On the Price of Concurrency in Group Ratcheting Protocols*

Alexander Bienstock[1], Yevgeniy Dodis[1], and Paul Rösler[2]

[1] New York University
{abienstock,dodis}@cs.nyu.edu
[2] Chair for Network and Data Security, Ruhr University Bochum
paul.roesler@rub.de

Abstract. *Post-Compromise Security*, or PCS, refers to the ability of a given protocol to recover—by means of normal protocol operations—from the exposure of local states of its (otherwise honest) participants. While PCS in the two-party setting has attracted a lot of attention recently, the problem of achieving PCS in the group setting—called *group ratcheting* here—is much less understood. On the one hand, one can achieve excellent security by simply executing, in parallel, a two-party ratcheting protocol (e.g., Signal) for each pair of members in a group. However, this incurs $\mathcal{O}(n)$ communication overhead for every message sent, where n is the group size. On the other hand, several related protocols were recently developed in the context of the IETF Messaging Layer Security (MLS) effort that improve the communication overhead per message to $\mathcal{O}(\log n)$. However, this reduction of communication overhead involves a great restriction: group members are not allowed to send and recover from exposures concurrently such that reaching PCS is delayed up to n communication time slots (potentially even more).

In this work we formally study the trade-off between PCS, concurrency, and communication overhead in the context of group ratcheting. Since our main result is a lower bound, we define the cleanest and most restrictive setting where the tension already occurs: *static* groups equipped with a *synchronous* (and authenticated) broadcast channel, where up to t arbitrary parties can concurrently send messages in any given round. Already in this setting, we show in a symbolic execution model that PCS requires $\Omega(t)$ communication overhead per message. Our symbolic model permits as building blocks black-box use of (even "dual") PRFs, (even key-updatable) PKE (which in our symbolic definition is at least as strong as HIBE), and broadcast encryption, covering all tools used in previous constructions, but prohibiting the use of exotic primitives.

To complement our result, we also prove an almost matching upper bound of $\mathcal{O}(t \cdot (1 + \log(n/t)))$, which smoothly increases from $\mathcal{O}(\log n)$ with no concurrency, to $\mathcal{O}(n)$ with unbounded concurrency, matching the previously known protocols.

* The full version [11] of this extended abstract is available as entry 2020/1171 in the IACR eprint archive.

1 Introduction

Post-Compromise Security. End-to-end (E2E) encrypted messaging systems including WhatsApp, Signal, and Facebook Messenger have increased in popularity. In these systems, intermediaries including the messaging service provider should not be able to read or modify messages. Moreover, as typical sessions in such E2E systems can last for a very long time, state compromise of some of the participants is becoming a real concern to the deployment of such systems. To address this security concern, modern E2E systems fulfill a novel property called *Post-Compromise Security* [16], which refers to the ability of a given protocol to recover—by means of normal protocol operations—from the exposure of local states of its (otherwise honest) participants. For example, the famous two-party Signal [28] protocol achieves PCS by having parties continuously run fresh sessions of Diffie-Hellman key agreement "in the background".

Group Messaging. By now, the setting of PCS-secure two-party encrypted messaging systems is relatively well understood [15,10,30,23,2,19,24]. In contrast, the setting of PCS-secure *group* messaging is much less understood. On the one extreme, several systems, including Signal Messenger itself, achieve PCS in groups by simply executing, in parallel, a two-party PCS-secure protocol (e.g., Signal) for each pair of members in a group. In addition to achieving PCS, this simple technique is also extremely resilient to asynchrony and concurrency: people can send messages concurrently, receive them out-of-order, or be offline for extended periods of time. However, it comes at a steep communication overhead $\mathcal{O}(n)$ for every message sent, where n is the group size.

On the other hand, several related protocols [14,3,4,5] (some of them introduced under the term *continuous group key agreement (CGKA)*[1]) were recently developed in the context of the IETF Message Layer Security (MLS) initiative for group messaging [7]. One of the main goals of this initiative was to achieve PCS with a significantly lower communication overhead. And, indeed, for static groups, these protocols improve this overhead per message to $\mathcal{O}(\log n)$. More precisely, these protocols separate protocol messages into two categories: *Payload* messages, used to actually encrypt messages, have no overhead, but also do not help in establishing PCS. In contrast, *update* messages carry no payload, but exclusively establish PCS: intuitively, an update message from user A refreshes all cryptographic material held by A. These update messages have size proportional to $\mathcal{O}(\log n)$ in MLS-related protocols, which is a significant saving for large groups, compared to the pairwise-Signal protocol.

Concurrency. Unfortunately, this reduction of communication overhead for MLS-related protocols involves a great restriction: *all update messages must be generated and processed one-by-one in the same order by all the group members.*

[1] By distinguishing between "CGKA" and "group ratcheting", these works differentiate between the asymmetric cryptographic parts of the protocols and the entire key establishment procedure, respectively [5]. In order to avoid this strict distinction, we call it "group ratcheting" here.

We stress that this does not just mean that update messages can be prepared concurrently, but processed in some fixed order. Instead, fresh update message cannot be *prepared* until all previous update messages are *processed*. In particular, it is critical to somehow implement what these protocols call a "delivery server", whose task is to reject all-but-one of the concurrently prepared update messages, and then to ensure that all group members process the "accepted updates" in the same correct order. Implementing such a delivery server poses a significant burden not only in terms of usability (which is clear), but also for *security* of these protocols, as it delays reaching PCS up to n communication time slots (potentially more in asynchronous settings, such as messaging). Indeed, the concurrency restriction of MLS is currently one of the biggest criticisms and hurdles towards its wide-spread use and adoption (see [3] for extensive discussion of this). In contrast, pairwise Signal does not have any such concurrency restriction, albeit with a much higher communication overhead. See Section 4 and Table 1 for more detailed comparison of various existing methods for group ratcheting.

Our Main Question. This brings us to the main question we study in this work:

What is the trade-off between PCS, concurrent sending and low communication complexity in encrypted group messaging protocols?

For our lower bound, we define the cleanest and most restrictive setting where the tension already occurs: *static* groups equipped with a *synchronous* (and authenticated) broadcast channel, where up to t arbitrary users can concurrently send messages in any given round. In particular, $t = 1$ corresponds to the restrictive MLS setting which, we term "no concurrency", and $t = n$ corresponds to unrestricted setting achieved by pairwise Signal, which we term "full concurrency". Also, without loss of generality, and following the convention already established in MLS-related protocols, we focus on the "key encapsulation" mechanism of group messaging protocols. Namely, our model is the following:

We have a static group of n members whose goal is to continuously share a group key k. Group members have private states st, and communicate in rounds over a public broadcast channel. Each round refreshes the current group key k into the next group key k' as follows: 1. At the beginning of a round, an arbitrary subset of up to t group members is selected by the adversary to update the current group key k. These groups members are called *senders* (of a given round). 2. During each round, each sender—*unaware of the identities of other senders*—tosses fresh random coins, sends a ciphertext c over the broadcast channel, and updates its private state st. 3. At the end of each round, all (up to t) ciphertexts c are received by all n users, who use them to update their state st, and output a new group key k'. 4. At the end of each round, the adversary can learn the current group key k', and is also allowed to expose an arbitrary number of group member states st.

For our lower bound, we will demand the following, rather weak, PCS guarantee. A key k after round i (not directly revealed to the attacker) is secure if:

(a) no user is exposed in round $i' \geq i$; (b) all users sent at least one update ciphertext between their latest exposure and round $i-1$; and (c) after all exposed users sent once without being exposed again, at least one user additionally sent in round $j \leq i$. Condition (a) will only be used in our lower bound (to make it stronger), to ensure that our lower bound is only due to the PCS, but not a complementary property called *forward*-secrecy, which states that past round keys cannot be compromised upon current state exposure. However, our upper bound will achieve forward-secrecy, dropping (a).

Condition (b) is the heart of PCS, demanding that security should be eventually restored once every exposed user updated its state. Condition (c) permits a one-round delay before PCS takes place. While not theoretically needed, avoiding this extra round seems to require some sort of multiparty non-interactive key exchange for *concurrent* state updates, which currently requires exotic cryptographic assumptions, such as multi-linear maps [12,13]. In contrast, the extra round allows to use traditional public-key cryptography techniques, such as the exposed user sending fresh public-keys, and future senders using these keys in the extra round to send fresh secret(s) to this user. While condition (c) strengthens our lower bound, our upper bound construction can be minimally adjusted to achieve PCS for *non-concurrent* state updates even without this "extra round".

For conciseness, we call any protocol in our model a *group ratcheting* scheme, taking inspiration from the "double ratchet" paradigm used in design of the Signal protocol [28].

Our Upper Bound. We show nearly matching lower and upper bounds on the efficiency of t-concurrent, PCS-secure group ratcheting schemes. With our upper bound we provide a group ratcheting scheme with message overhead $\mathcal{O}(t \cdot (1 + \log(n/t)))$, which smoothly increases from $\mathcal{O}(\log n)$ with no concurrency, to $\mathcal{O}(n)$ with unbounded concurrency, matching the upper bounds of the previously known protocols. Our upper bound is proven in the standard computational model. For the weak notion of PCS alone sketched above (i.e., conditions (a)-(c)), we only need public-key encryption (PKE) and pseudorandom functions (PRFs). Our construction carefully borrows elements from the complete subtree method of [27] used in the context of broadcast encryption (BE), and the TreeKEM protocol of the MLS standard [7,3] used in the context of non-concurrent group ratcheting. Similarly, one can view our construction as an adapted combination of components from Tainted TreeKEM [4] and the most recent MLS draft (verion-09) [8] with its propose-then-commit technique. By itself, none of these constructions is enough to do what we want: BE scheme of [27] allows to send a fresh secret to all-but-t senders from the previous round (this is needed for PCS), but needs centralized distribution of correlated secret keys to various users, while the TreeKEM schemes no longer need a group manager, but do not withstand concurrency of updates in a rather critical way. Finally, the propose-then-commit technique, when naively combined with (Tainted) TreeKEM as in MLS [8], in the worst case induces an overhead linear in the group size, and still does not completely achieve our desired concurrency and PCS guarantees. Nevertheless, we show how to combine

these structures together—in a very concrete and non-black-box way—to obtain our scheme with overhead $\mathcal{O}(t \cdot (1 + \log(n/t)))$.

Moreover, we can easily achieve forward-security in addition to PCS (i.e., drop restriction (a) on the attacker), by using the recent technique of [24,3], which basically replaces traditional PKE with so called *updatable* PKE (uPKE). Informally, such PKE is *stateful*, and only works if all the senders are synchronized with the recipient (which can be enforced in our model, even with concurrency). Intuitively, each uPKE ciphertext updates the public and secret keys in a correlated way, so that future ciphertexts (produced with new public key) can be decrypted with the new secret key, but old ciphertexts cannot be decrypted with the new secret key. Hence, uPKE provides an efficient and practical mechanism for forward-secrecy in such a synchronized setting, without the need of heavy, less efficient tools, such as hierarchical identity based encryption (HIBE), directly used as a building block for strongly secure group ratcheting [5], or used as an intermediary component to build stronger *key-updatable* PKE (kuPKE)[2] for secure two-party messaging [30,23].

Our Lower Bound. We prove a lower bound $\Omega(t)$ on the efficiency of any group ratcheting protocol which only uses "realistic" tools, such as (possibly key-updatable[2]) PKE, (possibly so called "dual") PRFs, and general BE (see Section 2 for explaining these terms). We define our symbolic notion of key-updatable PKE so that it even captures functionality and security guarantees at least as strong as one expects from HIBE. To the best of our knowledge, these primitives include all known tools used in all "practical" results on group ratcheting (including our upper bound). Thus, our result nearly matches our upper bound, and shows that the $\Omega(n)$ *overheard of pairwise Signal protocol is optimal for unbounded concurrency*, at least within our model.

To motivate our model for the lower bound, group ratcheting would be "easy" if we could use "exotic" tools, such as multiparty non-interactive key agreement (mNIKE), multi-linear maps, or general-purpose obfuscation. For example, using general mNIKE, one can easily achieve PCS and unbounded concurrency, by having each member simply broadcast its new public key, without any knowledge of other senders: at the end of each round, the union of latest keys of all the group members magically (and non-interactively) updates the previous group key to a new, unrelated value. Of course, we currently don't have any even remotely practical mNIKE protocols, so it seems natural that we must define a model which only permits the use of "realistic" tools, such as (ku)PKE, (dual) PRFs, BE, (HIBE,) etc.

To formally address this challenge, we use a *symbolic* modeling framework inspired by the elegant work of Micciancio and Panjwani [26], who used it to derive a lower bound for the efficiency of multi-cast encryption. Symbolic models treat all elements as symbols whose algebraic structure is entirely disregarded,

[2] While for our upper bound construction weaker and more efficient uPKE (based on DH groups) suffices as in [24,3], to strengthen our lower bound we allow constructions to use stronger and less efficient key-updatable PKE (thus far based on HIBE) as in [30,23,6].

and which can be used only as intended. E.g., a symbolic public key can be defined to only encrypt messages, and the only way to decrypt the resulting ciphertext is to have another symbol corresponding to the associated secret key. In particular, one cannot perform any other operations with the symbolic public key, such as verifying a signature, using it for a Diffie-Hellman key exchange, etc.

We use such a symbolic model to precisely define the primitives we allow, including the grammar of symbols and valid derivation rules between them (see Figure 1). We then formalize the intuition for our lower bound in Section 5 (that we formally prove in the full version [11]). Our bound is actually very strong: it is the *best-case* lower bound, which holds for any execution schedule of group ratcheting protocols within our model, and which is proven against highly restricted adversaries for extremely little security requirements. Specifically, we show that each sender for round i must send at least one fresh message over the broadcast channel "specific" to every sender of the previous round $i-1$.[3] While intuitively simple, the exact formalization of this result is non-trivial, in part due to the rather advanced nature of the underlying primitives we allow. For example, we must show that no matter what shared infrastructure was established before round $(i-1)$, and no matter what information a sender A sent in round $i-1$, there is no way for A to always recover at round i from potential exposure at round $(i-2)$, unless every sender B in round i sends some message "only to A".

Perspective. To put our symbolic result in perspective, early use of symbolic models in cryptography date to the Dolev-Yao model [18], and were used to prove "upper bounds", meaning security of protocols which were too complex to analyze in the standard "computational model" (with reductions to well established simpler primitives or assumptions). In contrast, Micciancio and Panjwani [26] observed that symbolic models can also be used in a different way to prove impossibility results (i.e., *lower* bounds) on the efficiency of building various primitives using a fixed set of (symbolic) building blocks. This is interesting because we do not have many other compelling techniques to prove such lower bounds.

To the best of our knowledge, the only other technique we know is that of "black-box separations" [22]. While originally used for black-box impossibility results [22], Gennaro and Trevisan [20] adapted this technique to proving efficiency limitations of black-box reductions, such as building psedorandom generators from one-way permutations. However, black-box separation lower bounds are not only complex (which to some extent is true for symbolic lower bounds as well), but also become exponentially harder, as the primitive in question becomes more complex to define, or more diverse building blocks are allowed. In particular, to the best of our knowledge, the setting of group ratcheting using kuPKE, HIBE, dual PRFs, and BE used in this paper, appears several orders of magnitude more complex than what can be done with the state-of-the-art black-box lower bounds.

[3] Except for itself, if the sender was active in the prior round. This intuitively explains why our "best-case" lower bound is actually $(t-1)$ and not t.

Thus, we hope that our paper renews the interests in symbolic lower bounds, and that our techniques would prove useful to study other settings where such lower bounds could be proven.

2 Preliminaries

We shortly introduce our notation as well as the syntax of the most important cryptographic building blocks. We also sketch their security guarantees that we formally define along the full proofs in our full version [11].

Notation We distinguish between deterministic and probabilistic assignments with symbols $\leftarrow$ and $\leftarrow_{\$}$, respectively; the latter denotes sampling of an element x from the uniform distribution over a set $\mathcal{X}$ ($x \leftarrow_{\$} \mathcal{X}$) and invoking a probabilistic algorithm alg on input a with output x ($x \leftarrow_{\$} \mathrm{alg}(a)$). In order to make the used random coins r of an invocation explicit (and turning it into a deterministic invocation), we write $x \leftarrow \mathrm{alg}(a; r)$. We denote the cardinality of a set $\mathcal{X}$ or the length of a string s with symbols $|\mathcal{X}|$ and $|s|$. Concatenations of two bit-strings s_1, s_2 is written as $s_1 \| s_2$.

Adversaries $\mathcal{A}$ in our computational models are probabilistic algorithms invoked in a security experiment denoted by the term **Game**. Therein they can call oracles, denoted by term **Oracle**.

In our symbolic model we describe grammar rules as follows. For three types of symbols X, Y, and Z, $X \mapsto Y|Z$ denotes that symbols of type X can be parsed as symbols of type Y or type Z. A type that cannot be parsed further is called *terminal type*. Using these grammar rules, we define derivation rules that describe how symbols can be derived from sets of (other) symbols. For a symbol m and set of symbols $\boldsymbol{M}$, $\boldsymbol{M} \vdash m$ means that m can be derived from the symbols in set $\boldsymbol{M}$ by using the grammar and derivation rules that we specify in our symbolic model.

(Dual) Pseudo-Random Function A pseudo-random function prf takes a symmetric key and some associated data, and outputs another symmetric key such that for sets $\mathcal{K}, \mathcal{AD}$: $\mathrm{prf}(k, ad) \rightarrow k'$ with $k, k' \in \mathcal{K}$ and $ad \in \mathcal{AD}$. A dual pseudo-random function dprf takes two symmetric keys and outputs another symmetric key such that for set $\mathcal{K}$: $\mathrm{dprf}(\{k_1, k_2\}) \rightarrow k'$ with $k_1, k_2, k' \in \mathcal{K}$ with the added property that $\mathrm{dprf}(k_1, k_2) = \mathrm{dprf}(k_2, k_1) = k'$. For simplicity (in our proof), we only consider symmetric dual PRFs [9].

A secure PRF outputs a key that is *secret*[4] if the input key is secret as well. A dual PRF additionally achieves secrecy of the output key in case at most one of the two input keys is known by an attacker.

[4] Where *secrecy* means indistinguishable from a random key in the computational model and underivable from public symbols in the symbolic execution model.

Key-Updatable Public Key Encryption Key-updatable public key encryption (kuPKE) is an extension of public key encryption that allows for independent updates of public and secret key with respect to some associated data. This primitive has been used in constructions of two-party ratcheting (e.g., [30,23,29,25]). Furthermore, a work by Balli et al. [6] recently showed that it is actually necessary for building optimally secure two-party ratcheting.

A kuPKE scheme UE is a tuple of algorithms $\mathsf{UE} = (\text{gen}, \text{up}, \text{enc}, \text{dec})$ where up takes some associated data together with either a public key or a secret key and produces a new public key or secret key respectively such that for sets $\mathcal{SK}, \mathcal{PK}, \mathcal{C}, \mathcal{M}, \mathcal{AD}$: $\text{gen}(sk) \to pk$, $\text{up}(sk, ad) \to sk'$, $\text{up}(pk, ad) \to pk'$, $\text{enc}(pk, m) \to_\$ c$, and $\text{dec}(sk, c) \to m$ with $sk, sk' \in \mathcal{SK}$, $pk, pk' \in \mathcal{PK}$, $ad \in \mathcal{AD}$, $m \in \mathcal{M}$, and $c \in \mathcal{C}$. A kuPKE scheme UE is correct if for synchronously updated public key and secret key, the latter can decrypt ciphertexts produced with the former: $\Pr[\forall n \in \mathbb{N}\ \text{dec}(sk_n, \text{enc}(pk_n, m)) = m : sk_0 \leftarrow_\$ \mathcal{SK}, pk_0 = \text{gen}(sk_0), \forall i \in [n]\ ad_i \leftarrow_\$ \mathcal{AD}, pk_{i+1} = \text{up}(pk_i, ad_i), sk_{i+1} = \text{up}(sk_i, ad_i), m \leftarrow_\$ \mathcal{M}] = 1$.

A secure kuPKE scheme intuitively guarantees that a message, encrypted to public key pk' that was derived from another public key pk via sequential updates under associated-data from vector $ad \in \mathcal{AD}^*$, cannot be decrypted by a (computationally bounded, or symbolic) adversary even with access to any secret keys, derived via updates from pk's secret key sk under an associated-data vector $ad' \in \mathcal{AD}^*$ such that ad' is not a prefix of ad. Note that this intuitive security notion matches security of HIBE when associated data is being parsed as identity strings.

Broadcast Encryption A broadcast encryption (BE) scheme BE is a tuple of four algorithms $\mathsf{BE} = (\text{gen}, \text{reg}, \text{enc}, \text{dec})$ where reg takes a (main) secret key and an integer and produces an accordingly *registered* secret key, enc takes, in addition to public key and message, a set of integers to indicate which registered secret keys must be unable to decrypt the message such that for sets $\mathcal{MSK}, \mathcal{SK}, \mathcal{MPK}, \mathcal{C}, \mathcal{M}$: $\text{gen}(msk) \to mpk$, $\text{reg}(msk, u) \to_\$ sk$, $\text{enc}(mpk, \boldsymbol{RM}, m) \to_\$ c$, and $\text{dec}(sk, c) \to m$ with $msk \in \mathcal{MSK}$, $mpk \in \mathcal{MPK}$, $u \in \mathbb{N}$, $sk \in \mathcal{SK}$, $\boldsymbol{RM} \subset \mathbb{N}$, $m \in \mathcal{M}$, and $c \in \mathcal{C}$. A broadcast encryption scheme BE is correct if all registered secret keys that were not excluded when encrypting with the public key can decrypt the corresponding encrypted message: $\Pr[\text{dec}(sk, \text{enc}(mpk, \boldsymbol{RM}, m)) = m : msk \leftarrow_\$ \mathcal{MSK}, mpk = \text{gen}(msk), u \leftarrow_\$ \mathbb{N}, sk \leftarrow_\$ \text{reg}(msk, u), \boldsymbol{RM} \subset \mathbb{N} \setminus \{u\}] = 1$.

A secure BE scheme intuitively guarantees that a message, encrypted to a (main) public key mpk with a set of removed users $\boldsymbol{RM}$, cannot be decrypted by a (computationally bounded, or symbolic) adversary even with access to any secret keys, registered under mpk's main secret key msk for numbers $u \in \boldsymbol{RM}$.

3 Security of Concurrent Group Ratcheting

In this work we consider an abstraction of group ratcheting under significant relaxations and restrictions with respect to the real-world. The purpose of this

approach is to disregard irrelevant aspects in order to highlight the immediate effects of concurrent state updates in group ratcheting.

In the following, we define syntax and (restricted) security of ratcheting in static groups against computationally bounded adversaries. We assume in our model that all group members have access to a round-based reliable and authenticated broadcast. Additionally, since our focus are concurrent operations in an initialized group, we consider an abstract initialization algorithm for deriving initial user states.[5]

Syntax A static group ratcheting protocol is a tuple of three algorithms $\mathsf{GR} = (\text{init}, \text{snd}, \text{rcv})$ such that for sets $\mathcal{ST}_{\mathsf{GR}}, \mathcal{C}_{\mathsf{GR}}, \mathcal{K}_{\mathsf{GR}}, \mathcal{R}$:

- $\text{init}(n; r) \rightarrow (st_1, \ldots, st_n)$ with $n \in \mathbb{N}$, $r \in \mathcal{R}$, and $st_1, \ldots, st_n \in \mathcal{ST}_{\mathsf{GR}}$; creates an initial local state for every participating group member.
- $\text{snd}(st; r) \rightarrow (st', c)$ with $st, st' \in \mathcal{ST}_{\mathsf{GR}}$, $r \in \mathcal{R}$, and $c \in \mathcal{C}_{\mathsf{GR}}$; takes the current state of an instance (in addition to freshly sampled random coins) and outputs the updated state and update information within a ciphertext that is to be sent via the broadcast.
- $\text{rcv}(st, \boldsymbol{c}) \rightarrow (st', k)$ with $st, st' \in \mathcal{ST}_{\mathsf{GR}}$, $\boldsymbol{c} \subset \mathcal{C}_{\mathsf{GR}}$, and $k \in \mathcal{K}_{\mathsf{GR}}$; takes the current state of an instance and a set of update ciphertexts (e.g., all broadcast ciphertexts since this instance's last receiving), and outputs the updated state and the current (joint) group key.

Security Security experiments $\text{KIND}^b_{\mathsf{GR}}$ in which adversary $\mathcal{A}$ attacks scheme GR proceed as follows:

1. $\mathcal{A}$ determines the number of group members n. Afterwards the challenger invokes the init algorithm to generate initial secret states for all members. Then the security experiment continues in rounds. In every round i
   - adversary $\mathcal{A}$ chooses set $\boldsymbol{U}^i_{\mathbf{S}}$ of senders. For each sender $u \in \boldsymbol{U}^i_{\mathbf{S}}$ algorithm snd is invoked. All resulting ciphertexts are both given to $\mathcal{A}$ and received by all group members via invocations of algorithm rcv.
   - adversary $\mathcal{A}$ chooses set $\boldsymbol{U}^i_{\mathbf{X}}$ of exposed users. The local state of each user $u \in \boldsymbol{U}^i_{\mathbf{X}}$ after receiving in round i is given to $\mathcal{A}$.
2. During the entire security experiment, $\mathcal{A}$ can challenge group keys established in any round i^*. $\mathcal{A}$ either obtains a random key (if $b = 0$) or the actual group key from round i^* (if $b = 1$) in response.
3. When terminating, $\mathcal{A}$ returns a guess b' such that it wins if $b = b'$ and for all challenged group keys it holds that:
   (a) no user was exposed after a challenged group key was computed,
   (b) every user sent at least once after being exposed and before a challenged group key was computed, and

[5] We note that we only consider a single independently established group session. For protocols in which participants use the same secrets simultaneously across multiple (thereby dependent) sessions, we refer the reader to a work by Cremers et al. [17]. Both the problems and the solutions for these two considerations appear to be entirely distinct.

(c) after all exposed users sent once without being exposed again, at least one user additionally sent before a challenged group key was computed.

Group keys for which conditions 3a-3c hold are marked *secure*.

We restrict the adversary with condition (3a) only because the resulting weaker security definition already suffices to prove our *lower* bound of communication complexity. For our full model in which we prove the construction of our *upper* bound secure , we strengthen adversaries by lifting restriction (3a). This reflects that our upper bound construction achieves immediate forward-secrecy while our lower bound already holds without requiring any form of forward-secrecy.

Condition (3b) models that a user who was exposed must generate fresh secrets and send the respective public values to the group before it can receive confidential information for establishing new secure group keys. After all exposed users recovered by sending subsequently, their sent contribution must be used effectively to establish a new secret group key. Therefore, condition (3c) additionally requires one further response from a user as a reaction to all newly contributed public values.

For removing condition (3c) either 1. the last users who recovered did so concurrently at most as a pair of two (such that their new public contributions can be merged into a shared group key non-interactively with NIKE mechanisms), or 2. multiparty NIKE schemes exist (for resolving cases of more concurrently recovering users). In order to simplify our security definition by not introducing an according case distinction tracing occurrences of case 1, we generally restrict the adversary with condition (3c). We note that for proving our lower bound, restricting the adversary by this condition strengthens our result.

Intuitively, a group ratcheting scheme is secure if no adversary $\mathcal{A}$ exists that wins the above defined security experiment with probability non-negligibly higher than $1/2$.

Restrictions of the Model With the following abstractions, simplifications, and restrictions, we support clarity and comprehensibility of our results and strengthen the statement of our lower bound. We consider: 1. A round-based communication setting, 2. Static groups, 3. All group members receive in every round, 4. Only passive adversaries 5. Adversaries can expose users only after receiving, and 6. Adversaries cannot attack used randomness. As we do not aim to develop a functional and secure group messenger but to theoretically analyze the foundations of concurrent group ratcheting, we believe this is justified.

4 Deficiencies of Existing Protocols

The problem of constructing group ratcheting could be solved trivially if efficient *multiparty non-interactive key exchange* schemes existed. Especially for the concurrent recovery from state exposures in group ratcheting, the lack of this tool appears to be crucial: Due to not being able to combine independently

proposed fresh public key material, existing efficient group ratcheting constructions cannot process concurrent operations as we will explain in this section. In Table 1 we summarize the characteristics of previous group ratcheting schemes in comparison to our construction and the lower bound.

	PCS	Concurrency	Overhead
Sender Key Mechanism [31]	○	●	1
Parallel Pairwise Signal [31,15,2]	●	●	n
Asynchronous Ratcheting Trees [14]	●	○	$\log(n)$
Causal TreeKEM [32]	◐	◑	$\log(n)$
TreeKEM Familiy [3,4]	●	○	$\log(n)$
MLS Draft-09 [8]	◐	◑	n
Optimally Secure Tainted TreeKEM [5]	◐	◑	$\log(n)$
Our Construction	●	●	$t \cdot (1 + \log(n/t))$
Our Lower Bound	●	●	$t - 1$

Table 1: Properties of group ratcheting constructions and our lower bound. $t = |\boldsymbol{U}_{\mathbf{S}}^{i-1}|$ is the number of members who sent concurrently in the previous round. For the overhead we consider a worst-case scenario in a constant size group. Constructions denoted with '◐'/'◑' provide PCS under no concurrency and can handle concurrent state updates without reaching PCS with them.

Sender Key Mechanism WhatsApp uses the so called *sender key mechanism* for implementing group chats [31]. This mechanism distributes a symmetric *sender key* for each member in a group. When sending a group message, the sender protects the payload with its own sender key, transmits the resulting (single) ciphertext, and hashes the used sender key to obtain its next sender key. The receivers decrypt the ciphertext with the sender's sender key and also update the sender's sender key by hashing it.

While the deterministic derivation of sender keys induces no communication overhead after the initial distribution of sender keys, it implies the reveal of all future sender keys as soon as a member state is exposed (breaking post-compromise security). However, as each group member's key material is processed and used independently, concurrently initiated group operations can be processed naturally.

Parallel Execution of Pairwise Signal The group ratcheting mechanism implemented in the Signal messenger bases on parallel executions of the two-party Double Ratchet Algorithm [28,15,2] between each pair of members in a group [31]. Due to splitting the group of size n into its n^2 independent pairwise components, this construction can naturally handle concurrency. At the same time, this approach induces a communication overhead of $\mathcal{O}(n)$ ciphertexts per sent group payload.

Since the Double Ratchet Algorithm reaches post-compromise security (PCS) for each pair of members, also its parallel execution achieves this goal for the group against passive adversaries or if the member set remains static. Rösler et

al. [31] describe an active attack against PCS in dynamic groups that exploits the implemented decentralized membership management. Furthermore, the delayed recovery from state exposures in the Double Ratchet Algorithm due to a strictly alternating update schedule between protocol participants (cf. analysis and fix in [2]) lets recoveries from state exposures in the group become effective only after every group member sent once at worst. With stronger two-party ratcheting protocols (e.g., [30,29,23,2,24]) this problem can be solved.

Asynchronous Ratcheting Tree While the two above described approaches compute and use multiple symmetric keys in parallel for protecting communication in groups, the following constructions do so by deriving a single shared group key at each step of the group's lifetime. Therefore they arrange asymmetric key material on nodes in a tree structure in which each leaf represents a group member and the common root represents the shared group secret. Every group member stores the asymmetric secrets on the path from its leaf to the common root in its local state. For updating the local state, in order to recover from an adversarial exposure, all constructions let the updating member generate new asymmetric secrets for each node on their path to the root.

In the Asynchronous Ratcheting Trees (ART) design [14], these asymmetric secrets are exponents in a Diffie–Hellman (DH) group. State updates of a member's path is conducted as follows: the updating member freshly samples a new secret exponent for its own leaf and then deterministically derives every ancestor node's secret exponent as the shared DH key from its two children's public DH shares. All resulting new public DH shares on the path are sent to the group, inducing a communication overhead of $\mathcal{O}(\log(n))$ per update operation. Other members perform the same derivations for updated nodes on their own paths to the root to obtain the new exponents. Since all secrets in the updating member's local state are renewed based on fresh random coins, this mechanism achieves PCS.

The reason for ART not being able to process concurrent update operations is that simultaneous updates of nodes in the tree with independently computed DH exponents cannot be merged into a joint tree structure while reaching PCS. For t concurrent updates, a t-party NIKE would be needed to combine the resulting t new proposed DH shares into a shared secret exponent for the ancestor node at which all updating members' paths to the root join together. (As mentioned before, if multiparty NIKE existed, group ratcheting can be solved trivially without complex tree structures.)

Causal TreeKEM As in the ART design [14], Causal TreeKEM [32] uses exponents in a DH group as asymmetric secrets on nodes in the tree. Also the update procedure is conceptually the same. However, in case of concurrently proposed path updates, the conflicting new exponents on a node are combined via exponent-addition and the conflicting public DH shares on a node are combined via multiplying these group elements.

Although this merge-mechanism resolves conflicts caused by concurrency, the combination of updated path secrets is not post-compromise secure: the old

exponents of two nodes (from which their updating users A and B aimed to recover), whose common parent was updated via a combination of concurrent path updates, suffice to derive their parent's resulting new exponent. (The new exponent is the old exponent mixed with random values from A and B that they encrypt to the other's old node key.)

TreeKEM Family In the family of TreeKEM constructions [3,4], the asymmetric key material of nodes in the tree are key encapsulation mechanism (KEM) key pairs or, in forward-secure TreeKEM, updatable KEM key pairs. For updating its local state, a group member samples a fresh secret from which it deterministically derives seeds for each node on its path to the root, such that all ancestor seeds can be derived from their descendant seeds (but not vice versa). The updating member generates the new key pair for each updated node from its seed deterministically, and encapsulates the node's seed to the public key of the child which is not on the member's path to the root. This mechanism achieves PCS and induces a communication overhead of $\mathcal{O}(\log(n))$ per update.

The idea of recovery from exposures is undermined in case of concurrency, since updating members send their new seeds for a node on their path to public keys of siblings, simultaneously being updated and replaced by new key material of members who concurrently update: the potentially exposed secrets *from which* one updating member aims to recover can then be used to obtain the new secrets *with which* the other updating user aims to recover (as in the case of Causal TreeKEM). Consequently, concurrent updates in TreeKEM are essentially ineffective with respect to PCS.

Forward-secure TreeKEM [3] uses an updatable KEM for enhancing forward-security guarantees of the above described mechanism. Tainted TreeKEM [4] enhances PCS guarantees with respect to dynamic membership changes in groups. Neither of these changes affect the trade-offs discussed here.

MLS Draft-09 Based on TreeKEM, the most recent draft of MLS [8] distinguishes between two state update variants: (a) In an *update proposal* a member refreshes only its own leaf key pair, removes all other nodes on the path from this leaf to the root, and makes the root parent of all nodes that thereby became parentless. (b) In a *commit* a member combines previous update proposals and refreshes all key pairs on the path from its own leaf to the root (matching the normal TreeKEM update as described in the last paragraph).

In principle, both update variants achieve PCS for respective the sender. However, for simultaneously sent *commits*, all but one are rejected (e.g., by a central server) meaning that PCS under concurrency is not achieved for rejected updating commits. Furthermore, while *update proposals* can be processed concurrently, they eventually let the tree's depth degrade to 1, inducing a worst-case overhead of $\mathcal{O}(n)$ for later commits.[6]

[6] Consider, for example, a scenario in which the same majority of members always sends update proposals and a fixed disjoint set of few members always commits. In this case, the overhead of commits for these few members converges to $\mathcal{O}(n)$.

Optimally Secure Tainted TreeKEM Recently and concurrent to our work, an optimally secure variant of group ratcheting, based on a combination of Tainted TreeKEM and MLS draft-09, was proposed by Alwen et al. [5]. In addition to authentication guarantees (which is independent of our focus), their protocol achieves strong security guarantees for group partitions due to concurrency: instead of assuming that a (consensus) mechanism rejects conflicting commits as in MLS, they anticipate that different sub-groups of group members may process different of these commits such that the overall perspective on the group diverges. Their protocol guarantees that, after diverging, exposing states of one sub-group's members does not affect the security of another sub-groups' secrets. Intuitively, this is achieved by using HIBE key pairs on the tree's nodes that are regularly updated via secret-key-delegation based on identity strings that reflect the current perspective on the group. (For details, we refer the interested reader to [5].)

While these changes increase security with respect to some form of forward-secrecy under group partitions, they do not entirely solve the issue of conflicting commits as in MLS: committed state updates still only have an effect in a sub-group that processes the commit such that only one user at a time can update secrets on the path from its leave to the root whereas other user's path updates remain ineffective.

Our construction from Section 6 bypasses the issue of concurrently generated, incompatible path proposals by postponing the update of affected nodes in the tree by one communication round. However, "immediate" PCS can still be reached for non-concurrent updates by composing our construction with one of the above described ones without loss in efficiency. We note that some of the above constructions provide strong security guarantees with respect to active adversaries, dynamic groups, entirely asynchronous communication, or weak randomness, which is out (and partially independent) of our consideration's scope.

5 Intuition for Lower Bound

Our lower bound proof intuitively says that every group ratcheting scheme with better communication complexity than this bound is either insecure, or not correct, or cannot be built from the building blocks we consider. In the following, we first list these considered building blocks and argue why the selection of those is indeed justified (and not too restrictive). We then abstractly explain the symbolic security definition of group ratcheting, and finally sketch the steps of our proof that is formally given in the full version [11].

5.1 Symbolic Building Blocks

The selection of primitives which a group ratcheting construction may use to reach minimal communication complexity in our symbolic model is inspired by the work of Micciancio and Panjwani [26]. For their lower bound of communication complexity in multi-cast encryption—which can also be understood as group

key exchange—, Micciancio and Panjwani allow constructions to use pseudo-random generators, secret sharing, and symmetric encryption. We instead consider 1. *(dual) pseudo-random functions*, 2. *key-updatable public key encryption* (with functionality and symbolic security guarantees at least as strong as those of *hierarchical identity based encryption*), and 3. *broadcast encryption* and thereby significantly extend the power of available building blocks. As secret sharing appears to be rather irrelevant in our setting—as well as it is irrelevant in their setting—, we neglect it to achieve better clarity in model and proof.

Bulding Blocks in Related Work To support the justification of our selection, we note that all previous constructions of group ratcheting base on less powerful building blocks than we consider here: The ART construction [14] relies on a combination of dual PRF and Diffie-Hellman (DH) group. The actual properties used from the DH group can also be achieved by using generic public key encryption (PKE)—as demonstrated by its following successors. TreeKEM as proposed in the MLS initiative [3,8] relies on a PRG and a PKE scheme. TreeKEM with extended forward-secrecy [3] relies on a PRG and an updatable PKE scheme. The syntax of the latter in combination with the respective computational security guarantees can be considered weaker than our according symbolic variant of kuPKE. Tainted TreeKEM [4] relies on a PKE scheme in the random oracle model. Optimally secure Tainted TreeKEM [5] relies on an HIBE scheme in the random oracle model. As noted before, functionality and security guarantees of HIBE are captured in our symbolic notion of kuPKE. The property of the random oracle that allows for mixing multiple input values of which at least one is confidential to derive a confidential random output can be achieved similarly by using (a cascade of) dual PRF invocations.[7]

Only the post-compromise *insecure* merge-mechanism of DH shares from Causal TreeKEM [32] is not captured in our symbolic model. However, turning this mechanism post-compromise *secure* results in multi-party NIKE, which we intentionally exclude.

Grammar The grammar definition of the considered building blocks bases on five types of symbols: messages M, secret keys SK, symmetric keys K, public keys PK, and random coins R (which is a terminal type). These types and their relation are specified in the lower right corner of Figure 1. For simplicity (and in order to strengthen our lower bound result), we consider algorithms gen and enc interoperable for kuPKE and BE.[8]

Derivation Rules Symbolic security for the building blocks is defined via derivation rules that describe the conditions under which symbols can be derived from

[7] If the constructions in [4,5] would rely on stronger (security) guarantees of the random oracle model, their practicability might be questionable.

[8] As a simplification we use $\mathbb{N}$ to denote the user input symbol of BE, $\mathcal{S}(\cdot)$ to denote an unordered compilation of multiple such symbols, and $\{\cdot,\cdot\}$ to denote an unordered compilation of two key symbols. For kuPKE encryptions the second parameter in our symbolic model can be ignored.

Derivation of protected values:

a) $m \in \boldsymbol{M} \implies \boldsymbol{M} \vdash m$

b) $\boldsymbol{M} \vdash k \implies \forall ad\ \boldsymbol{M} \vdash \mathrm{prf}(k, ad)$

c) $\boldsymbol{M} \vdash k_1, k_2 \implies \boldsymbol{M} \vdash \mathrm{dprf}(\{k_1, k_2\})$

d) $\boldsymbol{M} \vdash \mathrm{enc}(pk, \boldsymbol{RM}, m), sk :$
$\mathrm{Fit}(pk, \boldsymbol{RM}, sk) \implies \boldsymbol{M} \vdash m$

Derivation of public values:

g) $\boldsymbol{M} \vdash sk \implies \boldsymbol{M} \vdash \mathrm{gen}(sk)$

h) $\boldsymbol{M} \vdash pk \implies \forall ad\ \boldsymbol{M} \vdash \mathrm{up}(pk, ad)$

i) $\boldsymbol{M} \vdash pk, m \implies \forall \boldsymbol{RM}\ \boldsymbol{M} \vdash \mathrm{enc}(pk, \boldsymbol{RM}, m)$

Derivation of secret keys:

e) $\boldsymbol{M} \vdash sk \implies \forall ad\ \boldsymbol{M} \vdash \mathrm{up}(sk, ad)$

f) $\boldsymbol{M} \vdash sk \implies \forall u\ \boldsymbol{M} \vdash \mathrm{reg}(sk, u)$

Grammar rules:

1. $M \mapsto SK|PK|\mathrm{enc}(PK, \mathcal{S}(\mathbb{N}), M)$
2. $SK \mapsto K|\mathrm{up}(SK, M)|\mathrm{reg}(SK, \mathbb{N})$
3. $K \mapsto R|\mathrm{prf}(K, M)|\mathrm{dprf}(\{K, K\})$
4. $PK \mapsto \mathrm{gen}(SK)|\mathrm{up}(PK, M)$

Fig. 1: Grammar and derivation rules of building blocks in the symbolic model.

sets of (other) symbols. These rules are defined in Figure 1 clustered into those with which protected values can be obtained, with which secret keys can be updated or registered, and with which public values can be obtained.

Rules b) and c) describe the security of (dual) PRFs, rules d), e), and g) to i) describe the security and functionality of kuPKE (and HIBE), and rules d), f), g), and i) describe the security and functionality of BE.

Rule d), describing the conditions under which a ciphertext can be decrypted, uses predicate Fit that validates the compatibility of public key and secret key (and set of removed registered users). Intuitively, a secret key sk is compatible with a public key pk if all updates for obtaining sk correspond to updates for obtaining pk in the same order and under the same associated data with respect to an initial key pair, or if the former was registered under the main secret key of the latter.

5.2 Symbolic Group Ratcheting

The syntax of group ratcheting was introduced in Section 3. In the following we map this syntax to the grammar definition above, and shortly give an intuition for the correctness and security of group ratcheting in the symbolic model.

Inputs and outputs of group ratcheting algorithms init, snd, and rcv are random coins $\mathcal{R}$, local user states $\mathcal{ST}_{\mathsf{GR}}$, ciphertexts $\mathcal{C}_{\mathsf{GR}}$, and group keys $\mathcal{K}_{\mathsf{GR}}$. In our grammar these random coins are sets of type R symbols, local states and ciphertexts are sets of type M symbols, and group keys are symbols of type K.

According to this grammar, we require from symbolic constructions of group ratcheting for being *correct* that 1. all outputs of a group ratcheting algorithm invocation can be derived from its inputs via the derivation rules defined above and 2. in each round the group keys, computed by all users, are equal. The first condition is necessary to allow for symbolic adversaries. We note that this condition furthermore implies "inverse derivation guarantees", meaning that symbols can *only* be obtained via our derivation rules. For example, for inputs IN and outputs OUT of an algorithm invocation, output $k' \in \mathrm{OUT}$ with $\mathrm{prf}(k, ad) = k'$ is either also element of set IN (i.e., $k' \in \mathrm{IN}$), or k' is encrypted in a cipher-

text contained in set IN, or IN $\vdash k$ holds. We explicitly provide these inverse derivation guarantees in our full version [11].

Security To transfer the computational security experiment from Section 3 to the execution of symbolic attackers against group ratcheting, only few small changes are necessary: 1. a symbolic adversary $\mathcal{A}$ follows the above defined derivation rules for an unbounded time, 2. the target of $\mathcal{A}$ is not to distinguish *securely* marked real group keys from random ones but to derive such *securely* marked keys from the ciphertexts, sent in each round, and the states, exposed at the end of each round, with these derivation rules.

A group ratcheting scheme is *secure in the symbolic model* if an unbounded adversary cannot derive any of the securely marked group keys from the combination of all rounds' ciphertexts and exposed states via the above defined rules. The fully formal variant of this definition is in Figure 2.

Game $\mathrm{SYM}_{\mathsf{GR}}(n, \boldsymbol{U}^0_{\mathbf{X}}, \boldsymbol{U}^1_{\mathbf{S}}, \boldsymbol{U}^1_{\mathbf{X}}, \ldots, \boldsymbol{U}^q_{\mathbf{S}}, \boldsymbol{U}^q_{\mathbf{X}})$
00 $\boldsymbol{XU} \leftarrow \emptyset$; $\boldsymbol{SEC} \leftarrow \emptyset$
01 $\boldsymbol{XST} \leftarrow \emptyset$; $\boldsymbol{C} \leftarrow \emptyset$
02 $XST[\cdot] \leftarrow \emptyset$; $C[\cdot] \leftarrow \emptyset$
03 $K[\cdot] \leftarrow \bot$; $r \leftarrow_{\$} \mathcal{R}$
04 $(st_1, \ldots, st_n) \leftarrow \mathrm{init}(n; r)$
05 Call Expose($\boldsymbol{U}^0_{\mathbf{X}}$)
06 For i from 1 to q:
07 Call Round($\boldsymbol{U}^i_{\mathbf{S}}$)
08 Call Expose($\boldsymbol{U}^i_{\mathbf{X}}$)
09 $\boldsymbol{C} \leftarrow \bigcup_{j\in[q]} C[i]$
10 If $\exists i' \in \boldsymbol{SEC}$: $K[i'] \in \mathrm{Der}(\boldsymbol{C} \cup \boldsymbol{XST})$:
11 Stop with 1
12 Stop with 0

Proc Round($\boldsymbol{U}$)
13 Require $\boldsymbol{U} \subseteq [n]$
14 For all $u \in \boldsymbol{U}$:
15 $r_u \leftarrow_{\$} \mathcal{R}$
16 $(st_u, c_u) \leftarrow \mathrm{snd}(st_u; r_u)$
17 $C[i] \leftarrow \bigcup_{u\in\boldsymbol{U}} c_u$
18 For all $u \in [n]$:
19 $(st_u, k_u) \leftarrow \mathrm{rcv}(st_u, C[i])$
20 If $\boldsymbol{XU} = \emptyset \wedge (\boldsymbol{U} \neq \emptyset \vee i-1 \in \boldsymbol{SEC})$:
21 $\boldsymbol{SEC} \leftarrow \{i\}$
22 $\boldsymbol{XU} \leftarrow \boldsymbol{XU} \setminus \boldsymbol{U}$
23 $K[i] \leftarrow k_1$
24 Return

Proc Expose($\boldsymbol{U}$)
25 Require $\boldsymbol{U} \subseteq [n]$
26 $\boldsymbol{XU} \leftarrow \boldsymbol{XU} \cup \boldsymbol{U}$
27 $XST[i] \leftarrow \bigcup_{u\in\boldsymbol{U}} st_u$
28 $\boldsymbol{XST} \leftarrow \boldsymbol{XST} \cup XST[i]$
29 $\boldsymbol{SEC} \leftarrow \boldsymbol{SEC} \setminus [i-1]$
30 Return

Fig. 2: Security definition of concurrent group ratcheting in our symbolic model.

5.3 Lower Bound

Using this symbolic framework, we formulate (a sketched variant of) the lower bound of communication complexity for secure (and correct) group ratcheting constructions:

> *Let* GR *be a secure and correct group ratcheting scheme. For every round i in a symbolic execution of* GR *with senders $\boldsymbol{U}^i_{\mathbf{S}}$ and exposed users $\boldsymbol{U}^i_{\mathbf{X}}$, the number of sent symbols is $|C[i]| \geq |\boldsymbol{U}^i_{\mathbf{S}}| \cdot (|\boldsymbol{U}^{i-1}_{\mathbf{S}}| - 1)$.*

For our proof, we consider a symbolic adversary that proceeds as follows:

1. In round $i-2$ a set of members $\boldsymbol{U}^{i-2}_{\mathbf{X}} \subseteq [n]$ with $|\boldsymbol{U}^{i-2}_{\mathbf{X}}| > 1$ is exposed.

2. In subsequent round $i-1$ these exposed users send (i.e., $\boldsymbol{U}_{\mathbf{S}}^{i-1} \coloneqq \boldsymbol{U}_{\mathbf{X}}^{i-2}$).
3. In round i a non-empty set of members $\emptyset \neq \boldsymbol{U}_{\mathbf{S}}^{i} \subseteq [n]$ sends.

Assuming no user was exposed in any round before or after $i-2$, our symbolic security definition requires the group key in round i to be secure (i.e., not derivable from exposed states and sent ciphertexts up to round i). In order to show that each sender in round i must send at least $|\boldsymbol{U}_{\mathbf{S}}^{i-1}|-1$ ciphertexts to establish this secure group key, we analyze the effects of exposures in round $i-2$, sending in round $i-1$, and sending in round i in the following paragraphs.

At the end of round $i-2$ any symbol derivable by users in set $\boldsymbol{U}_{\mathbf{X}}^{i-2}$ is also derivable by the adversary. After generating new secret random coins at the beginning of round $i-1$, users in set $\boldsymbol{U}_{\mathbf{S}}^{i-1}$ can derive symbols, that the adversary cannot derive, from these new random coins and public symbols from their (exposed) state. We call such derivable symbols of types SK, K, and R that the adversary cannot derive *useful secrets*. Symbols of these types that are derivable by the adversary are called *useless secrets* (resulting in two complementary sets). Before sending in round $i-1$, new useful secrets of a user $u^* \in \boldsymbol{U}_{\mathbf{S}}^{i-1}$ are only derivable for u^* itself but not for any other user $u \in [n] \setminus \{u^*\}$. This is because the origin of these new useful secrets are the new secret random coins generated at the beginning of round $i-1$ and no communication took place after their generation yet. Hence, at sending in round $i-1$ users in set $\boldsymbol{U}_{\mathbf{S}}^{i-1}$ share no *compatible* useful secrets with other users. Secrets are called *compatible* if they are equal or if they are registered via rule f) under the same (main) secret key.

We formulate three observations: I) For deriving a public key pk from a set of type R symbols it is necessary according to grammar rule 4. and derivation rules g) and h) (with their inverse derivation guarantees) that its secret key sk (or one of its update-ancestors' secret key sk) is derivable from this set as well. II) For deriving a ciphertext c, encrypted to a public key pk, from a set of type R symbols it is necessary according to grammar rule 1. and derivation rule i) (with its inverse derivation guarantees) that this public key pk is derivable from it as well. III) Unifying all random coins generated by all users up to (including) round $i-1$ except those generated by user $u^* \in \boldsymbol{U}_{\mathbf{S}}^{i-1}$ in round $i-1$ forms a set of type R symbols from which all useful secrets at the beginning of round $i-1$ can be derived except those that are new to user u^* at that point. Combining these observations shows that at the beginning of round $i-1$ no user $u \neq u^*$ can derive public keys to useful secrets of user $u^* \in \boldsymbol{U}_{\mathbf{S}}^{i-1}$. This further implies that user u cannot derive ciphertexts encrypted to such public keys. As a result, the set of symbols sent by one user $u \in \boldsymbol{U}_{\mathbf{S}}^{i-1}$ in round $i-1$ contains no ciphertexts directed to useful secrets derivable by another user $u^* \in \boldsymbol{U}_{\mathbf{S}}^{i-1} \setminus \{u\}$ that would transport useful secrets between such users.

We further observe: According to the inverse derivation guarantees of rule c), both inputs to a dual PRF invocation must be derivable for deriving its output. As this requires a shared useful secret on input for deriving a shared useful secret as output, also a dual PRF establishes no shared (compatible) useful secrets in round $i-1$. All remaining derivation rules either output no secrets, or are *unidimensional*, meaning that they only immediately derive one (useful) secret

from another. As a result, also after receiving in round $i-1$ users in set $\boldsymbol{U}_{\mathbf{S}}^{i-1}$ share no compatible useful secrets.

Sampling random coins before sending in round i again produces no shared compatible useful secrets between users that shared none before. Hence, also before receiving in round i, users in set $\boldsymbol{U}_{\mathbf{S}}^{i-1}$ share no compatible useful secrets. We remark that our symbolic correctness and security definition requires for the given adversary that the shared group key derived in round i (after receiving) is a *useful secret*.

For quantifying the number of ciphertexts sent in round i, we define two *key graphs* $\mathcal{G}_i^{\text{before}}$ and $\mathcal{G}_i^{\text{after}}$ that represent useful secrets as nodes and derivations among them as edges. Secret y being derivable from secret x is represented by a directed edge from x to y. Although inspired by the proof technique of Micciancio and Panjwani [26], the use of key (derivation) graphs in our proof is entirely new.

Graph $\mathcal{G}_i^{\text{before}}$ includes a node for each useful secret that exists after receiving in round i and an edge for each derivation among them except for derivations possible only due to ciphertexts sent in round i. Graph $\mathcal{G}_i^{\text{after}}$ contains $\mathcal{G}_i^{\text{before}}$ and additionally includes edges for derivations possible due to ciphertexts sent in round i. Thus, the number of additional edges in $\mathcal{G}_i^{\text{after}}$ equals the number of sent ciphertexts in round i. Mapping our derivation rules to edges is highly non-trivial (e.g., each sent ciphertext must appear at most once). All details are in the full version [11].

The fact that users in set $\boldsymbol{U}_{\mathbf{S}}^{i-1}$ share no compatible useful secrets before receiving in round i finds expression in graph $\mathcal{G}_i^{\text{before}}$ as follows: Every such user $u \in \boldsymbol{U}_{\mathbf{S}}^{i-1}$ is represented by nodes in a set $\mathcal{V}_u^i$ that stand for its useful secret random coins from rounds $i-1$ and i (the latter only if u also sent in round i). For every pair of users $u_1, u_2 \in \boldsymbol{U}_{\mathbf{S}}^{i-1}$ with $u_1 \neq u_2$ there exists no node in graph $\mathcal{G}_i^{\text{before}}$ that is reachable via a path from a node in set $\mathcal{V}_{u_1}^i$ and a path from a node in set $\mathcal{V}_{u_2}^i$ simultaneously (including trivial paths). In contrast, every set $\mathcal{V}_u^i$ with $u \in \boldsymbol{U}_{\mathbf{S}}^{i-1}$ must contain a node from which a path in graph $\mathcal{G}_i^{\text{after}}$ reaches node v^* that represents the group key in round i.

In graph $\mathcal{G}_i^{\text{before}}$ node v^* was reachable via a path from nodes $\mathcal{V}_u^i$ of at most one user $u \in \boldsymbol{U}_{\mathbf{S}}^{i-1}$. Otherwise v^* would have been a compatible useful secret for two users in set $\boldsymbol{U}_{\mathbf{S}}^{i-1}$ before receiving in round i. Consequently, at least one edge per user $u^* \in \boldsymbol{U}_{\mathbf{S}}^{i-1} \setminus \{u\}$ must be included in $\mathcal{G}_i^{\text{after}}$ in addition to those contained in $\mathcal{G}_i^{\text{before}}$. Hence, $\mathcal{G}_i^{\text{after}}$ contains at least $|\boldsymbol{U}_{\mathbf{S}}^{i-1}| - 1$ more edges than $\mathcal{G}_i^{\text{before}}$, implying that at least $|\boldsymbol{U}_{\mathbf{S}}^{i-1}| - 1$ ciphertexts were sent in round i.

We now observe that invocations of algorithm snd in every round are independent of sets $\boldsymbol{U}_{\mathbf{X}}^j$ for all j, and invocations of algorithm snd in round i are independent of set $\boldsymbol{U}_{\mathbf{S}}^i$. As a consequence, every sender $u \in \boldsymbol{U}_{\mathbf{S}}^i$ must send $|\boldsymbol{U}_{\mathbf{S}}^{i-1}| - 1$ ciphertexts, anticipating the worst case that it is the only sender in that round. Therefore, $|\boldsymbol{U}_{\mathbf{S}}^i| \cdot (|\boldsymbol{U}_{\mathbf{S}}^{i-1}| - 1)$ ciphertexts are sent in (every) round i.

Interpretation This lower bound, formally proved in the full version of this article [11], describes the best case of communication complexity both within our model but partially also with respect to the real-world: it holds against very weak adversaries for significantly reduced functionality requirements of group

ratcheting without any form of required forward-secrecy. Lower bounds, induced by forward-secrecy for group key exchange [26], may furthermore apply to practical group ratcheting and therefore increase necessary communication complexity thereof.[9] We note that our result even applies to any two rounds between which no user sent.

Bypassing our lower bound is possible for constructions that exploit the algebraic structure of elements (which is forbidden in symbolic models), base on building blocks that we do not allow here (e.g., multiparty NIKE), or provide weaker security guarantees (e.g., recover from state exposures only with an additional delay in rounds).

For clarity we note that the key graph concept used here is independent of the tree structure of keys within our upper bound construction in Section 6.

6 Upper Bound of Communication Complexity

In order to overcome the deficiencies of existing protocols, we postpone the refresh of parts of the key material in the group by one operation. The resulting construction closely (up to a factor of $\approx \log(n/t)$) meets our communication complexity lower bound.

For computational security of group ratcheting, games $\text{KIND}^b_{\mathsf{GR}}$ from Section 3 are slightly adapted to additionally require immediate forward-secrecy. We note that the use of (a weak form of) kuPKE instead of standard PKE in our construction is only due to required forward-secrecy. Furthermore, the weak kuPKE used can be efficiently built from standard assumptions (see e.g., a construction from DDH in [24]).

6.1 Construction

Our construction uses ideas from the complete subtree method of broadcast encryption [27] and resembles concepts from TreeKEM [3,4]. More specifically, the construction bases on a static complete (directed) binary tree structure τ with n leaves (i.e., one leaf per group member), on top of which at every node, there is an evolving kuPKE key pair. The secret key at each of the n leaves is known only by the unique user that occupies that leaf. For the remaining nodes we maintain the invariant that the only secret keys in a user's state at a given time are those that are at nodes along the direct path of its corresponding leaf to the root of the tree.

We refer to the children of a node v in a tree as $v.c_0$ (left child) and $v.c_1$ (right child), and its parent as $v.p$. Furthermore we let $i, j, i > j$ be two rounds in which the set of sending group members is non-empty and there is no intermediate round l, $i > l > j$, with non-empty sending set. For simplicity in the description we define $j := i - 1$.

[9] We observe that if a group-ratcheting-pendant of the amortized $\log(n)$ lower bound for forward-secure group key exchange by Micciancio and Panjwani [26] applies as a factor on our lower bound, then our construction from Section 6 has optimal communication complexity.

Sending. To recover from state exposures, our construction lets senders in round $i-1$ refresh only their own individual leaf key pair. Senders in round i then refresh all remaining secret keys stored in the local states of round $i-1$ senders (i.e., for nodes on their direct paths to the root) on their behalf. This is illustrated in Figure 3. Note that (as explained below in paragraph *Receiving*) all group members collect the senders of round $i-1$ into a set $\boldsymbol{U}_{i-1}$ in the rcv algorithm of round $i-1$. Our construction, formally defined in Figure 4, accordingly lets all senders in a round perform five tasks:

1) To refresh their own individual secret key: Generate a fresh secret key for their corresponding leaf and send the respective public key to the group (lines 42-43, 63).
2) To refresh and rebuild direct paths of last round's senders: Sample a new seed for the leaf of each sender of the last round and encrypt it to the respective sender's (refreshed) leaf public key (lines 46-49). Then derive a seed for each non-leaf node on the direct paths from these leaves to the root using the new seeds at the leaves (line 50). Each seed will be used to deterministically generate a fresh key pair for its node.
3) To share refreshed secrets with members who did not send in the last round: Encrypt the new seed of each refreshed non-leaf node to the public key of its child from which it was not derived (lines 52-55, 58-61). Update the used public keys via kuPKE algorithm up (lines 56, 62).
4) To inform the group of changed public keys: Send all changed public keys to the group, including those for which seeds were renewed, and those that were updated via kuPKE (lines 50, 56, 62, 63).
5) Sample and encrypt a group key k for the round to all other users in the group (lines 44, 48, 54, 63).

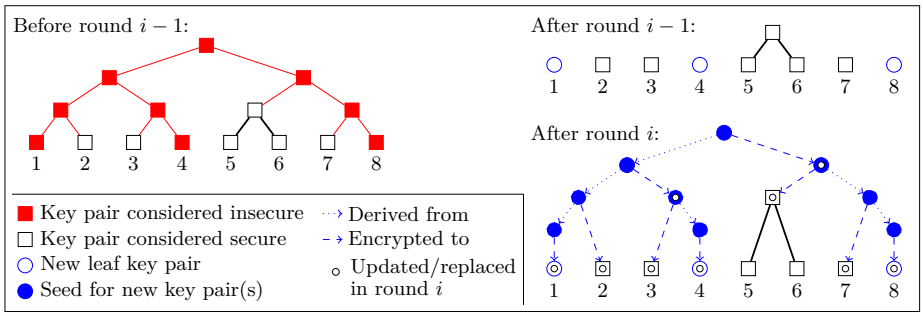

Fig. 3: Example tree for two rounds $i-1$ and i with $n=8$, $\boldsymbol{U}_{i-1}=\{1,4,8\}$, and $\boldsymbol{U}_i \neq \emptyset$. In round $i-1$, senders generate new key pairs for their leaves. In round i, senders generate seeds for all nodes considered insecure from round $i-1$ and replace leaf key pairs for round $i-1$ senders, as shown in the bottom-right corner.

In step 2), one seed is individually encrypted to each user in set $\boldsymbol{U}_{i-1}$ via public key encryption, which will allow them to reconstruct their direct path in

the tree. The purpose of this individual encryption is to let the recent senders forget their old (potentially exposed) secrets and use their fresh secret (which they generated during their last sending) to obtain new, secure secrets on their direct path.

We now describe how all remaining group members are able to rebuild the tree in their view. The reader is invited to follow the explanation and focus their attention on the tree in the lower right corner of Figure 3. In this tree, directed edges represent the derivation of a seed at a node from one of its children (dotted) or encryption of a seed at a node to one of its children (dashed). We consider the Steiner Tree $ST(\boldsymbol{U}_{i-1})$ induced by the set of leaves of users in $\boldsymbol{U}_{i-1}$. $ST(\boldsymbol{U}_{i-1})$ is the minimal subtree of the full tree that connects all of the leaves of $\boldsymbol{U}_{i-1}$ and the root; in the lower right corner tree of Figure 3, $ST(\boldsymbol{U}_{i-1})$ is the subtree of blue filled circles and edges between them. For each *degree-one node* v of $ST(\boldsymbol{U}_{i-1})$ (i.e., nodes with only one child in the Steiner Tree), its seed is encrypted to the public key of its child which is not in $ST(\boldsymbol{U}_{i-1})$. This seed can be used to derive some (possibly all) of the secret keys for the nodes on the direct path of v, including v itself (lines 51-56). We denote the set of such degree one nodes of the Streiner Tree as $ST(\boldsymbol{U}_{i-1})_1$ and the child of a node v in $ST(\boldsymbol{U}_{i-1})_1$ that is not in the Steiner Tree as $v.c_{\notin ST(\boldsymbol{U}_{i-1})}$.[10] For each *degree-two node* v of $ST(\boldsymbol{U}_{i-1})$ (i.e., nodes with two children in the Steiner Tree), its seed is encrypted to the public key of its right child (lines 57-62). We denote the set of such degree-two nodes of the Steiner Tree as $ST(\boldsymbol{U}_{i-1})_2$. All of these encrypted seeds are derived from the fresh leaf seeds of users in set $\boldsymbol{U}_{i-1}$ via prf computations, as explained below in paragraph *Construction Subroutines*.

Alongside the seeds, some randomly sampled associated data ad is also encrypted in the ciphertexts of the above paragraph (lines 52, 58). Public keys used for the encryption are afterwards updated with this associated data ad (lines 56, 62). Upon receipt, this associated data is used correspondingly to update the secret keys as well. Due to this mechanism, immediate forward-secrecy is achieved since secret keys stored in users' local states are updated as soon as they are used for decryption.

We refer to the union of nodes that are in the Steiner Tree with nodes that are children of degree-one nodes in the Steiner Tree as $CST = \{v : v \in ST(\boldsymbol{U}_{i-1}) \vee v = w.c_{\notin ST(\boldsymbol{U}_{i-1})} \forall w \in ST(\boldsymbol{U}_{i-1})_1\}$. For step 4) above, senders must publish the new public keys corresponding to all nodes of $CST(\boldsymbol{U}_{i-1})$ (lines 50, 56, 62, 63).

Receiving. For rounds in which no member sent, the recipients forward-securely derive symmetric keys (one output group key, and one saved key) from last round's secrets (lines 87-88). In addition, they assign $\boldsymbol{U}_i \leftarrow \boldsymbol{U}_{i-1}$ (line 68), so that senders of subsequent rounds can refresh the secrets of the senders of round $i-1$.

In case members sent in a round, a receiver determines the first message bc^* among all sent in this round, via some definite order (e.g., lexicographic). The

[10] We overload the set theoretic symbol $\notin$ here for brevity.

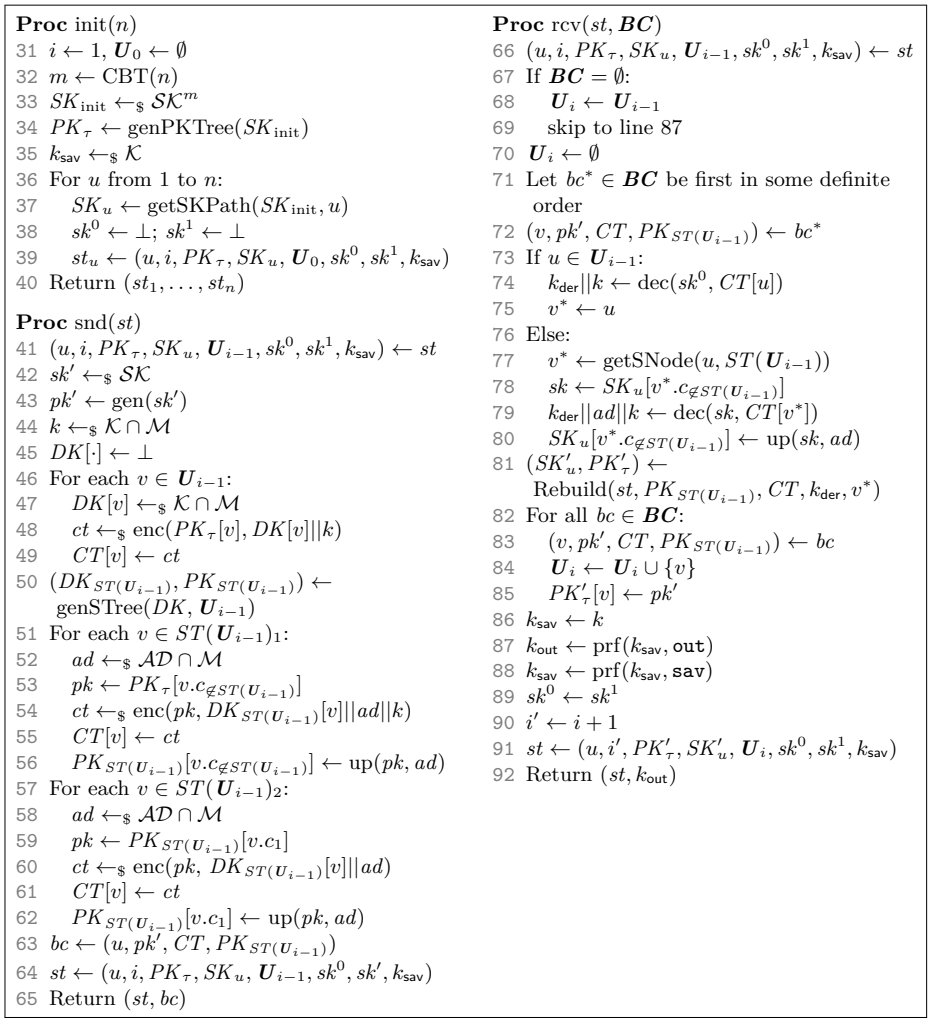

Fig. 4: Construction of concurrent group ratcheting in the computational model. $\mathrm{CBT}(n)$ calculates the number of nodes in a complete binary tree with n leaves. $\mathrm{getSNode}(u, ST(\boldsymbol{U}_{i-1}))$ finds the first node v on the direct path of u that is in $ST(\boldsymbol{U}_{i-1})$.

receiver then retrieves from this message the ciphertext set CT for decrypting the symmetric secret k and the first seed needed to rebuild the tree: If the receiver sent in the last active round (in which anyone sent), it uses its individual (fresh) secret key (lines 74-75). Otherwise, it uses the secret key of the first node on its direct path that is the child of some node in $ST(\boldsymbol{U}_{i-1})$ (lines 76-79). The decrypted seed, as well as the rest of CT, and the public keys of the Steiner Tree within bc^* are then used to rebuild the secret path for the receiver, as well as

the public key tree, as described below in paragraph *Construction Subroutines* (line 81). The resulting symmetric secret is then used to derive the output group key and a new saved key (as described above for rounds without ciphertexts).

Additionally, secret keys used to decrypt ciphertexts (including those as described in the *Construction Subroutines* paragraph below), are updated with the associated data that was also decrypted from the respective ciphertexts (lines 79, 80, 111, 112). Finally, all senders of the round are collected into $\boldsymbol{U}_i$ and their new public keys are saved (lines 82-85) in order to later achieve post-compromise security.

Construction Subroutines. In the common state initialization algorithm init, a complete binary tree of n leaves with a public key at each node is initialized using a list of corresponding secret keys SK_{init} with procedure $PK_\tau \leftarrow \text{genPKTree}(SK_{\text{init}})$ (line 34). Also, the secret keys along the direct path to the root of leaf u for each user are retrieved for that user, using $SK_u \leftarrow \text{getSKPath}(SK_{\text{init}}, u)$.

Figure 5 details the subroutines for genSTree and Rebuild (lines 50 and 81). Subroutine genSTree is used in the snd algorithm to compute the seeds and public keys at each node of the Steiner tree $ST(\boldsymbol{U}_{i-1})$ using the seeds $DK[v]$ sampled for the leaves $v \in \boldsymbol{U}_{i-1}$ (lines 46-49). For each $v \in \boldsymbol{U}_{i-1}$, the receiver uses $DK[v]$ to compute the node's secret key, public key, and (possibly) the seed to be used for its parent (lines 97-100), continuing up the tree until there has already been a seed generated for some node w on the path.

Rebuild is used in the rcv algorithm, by each user u to rebuild its "secret key path" as well as the "public key tree" using the public keys of the Steiner Tree $PK_{ST(\boldsymbol{U}_{i-1})}$, the set of ciphertexts CT, and the seed k_{der} obtained from CT corresponding to a node v^* in the tree. First, for every $v \in CST(\boldsymbol{U}_{i-1})$, the receiver sets its public key to that which is in the dictionary $PK_{ST(\boldsymbol{U}_{i-1})}$ (lines 104-105). Then, starting from node v^* using k_{der}, the receiver derives the secret key for v^* and a new seed for its parent if the node is the left child of its parent. Otherwise the receiver uses the secret key just derived to decrypt the seed to be used at its parent (lines 107-113). The receiver continues up the tree until the root is reached.

Efficiency. We here provide a short and simple proof of our communication complexity upper bound.[11]

Lemma 1. *For every round $i \in [q]$, the communication costs in an execution $(n, \boldsymbol{U}^0_{\mathbf{X}}, \boldsymbol{U}^1_{\mathbf{S}}, \boldsymbol{U}^1_{\mathbf{X}}, \ldots, \boldsymbol{U}^1_{\mathbf{S}}, \boldsymbol{U}^q_{\mathbf{X}})$ are*

$$|C[i]| = \mathcal{O}\left(|\boldsymbol{U}^i_{\mathbf{S}}| \cdot |\boldsymbol{U}^{i-1}_{\mathbf{S}}| \cdot \left(1 + \log\left(\frac{n}{|\boldsymbol{U}^{i-1}_{\mathbf{S}}|}\right)\right)\right).$$

[11] One might observe that using ideas from the Layered Subset Difference BE method [21] could lower the communication complexity of our construction, however we failed to do so due to potential security issues.

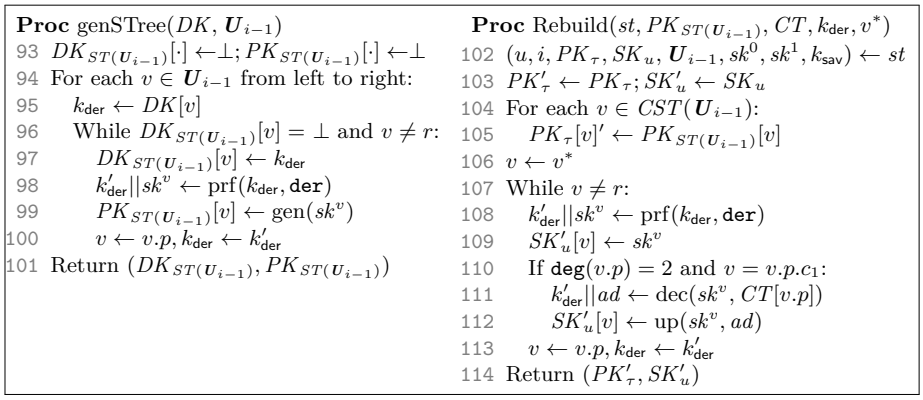

Fig. 5: Subroutines for construction upper bound. $\mathtt{deg}(v)$ refers to the degree of a node v in a tree, i.e. number of children.

We note that $|C[i]|$ denotes the number of sent items (i.e., ciphertexts and public keys) per round. Their individual length depends on the respectively deployed kuPKE scheme. (In a setting that defines a *security parameter*, the factor with which the communication costs are multiplied is (asymptotically) constant in this security parameter.)

Proof. We track communication of each user $u \in \boldsymbol{U}_{\mathbf{S}}^i$ that sends in round i. From this, the result follows easily. In round i, user u sends one ciphertext and one public key for each $v \in ST(\boldsymbol{U}_{\mathbf{S}}^{i-1})$ (plus an additional public key for at most one child c_v of each v). It is shown in [27] that $|ST(\boldsymbol{U}_{\mathbf{S}}^{i-1})_1| = \mathcal{O}\left(|\boldsymbol{U}_{\mathbf{S}}^{i-1}| \cdot \log\left(\frac{n}{|\boldsymbol{U}_{\mathbf{S}}^{i-1}|}\right)\right)$. Moreover, it follows from the analysis in [27] that $|ST(\boldsymbol{U}_{\mathbf{S}}^{i-1})_2| + |\boldsymbol{U}_{\mathbf{S}}^{i-1}| = \mathcal{O}\left(|\boldsymbol{U}_{\mathbf{S}}^{i-1}|\right)$. Since $ST(\boldsymbol{U}_{\mathbf{S}}^{i-1}) = ST(\boldsymbol{U}_{\mathbf{S}}^{i-1})_1 \cup ST(\boldsymbol{U}_{\mathbf{S}}^{i-1})_2 \cup \boldsymbol{U}_{\mathbf{S}}^{i-1}$,[12] we have accounted for each node $v \in ST(\boldsymbol{U}_{\mathbf{S}}^{i-1})$.

Therefore, each user $u \in \boldsymbol{U}_{\mathbf{S}}^i$ communicates $\mathcal{O}\left(|\boldsymbol{U}_{\mathbf{S}}^{i-1}| \cdot \left(1 + \log\left(\frac{n}{|\boldsymbol{U}_{\mathbf{S}}^{i-1}|}\right)\right)\right)$ information. □

Theorem 1 (informal). *Assuming secure kuPKE (as proposed in [24,3]) and PRF constructions, the construction of Figure 4 is a secure group ratcheting scheme according to the forward-secure variant of game* $\mathrm{KIND}^b_{\mathsf{GR}}$ *from Section 3, with security loss at most* $(q_{\mathrm{Round}}+1)\cdot((\lceil\log(n)\rceil+1)\cdot \mathrm{Adv}^{\mathrm{prfind}}_{\mathsf{PR}}(\mathcal{B}_{\mathsf{PR}}) + \lceil\log(n)\rceil \cdot \mathrm{Adv}^{\mathrm{kind}}_{\mathsf{UE}}(\mathcal{B}_{\mathsf{UE}}))$*, where* n *is the number of group members,* q_{Round} *is the number of executed rounds, and* $\mathrm{Adv}^{\mathrm{prfind}}_{\mathsf{PR}}(\mathcal{B}_{\mathsf{PR}})$, $\mathrm{Adv}^{\mathrm{kind}}_{\mathsf{UE}}(\mathcal{B}_{\mathsf{UE}})$ *are upper bounds on the advantage of any adversaries* $\mathcal{B}_{\mathsf{PR}}$, $\mathcal{B}_{\mathsf{UE}}$ *against the security of PRF and kuPKE, respectively.*

[12] We overload $\boldsymbol{U}_{\mathbf{S}}^{i-1}$ to also refer to the set of leaves corresponding to the users $u' \in \boldsymbol{U}_{\mathbf{S}}^{i-1}$.

For the formal version of this theorem and the full security proof, we refer the reader to our full version [11]. Below we provide a proof sketch that intuitively summarizes our proof idea.

Proof (sketch). Recall that in our construction, for each round i (with senders) initiated by the adversary, the initial secret key generated at each node in the Steiner Tree $ST(\boldsymbol{U}_{i-1})$ is derived via a PRF computation (lines 98,108). The key idea behind our proof is that we slowly replace these initial secret keys with keys that are drawn uniformly from the space of secret keys. Then, we replace all encryptions to such keys (lines 48,54,60) with fake ciphertexts that are independent of the actual contents of the message. Furthermore, in the rcv() algorithm of our hybrid experiments, we hardcode the associated data to be used to update the secret key to which it is encrypted (lines 80,112), so that all users maintain consistent views of the key pairs at each node v in τ, despite the fake ciphertexts.

However, we must be careful to only replace the secret keys and ciphertexts which the adversary cannot compute directly because of the corruption of some user $u \in [n]$. Specifically, after corruption of a user $u \in [n]$, we generate the secret keys along their direct path in τ as well as any ciphertexts encrypted to these secret keys as in Figure 4. For any node v in τ, we then wait until each of the corrupted users corresponding to the leaves of the subtree rooted at v send in a subsequent round. It is not until this point that we can again replace any of the secret keys and ciphertexts in our hybrids. This does not violate security because if some corrupted user u in the subtree rooted at v has not yet refreshed their leaf key pair in some round i, the adversary can trivially compute the secret key at v, as well as the output group secret of that round. Thus, the output group secret is not considered secure for round i anyway. Moreover, by forward secrecy of the kuPKE scheme's updates, any ciphertexts encrypted to *previous* versions of the key pair (i.e., before the latest update) of a node along the direct path of a corrupted user u are still secure. Thus all previous secret keys along the direct path of u and any previous output group secrets are still secure (provided that no other users were corrupted).

Now recall that the secret key of an interior node v in $ST(\boldsymbol{U}_{i-1})$ for some round i is generated via a PRF computation on a key output at one of the children of v (lines 98-100). Therefore, our hybrid experiments must proceed by first replacing the secret keys (resp. subsequent encryptions to them) of leaves in $ST(\boldsymbol{U}_{i-1})$ with uniformly random (resp. fake) values, followed by the secret keys of their parents, and so on, until we reach the root. When we reach the hybrids corresponding to the root, for all rounds in which the adversary cannot anyway trivially compute the uniformly random key encrypted to all users that will be used to derive the output group secret, all ciphertexts broadcast are independent of it. We finally add hybrids replacing the output group secret keys (line 87) and any intermediate saved keys for rounds with no senders (line 88) with uniformly random keys. Therefore, in our final hybrid, the output group secrets for non-trivially attackable rounds are uniformly random and independent of all ciphertexts broadcast throughout the protocol. □

6.2 Discussion

We shortly reflect on our construction, compare it to previous works, discuss its limitations with respect to the security model, and propose possible efficiency improvements.

The main purpose of our protocol is to give an upper bound that confirms our lower bound, but not to provide optimal security and maximal functionality under concurrency. Nevertheless, our construction provides the same security as parallel pairwise Signal executions, i.e. FS and PCS one round with non-empty sender set after all exposed users updated their states. In addition, it provides full concurrency for user updates unlike those in [14,32,8,3,4,5].

When using a variant of our construction for dynamic groups, removed members in such groups may maliciously store secrets that they saw during their membership for breaking confidentiality of group secrets after their membership. Effectively solving this problem—discussed as "double-join"—could be achieved by using ideas from protocols constructed for dynamic groups, such as MLS and Tainted TreeKEM. Without these ideas, it would be required that siblings of all removed users that are still in the group issue state updates before any removed user would be unable to derive the output secrets. Yet, as we discuss below, dynamic member changes appear to happen rather seldom in many practical applications such that this restriction might be insignificant.

Our security model is somewhat weak: we require an honest (but curious) mechanism that clocks rounds, we do not allow the adversary access to random coins used by senders in a round that are not saved to their state, and we do not allow the adversary to alter broadcast messages. Clock synchronization could, however, be rather coarse (resulting in long round periods) as our protocol's speedup in reaching PCS, compared to non-concurrent alternatives that require members to update their states one after another, is already significant. Furthermore, we note that all members processing all ciphertexts in a round (as defined in our model) is not mandatory but allows for immediate forward-secrecy due to kuPKE key pair updates. Processing all previous ciphertexts before sending is usually also unproblematic as sending anyways requires a user to come online, such that all cryptographic operations can be executed at that moment. Especially for reaching authentication and handling out-of-order receipts, tools that are independent of our core state update mechanism can be added (maybe even generically) to our construction. The problem of weak random coins is indeed an open problem for concurrent group ratcheting that we leave for future research.

As stated earlier, it is not ultimately clear whether our lower bound or upper bound is loose (or even both of them). One technique to improve our upper bound would be to utilize more sophisticated broadcast encryption methods than the Complete Subtree method [27], such as the Layered Subset Difference method [21] or techniques from the recently proposed optimal broadcast encryption scheme [1], while still preserving security. Additionally, if one allows a slight relaxation in the model by allowing for delayed PCS, i.e. PCS in some $\Delta > 1$ rounds, then better communication complexity could be achieved. This is because if users update their state in a given round i by publishing a fresh

public key, other users could send secrets to these users to help them recover in all rounds $i' \in \{i+1, i+2, \ldots, i+\Delta\}$, spreading out the communication costs across these rounds and allowing for some adaptivity between senders therein.

6.3 Insights for Practice

We shortly summarize concepts from our construction that could enhance, and insights from our lower bound that could influence real-world protocols (like the MLS initiative's design).

Almost-immediate PCS As mentioned many times before, immediate PCS under t-concurrency appears to require t-party NIKE (which is currently inaccessible). Postponing the update of *shared* secrets to a reaction in the next protocol execution step, as implemented in our construction, bypasses this problem. The major advantages of this bypass are a significant speedup for PCS, compared to sequential state updates, and a maintained balanced tree structure, compared to tree modifications, resulting in a reduced tree depth, or group partitions. An open question remains to analyze our scheme's resilience against weak randomness.

Static Groups are Practical Some deficiencies of our protocol are only relevant in dynamic settings. In contrast, constant groups can benefit from this construction significantly as it maintains communication complexity in all cases nearly optimally. We emphasize that many groups in real-world applications indeed seldom or never change the set of members (e.g., family groups, friendship group, smaller working groups, etc).

To resolve issues with respect to membership changes, the mechanism proposed in Tainted TreeKEM [4] could be applied on path updates in our protocol. Thereby, the "double-join"-problem could be prevented.

Better Solutions In the light of our lower bound, finding better solutions for reaching PCS under concurrency seems very complicated, if not unlikely. The set of permitted building blocks in our symbolic model is very powerful, the functionality required by constructions in this setting is very restricted, and the adversarial power in the lower bound security definition is very limited. Hence, it seems necessary to utilize "more exotic" primitives or relax the required PCS guarantees for obtaining better constructions.

References

1. Agrawal, S., Yamada, S.: Optimal broadcast encryption from pairings and LWE. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 13–43. Springer, Heidelberg (May 2020)
2. Alwen, J., Coretti, S., Dodis, Y.: The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 129–158. Springer, Heidelberg (May 2019)

3. Alwen, J., Coretti, S., Dodis, Y., Tselekounis, Y.: Security analysis and improvements for the IETF MLS standard for group messaging. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 248–277. Springer, Heidelberg (Aug 2020)
4. Alwen, J., Capretto, M., Cueto, M., Kamath, C., Klein, K., Pascual-Perez, G., Pietrzak, K., Walter, M.: Keep the dirt: Tainted treekem, an efficient and provably secure continuous group key agreement protocol. Cryptology ePrint Archive, Report 2019/1489 (2019), https://eprint.iacr.org/2019/1489
5. Alwen, J., Coretti, S., Jost, D., Mularczyk, M.: Continuous group key agreement with active security. In: Theory of Cryptography - 18th International Conference, TCC 2020, November 15-19, 2020, Proceedings. Lecture Notes in Computer Science, Springer (2020)
6. Balli, F., Rösler, P., Vaudenay, S.: Determining the core primitive for optimally secure ratcheting. In: Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Virtual, December 7-11, 2020, Proceedings. Lecture Notes in Computer Science (2020)
7. Barnes, R., Beurdouche, B., Millican, J., Omara, E., Cohn-Gordon, K., Robert, R.: The messaging layer security (mls) protocol (2020), https://datatracker.ietf.org/doc/draft-ietf-mls-protocol/
8. Barnes, R., Beurdouche, B., Millican, J., Omara, E., Cohn-Gordon, K., Robert, R.: The messaging layer security (MLS) protocol draft-ietf-mls-protocol-09. Internet-draft (September 2020), https://www.ietf.org/archive/id/draft-ietf-mls-protocol-09.txt
9. Bellare, M., Lysyanskaya, A.: Symmetric and dual PRFs from standard assumptions: A generic validation of an HMAC assumption. Cryptology ePrint Archive, Report 2015/1198 (2015), http://eprint.iacr.org/2015/1198
10. Bellare, M., Singh, A.C., Jaeger, J., Nyayapati, M., Stepanovs, I.: Ratcheted encryption and key exchange: The security of messaging. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 619–650. Springer, Heidelberg (Aug 2017)
11. Bienstock, A., Dodis, Y., Rösler, P.: On the price of concurrency in group ratcheting protocols. Cryptology ePrint Archive, Report 2020/1171 (2020), https://eprint.iacr.org/2020/1171
12. Boneh, D., Glass, D., Krashen, D., Lauter, K., Sharif, S., Silverberg, A., Tibouchi, M., Zhandry, M.: Multiparty non-interactive key exchange and more from isogenies on elliptic curves. Cryptology ePrint Archive, Report 2018/665 (2018), https://eprint.iacr.org/2018/665
13. Boneh, D., Silverberg, A.: Applications of multilinear forms to cryptography. Cryptology ePrint Archive, Report 2002/080 (2002), http://eprint.iacr.org/2002/080
14. Cohn-Gordon, K., Cremers, C., Garratt, L., Millican, J., Milner, K.: On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 1802–1819. ACM Press (Oct 2018)
15. Cohn-Gordon, K., Cremers, C.J.F., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the signal messaging protocol. In: 2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017. pp. 451–466. IEEE (2017)

16. Cohn-Gordon, K., Cremers, C.J.F., Garratt, L.: On post-compromise security. In: IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016. pp. 164–178. IEEE Computer Society (2016)
17. Cremers, C., Hale, B., Kohbrok, K.: Efficient post-compromise security beyond one group. Cryptology ePrint Archive, Report 2019/477 (2019), https://eprint.iacr.org/2019/477
18. Dolev, D., Yao, A.C.C.: On the security of public key protocols (extended abstract). In: 22nd FOCS. pp. 350–357. IEEE Computer Society Press (Oct 1981)
19. Durak, F.B., Vaudenay, S.: Bidirectional asynchronous ratcheted key agreement with linear complexity. In: Attrapadung, N., Yagi, T. (eds.) IWSEC 19. LNCS, vol. 11689, pp. 343–362. Springer, Heidelberg (Aug 2019)
20. Gennaro, R., Trevisan, L.: Lower bounds on the efficiency of generic cryptographic constructions. In: 41st FOCS. pp. 305–313. IEEE Computer Society Press (Nov 2000)
21. Halevy, D., Shamir, A.: The LSD broadcast encryption scheme. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 47–60. Springer, Heidelberg (Aug 2002)
22. Impagliazzo, R., Rudich, S.: Limits on the provable consequences of one-way permutations. In: 21st ACM STOC. pp. 44–61. ACM Press (May 1989)
23. Jaeger, J., Stepanovs, I.: Optimal channel security against fine-grained state compromise: The safety of messaging. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 33–62. Springer, Heidelberg (Aug 2018)
24. Jost, D., Maurer, U., Mularczyk, M.: Efficient ratcheting: Almost-optimal guarantees for secure messaging. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 159–188. Springer, Heidelberg (May 2019)
25. Jost, D., Maurer, U., Mularczyk, M.: A unified and composable take on ratcheting. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part II. LNCS, vol. 11892, pp. 180–210. Springer, Heidelberg (Dec 2019)
26. Micciancio, D., Panjwani, S.: Optimal communication complexity of generic multicast key distribution. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 153–170. Springer, Heidelberg (May 2004)
27. Naor, D., Naor, M., Lotspiech, J.: Revocation and tracing schemes for stateless receivers. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 41–62. Springer, Heidelberg (Aug 2001)
28. Perrin, T., Marlinspike, M.: The double ratchet algorithm (2016), https://signal.org/docs/specifications/doubleratchet/
29. Poettering, B., Rösler, P.: Asynchronous ratcheted key exchange. Cryptology ePrint Archive, Report 2018/296 (2018), https://eprint.iacr.org/2018/296
30. Poettering, B., Rösler, P.: Towards bidirectional ratcheted key exchange. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 3–32. Springer, Heidelberg (Aug 2018)
31. Rösler, P., Mainka, C., Schwenk, J.: More is less: on the end-to-end security of group chats in Signal, WhatsApp, and Threema. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 415–429. IEEE (2018)
32. Weidner, M.: Group messaging for secure asynchronous collaboration. Ph.D. thesis, MPhil Dissertation, 2019. Advisors: A. Beresford and M. Kleppmann, 2019 (2019), https://mattweidner.com/acs-dissertation.pdf