t_{1,u,t} \in \mathtt{ST}_u^{-1} \cup \bigcup_{t'=0}^{t} \mathtt{R}_u^{t'}$, so this proves Property 10.

Property 11 follows directly from the properties of the symbolic model.

Property 12 is a direct consequence of the two previous properties.

Property 13 follows from the observation that the node $\mathtt{k}_{\ell_u,u,t} = \mathtt{k}^t$ satisfies the condition that $v \in S(t, \mathtt{k}^t)$ for all users $v \in D_{t+1}$ and the node $\mathtt{k}_{1,u,t}$ satisfies the second condition (by Property 10). Since $D_{t+1} \neq \emptyset$ by assumption, there must exist an edge with the required property. □

A.2.2 Proofs of Appendix A.1.2

The proof is essentially the same as the one given in the case of multicast encryption for Lemma 4.3.5, but adapted to the CGKA setting.

Proof of Lemma A.1.6. We start proving Property (ii). Let $S = S(t, \mathtt{r}) \in \mathcal{S}_t$ and $u \in S$. By definition of $S(t, \mathtt{r})$,

$$\mathtt{r} \in \mathsf{Der}(\mathtt{pub}, \mathtt{ST}_u^{-1}, (\mathtt{M}_{t'})_{0 \le t' \le t:\, u \in G_{t'}}, (\mathtt{R}_u^{t'})_{t'=0}^{t})$$

and since $\mathtt{r}$ is useful at time t,

$$\mathtt{r} \notin \mathsf{Der}\left(\mathtt{pub}, (\mathtt{M}_{t'})_{t'=0}^{t}, ((\mathtt{ST}_u^{t'})_{t'=-1}^{t}, (\mathtt{R}_u^{t'})_{t'=0}^{t})_{u \in [n_{\max}] \setminus G_t}\right),$$

Thus $u \in G_t$ as claimed in Property (ii).

Now we prove Property (i). By correctness there exists a group key at time t, $\mathtt{k}^t$, such that $\mathtt{k}^t = \mathsf{CGKA.Key}(\mathtt{ST}_u^t)$ for all users $u \in G_t$. Therefore $G_t \subseteq S(t, \mathtt{k}^t)$. By security

$$\mathtt{k}^t \notin \mathsf{Der}\left(\mathtt{pub}, (\mathtt{M}_{t'})_{t'=0}^{t}, ((\mathtt{ST}_u^{t'})_{t'=-1}^{t}, (\mathtt{R}_u^{t'})_{t'=0}^{t})_{u \in [n_{\max}] \setminus G_t}\right),$$

so $S(t, \mathtt{k}^t) \subseteq G_t$ and $\mathtt{k}^t$ is useful at time t. This shows that $S(t, \mathtt{k}^t) = G_t \in \mathcal{S}_t$. It remains to prove that $G_t \in \tilde{\mathcal{S}}_t$ as claimed in Property (i). Let's assume that there exist $\mathtt{r}_1, \mathtt{r}_2$ such that $\mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2) = \mathtt{k}^t$ and $S(t, \mathtt{r}_1) \cap S(t, \mathtt{r}_2) = S(t, \mathtt{k}^t) = G_t$. The fact that $\mathtt{k}^t$ is useful at time t implies that there exists $i \in \{1, 2\}$ such that $\mathtt{r}_i$ is useful at time t. By Property (ii) it must hold that $S(t, \mathtt{r}_i) \subseteq G_t$. By assumption $S(t, \mathtt{r}_1) \cap S(t, \mathtt{r}_2) = S(t, \mathtt{k}^t) = G_t$, so we also have $G_t \subseteq S(t, \mathtt{r}_i)$. Therefore $S(t, \mathtt{r}_i) = G_t$. By repeating this process we can find a secret $\mathtt{r}$ that is useful at time t and that satisfies the properties that $S(t, \mathtt{r}) = G_t$ and $\nexists \mathtt{r}_1, \mathtt{r}_2$ such that $\mathsf{dPRF}(\mathtt{r}_1, \mathtt{r}_2) = \mathtt{r}$ and $S(t, \mathtt{r}_1) \cap S(t, \mathtt{r}_2) = S(t, \mathtt{r})$. This completes the proof of Property (i), i.e., $G_t \in \tilde{\mathcal{S}}_t$.

Finally, in order to prove Property (iii), it suffices to observe that the graph $\mathcal{G}'_t$ from the modified protocol CGKA$'$ satisfies the same properties according to Lemma A.1.5 than the ones we needed in the multicast encryption setting from Lemma 4.3.4. This shows the cost inequality for CGKA$'$ and this protocol preserves the cost of the original protocol CGKA as well as the value of the function $\mathrm{Cost}(t)$, which in turn proves inequality(iii) for CGKA. □

APPENDIX B

Supplementary material for Chapter 5

B.1 Formal Security Definition

For the definition of the PKI functionalities see Figures B.1 and B.2.

The reader may find a pseudo-code description of the ideal functionality $\mathcal{F}_{\mathsf{CGKA}}$ in Figures B.3, B.4, B.5 and B.6.

The helpers used to describe $\mathcal{F}_{\mathsf{CGKA}}$ can be found in Figures B.7, B.8, B.9, B.10 and B.11.

The reader may find the definition of the predicates safe(c) and inj-allowed(c, u) in Figure B.12.

B.2 Protocol Details

B.2.1 Formal description of MLS

In this section we provide pseudocode for the CGKA underlying MLS. Our protocol description closely follows the one of [AJM22], more precisely the one of protocol Insider TreeKEM (ITK), a CGKA introducing fixes to a prior draft of the MLS protocol that were later adapted into the standard. As the focus on this work is on the impact of blanking on the protocol's communication complexity, makes the same simplifying assumptions. I.e., we focus on the CGKA part of the protocol and do not include its secure messaging layer, assume a fixed protocol version and cipher suite, and omit

features like metadata protection and the additional proposal types PreSharedKey, ReInit, ExternalInit, GroupContextExtensions and external proposals. As a minor difference we use the parent hash mechanism as defined in the IETF standard [BBR+23] that slightly differs from the one used by ITK.

The protocol is defined with respect to the following primitives:

- a public-key encryption scheme $\mathsf{PKE} = (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ with key pairs $(\mathsf{pk}, \mathsf{sk})$,
- a signature scheme $\mathsf{SIG} = (\mathsf{GenS}, \mathsf{Sign}, \mathsf{Vrfy})$ with key pairs $(\mathsf{svk}, \mathsf{ssk})$,
- a message authentication code $\mathsf{MAC} = (\mathsf{MAC.Tag}, \mathsf{MAC.Ver})$,
- a key-derivation function the use of which is modeled as hash functions $\mathsf{H}_1, \mathsf{H}_2$, and
- a hash function H.

Protocol state. We list the content of users' states, as well as the intuition behind additional values computed throughout the protocol execution in Table B.1.

Interaction with KS and AS. Users interact with the key service and authentication service via the following.

fetch-ssk-if-nec is used to retrieve (and potentially update) a user's secret signing key (see Figure B.13).

get-sks is used to fetch the secret keys of key packages when joining a group (see Figure B.2).

fetch-kp: fetches a user's key package from the KS (See Figure B.2).

validate-kp is used to validate key packages (See Figure B.13).

Gen-kp generates a key package (Figure B.13).

Ratchet tree state and modifications. The state of nodes in the protocol's underlying ratchet tree is listed in Table B.2. The table further list functions returning particular properties of the tree, as well as functions implementing changes to the ratchet tree.

state	
st.groupID	the group identifier
st.ep	the current epoch
st.id	the user's leaf identifier
st.T	the user's view of the ratchet tree
st.tHash	tree hash of st.T
st.ssk	the user's current secret signing key
st.cert-SPks	the signature verification keys associated to each user's id
st.pendUpd	a list of pending updates
st.pendComm	a list of pending commits
key schedule stored in state	
st.appSec	the current epoch's CGKA group key
st.interim-tranHash	the interim transcript hash for the following epoch
st.memKey	membership key used to frame protocol messages
st.initSec	the next epoch's initialization secret
additional values	
comSec	commit secret generated by rekey-path
joinerSec	secret for added group members, derived from initSec, comSec
confKey	key to tag confirmed transcript hash
conf-TransHash	hash encoding operations performed so far

Table B.1: Users' state in MLS.

Protocol algorithms. The group creation, commit, process, join and key-recovery algorithms of MLS can be found in Figure B.14. Its proposal algorithm, contrasted against the one of MLS-Cutoff, is in Figure B.15. The algorithms make use of helper functionalities rekey-path, apply-rekey (both in Figure B.17), and apply-props. The latter is in Figure B.18, contrasted against its MLS-Cutoff variant.

Further, the protocol employs several additional helper functionalities. While these play an important role in deriving the key schedule and ensuring consistency and authenticity of protocol messages, they do not play a role in modifications to ratchet tree's state, which is the main focus of this work. Thus for formal definitions of these functions we refer to [AJM22], providing only an overview on their roles below.

- init-epoch on input a user's state st returns a copy st′ of the state with incremented epoch st′.ep $=$ st.ep $+ 1$ and empty list of pending updates and commits.

- derive-keys on input state st, pending state st′ and commit secret comSec first computes the joiner secret joinerSec from comSec and initSec. From this it computes confirmation key confKey, application secret appSec, membership

public state of node v	
v.index	index of the node
v.pk	public PKE key
v.pHash	parent hash value
v.unm	unmerged leaves
v.svk	signature verification key (if v is leaf)
v.cred	credential (if v is leaf)
v.kp	key package $v.\mathsf{kp} = (v.\mathsf{pk}, v.\mathsf{svk}, v.\mathsf{cred})$ (if v is leaf)
private state of node v	
v.sk	secret PKE key
tree state	
root(T)	root of the tree
leaves(T)	returns the leaves of T
T.public	public part of the ratchet tree
v_{id}	leaf associated to leaf identity id
id(v)	leaf identity of node v
path(v)	direct path from v to the root
fil-path(v)	filtered direct path from v_0 to the root
lca(u, v)	least common ancestor w.r.t filtered paths of leaves u, v
ind-lca(u, v)	index (from root) of the least common ancestor w.r.t filtered paths of leaves u, v
Res(v)	resolution of node v
roster(T)	returns list of leaf IDs
modifications to tree	
init-tree	creates ratchet tree of size 1
assign-kp(T, id, kp)	assigns leaf associated to id key package kp
add-leaf(T, id)	assign id leftmost free leaf, doubles size of T if necessary
truncate(T)	removes right half of T if it contains no populated leaves
blank-leaf(T, v)	calls blank(v) for leaf v
blank-path(T, v, i)	calls blank(v_j) for $v_j \in$ path(v) with $j \geq i$
set-as-unmerged(T, v_0)	sets v.unm $\leftarrow_\cup \{v_0\}$ for all $v \in$ path(v_0)
merge-leaves(T, v_0)	sets v.unm $\leftarrow \emptyset$ for all $v \in$ path(v_0)

Table B.2: Ratchet tree state and functions modifying the ratchet tree.

key memKey, and updated initialization secret initSec. The latter three are stored in st′, and the function's output is $(\text{st}', \text{confKey})$.

- derive-epochKeys on input pending state st′ and joiner secret joinerSec computes confKey, appSec, memKey, and initSec in the same way as derive-keys and returns $(\text{st}', \text{confKey}, \text{joinerSec})$.

- sign-commit receives as input the state st and value C that contains hashes of the applied commits and the UpdPath object. It returns a signature σ on the concatenation of C with the group context, id, epoch, committer leaf id.

- set-int-transHash on input pending state st′ and confirmation tag confTag computes the interim transcript hash interim-tranHash as a hash of the confirmed transcript hash stored in st′ and confTag. It stores the resulting value in st′.

- set-conf-trans-hash on input state st, pending state st′, sender id id-snd, C, and signature σ as above computes the confirmed transcript hash conf-TransHash as a hash of the latter three values and the interim transcript hash stored in st′. The confirmed transcript hash conf-TransHash is stored in st′.

- compute-confTag on input pending state st′ and confirmation key confKey returns a MAC confTag of the confirmed transcript hash conf-TransHash contained in st′.

- verify-confTag on input pending statest′, confirmation key confKey, and confirmation tag confTag verifies the tag.

- frame-prop frames proposal $P = (\text{`type'}, \text{ad})$ by signing and MACing (under the membership key memKey) P together with group context, group id, epoch, leaf id of the sender.

- unframe-prop on input state st and framed proposal pmsg, verifies signature and MAC tag and returns P as well as the sender id.

- frame-comm on input state st, confirmation tag confTag, signature σ (as in sign-commit), and membership tag memTag frames the commit by appending group id, epoch, and id of the committer.

- unframe-comm on input state st and framed commit cmsg, verifies the contained signature and MAC tag. If both verify it returns the sender's id, C, the confirmation tag, signature, and the membership tag.

B.2.2 Formal description of MLS-Cutoff

We provide a formal description of protocol MLS-Cutoff. It is defined with respect to public key encryption scheme PKE, Signature scheme SIG, message authentication code MAC, hash function H, and a key-derivation function the use of which is denoted by H_1 and H_2. The protocol is parameterized by cutoff parameter i_{cut}. The protocol shares most algorithms with MLS differing only in two aspects:

MLS-Cutoff generates proposal messages in a different way. Its proposal algorithm CGKA.Propcan be found in Figure B.15 (right).

MLS-Cutoff uses a different method of applying proposals to the ratchet tree. I.e., algorithms CGKA.Com and CGKA.Proc make use of the subroutine apply-props of Figure B.18 (right).

All remaining aspects are handled as in MLS. I.e., the protocol has the same user state (except for storing the cutoff bound in st.i_{cut} that remains unchanged throughout the protocol execution), uses CGKA.Create, CGKA.Com, CGKA.Proc, CGKA.Join, and CGKA.Key as defined in Figure B.14, and makes use of the subroutines listed in Appendix B.2.1 and Figures B.17 and B.13. Note that when rekeying paths while applying updates MLS-Cutoff calls apply-rekey with option $\mathrm{copySt} = \mathrm{true}$. This means the parent hashes contained in each update proposal are verified with respect to a tree resulting from applying the modifications to st.T instead of the pending tree T′. Thus, the contained key package verifies if the update message was honestly generated, which implies the parent hash was computed with respect to such a tree.

Security of MLS-Cutoff**.**

Proof of Theorem 5.5.1. We are able to closely follow the security proof for ITK in [AJM22]. The proof is divided into two parts. First a sequence of hybrids shows that a version of the MLS protocol (MLS*) that has a different parent hash securely realizes $\mathcal{F}_{\mathsf{CGKA}}$ with a slightly different version of the safe predicate denoted safe*. The second step consists in showing that using the actual parent hash guarantees and the actual safe predicate satisfy the following property: if $\mathsf{safe}(c) = \mathrm{true}$, then no node in the ratchet tree of the epoch that corresponds to c contains an exposed key pk.

We argue that both steps can be adapted to the case of MLS-Cutoff. We consider the following hybrids:

- Hybrid $\mathcal{H}_0$: Real world.

- Hybrid $\mathcal{H}_1$: Instead of the protocol as in the real world, now we consider a dummy functionality $\mathcal{F}_{\mathsf{Dummy}}$ that simply routs its inputs and outputs through a simulator $\mathcal{S}_1$ that executes the protocol.

- Hybrid $\mathcal{H}_2$: $\mathcal{F}_{\mathsf{Dummy}}$ is substituted by the ideal functionalities $\mathcal{F}_{\mathsf{AS}}, \mathcal{F}_{\mathsf{KS}}$ and $\mathcal{F}_{\mathsf{CGKA}}$ with the only difference that $\mathcal{F}_{\mathsf{CGKA}}$ uses as predicates safe$(c) = \mathrm{false}$ and inj-allowed$(c, \mathsf{id}) = \mathrm{true}$. The simulator $\mathcal{S}_2$ still executes the actual protocol.

- Hybrid $\mathcal{H}_3$: now $\mathcal{F}_{\mathsf{CGKA}}$ uses the actual predicate safe*. The simulator $\mathcal{S}_3$ proceeds as before except that it only sets applications secrets when safe$^*(c) = \mathrm{false}$.

- Hybrid $\mathcal{H}_4$: Ideal world. The difference with the previous hybrid consists in also using the actual predicate inj-allowed. The simulator $\mathcal{S}_4$ is the same as $\mathcal{S}_3$.

Clearly the first two hybrids are indistinguishable. Let's argue that $\mathcal{H}_1$ and $\mathcal{H}_2$ are indistinguishable. Despite the fact that in MLS-Cutoff update proposals are created differently, we still frame the proposals in the same way and this guarantees consistency by the same argument as in the proof for MLS.

Concerning $\mathcal{H}_2$ and $\mathcal{H}_3$ the only change in the argument that has to be made corresponds to the fact that rekey-path is also used in proposals. Therefore the GSD graph also contains nodes and edges corresponding to these new keys just as is the case for commits in [AJM22].

The indistinguishability of hybrids $\mathcal{H}_3$ and $\mathcal{H}_4$ follows from the argument made in [AJM22]. Therefore it just remains to show that: if safe$(c) = \mathrm{true}$, then no node in the ratchet tree of the epoch that corresponds to c contains an exposed key pk. Here we make use of the fact that keys ending up in the ratchet tree after processing commits to update proposals were never encrypted under keys being concurrently updated. In the inductive step we have to consider an additional case, namely, that in which the node v for which v.pk has been exposed is set as part of proposal that is included in some commit c and the user who generated the commit is a user outsider v's subtree (in the ratchet tree T that corresponds to that commit). We denote by id_c the leaf of the party that generated the commit c.

It may be the case that the parent hash of v, pHash_v, cannot be verified by using the parent hash of its parent v.par in T as it would be the case when v.par is blanked. However, all nodes contained in the ratchet tree are parent hash valid, as defined in the MLS standard, which uses a slightly modified parent hash definition compared to ITK.

However the node at which the path of v to the root and the path of id_c to the root merge contains a commitment to the value of pHash_v (by definition of par-hash-cochild) which can be verified and this is done by verify-treeState. □

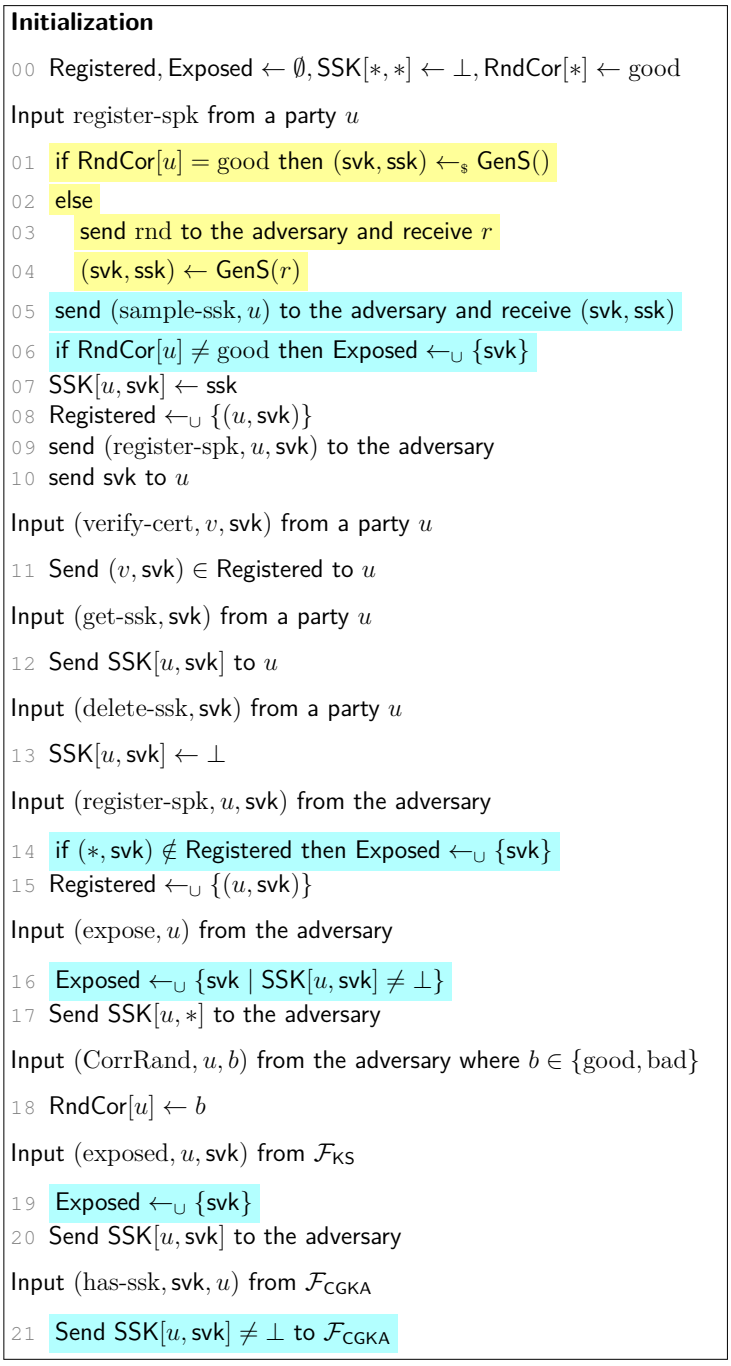

Figure B.1: Functionality $\mathcal{F}_{\mathsf{AS}}$ (parametrized by a key generation algorithm for a signature scheme GenS) and the ideal functionality $\mathcal{F}_{\mathsf{AS}}^{\mathsf{I}}$. Yellow lines (●) are only executed by $\mathcal{F}_{\mathsf{AS}}$ while cyan lines (●) are only executed by $\mathcal{F}_{\mathsf{AS}}^{\mathsf{I}}$.

Initialization

00 $\mathsf{SK}[*,*], \mathsf{SVK} \leftarrow \perp, \mathsf{RndCor}[*] \leftarrow \text{good}$

Input $(\text{register-kp}, \mathsf{svk}, \mathsf{ssk})$ from a party u

01 if $\mathsf{RndCor}[u] = \text{good}$ then $(\mathsf{kp}, \mathsf{sk}) \leftarrow_{\$}$ Gen-kp()
02 if $\mathsf{kp} \neq \perp$ then return
03 else
04 send rnd to the adversary and receive r
05 $(\mathsf{kp}, \mathsf{sk}) \leftarrow$ Gen-kp(r)
06 if $\mathsf{kp} \neq \perp$ then return
07 send $(\text{exposed}, u, \mathsf{svk})$ to $\mathcal{F}_{\mathsf{AS}}$
08 send ssk to the adversary
09 send $(\text{sample-sk}, u, \mathsf{svk}, \mathsf{ssk})$ to the adversary and receive $(\mathsf{kp}, \mathsf{sk}, ack)$
10 if $\neg ack$ then return
11 if $\mathsf{RndCor}[u] \neq \text{good}$ then
12 send $(\text{exposed}, u, \mathsf{svk})$ to $\mathcal{F}_{\mathsf{AS}}$
13 $\mathsf{SK}[u, \mathsf{kp}] \leftarrow \mathsf{sk}, \mathsf{SVK}[u, \mathsf{kp}] \leftarrow \mathsf{svk}$
14 send $(\text{register-kp}, u, \mathsf{svk}, \mathsf{kp})$ to the adversary
15 send kp to u

Input get-sks from a party u

16 Send $\{(\mathsf{kp}, \mathsf{SK}[u, \mathsf{kp}]) \mid \mathsf{SK}[u, \mathsf{kp}] \neq \perp\}$ to u

Input $(\text{fetch-kp}, v)$ from a party u

17 send $(\text{fetch-kp}, u, v)$ to the adversary and receive kp
18 sendkp to u

Input $(\text{delete-sk}, \mathsf{svk})$ from a party u

19 $\mathsf{SK}[u, \mathsf{kp}] \leftarrow \perp$

Input (expose, u) from the adversary

20 Send $\mathsf{SK}[u, *]$ to the adversary

Input $(\text{CorrRand}, u, b)$ from the adversary where $b \in \{\text{good}, \text{bad}\}$

21 $\mathsf{RndCor}[u] \leftarrow b$

Figure B.2: Functionality $\mathcal{F}_{\mathsf{KS}}$ (parametrized by a key generation algorithm for key packages Gen-kp) and the ideal functionality $\mathcal{F}_{\mathsf{KS}}^{\mathsf{I}}$. Yellow lines (•) are only executed by $\mathcal{F}_{\mathsf{KS}}$ while cyan lines (•) are only executed by $\mathcal{F}_{\mathsf{KS}}^{\mathsf{I}}$.

Initialization

00 $\mathsf{Ptr}[*], \mathsf{Node}[*], \mathsf{Prop}[*], \mathsf{Wel}[*] \leftarrow \perp$
01 $\mathsf{RndCor}[*] \leftarrow \mathrm{good}$
02 $\mathsf{HasKey}[*] \leftarrow \mathrm{false}$
03 $\mathsf{rootCtr} \leftarrow 0$

Input $(\mathrm{create}, \mathsf{svk})$ from u_{creator} \\ Used only once

04 req $\mathsf{Node}[root_0] = \perp$
05 req $\mathsf{valid\text{-}vsk}(u_{\mathsf{creator}}, \mathsf{svk})$
06 $\mathsf{orig} \leftarrow u_{\mathsf{creator}}$
07 $\mathsf{mem} \leftarrow \{(u_{\mathsf{creator}}, \mathsf{svk})\}$
08 $\mathsf{Node}[root_0] \leftarrow \mathsf{create\text{-}root}(\mathsf{orig}, \mathsf{mem}, \mathsf{RndCor}[\mathsf{orig}])$
09 $\mathsf{HasKey}[u_{\mathsf{creator}}] \leftarrow \mathrm{true}$
10 $\mathsf{Ptr}[u_{\mathsf{creator}}] \leftarrow root_0$

Figure B.3: Initialization of $\mathcal{F}_{\mathsf{CGKA}}$.

Input $(\text{propose}, \mathsf{act})$ from u

00 req $\mathsf{Ptr}[u] \neq \bot$
01 send $(\text{propose}, u, \mathsf{act})$ to the adversary and receive (p, svk_v, ack)
02 if $\neg$req-correct-prop(u, act) then req ack
03 if $\mathsf{act} = (\mathsf{upd}, \mathsf{svk})$ then assert valid-vsk(u, svk)
04 if $\mathsf{act} = (\mathsf{add}, v)$ then $\mathsf{act} \leftarrow (\mathsf{add}, v, \mathsf{svk}_v)$
05 if $\mathsf{Prop}[p] = \bot$ then
06 $\mathsf{Prop}[p] \leftarrow$ create-prop$(\mathsf{Ptr}[u], u, \mathsf{act}, \mathsf{RndCor}[u])$
07 else consistent-prop$(p, u, \mathsf{act}, \mathsf{RndCor}[u])$
08 if $\mathsf{RndCor}[u] = \text{bad}$ then send $(\text{exposed}, u, \mathsf{svk})$ to $\mathcal{F}_{\mathsf{AS}}$
09 return p

Input $(\text{commit}, P, \mathsf{svk})$ from u

10 req $\mathsf{Ptr}[u] \neq \bot$
11 send $(\text{commit}, u, P, \mathsf{svk})$ to the adversary and receive (c, w, ack, rt)
12 if $\neg$req-correct-commit(u, P, svk) then req ack
13 fill-props(u, P)
14 assert valid-vsk(u, svk)
15 $\mathsf{mem} \leftarrow$ members(c, u, P, svk)
16 assert $\mathsf{mem} \neq \bot \ \wedge (u, \mathsf{svk}) \in \mathsf{mem}$
17 if $\mathsf{Node}[c] = \bot \ \wedge \ rt = \bot$ then
18 $\mathsf{Node}[c] \leftarrow$ create-child$(\mathsf{Ptr}[u], u, P, \mathsf{mem}, \mathsf{RndCor}[u])$
19 else
20 if $\mathsf{Node}[c] = \bot$ then $c' \leftarrow root_{rt}$
21 else $c' \leftarrow c$
22 consistent-commit(c', u, P, mem)
23 if $c \neq c'$ then attach(c, c', u, P)
24 assert $w \neq \bot$ if and only if $\exists p \in P$: $\mathsf{Node}[p].\mathsf{act} = (\mathsf{add}, *)$
25 if $w \neq \bot$ then
26 assert $\mathsf{Wel}[w] \in \{\bot, c\}$
27 $\mathsf{Wel}[w] \leftarrow c$
28 assert cons-invariant $\wedge$ auth-invariant
29 if $\mathsf{RndCor}[u] = \text{bad}$ then send $(\text{exposed}, u, \mathsf{Node}[u].\mathsf{mem}[u])$ to $\mathcal{F}_{\mathsf{AS}}$
30 return (c, w)

Figure B.4: Proposals, commits and process of $\mathcal{F}_{\mathsf{CGKA}}$.

Input $(\text{process}, c, P)$ from u

00 send $(\text{process}, u, c, P)$ to the adversary and receive $(rt, \mathsf{orig}', \mathsf{svk}', ack)$
01 if $\neg$req-correct-proc(u, c, P) then req ack
02 fill-props(u, P)
03 if Node$[c] = \bot \ \wedge \ rt = \bot$ then
04 mem $\leftarrow$ members(Ptr$[u], \mathsf{orig}', P, \mathsf{svk}')$
05 assert mem $\neq \bot \ \wedge$ inj-allowed(Ptr$[u], u)$
06 else
07 if Node$[c] = \bot$ then $c' \leftarrow root_{rt}$
08 else $c' \leftarrow c$
09 $v \leftarrow$ Node$[c']$.orig
10 $\mathsf{svk}_v \leftarrow$ Node$[c']$.mem$[v]$
11 mem $\leftarrow$ members(Ptr$[u], v, P, \mathsf{svk}_v)$
12 assert mem $\neq \bot$
13 valid-succesor(c', u, P, mem)
14 if $c \neq c'$ then attach(c, c', u, P)
15 if $\exists p \in P$: Prop$[p]$.act $= (\mathsf{rem}, u)$ then Ptr$[u] \leftarrow \bot$
16 else
17 assert $u \in$ Node$[c]$.mem
18 Ptr$[u] \leftarrow c$
19 HasKey$[u] \leftarrow \text{true}$
20 assert cons-invariant $\wedge$ auth-invariant
21 $(*, \mathsf{PropSemantics}) \leftarrow$ apply-props$(c,$ Node$[c]$.pro$)$
22 return (Node$[c]$.orig, PropSemantics)

Input (join, w) from u

23 send (join, u, w) to the adversary and receive $(c', \mathsf{orig}', \mathsf{mem}', ack)$
24 req ack
25 $c \leftarrow$ Wel$[w]$
26 if $c = \bot$ then
27 if Node$[c'] \neq \bot$ then $c \leftarrow c'$
28 else
29 rootCtr $\leftarrow$ rootCtr $+ 1$
30 $c \leftarrow root_{\mathsf{rootCtr}}$
31 Node$[c] \leftarrow$ create-root$(\mathsf{orig}', \mathsf{mem}', \text{adv})$
32 Wel$[w] \leftarrow c$
33 Ptr$[u] \leftarrow c$
34 HasKey$[u] \leftarrow \text{true}$
35 assert $u \in$ Node$[c]$.mem $\wedge$ cons-invariant $\wedge$ auth-invariant
36 return (Node$[c]$.mem, Node$[c]$.orig)

Input key from u

37 req Ptr$[u] \neq \bot \ \wedge$ HasKey$[u]$
38 if Node$[u]$.key $= \bot$ then set-key(Ptr$[u]$)
39 HasKey$[u] \leftarrow \text{false}$
40 return Node[Ptr$[u]$].key

Figure B.5: Process, join and key in $\mathcal{F}_{\mathsf{CGKA}}$.

Input (expose, u) from the adversary

00 This input is not allowed if $\exists c$ such that Node$[c]$.chall $\wedge$ $\neg$safe(c)
01 if Ptr$[u] \neq \perp$ then
02 Node[Ptr$[u]$].exp $\leftarrow_\cup \{(u, \text{HasKey}[u])\}$
03 update-stat-after-exp(u)
04 send $(\text{exposed}, u, \text{Node}[u].\text{mem}[u])$ to $\mathcal{F}_{\text{AS}}$
05 send get-sks to $\mathcal{F}_{\text{KS}}$ and receive SK and SVK.
06 for each kp such that SK$[u, \text{kp}] \neq \perp \wedge$ SVK$[u, \text{kp}] = \text{svk}$ do
07 for each c such that $\exists p \in$ Node$[c]$.pro: Prop$[p]$.act $= (\text{add}, u, \text{svk})$ do
08 Node$[c]$.exp $\leftarrow_\cup \{(u, \text{true})\}$

Input $(\text{CorrRand}, u, b)$ from the adversary where $b \in \{\text{good}, \text{bad}\}$

09 RndCor$[u] \leftarrow b$

Figure B.6: Corruptions in $\mathcal{F}_{\text{CGKA}}$.

Helper create-root$(u, \text{mem}, \text{stat})$

00 return new node with par $\leftarrow \perp$, orig $\leftarrow u$, pro $\leftarrow \perp$, mem $\leftarrow$ mem, stat $\leftarrow$ stat

Helper create-prop$(c, u, \text{act}, \text{stat})$

01 return new proposal node with par $\leftarrow c$, orig $\leftarrow u$, act $\leftarrow$ act, stat $\leftarrow$ stat

Helper create-child$(c, u, P, \text{mem}, \text{stat})$

02 return new node with par $\leftarrow c$, orig $\leftarrow u$, pro $\leftarrow P$, mem $\leftarrow$ mem, stat $\leftarrow$ stat

Helper fill-props(u, P)

03 for $p \in P$ such that Prop$[p] = \perp$ do
04 send $(\text{propose}, p)$ to the adversary and receive (orig, act)
05 Prop$[p] \leftarrow$ create-prop(Ptr$[u]$, orig, act, adv)

Figure B.7: Helpers for creating new nodes in $\mathcal{F}_{\text{CGKA}}$.

Helper valid-vsk(u, svk')

00 $\mathsf{svk} \leftarrow \mathsf{Node}[\mathsf{Ptr}[u]].\mathsf{mem}[u]$
01 if $\mathsf{svk} \neq \bot \ \wedge \ \mathsf{svk} = \mathsf{svk}'$ then return true
02 send $(\text{has-ssk}, \mathsf{svk}', u)$ to $\mathcal{F}_{\mathsf{AS}}$ and receive ack
03 return ack

Helper set-key(c)

04 if $\neg\mathsf{safe}(c)$ then
05 send (key, u) to the adversary and receive k
06 $\mathsf{Node}[c].\mathrm{key} \leftarrow \mathsf{k}, \mathsf{Node}[c].\mathsf{chall} \leftarrow \mathrm{false}$
07 else $\mathsf{Node}[c].\mathrm{key} \leftarrow_{\$} \mathcal{K}, \mathsf{Node}[c].\mathsf{chall} \leftarrow \mathrm{true}$

Helper update-stat-after-exp(u)

08 for each p such that $\mathsf{Prop}[p] \neq \bot$ and
09 $\mathsf{Prop}[p].\mathsf{par} = \mathsf{Ptr}[u]$ and
10 $\mathsf{Prop}[p].\mathsf{orig} = u$ and
11 $\mathsf{Prop}[p].\mathsf{act} = \mathsf{upd}$
12 do $\mathsf{Prop}[p].\mathsf{stat} \leftarrow \mathrm{bad}$
13 for each c such that $\mathsf{Node}[c] \neq \bot$ and
14 $\mathsf{Node}[c].\mathsf{par} = \mathsf{Ptr}[u]$ and
15 $\mathsf{Node}[c].\mathsf{orig} = u$
16 do $\mathsf{Node}[c].\mathsf{stat} \leftarrow \mathrm{bad}$

Helper members(c, u, P, svk)

17 $(G, *) \leftarrow$ apply-props(c, u, P, svk)
18 if $(G, *) = \bot$ then return $\bot$
19 else return G

Helper apply-props(c, u, P, svk)

20 req $\mathsf{Node}[c] \neq \bot \ \wedge \ (u, *) \in \mathsf{Node}[c].\mathsf{mem}$
21 req $\forall p \in P$: $\mathsf{Prop}[p] \neq \bot \ \wedge \ \mathsf{Prop}[p].\mathsf{par} = c$
22 $G \leftarrow \mathsf{Node}[c].\mathsf{mem}$
23 $G \leftarrow G \setminus \{(u, *)\}$
24 $G \leftarrow_{\cup} \{(u, \mathsf{svk})\}$
25 $L \leftarrow \{u\}$
26 for each $p \in P$ such that $\mathsf{Prop}[p].\mathsf{act} = (\mathsf{upd}, *)$ do
27 $(v, (\mathsf{upd}, \mathsf{svk}')) \leftarrow (\mathsf{Prop}[p].\mathsf{orig}, \mathsf{Prop}[p].\mathsf{act})$
28 req $v \in G \setminus L$
29 $G \leftarrow G \setminus \{(v, *)\}$
30 $G \leftarrow_{\cup} \{(v, \mathsf{svk}')\}$
31 $L \leftarrow_{\cup} \{v\}$
32 for each $p \in P$ such that $\mathsf{Prop}[p].\mathsf{act} = (\mathsf{rem}, *)$ do
33 $(v, (\mathsf{rem}, v')) \leftarrow (\mathsf{Prop}[p].\mathsf{orig}, \mathsf{Prop}[p].\mathsf{act})$
34 req $v \in G \ \wedge \ v' \in G \setminus L$
35 $G \leftarrow G \setminus \{(v', *)\}$
36 for each $p \in P$ such that $\mathsf{Prop}[p].\mathsf{act} = (\mathsf{add}, *)$ do
37 $(v, (\mathsf{add}, v', \mathsf{svk}_{v'})) \leftarrow (\mathsf{Prop}[p].\mathsf{orig}, \mathsf{Prop}[p].\mathsf{act})$
38 req $v \in G \ \wedge \ v' \notin G$
39 $G \leftarrow_{\cup} \{(v', \mathsf{svk}_{v'})\}$
40 PropSemantics $\leftarrow ((\mathsf{Prop}[p].\mathsf{orig}, \mathsf{Prop}[p].\mathsf{act}) : p \in P)$
41 return $(G, \mathsf{PropSemantics})$

Figure B.8: Other helpers for $\mathcal{F}_{\mathsf{CGKA}}$.

Helper consistent-prop$(p, u, \mathsf{act}, \mathsf{stat})$

00 assert $\mathsf{Prop}[p].\mathsf{orig} = u \ \wedge \ \mathsf{Prop}[p].\mathsf{act} = \mathsf{act} \ \wedge \ \mathsf{Prop}[p].\mathsf{par} = \mathsf{Ptr}[u]$

Helper valid-succesor(c, u, P, mem)

01 $\mathsf{Node}[c] \neq \bot \wedge \mathsf{Node}[c].\mathsf{mem} = \mathsf{mem} \wedge \mathsf{Node}[c].\mathsf{pro} \in \{\bot, P\} \wedge \mathsf{Node}[c].\mathsf{par} \in \{\bot, \mathsf{Ptr}[u]\}$

Helper consistent-commit(c, u, P, mem)

02 assert valid-succesor(c, u, P, mem)
03 assert $\mathsf{RndCor}[u] \neq \mathrm{good} \ \wedge \ \mathsf{Node}[c].\mathsf{orig} = u$

Helper attach(c, c', u, P)

04 assert $c' \neq root_0$
05 $\mathsf{Node}[c'].\mathsf{par} \leftarrow \mathsf{Ptr}[u], \mathsf{Node}[c'].\mathsf{pro} \leftarrow P, \mathsf{Node}[c] \leftarrow \mathsf{Node}[c'], \mathsf{Node}[c'] \leftarrow \bot$
06 for each w such that $\mathsf{Wel}[w] = c'$ do $\mathsf{Wel}[c] \leftarrow c$

Figure B.9: Consistency helpers for $\mathcal{F}_{\mathsf{CGKA}}$.

```
Helper req-correct-prop(u, act)

00 if act = (rem, v) then
01    return v ∈ Node[Ptr[u]].mem
02 else if act = (upd, svk) then
03    return valid-vsk(u, svk)
04 else return false

Helper req-correct-commit(u, P, svk)

05 return apply-props(u, P, svk) ≠ ⊥ ∧ valid-vsk(u, svk)

Helper req-correct-proc(u, c, P)

06 return Node[c] ≠ ⊥ ∧ Node[c].par = Ptr[u] ∧ Node[c].pro = P ∧
07    Node[c].stat ≠ adv ∧ ∀p ∈ P: Prop[p].stat ≠ adv
```

Figure B.10: Correctness helpers for $\mathcal{F}_{\mathsf{CGKA}}$.

```
Helper auth-invariant

00 return true iff
01    ∀c if Node[c].stat = adv then inj-allowed(Node[c].par, Node[c].orig) and
02    ∀p if Prop[p].stat = adv then inj-allowed(Prop[p].par, Prop[p].orig)

Helper cons-invariant

03 return true iff
04    ∀c such that Node[c].par ≠ ⊥ it holds that
05       Node[c].pro ≠ ⊥ and ∀p ∈ Node[c].pro: Prop[p].par = Node[c].par
06    and ∀u such that Ptr[u] ≠ ⊥: u ∈ Node[Ptr[u]].mem
07    and the history graph contains no cycles
```

Figure B.11: Invariant helpers for $\mathcal{F}_{\mathsf{CGKA}}$.

safe(c)

00 safe(c) $=$ true if and only if $\neg((*, \text{true}) \in$ Node[c].exp $\vee$ can-traverse(c))

inj-allowed(c, u)

01 inj-allowed(c, u) $=$ true if and only if Node[c].exp $\in$ Exposed $\wedge$ know(c, 'ep')

know(c, u) $=$ true if and only if

02 1) state-directly-leaks(c, u) $\vee$
03 2) (Node[c].par $\neq \perp$ $\wedge$ $\neg$secrets-replaced(c, u) $\wedge$ know(Node[c].par, u)) $\vee$
04 3) $\exists c'$: (Node[c'].par $= c$ $\wedge$ $\neg$secrets-replaced(c', u) $\wedge$ know(c', u))

state-directly-leaks(c, u) $=$ true if and only if

05 1) $(u, *) \in$ Node[c].exp $\vee$
06 2) $\exists$rootCtr: ($root_{\text{rootCtr}}$ is ancestor of c $\wedge$ $\exists$svk $\in$ Exposed: $(u, \text{svk}) \in$ Node[c].mem) $\vee$
07 3) $(u, *) \in$ Node[c].mem $\wedge$ secrets-injected(c, u)

secrets-injected(c, u) $=$ true if and only if

08 1) (Node[c].orig $= u$ $\wedge$ Node[c].stat $\neq$ good) $\vee$
09 2) $\exists p \in$ Node[c].pro: (Prop[p].act $= (\text{add}, u, \text{svk})$ $\wedge$ svk $\in$ Exposed) $\vee$
10 3) $\exists p \in$ Node[c].pro: (Prop[p].act $= (\text{upd}, *) \wedge$ Prop[p].orig $= u \wedge$ Prop[p].stat $\neq$ good)

secrets-replaced(c, u) $=$ true if and only if

11 Node[c].orig $= u$ $\vee$ ($\exists p \in$ Node[c].pro:
12 Prop[p].act $\in \{(\text{add}, u), (\text{rem}, u)\}$ $\vee$ (Prop[p].act $= (\text{upd}, *)$ $\wedge$ Prop[p].orig $= u$))

know(c, 'ep') $=$ true if and only if

13 Node[c].exp $\neq \emptyset$ $\vee$ can-traverse(c)

can-traverse(c) $=$ true if and only if

14 1) (Node[c].par $= \perp$ $\wedge$ $\exists$svk $\in$ Exposed: $(*, \text{svk}) \in$ Node[c].mem) $\vee$
15 2) $\exists p \in$ Node[c].pro: (Prop[p].act $= (\text{add}, u, \text{svk})$ $\wedge$ svk $\in$ Exposed) $\vee$
16 3) leaf-welcome-key-reuse(c) $\vee$
17 4) (know($c, *$) $\wedge$ ($c = root_*$ $\vee$ know(Node[c].par, 'ep')))

leaf-welcome-key-reuse(c) $=$ true if and only if

18 $\exists u$ $\exists p \in$ Node[c].pro:
19 1) Prop[p].act $= (\text{add}, u)$ $\wedge$
20 2) $\exists c'$: c is ancestor of c' $\wedge$ $(u, *) \in$ Node[c'].exp $\wedge$
21 ($\nexists c''$ in the path between c and c' such that secrets-replaced(c'', u))

Figure B.12: Predicates safe(c) and inj-allowed(c, u).

Helper fetch-ssk-if-nec(st, svk)

00 if $v_{\mathsf{st.id}}$.svk $\neq$ svk then
01 send $(\text{get-ssk}, \mathsf{svk})$ to $\mathcal{F}_{\mathsf{AS}}$ and receive ssk
02 else ssk $\leftarrow$ st.ssk
03 return ssk

Helper validate-kp(st, kp, id, pHash)

04 parse $(\mathsf{id}', \mathsf{pk}, \mathsf{svk}, \mathsf{pHash}', \sigma) \leftarrow \mathsf{kp}$
05 req $\mathsf{id} = \mathsf{id}' \ \wedge \ \mathsf{pHash} = \mathsf{pHash}'$
06 if svk $\notin$ st.cert-SPks[id] then
07 send $(\text{verify-cert}, \mathsf{id}, \mathsf{svk})$ to $\mathcal{F}_{\mathsf{AS}}$ and receive b
08 req b
09 st.cert-SPks[id] $\leftarrow_{\cup} \{\mathsf{svk}\}$
10 req $\mathsf{Vrfy}(\mathsf{svk}, \sigma, (\mathsf{id}', \mathsf{pk}, \mathsf{svk}, \mathsf{pHash}', \sigma))$
11 return st

Helper Gen-kp(id, svk, ssk)

12 $(\mathsf{pk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{PKE.Gen}()$
13 $\sigma \leftarrow \mathsf{Sign}(\mathsf{ssk}, (\mathsf{id}, \mathsf{pk}, \mathsf{svk}, \mathsf{pHash}))$
14 $\mathsf{kp} \leftarrow (\mathsf{id}, \mathsf{pk}, \mathsf{svk}, \mathsf{pHash}, \sigma)$
15 return $(\mathsf{kp}, \mathsf{sk})$

Figure B.13: Helper functions for the interaction with KS and AS.

Algorithm CGKA.Create(svk)

00 require $\mathsf{st} = \bot \wedge u = u_{\mathsf{creator}}$
01 $\mathsf{st.T} \leftarrow$ init-tree
02 $\mathsf{st.groupID} \leftarrow_{\$} \{0,1\}^{\kappa}$
03 $\mathsf{st.appSec} \leftarrow_{\$} \{0,1\}^{\kappa}$
04 $\mathsf{st.ep} \leftarrow 0$
05 $\mathsf{st.interim\text{-}tranHash} \leftarrow \varepsilon$
06 try $\mathsf{st.ssk} \leftarrow$ fetch-ssk-if-nec(st, svk)
07 $(\mathsf{kp}, \mathsf{sk}) \leftarrow$ Gen-kp(svk, ssk)
08 $\mathsf{id}(\mathsf{leaves}(\mathsf{T})) \leftarrow \mathsf{H}(\mathsf{pk})$
09 assign-kp($\mathsf{T}, v_{\mathsf{id}}, \mathsf{kp}$)
10 $v_{\mathsf{id}}.\mathsf{sk} \leftarrow \mathsf{sk}$

Algorithm CGKA.Com(st, PMSG)

11 $\mathsf{st}' \leftarrow$ init-epoch(st)
12 $\mathsf{T}' \leftarrow \mathsf{st}'.\mathsf{T}$
13 try $(\mathsf{st}', \mathrm{upd}, \mathrm{rem}, \mathrm{add}) \leftarrow$ apply-props(st, PMSG)
14 require $(\mathsf{st.id}, \cdot, \cdot) \notin \mathrm{rem} \wedge (\mathsf{st.id}, \cdot, \cdot) \notin \mathrm{upd}$
15 try $(\mathsf{st}', \mathsf{comSec}, \mathsf{UpdPath}, \mathsf{pathSec})$
$\leftarrow$ rekey-path($\mathsf{st}', \mathsf{st.id}, v_{\mathsf{id}}.\mathsf{svk}, 0$)
16 propIds $\leftarrow ()$
17 for pmsg $\in$ PMSG:
18 propIds $\leftarrow_{\cup} \mathsf{H}(\mathsf{pmsg})$
19 $C \leftarrow (\mathsf{propIds}, \mathsf{UpdPath})$
20 $\sigma \leftarrow$ sign-commit(st, C)
21 $\mathsf{st}' \leftarrow$ set-conf-trans-hash(st, $\mathsf{st}', \mathsf{st.id}, C, \sigma$)
22 $(\mathsf{st}', \mathsf{confKey}, \mathsf{joinerSec}) \leftarrow$ derive-keys(st, $\mathsf{st}', \mathsf{comSec}$)
23 confTag $\leftarrow$ compute-confTag($\mathsf{st}', \mathsf{confKey}$)
24 if $\mathrm{rem} \neq ()$
25 memTag $\leftarrow$ MAC.Tag(st.memKey, C)
26 else memTag $\leftarrow \bot$
27 cmsg $\leftarrow$ frame-comm(st, confTag, σ, memTag)
28 if $\mathrm{add} \neq ()$
29 $(\mathsf{st}', \mathsf{wmsg}) \leftarrow$ comp-wmsg(st',
, add, joinerSec, pathSec, confTag)
30 else wmsg $\leftarrow \bot$
31 $\mathsf{st}' \leftarrow$ set-int-transHash(st', confTag)
32 $\mathsf{st.pendComm}(\mathsf{cmsg}) \leftarrow (\mathsf{st}', \mathsf{PMSG}, \mathrm{upd}, \mathrm{rem}, \mathrm{add})$
33 return (cmsg, wmsg)

Algorithm CGKA.Key

34 $(k, \mathsf{st.appSec}) \leftarrow (\mathsf{st.appSec}, \bot)$
35 return k

Algorithm CGKA.Proc(cmsg, PMSG)

36 $(\mathsf{id\text{-}snd}, C, \mathsf{confTag}, \sigma, \mathsf{memTag}) \leftarrow$ unframe-comm(st, cmsg)
37 if id-com $=$ st.id
38 $(\mathsf{st}', \mathsf{PMSG}', \mathrm{upd}, \mathrm{rem}, \mathrm{add}) \leftarrow \mathsf{st.pendComm}(\mathsf{cmsg})$
39 require $\mathsf{PMSG}' = \mathsf{PMSG}$
40 $\mathsf{st} \leftarrow \mathsf{st}'$
41 return
42 parse $(\mathsf{propIds}, \mathsf{UpdPath}) \leftarrow C$
43 require $\mathsf{propIds}(i) = \mathsf{H}(\mathsf{PMSG}(i))$ for all i
44 $\mathsf{st}' \leftarrow$ init-epoch(st)
45 try $(\mathsf{st}', \mathrm{upd}, \mathrm{rem}, \mathrm{add}) \leftarrow$ apply-props(st, PMSG)
46 require $\mathsf{id\text{-}com} \notin \mathrm{rem} \wedge \mathsf{id\text{-}com} \notin \mathrm{upd}$
47 if $\mathsf{st.id} \in \mathrm{rem}$
48 require MAC.Ver(st.memKey, memTag)
49 $\mathsf{st} \leftarrow \bot$
50 else
51 $(\mathsf{st}', \mathsf{comSec}) \leftarrow$ apply-rekey(st,
$\mathsf{st}', \mathsf{id\text{-}com}, \mathsf{UpdPath}, 0, \mathtt{False}$)
52 $\mathsf{st}' \leftarrow$ set-conf-trans-hash($\mathsf{st}, \mathsf{st}', \mathsf{id\text{-}com}, C, \sigma$)
53 $(\mathsf{st}', \mathsf{confKey}, \mathsf{joinerSec}) \leftarrow$ derive-keys($\mathsf{st}, \mathsf{st}', \mathsf{comSec}$)
54 require verify-confTag($\mathsf{st}', \mathsf{confKey}, \mathsf{confTag}$)
55 $\mathsf{st}' \leftarrow$ set-int-transHash(st', confTag)
56 $\mathsf{st} \leftarrow \mathsf{st}'$
57 return $((\mathrm{upd}, \mathrm{rem}, \mathrm{add}), \mathsf{id\text{-}snd})$

Algorithm CGKA.Join(wmsg)

58 parse $(\mathsf{encGrpSec}, \mathsf{GrpInfo}) \leftarrow \mathsf{wmsg}$
59 parse $(\mathsf{GrpInfoTBS}, \sigma) \leftarrow \mathsf{GrpInfo}$
60 parse $(\mathsf{st.groupID}, \mathsf{st.ep}, \mathsf{st.tHash}, \mathsf{st.conf\text{-}TransHash},$
$\mathsf{st.interim\text{-}tranHash}, \mathsf{st.confTag}, \mathsf{id\text{-}com}) \leftarrow \mathsf{GrpInfoTBS}$
61 require $\mathsf{SIG.Vrfy}(v_{\mathsf{id\text{-}com}}.\mathsf{svk}, \sigma, \mathsf{GrpInfoTBS})$
62 try verify-treeState(st)
63 try fetch-ssk-if-nec(st, $v_{\mathsf{st.id}}.\mathsf{svk}$)
64 kbs $\leftarrow$ get-sks
65 $(\mathsf{joinerSec}, \mathsf{pathSec}) \leftarrow (\bot, \bot)$
66 for $e \in \mathsf{encGrpSec}$
67 parse $(h, c) \leftarrow e$
68 for $(\mathsf{kp}, \mathsf{sk}) \in \mathsf{kbs}$
69 if $h = \mathsf{H}(\mathsf{kp})$
70 $v.\mathsf{sk} \leftarrow \mathsf{sk}$
71 require $v_{\mathsf{st.id}}.\mathsf{kp} = \mathsf{kp}$
72 parse $(\mathsf{joinerSec}, \mathsf{pathSec}) \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, c)$
73 if $s_j = \mathsf{pathSec} \neq \bot$
74 $a \leftarrow \mathsf{st.T.ind\text{-}lca}(v_{\mathsf{st.id}}, v_{\mathsf{id\text{-}com}})$
75 $(v_1, \ldots, v_\ell) \leftarrow$ fil-path(v)
76 for $i \in \{\ell - a, \ldots, \ell - 1\}$
77 $(\mathsf{pk}, v_i.\mathsf{sk}) \leftarrow \mathsf{PKE.Gen}(\mathsf{H}_2(s_i))$
78 require $\mathsf{pk} = v_i.\mathsf{pk}$
79 $s_{i+1} \leftarrow \mathsf{H}_1(s_i)$
80 $(\mathsf{st}, \mathsf{confKey}) \leftarrow$ derive-epochKeys(st, joinerSec)
81 require verify-confTag(st, confKey, confTag)
82 return $(\mathsf{roster}(\mathsf{T}), \mathsf{st.id})$

Figure B.14: Algorithms CGKA.Create, CGKA.Com, CGKA.Proc, CGKA.Join, CGKA.Key shared between MLS and MLS-Cutoff. For subroutines rekey-path, apply-props, comp-wmsg, see Figure B.17, B.18, B.13, respectively.

Algorithm CGKA.Prop(op, ad) (used in MLS)

```
00 T ← st.T
01 if op = 'upd':
02    try st.ssk ← fetch-ssk-if-nec(st, svk)
03
04    (kp, sk) ← Gen-kp(svk, ssk)
05    pmsg ← frame-prop(st, ('upd', kp))
06    st.pendUpd(pmsg) ← (ssk, sk)
07 if op = ('add'):
08    parse u ← ad
09    require u ∉ roster(T)
10    send (fetch-kp, u) to F_KS and get kp_a
11    try validate-kp(kp_a)
12    pmsg ← frame-prop(st, ('add', kp_a))
13 if op = 'rem':
14    parse u ← ad
15    require u ∈ roster(T)
16    pmsg ← frame-prop(st, ('rem', id-rem))
17 return pmsg
```

Algorithm CGKA.Prop(op, ad) (used in MLS-Cutoff)

```
18 T ← st.T
19 if op = 'upd':
20    try st.ssk ← fetch-ssk-if-nec(st, svk)
21    st' ← init-epoch(st)
22    (id-own, svk) ← (st'.id-own, v_id-own.svk)
23    try (st', ·, UpdPath, pathSec) ← rekey-path(st',
                 id-own, svk, st.i_cut)
24    pmsg ← frame-prop(st, ('upd', UpdPath))
25    st.pendUpd(pmsg) ← (ssk, v'_id-own.sk, pathSec)
26 if op = ('add'):
27    parse u ← ad
28    require u ∉ roster(T)
29    send (fetch-kp, u) to F_KS and get kp_a
30    try validate-kp(kp_a)
31    pmsg ← frame-prop(st, ('add', kp_a))
32 if op = 'rem':
33    parse u ← ad
34    require u ∈ roster(T)
35    pmsg ← frame-prop(st, ('rem', id-rem))
36 return pmsg
```

Figure B.15: Generating proposals in MLS (left) and protocol variant MLS-Cutoff (right). Differences between the protocols are marked in yellow (●) and cyan (●). In line 25 $v'_{\text{id-own}}$ is to be understood as the issuing user's leaf with respect to st′.

Algorithm set-tHash(st′)

```
00 T′ ← st′.T
01 st′.tHash ← tree-hash(T′, root(T′))
```

Algorithm tree-hash(T′, v)

```
02 if v is leaf
03    return H(v.index, v.kp)
04 else
05    lHash ← tree-hash(T′, left(v))
06    rHash ← tree-hash(T′, right(v))
07    return H(v.index, v.pk,
              v.unm, v.pHash, lHash, rHash)
```

Algorithm set-pHash(st′, id)

```
08 T′ ← st.T
09 v_0 ← v_id
10 parse (v_1, ..., v_ℓ) ← fil-path(v_0)
11 v_ℓ.pHash ← ε
12 for j = ℓ − 1, ..., 0
13    v_j.pHash ← par-hash-cochild(v_{j+1}, sib(v_j))
14 return st′
```

Algorithm par-hash-cochild(T′, v, u)

```
15 oriRes ← Res(u) ∪ (u.unm \ v.unm)
16 copy T′ to T″
17 for v″ ∈ v.unm
18       blank-path(T″, v″)
19       for w ∈ path(v″)
20          w.unm ← w.unm \ {v″}
21 origSibTreeHash ← tree-hash(T″, u)
22 return H(v.pk, v.pHash, origSibTreeHash)
```

Algorithm verify-treeState(st′)

```
23 T′ ← st′.T
24 require st′.tHash = tree-hash(T′, root(T′))
25 for v ∈ T′.V
26    if v not blank ∧ v not leaf
27       l ← par-hash-cochild(v, left(v))
28       r ← par-hash-cochild(v, right(v))
29       require (left(v) not blank ∧ left(v).pHash = r) ∨ ...
                 ... ∨ (right(v) not blank ∧ right(v).pHash = l)
30 M ← ()
31 for v ∈ T′.V
32    if v not blank ∧ v is leaf
33       require id(v) ∉ M
34       M ←∪ {id(V)}
35       try st′ ← validate-kp(st′, v.kp, id(v), v.pHash)
36 return st′
```

Figure B.16: Helper functions set-tHash, set-pHash, and verify-treeState, used to authenticate the ratchet tree's state.

Algorithm rekey-path($\mathsf{st}', \mathsf{id}, \mathsf{svk}, i$)

$\mathsf{T}' \leftarrow \mathsf{st}'.\mathsf{T}$
$\mathsf{UpdPathNodes} \leftarrow ()$
$\mathsf{pathSec} \leftarrow ()$
$(v_1, \ldots, v_\ell) \leftarrow \mathsf{fil\text{-}path}(v_{\mathsf{id}})$
$\mathrm{s}_0 \leftarrow_{\$} \{0,1\}^\lambda$
for $j = 1, \ldots, \ell - i$:
  $\mathrm{s}_j \leftarrow \mathsf{H}_1(\mathrm{s}_{j-1})$
  if $j < \ell$:
    $(v_j.\mathsf{pk}, v_j.\mathsf{sk}_j) \leftarrow \mathsf{Gen}(\mathsf{H}_2(\mathrm{s}_j))$
  $C \leftarrow ()$
  $v_{\mathsf{sib}} \leftarrow \mathsf{sib}(v_{j-1})$
  for $w \in \mathsf{Res}(v_{\mathsf{sib}}) \cup v_{\mathsf{sib}}.\mathsf{unm}$:
    $C \leftarrow_\cup c_w := \mathsf{PKE.Enc}(w.\mathsf{pk}, \mathrm{s}_j)$
  $\mathsf{UpdPathNodes} \leftarrow_\cup (v_j.\mathsf{pk}, C)$
  $\mathsf{pathSec} \leftarrow_\cup (s_j)$
if $i = 0$
  $\mathsf{comSec} \leftarrow s_\ell$
else $\mathsf{comSec} = \bot$
$\mathsf{merge\text{-}leaves}(\mathsf{T}', v_{\mathsf{id}})$
$\mathsf{set\text{-}pHash}(\mathsf{T}', v_{\mathsf{id}})$
try $\mathsf{fetch\text{-}ssk\text{-}if\text{-}nec}(\mathsf{T}', \mathsf{svk})$
$(\mathsf{kp}, \mathsf{sk}) \leftarrow \mathsf{Gen\text{-}kp}(\mathsf{id}, \mathsf{svk}, \mathsf{ssk}, v.\mathsf{pHash}; s_0)$
require $v_{\mathsf{H(kp)}} = \bot$
$\mathsf{id}(v_{\mathsf{id}}) \leftarrow \mathsf{H}(\mathsf{kp})$
$\mathsf{assign\text{-}kp}(\mathsf{T}', \mathsf{H}(\mathsf{kp}), \mathsf{kp})$
$v_{\mathsf{id}}.\mathsf{sk} \leftarrow \mathsf{sk}$
$\mathsf{set\text{-}tHash}(\mathsf{T}')$
$\mathsf{UpdPath} \leftarrow (\mathsf{UpdPathNodes}, \mathsf{kp})$
return $(\mathsf{T}', \mathsf{comSec}, \mathsf{UpdPath}, \mathsf{pathSec})$

Algorithm comp-wmsg($\mathsf{st}', \mathrm{add}, \mathsf{joinerSec}, \mathsf{pathSec}, \mathsf{confTag}$)

$\mathsf{tbs} \leftarrow (\mathsf{st}'.\mathsf{groupID}, \mathsf{st}'.\mathsf{ep}, \mathsf{st}'.\mathsf{tHash},$
  $\mathsf{st.conf\text{-}TransHash}, \mathsf{st}'.\mathsf{interim\text{-}tranHash},$
  $\mathsf{st}'.\mathsf{T}.\mathsf{public}, \mathsf{st}'.\mathsf{confTag}, \mathsf{st}'.\mathsf{id})$
$\sigma \leftarrow \mathsf{Sign}(\mathsf{st}'.\mathsf{ssk}, \mathsf{tbs})$
$\mathsf{GrpInfo} \leftarrow (\mathsf{tbs}, \sigma))$
$\mathsf{encGrpSec} \leftarrow ()$
for $(\cdot, \mathsf{id\text{-}add}, \mathsf{svk}) \in \mathrm{add}$
  $a \leftarrow \mathsf{ind\text{-}lca}(v_{\mathsf{st}'.\mathsf{id}}, v_{\mathsf{id\text{-}add}})$
  $c \leftarrow \mathsf{PKE.Enc}(v_{\mathsf{id\text{-}add}}.\mathsf{pk}, (\mathsf{joinerSec}, \mathsf{pathSec}(\ell - a)))$
  $\mathsf{encGrpSec} \leftarrow_\cup (\mathsf{H}(v_{\mathsf{id\text{-}add}}.\mathsf{kp}), c)$
return $(\mathsf{st}', \mathsf{wmsg})$

Algorithm apply-rekey($\mathsf{st}, \mathsf{st}', \mathsf{id\text{-}snd}, \mathsf{UpdPath}, i, \mathrm{copySt}$)

$\mathsf{T}' \leftarrow \mathsf{st}'.\mathsf{T}$
parse $(\mathsf{UpdPathNodes}, \mathsf{kp}) \leftarrow \mathsf{UpdPath}$
$(v_1, \ldots, v_\ell) \leftarrow \mathsf{fil\text{-}path}(\mathsf{T}', v_{\mathsf{st.id}})$
$(v'_1, \ldots, v'_{\ell'}) \leftarrow \mathsf{fil\text{-}path}(\mathsf{T}', v_{\mathsf{id\text{-}snd}})$
parse $((\mathsf{pk}_1, C_1), \ldots, (\mathsf{pk}_m, C_m)) \leftarrow \mathsf{UpdPathNodes}$
for $i \in \{1, \ldots, m\}$
  $v'_i.\mathsf{pk} \leftarrow \mathsf{pk}_i$
$a \leftarrow \mathsf{ind\text{-}lca}(v_{\mathsf{st.id}}, v_{\mathsf{id\text{-}snd}})$
if $i > a$:
  $\mathsf{comSec} \leftarrow \bot$
else
  parse $(c_w)_w \leftarrow C_{\ell - a}$
  if $v'_0 \in v'_{\ell' - a - 1}.\mathsf{unm}$:
    $w \leftarrow v'_0$
  else:
    $m \leftarrow \max(m' \in \{0, \ldots, \ell' - a\} \mid v'_{m'} \text{ not blank})$
    $w \leftarrow v'_m$
  $s_{\ell - a} \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_w, c_w)$
  for $j = \ell' - a + 1, \ldots, \ell' - i$:
    $\mathrm{s}_j \leftarrow \mathsf{H}_1(\mathrm{s}_{j-1})$
    if $j < \ell$
      $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathsf{Gen}(\mathsf{H}_2(\mathrm{s}_j))$
      $v'_j.\mathsf{sk} \leftarrow \mathsf{sk}_j$
      require $v'_j.\mathsf{pk} = \mathsf{pk}_j$
if $i = 0$
  $\mathsf{comSec} \leftarrow s_\ell$
else $\mathsf{comSec} = \bot$
$\mathsf{merge\text{-}leaves}(\mathsf{T}', v_{\mathsf{id}})$
$\mathsf{st}' \leftarrow \mathsf{set\text{-}pHash}(\mathsf{T}', v_{\mathsf{id}})$
if $\mathrm{copySt} = \texttt{False}$
  try $\mathsf{validate\text{-}kp}(\mathsf{st}'', \mathsf{kp}, \mathsf{id\text{-}snd}, \mathsf{st}''.\mathsf{pHash})$
else
  $\mathsf{st}'' \leftarrow \mathsf{init\text{-}epoch}(\mathsf{st})$
  $\mathsf{T}'' \leftarrow \mathsf{st}''.\mathsf{T}$
  $(v''_1, \ldots, v''_{\ell'}) \leftarrow \mathsf{fil\text{-}path}(\mathsf{T}'', v_{\mathsf{id\text{-}snd}})$
  for $i \in \{1, \ldots, m\}$
    $v''_i.\mathsf{pk} \leftarrow \mathsf{pk}_i$
  $\mathsf{merge\text{-}leaves}(\mathsf{T}'', v_{\mathsf{id}})$
  $\mathsf{st}'' \leftarrow \mathsf{set\text{-}pHash}(\mathsf{T}'', v_{\mathsf{id}})$
  try $\mathsf{validate\text{-}kp}(\mathsf{st}'', \mathsf{kp}, \mathsf{id\text{-}snd}, \mathsf{st}''.\mathsf{pHash})$
require $v_{\mathsf{H(kp)}} = \bot$
$\mathsf{id}(v_{\mathsf{id\text{-}snd}}) \leftarrow \mathsf{H}(\mathsf{kp})$
$\mathsf{assign\text{-}kp}(\mathsf{T}', \mathsf{H}(\mathsf{kp}), \mathsf{kp})$
$\mathsf{set\text{-}tHash}(\mathsf{T}')$
return $(\mathsf{st}', \mathsf{comSec})$

Figure B.17: Helper functions rekey-path and apply-rekey used to rekey users' paths.

Algorithm apply-props(st, st′, PMSG) (used in MLS)

```
00 (upd, rem, add) ← ((), (), ())
01 fetch T′ ← st′.T
02 for pmsg ∈ PMSG:
03   ((‘type’, ad), id-snd) ← unframe-prop(st, pmsg)
04   if ‘type’ = ‘upd’:
05     require (id-snd, ·) ∉ upd ∧ rem = () ∧ add = ()
06     try validate-kp(st′, ad, id-snd, ε)
07     assign-kp(T′, v_id-snd, ad)
08     if st′.id = id-snd
09       parse (ssk, sk) ← pendUpd(pmsg)
10       v_id-snd.sk ← sk
11       st′.ssk ← ssk
12     blank-path(T′, v_id-snd, 1)
13     svk ← leaf(T′, id-snd).svk
14     require v_H(kp) = ⊥
15     id(v_id-snd) ← H(kp)
16     upd ←∪ (id-snd, ‘upd’, svk)
17   else if ‘type’ = ‘rem’:
18     require id-rem := ad ≠ id-snd ∧ v_id-rem ≠ ⊥
19     require (ad, ·) ∉ upd ∧ add = ()
20     blank-leaf(T′, v_id-rem)
21     blank-path(T′, v_id-rem, 1)
22     T′ ← truncate(T′)
23     rem ←∪ (id-snd, ‘rem’, id-rem)
24   else if ‘type’ = ‘add’:
25     try st′ ← validate-kp(st′, ad, u_new, ε)
26     id-add ← H(ad)
27     add-leaf(T′, id-add)
28     set-as-unmerged(T′, id-add)
29     add ←∪ (id-snd, ‘add’, v_id-add.svk)
30   else return ⊥
31 return (st′, upd, rem, add)
```

Algorithm apply-props(st, st′, PMSG) (used in MLS-Cutoff)

```
32 (upd, rem, add) ← ((), (), ())
33 fetch T′ ← st′.T
34 for pmsg ∈ PMSG:
35   ((‘type’, ad), id-snd) ← unframe-prop(st, pmsg)
36   if ‘type’ = ‘upd’:
37     require (id-snd, ·) ∉ upd ∧ rem = () ∧ add = ()
38     if st′.id ≠ id-snd
39       (st′, ·) ← apply-rekey(st, st′, id-snd, ad, st.i_cut, true)
40     if st′.id = id-snd
41       parse ((pk_1, ·), ..., (pk_ℓ−i_cut, ·), kp) ← ad
42       assign-kp(T′, v_id-snd, kp)
43       parse (ssk, sk, pathSec) ← pendUpd(pmsg)
44       v_id-snd.sk ← sk
45       st′.ssk ← ssk
46       parse (v_1, ..., v_ℓ) ← path(id-snd)
47       for j = 1, ..., ℓ − st.i_cut
48         (v_j.pk, v_j.sk) ← (pk_j, pathSec(j))
49     blank-path(T′, v_id-snd, ℓ − st.i_cut + 1)
50     parse (v_1, ..., v_ℓ) ← path(v_id-snd)
51     for j = 1, ..., ℓ − st.i_cut
52       if v_j ∈ V_col
53         blank(v_j)
54       V_col ←∪ {v_j}
55     svk ← leaf(T′, id-snd).svk
56     require v_H(kp) = ⊥
57     id(v_id-snd) ← H(kp)
58     upd ←∪ (id-snd, ‘upd’, svk)
59   else if ‘type’ = ‘rem’:
60     require id-rem := ad ≠ id-snd ∧ v_id-rem ≠ ⊥
61     require (ad, ·) ∉ upd ∧ add = ()
62     blank-leaf(T′, v_id-rem)
63     blank-path(T′, v_id-rem, 1)
64     T′ ← truncate(T′)
65     rem ←∪ (id-snd, ‘rem’, id-rem)
66   else if ‘type’ = ‘add’:
67     try st′ ← validate-kp(st′, ad, u_new, ε)
68     id-add ← H(ad)
69     add-leaf(T′, id-add)
70     set-as-unmerged(T′, id-add)
71     add ←∪ (id-snd, ‘add’, v_id-add.svk)
72   else return ⊥
73 return (st′, upd, rem, add)
```

Figure B.18: Helper function apply-props for MLS (left) and MLS-Cutoff (right). Differences between the protocols are marked in yellow (•) and cyan (•).

APPENDIX C

Declaration of the use of Generative AI Tools.

In this thesis, generative AI tools, specifically Gemini Flash 3.5, have been used to detect typos and rephrase some paragraphs in order to improve clarity. The author accepts full responsibility for all content submitted.